LLM-Symbolic Integration for Robust Temporal Tabular Reasoning

Anonymous ACL submission

Abstract

Temporal tabular question answering presents a significant challenge for Large Language Models (LLMs), requiring robust reasoning over structured data—a task where traditional prompting methods often fall short. These methods face challenges such as memorization, sensitivity to table size, and reduced performance on complex queries. To overcome these limitations, we introduce TEMPTABQA-C , a synthetic dataset designed for systematic and controlled evaluations, alongside a symbolic intermediate representation that transforms tables into database schemas. This structured approach allows LLMs to generate and execute SQL queries, enhancing generalization and mitigating biases. By incorporating adaptive few-shot prompting with contextually tailored examples, our method achieves superior robustness, scalability, and performance. Experimental results consistently highlight improvements across key challenges, setting a new benchmark for robust temporal reasoning with LLMs.

1 Introduction

Large Language Models (LLMs) have revolutionized natural language processing, achieving remarkable success across a variety of tasks. However, answering questions about temporal tabular data task that requires precise reasoning over structured information with time-based attributes remains a significant challenge. This capability is crucial in fields such as finance, healthcare, and policymaking, where actionable insights often depend on understanding and analyzing evolving datasets. Yet, existing methods often fall short, struggling with complex queries, large datasets, and scenarios that require nuanced reasoning. An example of such task, with a long table and corresponding query and it's answer is shown in 1. These limitations underscore the urgent need for robust, scalable, and interpretable solutions.

Áron Szilágyi

Personal information		
Born	14 January 1990 (age 34)	
Olympic Games		
G	2012 London	Individual
G	2016 Rio de Janeiro	Individual
G	2020 Tokyo	Individual
S	2024 Paris	Team
B	2020 Tokyo	Team
World Championships		
G	2007 Saint Petersburg	Team
G	2022 Cairo	Individual
G	2023 Milan	Team
S	2016 Rio de Janeiro	Team
B	2023 Milan	Individual
European Games		
B	2023 Kraków–Małopolska	Individual
European Championships		
G	2015 Montreux	Individual
G	2018 Novi Sad	Team
G	2022 Antalya	Team
G	2024 Basel	Team
S	2013 Zagreb	Team

Q.** In which year did Áron Szilágyi achieve his personal highest number of gold medal wins?* ***Gold Answer. *2022*

Direct prompt (CoT): In 2020, Áron Szilágyi won a Gold Medal in the Individual format at the Tokyo Olympics. This was his most successful year in terms of highest Gold Medal wins since he did not win multiple Gold Medals in any other format or competition. **Answer:** 2020.

Symbolic Intermediate Representation (SQL):

```
WITH gold_medal_counts AS (
    SELECT m.year, COUNT(m.medal_id) AS
        gold_medals
    FROM Medal m JOIN Format f ON m.format_id = f
        .format_id
    JOIN Tournament t ON f.tournament_id = t.
        tournament_id
    JOIN Athlete a ON t.athlete_id = a.athlete_id
    WHERE a.name = 'Áron Szilágyi' AND m.type = '
        MedalGold'
    GROUP BY m.year )
SELECT year FROM gold_medal_counts
WHERE gold_medals = (SELECT MAX(gold_medals) FROM
     gold_medal_counts);
```

Answer: 2022

Figure 1: Structured table of Áron Szilágyi's achievements with question and answers. Direct prompting fails, whereas, Symbolic Intermediate Representation give correct answer.

A key obstacle lies in the lack of benchmarks that adequately capture the complexity and diversity of temporal reasoning tasks. Existing benchmarks, typically created manually, are inconsistent and fail to provide the variability needed to thoroughly evaluate models. Without rigorous evaluation frameworks, it becomes difficult to diagnose weaknesses or ensure models can handle real-world scenarios. This raises an essential question: *How can we design benchmarks that effectively evaluate temporal reasoning across a range of challenging contexts?*

Equally important is the need for robust methods. Many existing approaches rely on direct prompting, which often depends on heuristics and memorized patterns rather than true reasoning. This results in semantic biases and poor performance in demanding scenarios, such as counterfactual reasoning, large table contexts, or multi-step queries. This leads to a second critical question: *How can we develop methods that remain robust across diverse table structures, dynamic data, and complex queries?*

To address these challenges, we propose a comprehensive framework that reimagines how LLMs approach temporal tabular data. At its core is TEMPTABQA-C , a synthetic dataset generation method designed to fill the gaps in existing benchmarks. TEMPTABQA-C provides precise control over data characteristics, enabling consistent and systematic evaluation across a wide range of scenarios, including counterfactual reasoning and intricate temporal queries. Building on this foundation, we introduce a **symbolic intermediate representation approach** that transforms unstructured tables into structured database schemas. LLMs are guided to generate SQL queries based on these schemas, which are executed to produce accurate answers. E.g. in Figure 1, the SQL query serves as a symbolic representation and provides the correct answer, whereas direct prompting fails on given query. This structured pipeline reduces semantic biases, enhances interpretability, and significantly improves the generalization of models across different table configurations. Additionally, we incorporate **adaptive few-shot prompting**, a dynamic approach that selects contextually relevant examples tailored to each query. This method overcomes the limitations of static examples, further improving the robustness of the system in complex scenarios.

Our experiments demonstrate that this framework delivers substantial improvements over direct prompting methods. It excels in critical areas such as counterfactual reasoning, scalability to larger datasets, and the handling of complex queries. Beyond these technical advancements, our work establishes a new benchmark for temporal tabular question answering by addressing fundamental weaknesses in existing approaches and introducing innovative tools for evaluation and reasoning. These contributions pave the way for building more interpretable, scalable, and robust AI systems with implications for critical real-world applications. Our contributions are as follows:

1. We introduce TEMPTABQA-C , a synthetic dataset designed for **precise and robust evaluation** of temporal tabular reasoning across diverse and challenging scenarios.
2. We analyze the **limitations of direct prompting**, including reliance on **memorization**, sensitivity to **table size**, and struggles with **complex multi-step or counterfactual** reasoning.
3. We propose a **symbolic intermediate representation approach** that enhances interpretability, reduces biases, and improves generalization by guiding LLMs to generate and execute SQL queries on structured schemas.
4. We enhance this approach with **adaptive few-shot prompting**, enabling context-specific example selection for improved flexibility and performance in diverse scenarios.

To support future research, we will release the TEMPTABQA-C dataset and source code (prompts etc) upon acceptance.

2 The TEMPTABQA-C Dataset

The TEMPTABQA-C dataset is a large-scale, semi-automatically generated resource designed for evaluating temporal reasoning in Large Language Models (LLMs). It provides a benchmark for analyzing the temporal qualities of LLMs by enabling controlled variations in data characteristics, making it superior to traditional human-curated datasets. This section describes the dataset creation process, its schema, and key characteristics.

2.1 TEMPTABQA-C Creation Pipeline

The creation of TEMPTABQA-C follows a systematic pipeline to extract, structure, and store temporal information from Wikipedia infoboxes. Below, we describe the steps involved in detail.

Extracting Temporal Information. Temporal information about athletes, tournaments, events, and achievements is extracted from Wikipedia infoboxes. These tables contain attributes such as *"Name," "Date of Birth," "Tournaments Played,"* and *"Medals Won,"* which are programmatically extracted and input into a relational database using a predefined schema. This step ensures that the raw tabular data is converted into a structured format for efficient querying and storage.

Relational Database Creation. The structured temporal data is converted into a relational database schema to enable efficient storage and querying. The schema is designed to represent key entities and their relationships comprehensively:

- **Athlete Table:** Contains a unique `athlete_id` and the corresponding athlete's name.
- **Personal Information Table:** Captures birth year, month, and day for each athlete, linked to the `athlete_id`.
- **Tournament Table:** Stores tournament details, such as the name (e.g., "Olympic Games") and the `athlete_id`.
- **Format Table:** Represents event formats (e.g., "100m Freestyle"), linked to tournaments through `tournament_id`.
- **Medal Table:** Documents medals, including type (e.g., "Gold"), year (e.g., "2016"), and location (e.g., "Rio de Janeiro"), linked to formats through `format_id`.

This schema ensures all entities are interconnected via primary and foreign keys, enabling complex queries like calculating an athlete's age at the time of their first medal or comparing performance across tournaments.

Question and Answer Generation Questions are generated using predefined templates filled with key attributes from the relational database. Templates capture a wide range of temporal reasoning scenarios, such as:

- At what age did [Athlete] win his most recent [Tournament] [Medal Type]?
- At what age did Michael Phelps win his most recent Pan Pacific Championships Silver Medal?

To generate answers, the relational database is queried using SQL-based logic, which systematically retrieves the necessary information. For instance, answering a question about the age of an athlete during a specific tournament involves retrieving the athlete's birth year and the tournament year from the database and calculating the difference. Similarly, questions about medal counts or locations are answered by aggregating or filtering data from the tables.

The SQL-based logic is generalized across various question types, allowing the generation of thousands of unique question-answer pairs. Examples include:

- At what age did Michael Phelps win his most recent Olympic Games Silver Medal? Answer: 29
- In which city did Caeleb Dressel win his most recent Olympic Games Silver Medal? Answer: Tokyo

This approach ensures the dataset is both scalable and robust for evaluating temporal reasoning in LLMs.

2.2 TEMPTABQA-C Composition and Splits

The TEMPTABQA-C dataset is divided into **Original** and **CounterFact** questions, with each category further subdivided based on table size and question reasoning difficulty. This structure enables fine-grained and comprehensive evaluations.

Original Questions. Original questions are derived directly from the structured database and are categorized as follows:

1. Table size: we make questions on the table with varied sizes: (a.) **Small Tables:** Contain concise data, typically representing athletes with fewer medals, (b.) **Large Tables:** Contain extensive data, often representing athletes with a larger number of medals.

2. Question Complexity: we answer questions on varied difficulty some requiring complex multi-hop reasoning: (a.) **Easy:** Require basic facts retrieval or single-step reasoning. E.g.: *"How many formats has Michael Phelps played?"*, (b.) **Medium:** Involve multi-step reasoning, such as calculations or comparisons. E.g.: *"At what age did Michael Phelps win his most recent Olympic Silver Medal?"*, and (c.) **Hard:** Demand complex reasoning, temporal analysis, and synthesis of multiple facts. E.g.: *"What is the shortest time span (in years) within which Michael Phelps won gold, silver, and bronze medals in the same format across any tournament?"*

Counterfactual Questions. Counterfactual questions modify specific facts in the original dataset while maintaining the same categorization based on table size and difficulty of the reasoning of the questions. This design challenges models to reason effectively under hypothetical scenarios.

Significance of TEMPTABQA-C . The dataset offers several unique advantages: (a.) **Controlled Evaluation:** Provides a framework for systematically testing LLMs across diverse data characteristics., (b.) **Scalability:** Comprises over 200,000 questions spanning a wide range of contexts and complexities., and (c.) **Fine-Grained Analysis:** Facilitates benchmarking of model biases and limitations, particularly for temporal reasoning.

By providing a controlled, scalable, and diverse dataset, TEMPTABQA-C establishes a robust foundation for advancing research on temporal reasoning in LLMs.

3 Experimental Setup

We designed experiments to address the following research questions:

1. **Robustness to Counterfactual Data**: How robust are direct LLM prompts to counterfactual data, and can symbolic intermediate representations improve this?
2. **Handling Large Tables**: Can a symbolic intermediate representation outperform direct prompting when applied to larger tables?
3. **Impact of Question Complexity**: How does increasing question complexity impact the performance of these two approaches?

To answer these questions, we evaluated two core approaches: **Direct Prompting** and **Symbolic Intermediate Representation**. Each approach was evaluated under three prompting configurations:

- **Zero-shot**: The model is prompted without examples, relying solely on its pretraining. For symbolic intermediate representation, this involves generating SQL queries without any in-context examples.
- **Non-Adaptive Few-shot**: This setup provides a fixed set of six example question-answer pairs with the prompt. These examples remain constant across all test questions, irrespective of their context. For symbolic intermediate representation, this includes fixed natural language-to-SQL mappings.
- **Adaptive Few-shot**: In this approach, six examples are dynamically chosen for each test question based on their relevance to the specific question. This ensures that the few-shot examples align closely with the current question's structure and content. For symbolic intermediate representation, this involves dynamically selecting natural language-to-SQL examples tailored to the test question.

In the Direct Prompting setup, models are queried directly in natural language, and the answers are returned as plain text. In the Symbolic Intermediate Representation setup, models generate SQL queries based on the question and context, which are executed on a structured database to obtain answers.

We used the TEMPTABQA-C dataset, which includes Original, counterfactual, and question difficulty (Easy, Medium, Hard) splits, along with small and large table contexts. Models were evaluated using Exact Match Score (EMS) [1], focusing on the following key splits:

- **Original vs. Counterfactual**: We examined whether the gap between Original and Counterfactual data reduces as we move toward symbolic intermediate reasoning.
- **Large Table vs. Small Table**: We evaluated if the gap between large and small tables decreases with symbolic intermediate reasoning.
- **Performance by Question Complexity**: We analyzed performance trends across Easy, Medium, and Hard questions, particularly the improvement brought by symbolic intermediate reasoning.

Through these experiments, we aim to demonstrate that symbolic intermediate reasoning reduces sensitivity to counterfactual data, scales better with table size, and handles increasing question complexity more effectively than direct prompting.

TEMPTABQA-C Test set. In order to evaluate the LLM's we created a subset of the TEMPTABQA-C dataset having the following number of question per category:

Category	#Examples	Category	#Examples
Original	578	Easy	732
Counterfactual	699	Medium	507
Small Table	855	Hard	719
Large Table	538	**Total**	**5067**

Table 1: Dataset Splits and Their Number of Examples.

In our test set, the average context length for the **Small Table Split** is **53.80** words when using the infobox as the context, while for the **Large Table Split**, it increases significantly to **348.85 words.**

4 Results and Analysis

In this section, we present the results for GPT-4o and Gemini 1.5 Pro. Additionally, we evaluated

[1]We also used a relaxed version of EMS (REMS), with similar results detailed in the appendix.

Gemini 1.5 Flash, GPT-4o Mini, Mixtral, Llama 3.1 70B, Code Llama, and SQL Coder, which demonstrated similar trends. The results of these additional experiments are included in the Appendix.

4.1 Robustness on Counterfactual Data.

To evaluate counterfactual robustness, we compare model performance on original and counterfactual datasets. Table 2 and 3 summarizes these results, including the performance gap (Δ) between the original and counterfactual performance for GPT 4o and Gemini-1.5-Pro.

Method	Adaptive	Original	CounterFact	Δ
Direct (CoT)	×	53.95	41.91	12.04
Direct (CoT)	✓	54.92	41.7	13.22
SQL	×	61.36	60.4	0.96
SQL	✓	**68.04**	**67.02**	**1.02**

Table 2: Original vs Counterfactual for GPT-4o.

Method	Adaptive	Original	CounterFact	Δ
Direct (CoT)	×	59.01	46.87	12.14
Direct (CoT)	✓	60.23	49.04	11.19
SQL	×	67.76	63.63	4.13
SQL	✓	**73.04**	**73.58**	**0.54**

Table 3: Original vs Counterfactual for Gemini 1.5 Pro.

Analysis: Comparing the performance of GPT-4o and Gemini 1.5 Pro across original and counterfactual datasets provides valuable insights into the robustness of Direct Prompting and SQL-based reasoning methods. *A model that truly reasons about data should not be affected by the origin of the data.*

However, for Direct Prompting, both models exhibit significant performance gaps between the original and counterfactual datasets, indicating a heavy reliance on memorized knowledge rather than robust reasoning capabilities. For GPT-4o, the performance gaps are 12.04 (non-adaptive) and 13.22 (adaptive), while Gemini 1.5 Pro shows slightly larger gaps of 12.14 and 11.19, respectively. Notably, the adaptive approach improves the performance on original data but increases sensitivity to counterfactuals for GPT-4o. In contrast, Gemini 1.5 Pro's adaptive Direct Prompting reduces the gap slightly but still fails to address the core issue of data sensitivity.

On the other hand, SQL-based methods, demonstrate superior robustness in both models, with performance gaps significantly smaller than those in Direct Prompting. For GPT-4o, the non-adaptive SQL gap is just 0.96, and the adaptive SQL gap is 1.02. Similarly, for Gemini 1.5 Pro, the non-adaptive SQL gap is 4.13, and the adaptive SQL gap reduces further to 0.54, showcasing its capability to reason effectively across different datasets. The use of symbolic intermediate representations in SQL methods explains this robustness, as these approaches operate independently of the data origin, focusing instead on schema-driven reasoning.

Finally, the adaptive approach enhances performance across both methods and models, particularly for SQL. For example, in GPT-4o, adaptive SQL improves counterfactual performance by 6.62 points compared to non-adaptive SQL, while in Gemini 1.5 Pro, it further narrows the performance gap to an almost negligible 0.54 points. This highlights the critical role of dynamic, context-sensitive few-shot examples in enhancing model reasoning capabilities and robustness across diverse datasets.

4.2 Impact of Table Size.

To evaluate the impact of data size, we compare model performance on small and large datasets. Table 4 and 5 presents the results, including the gap between small and large datasets for GPT 4o and Gemini 1.5 Pro.

Method	Adaptive	Small	Large	Δ
Direct (CoT)	×	71.11	46.84	24.27
Direct (CoT)	✓	73.92	48.88	25.04
SQL	×	71.93	70.82	1.11
SQL	✓	**72.16**	**73.23**	**1.07**

Table 4: Small Table vs Large Table for GPT-4o

Method	Adaptive	Small	Large	Δ
Direct (CoT)	×	65.79	43.86	21.93
Direct (CoT)	✓	66.26	41.86	24.40
SQL	×	62.22	54.65	7.57
SQL	✓	**71.47**	**72.43**	**0.96**

Table 5: Small Table vs Large Table for Gemini 1.5 Pro

Analysis: *A model capable of genuine reasoning should operate independently of data size. For example, the correctness of an SQL query's result is unaffected by the size of the tables—it impacts only the computation time, not the quality of the outcome.* However, the trends for small vs. large tables closely mirror those observed in the original vs. counterfactual analysis. Direct Prompting shows significant performance drops with larger tables, with GPT-4o and Gemini 1.5 Pro exhibiting gaps of 24.27 and 21.93 in non-adaptive settings, respectively. This underscores the method's sensitivity to data complexity and dependence on memory.

In contrast, SQL-based methods demonstrate remarkable robustness, maintaining minimal performance gaps across table sizes (e.g., 1.07 for adaptive SQL in GPT-4o and 0.96 in Gemini 1.5 Pro). This resilience stems from schema-driven reasoning, which abstracts away from the data's size or origin [2]. Adaptive few-shot examples further enhance performance, particularly for SQL-based methods, allowing them to consistently deliver high accuracy even with larger tables.

These findings emphasize that Direct Prompting struggles with data complexity and scale, mirroring its limitations in counterfactual settings. SQL-based methods, on the other hand, exemplify robustness and scalability by leveraging schema-driven symbolic representations that are agnostic to data size or source. The dynamic selection of adaptive examples further strengthens their reliability, making them a superior choice for reasoning over complex and evolving datasets.

4.3 Effect of question complexity.

To evaluate question complexity effects, we compare model performance on Easy, Medium, and Hard questions. Table 6 and 7 summarizes the results for GPT-4o and Gemini-1.5-Pro respectively.

Method	Adaptive	Easy	Medium	Hard
Direct (CoT)	✗	71.18	63.12	53.35
Direct (CoT)	✓	74.38	63.91	54.17
SQL	✗	78.58	75.54	62.62
SQL	✓	**79.83**	**73.57**	**66.32**

Table 6: Easy, Medium, and Hard results for GPT-4o

Method	Adaptive	Easy	Medium	Hard
Direct (CoT)	✗	73.26	60.71	57.43
Direct (CoT)	✓	78.04	66.43	59.28
SQL	✗	80.86	70.33	59.59
SQL	✓	**75.86**	**71.47**	**59.24**

Table 7: Easy, Medium, and Hard results on Gemini 1.5 Pro

Analysis: Performance consistently declines across all models and settings as question complexity increases from Easy to Hard, aligning with previous findings on the influence of data size and complexity. *While such a drop is expected for both models and humans (though less severe for the latter), the key question is: can we do better and reduce this decline?* Direct Prompting struggles as question complexity increases, with significant drops in accuracy (e.g., from 71.18 to 53.35 for non-adaptive GPT-4o). Adaptive prompting slightly mitigates this decline but remains limited in handling complex queries effectively.

SQL-based methods demonstrate greater resilience to complexity, maintaining higher accuracy across all levels. For example, non-adaptive SQL in GPT-4o drops moderately from 78.58 (Easy) to 62.62 (Hard), while adaptive SQL narrows this gap further, achieving 66.32 for Hard questions. Similarly, Gemini 1.5 Pro exhibits stable performance with SQL, with adaptive settings providing consistent improvements, particularly for harder questions.

These results reinforce SQL's robustness through schema-driven reasoning, which abstracts complexity and reduces reliance on memorization. Adaptive prompting enhances performance across all methods, particularly in SQL-based approaches, where tailored examples improve the model's ability to handle challenging queries. This underscores the importance of structured reasoning and adaptive techniques for tackling increasing data and query complexity effectively.

5 What Did We Learn?

1. Impact of Symbolic Representations. Parsing data into symbolic queries consistently boosts model performance. Symbolic representations bridge counterfactual gaps, reduce dependence on data size, and enhance the handling of complex questions. By structuring data more clearly, symbolic queries improve robustness and address challenges like noise and memorization.

2. Benefits of Schema-Based Reasoning. Schemas provide a clean, data-agnostic abstraction of database structures, removing irrelevant noise and simplifying reasoning. By presenting only the schema without any underlying data, we ensure there is no room for memorization. Unlike raw tables, which mix useful and irrelevant data, schemas provide a stable framework that ensures consistent performance, especially in counterfactual scenarios where structured reasoning is critical.

3. Effect of Data Size. Data size significantly affects model performance. Larger tables often introduce noise, increasing the risk of hallucinations. Schemas mitigate this by segmenting data into key components, reducing cognitive overload and clarifying the reasoning process, allowing models to perform more reliably on large, complex datasets.

[2]We tested counterfactual versions, showing similar findings to section 4.1.

4. Handling Complex Questions. Schema-based reasoning excels in answering complex questions by supporting logical, step-by-step reasoning. SQL query generation fosters clarity and reduces ambiguity. In contrast, raw text tables, especially those with counterfactual data, often lack structure, leading to errors or incomplete reasoning. By offering a predefined framework, schemas reduce cognitive demands, enabling models to handle nuanced queries more effectively.

6 Discussion of Model Failures

6.1 Inadequacy of Direct Complex Strategies

Several techniques, such as Program of Thought (PoT) Chen et al. (2023), Chain of Table(Wang et al., 2024a), Binder(Cheng et al.), Dater Ye et al. (2023b), and Plan and Solve Wang et al. (2023), aim to handle complex queries. However, these methods fall short when detailed query plans are needed. The complexity of tasks involving multiple steps, conditional logic, and dependencies cannot be captured by direct prompting alone. Each query introduces unique variables, making strategies like PoT fails for complex reasoning.

For example, a query requiring the join of three large tables with specific conditions cannot be effectively handled by PoT, which may only generate simple steps like *"select from Table A"* or *"filter Table B."* These methods fail to capture the necessary logic for combining tables or handling multiple joins and nested queries. Such complexity requires a carefully constructed query plan, which direct prompting cannot produce.

The core issue is the complexity of the underlying query plans. PoT may generate query plans, but they struggle with complex operations like joins, aggregations, and nested subqueries, which demand precise sequencing and optimization. Research, particularly by Akioyamen et al. (2024), argues that query planning requires structured approaches like SQL to manage these complexities, reinforcing that simpler prompting strategies are insufficient for intricate query reasoning.

6.2 Challenges with Symbolic Approach

Despite advancements in symbolic representation, several challenges remain in improving model reliability and performance:

1. SQL Query Inconsistencies The model often misuses SQL constructs, such as over-relying on `LIMIT 1` when multiple answers are needed *"List all the formats in which Carolina Marín has won medals?"* or adding redundant joins that slow execution *"How many tournaments did Michael Phelps win in 2008?"*. It also misaligns query objectives, failing to handle aggregates or `GROUP BY` clauses properly *"What are the medal counts for each athlete in the Olympics?"*.

2. Temporal and Positional Reasoning Errors The model struggles with temporal and positional reasoning, often hallucinating columns or misinterpreting data *"At what age did Michael Phelps win his most recent Olympic Gold Medal?"*. It also misaligns aggregations over time *"Which athlete had the most consistent medal wins over the last decade?"* and hierarchical relationships *"Which was Michael Phelps' most recent tournament medal?"*.

3. Nested and Conditional Logic Challenges Errors occur in nested and conditional logic, such as incorrect use of `WITH` clauses *"Which event had the shortest duration between P. V. Sindhu's medal wins?"* or failing to respect conditions *"List all tournaments where Carolina Marín won a medal after 2015?"*. The model also mishandles multi-field responses *"List the medal type, location, and year for Hugo Calderano's wins."*.

4. Aggregates, Joins, and Dependencies The model struggles with nested aggregates, non-standard joins, and dependency tracking. It fails to construct valid joins *"Which format had the highest number of gold medals in 2020?"* or align dependencies in complex queries *"Which medal did Michael Phelps win in the same tournament as his fastest recorded swim?"*. It also ignores group-level constraints, leading to overgeneralized results *"Which city hosted the most gold-medal-winning tournaments for P. V. Sindhu?"*.

5. Inconsistencies and Robustness Issues Inconsistent query structures lead to variable results for similar tasks *"How old was Hugo Calderano when he won his first medal?"* vs. *"At what age did Michael Phelps win his most recent Olympic Gold Medal?"*. The model struggles with entity disambiguation *"List all the medals won by Michael Phelps in the Olympic Games?"* and overlooks edge cases *"How many medals has an athlete with no wins received?"*. Ranking logic is often mishandled, such as ignoring ordering requirements *"Which city hosted the most tournaments in 2019?"*.

7 Comparison with Related Work

Temporal reasoning in LLMs is an evolving field intersecting with advancements in tabular reasoning, logic, and symbolic methods. Our work advances this area by introducing the TEMPTABQA-C dataset for detailed evaluation of temporal reasoning in tabular contexts. Key advancements in related areas are discussed below.

Tabular Reasoning. The application of LLMs to semi-structured tabular data has been widely studied across tasks like question answering, semantic parsing, and table-to-text generation (Chen et al., 2020; Gupta et al., 2020; Zhang et al., 2020; Zhang and Balog, 2020). Models such as TAPAS (Herzig et al., 2020), TaBERT (Yin et al., 2020), and TABBIE (Iida et al., 2021) enhance table comprehension by combining tabular and textual embeddings, while methods like Table2vec (Zhang et al., 2019) and TabGCN (Pramanick and Bhattacharya, 2021) explore alternative tabular representations to improve inference.

Recent studies have introduced symbolic reasoning for structured tables with predefined schemas (Cheng et al., 2023; Ye et al., 2023a; Wang et al., 2024b), enabling more effective navigation of structured data. Our work builds on these advancements by using SQL-based symbolic reasoning to address temporal queries in semi-structured tabular datasets. TEMPTABQA-C further contributes by offering a fine-grained evaluation framework for temporal reasoning across diverse data characteristics.

Temporal Reasoning. Temporal reasoning is central to question answering and event-centric tasks, with datasets like TIME-SENSITIVEQA (Chen et al., 2021) and TORQUE (Ning et al., 2020) addressing time-sensitive comprehension in text, and TEMPQA-WD (Neelam et al., 2022) and CRONQUESTIONS (Saxena et al., 2021) focusing on temporal links in knowledge graphs. Models like CRONKBQA (Saxena et al., 2021) further enhance performance by incorporating temporal reasoning during training.

Our work extends these efforts to structured tabular datasets. While datasets such as TempTabQA (Gupta et al., 2023) and TRAM (Wang and Zhao, 2024) tackle similar challenges, TEMPTABQA-C advances the field by introducing counterfactual reasoning, scalable table sizes, and diverse question difficulties, offering a broader framework for evaluating temporal reasoning.

Logical Reasoning and Symbolic Approaches Logical reasoning frameworks like LOGIC-LM (et al., 2023b) and neurosymbolic methods such as LINC (et al., 2023c) demonstrate the benefits of incorporating symbolic reasoning to enhance logical inference in LLMs. These approaches use external tools to handle complex logical tasks, enabling modular and interpretable reasoning pipelines. Similarly, auto-formalization techniques like NL2FOL (et al., 2023a) convert natural language inputs into structured symbolic representations, improving reasoning accuracy.

Building on these paradigms, our SQL-based symbolic representation focuses on temporal reasoning within tabular contexts, converting natural language queries into executable SQL to enable structured reasoning with scalability and precision. Positioned at the intersection of tabular reasoning, temporal reasoning, and symbolic methods, our work introduces the TEMPTABQA-C dataset—a comprehensive benchmark for evaluating LLM capabilities. This dataset includes original and counterfactual splits, varying table sizes, and questions of diverse difficulty levels, seamlessly integrating with the SQL-based reasoning approach to advance structured temporal reasoning in LLMs

7.1 Conclusion

This work investigates temporal tabular question answering with LLMs, tackling key challenges in counterfactual robustness, data sensitivity, and question complexity. We introduced TEMPTABQA-C , a controlled benchmark designed for systematic evaluations. By combining symbolic intermediate representations with adaptive few-shot prompting, our approach leverages database schemas and SQL query generation to address the limitations of direct prompting.

Our experiments demonstrate that symbolic representations improve generalization, counterfactual robustness, and scalability, especially when handling larger tables. Additionally, adaptive prompting enhances reasoning for complex queries. Immediate future work can focus on incorporating stronger baselines, conducting detailed error analysis, and exploring fine-tuning techniques. A deeper analysis of the results will further illuminate the strengths and limitations of the approach. TEMPTABQA-C lays a strong foundation for advancing structured temporal reasoning in LLMs and encourages future efforts to develop interpretable and robust temporal reasoning AI systems.

Limitations

We demonstrated the effectiveness of our approach through extensive experiments in English. However, extending the study to a multilingual context could reveal its applicability across diverse languages. While our work focuses on simple, entity-centric tables, real-world datasets are often more complex, such as hierarchical or multi-relational tables. Future research should explore these more intricate structures to expand the method's utility.

Our experiments assume static tables, yet many real-world scenarios involve dynamic data, such as streaming or frequently updated tables. Adapting the method to handle evolving datasets would enhance its practical relevance. Additionally, the approach does not leverage external domain knowledge, which could complement symbolic reasoning and broaden its applications.

The dataset may also exhibit inherent biases, such as domain-specific or entity-centric constraints, limiting generalizability. Future datasets should aim for greater diversity to better reflect real-world scenarios. Finally, due to computational constraints, we did not fine-tune models on the TEMPTABQA-C dataset. Future work should address this limitation by exploring fine-tuning on larger datasets and evaluating the approach in more resource-intensive and dynamic settings for a comprehensive assessment.

Ethics Statement

We are deeply committed to upholding the highest ethical standards in research and publication. To ensure transparency and reproducibility, we will publicly release our code, enhanced evaluation set, and detailed documentation, enabling the research community to validate, reproduce, and build upon our work. By sharing our resources, we aim to foster collaboration and accountability within the computational linguistics field.

Our methodology reflects a commitment to the responsible and fair use of tools and techniques, with all claims grounded in rigorously validated experimental results. To address the stochastic nature of black-box models, we maintained a fixed temperature throughout our experiments, ensuring consistent outcomes. AI tools were employed responsibly during the writing process, with careful oversight to prevent bias or inaccuracies. We provide comprehensive details about annotations, dataset splits, models, and prompting methods to ensure full reproducibility and empower researchers to evaluate our work rigorously.

Recognizing the importance of inclusivity and fairness, we acknowledge that our dataset may carry inherent biases, such as domain-specific or entity-centric limitations. While we strive for broad applicability, future iterations will prioritize greater diversity to enhance fairness and generalizability. By adhering to these principles, we aim to advance knowledge in computational linguistics while promoting ethical and responsible research practices that emphasize transparency, equity, and reproducibility.

References

Peter Akioyamen, Zixuan Yi, and Ryan Marcus. 2024. The unreasonable effectiveness of llms for query optimization. *arXiv preprint arXiv:2411.02862*.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Preprint*, arXiv:2211.12588.

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

Wenhu Chen, Xinyi Wang, and William Yang Wang. 2021. A dataset for answering time-sensitive questions. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

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. 2023. Binding language models in symbolic languages. *Preprint*, arXiv:2210.02875.

Zhoujun Cheng, Tianbao Xie, Peng Shi, Chengzu Li, Rahul Nadkarni, Yushi Hu, Caiming Xiong, Dragomir Radev, Mari Ostendorf, Luke Zettlemoyer, et al. Binding language models in symbolic languages. In *The Eleventh International Conference on Learning Representations*.

Chen et al. 2023a. Nl2fol: Natural language to first-order logic. *arXiv preprint arXiv:2304.09102*.

Liangming Pan et al. 2023b. Logic-lm: Empowering large language models with symbolic solvers for faithful logical reasoning. *Findings of the Association for Computational Linguistics*.

Theo X. Olausson et al. 2023c. Linc: Logical inference via neurosymbolic computation. *arXiv preprint arXiv:2310.15164*.

Vivek Gupta, Pranshu Kandoi, Mahek Vora, Shuo Zhang, Yujie He, Ridho Reinanda, and Vivek Srikumar. 2023. TempTabQA: Temporal question answering for semi-structured tables. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 2431–2453, Singapore. Association for Computational Linguistics.

Vivek Gupta, Maitrey Mehta, Pegah Nokhiz, and Vivek Srikumar. 2020. INFOTABS: Inference on tables as semi-structured data. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2309–2324, Online. Association for Computational Linguistics.

Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Eisenschlos. 2020. Tapas: Weakly supervised table parsing via pre-training. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics.

Hiroshi Iida, Dung Thai, Varun Manjunatha, and Mohit Iyyer. 2021. Tabbie: Pretrained representations of tabular data. *Preprint*, arXiv:2105.02584.

Sumit Neelam, Udit Sharma, Hima Karanam, Shajith Ikbal, Pavan Kapanipathi, Ibrahim Abdelaziz, Nandana Mihindukulasooriya, Young-Suk Lee, Santosh Srivastava, Cezar Pendus, Saswati Dana, Dinesh Garg, Achille Fokoue, G P Shrivatsa Bhargav, Dinesh Khandelwal, Srinivas Ravishankar, Sairam Gurajada, Maria Chang, Rosario Uceda-Sosa, Salim Roukos, Alexander Gray, Guilherme Lima, Ryan Riegel, Francois Luus, and L Venkata Subramaniam. 2022. A benchmark for generalizable and interpretable temporal question answering over knowledge bases. *Preprint*, arXiv:2201.05793.

Qiang Ning, Hao Wu, Rujun Han, Nanyun Peng, Matt Gardner, and Dan Roth. 2020. TORQUE: A reading comprehension dataset of temporal ordering questions. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1158–1172, Online. Association for Computational Linguistics.

Aniket Pramanick and Indrajit Bhattacharya. 2021. Joint learning of representations for web-tables, entities and types using graph convolutional network. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 1197–1206, Online. Association for Computational Linguistics.

Apoorv Saxena, Soumen Chakrabarti, and Partha Talukdar. 2021. Question answering over temporal knowledge graphs. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 6663–6676, Online. Association for Computational Linguistics.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. 2023. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL)*.

Yuqing Wang and Yun Zhao. 2024. Tram: Benchmarking temporal reasoning for large language models. *Preprint*, arXiv:2310.00835.

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. 2024a. Chain-of-table: Evolving tables in the reasoning chain for table understanding. In *International Conference on Learning Representations (ICLR)*.

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. 2024b. Chain-of-table: Evolving tables in the reasoning chain for table understanding. *Preprint*, arXiv:2401.04398.

Yunhu Ye, Binyuan Hui, Min Yang, Binhua Li, Fei Huang, and Yongbin Li. 2023a. Large language models are versatile decomposers: Decompose evidence and questions for table-based reasoning. *Preprint*, arXiv:2301.13808.

Yunhu Ye, Binyuan Hui, Min Yang, Binhua Li, Fei Huang, and Yongbin Li. 2023b. 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*, SIGIR '23, page 174–184, New York, NY, USA. Association for Computing Machinery.

Pengcheng Yin, Graham Neubig, Wen tau Yih, and Sebastian Riedel. 2020. Tabert: Pretraining for joint understanding of textual and tabular data. *Preprint*, arXiv:2005.08314.

Li Zhang, Shuo Zhang, and Krisztian Balog. 2019. Table2vec: Neural word and entity embeddings for table population and retrieval. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '19. ACM.

Shuo Zhang and Krisztian Balog. 2020. Web table extraction, retrieval, and augmentation: A survey. *ACM Trans. Intell. Syst. Technol.*, 11(2):13:1–13:35.

Shuo Zhang, Zhuyun Dai, Krisztian Balog, and Jamie Callan. 2020. Summarizing and exploring tabular data in conversational search. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '20, pages 1537–1540, New York, NY, USA. Association for Computing Machinery.

8 Appendix

8.1 Examples:

8.1.1 Example 1:

Q. Which Olympic year marked Michael Phelps' record for the most gold medals won?

Steps for SQL Reasoning

Step 1: Start with the infobox table of Michael Phelps' medals

Medal	Year	Event
Gold	2008 Beijing	100 m butterfly
Gold	2008 Beijing	200 m medley
Gold	2004 Indianapolis	200 m freestyle
Silver	2002 Yokohama	4×200 m freestyle

Step 2: Transform the data (all swimmer infoxes) into a relational schema and organize it into structured database tables for efficient querying.

Database Schema:

```
Athlete Table:
+--------------+----------------+
| Column       | Description    |
+--------------+----------------+
| athlete_id   | Primary Key    |
| name         | Athlete Name   |
+--------------+----------------+
Tournament Table:
+-----------------+------------------------+
| Column          | Description            |
+-----------------+------------------------+
| tournament_id   | Primary Key            |
| athlete_id      | Foreign Key (Athlete)  |
| name            | Tournament Name        |
+-----------------+------------------------+
Format Table:
+-----------------+----------------------------+
| Column          | Description                |
+-----------------+----------------------------+
| format_id       | Primary Key                |
| tournament_id   | Foreign Key (Tournament)   |
| name            | Event Name                 |
+-----------------+----------------------------+
Medal Table:
+--------------+----------------------------+
| Column       | Description                |
+--------------+----------------------------+
| medal_id     | Primary Key                |
| format_id    | Foreign Key (Format)       |
| type         | Medal Type                 |
| year         | Year of Achievement        |
| location     | Medal Location             |
+--------------+----------------------------+
PersonalInformation Table:
+-----------------+-----------------------------+
| Column          | Description                 |
+-----------------+-----------------------------+
| info_id         | Primary Key                 |
| athlete_id      | Foreign Key (Athlete)       |
| birth_year      | Birth Year                  |
| birth_month     | Birth Month                 |
| birth_day       | Birth Day                   |
+-----------------+-----------------------------+
```

Step 3: Write the SQL Query

The following query retrieves the year with the most gold medals:

```sql
WITH gold_medal_counts AS (
    SELECT m.year, COUNT(m.medal_id) AS
        gold_medals
    FROM Medal m
    JOIN Format f ON m.format_id = f.format_id
    JOIN Tournament t ON f.tournament_id = t.
        tournament_id
    JOIN Athlete a ON t.athlete_id = a.athlete_id
    WHERE a.name = 'Michael Phelps'
      AND m.type = 'MedalGold'
    GROUP BY m.year
)
SELECT year
FROM gold_medal_counts
WHERE gold_medals = (
    SELECT MAX(gold_medals)
    FROM gold_medal_counts
);
```

Step 4: Execute the Query

The query outputs the year with the highest number of gold medals.

Final Result: 2008

Direct Reasoning with Chain-of-Thought (CoT):

To perform direct reasoning using Chain-of-Thought (CoT), LLM arrange the medal in year and count the number of gold medals per year from the table:

```
Year 2008:
- 100 m butterfly (Gold)
Total: 1 gold medals

Year 2004:
- 200 m freestyle (Gold)
- 200 m medley (Gold)
Total: 2 gold medal

Year 2002:
- 4x200 m freestyle (Silver)
Total: 0 gold medals
```

Answer (CoT Reasoning): 2004 has the most gold medals with a count of 2.

However, due to direct reasoning errors or omissions, it misinterpret the complex table, and hence CoT fails whereas Symbolic succeed.

8.1.2 Example 2:

Q. Does Emma Weyant have more Bronze Medals than Gold Medals ?

Steps for SQL Reasoning

Step 1: Start with the infobox table of Emma Weyant's medals.

Step 2: Transform the data (all swimmer infoboxes) into a relational schema and organize

Emma Weyant

Olympic Games		
S	2020 Tokyo	400 m medley
B	2024 Paris	400 m medley
World Championships (LC)		
B	2022 Budapest	400 m medley
World Championships (SC)		
S	2021 Abu Dhabi	4×200 m freestyle
Junior Pan Pacific Championships		
G	2018 Suva	400 m medley
B	2018 Suva	800 m freestyle

Figure 2: Emma Weyant's Medal Infobox

it into structured database tables for efficient querying.

Database Schema:

```
Athlete Table:
+-------------+---------------+
| Column      | Description   |
+-------------+---------------+
| athlete_id  | Primary Key   |
| name        | Athlete Name  |
+-------------+---------------+

Tournament Table:
+---------------+------------------------+
| Column        | Description            |
+---------------+------------------------+
| tournament_id | Primary Key            |
| athlete_id    | Foreign Key (Athlete)  |
| name          | Tournament Name        |
+---------------+------------------------+

Format Table:
+---------------+----------------------------+
| Column        | Description                |
+---------------+----------------------------+
| format_id     | Primary Key                |
| tournament_id | Foreign Key (Tournament)   |
| name          | Event Name                 |
+---------------+----------------------------+

Medal Table:
+-------------+--------------------------+
| Column      | Description              |
+-------------+--------------------------+
| medal_id    | Primary Key              |
| format_id   | Foreign Key (Format)     |
| type        | Medal Type               |
| year        | Year of Achievement      |
| location    | Medal Location           |
+-------------+--------------------------+

PersonalInformation Table:
+---------------+----------------------------+
| Column        | Description                |
+---------------+----------------------------+
| info_id       | Primary Key                |
| athlete_id    | Foreign Key (Athlete)      |
| birth_year    | Birth Year                 |
| birth_month   | Birth Month                |
| birth_day     | Birth Day                  |
+---------------+----------------------------+
```

Step 3: Write the SQL Query

The following query checks whether Emma Weyant has more Bronze medals than Gold medals:

```sql
SELECT CASE
    WHEN SUM(CASE WHEN m.type = 'MedalBronze'
        THEN 1 ELSE 0 END) >
        SUM(CASE WHEN m.type = 'MedalGold' THEN
            1 ELSE 0 END)
    THEN 'Yes'
    ELSE 'No'
END AS has_more_bronze_than_gold
FROM Medal m
JOIN Format f ON m.format_id = f.format_id
JOIN Tournament t ON f.tournament_id = t.
    tournament_id
JOIN Athlete a ON t.athlete_id = a.athlete_id
WHERE a.name = 'Emma Weyant';
```

Step 4: Execute the Query

The query outputs whether Emma Weyant has more Bronze medals than Gold medals.

Final Result: Yes

Direct Reasoning with Chain-of-Thought (COT):

Using manual reasoning, the LLM counts the medals directly from the table:

```
Gold Medals:
- 2018 Suva: 400 m medley
Total: 1 Gold Medal

Bronze Medals:
- 2018 Suva: 800 m freestyle
- 2022 Budapest: 400 m medley
- 2024 Paris: 400 m medley
Total: 3 Bronze Medals

Final Count:
Gold: 1
Bronze: 3
```

LLM's Answer: No Emma Weyant has one Gold Medal and three Bronze Medals.

Why the LLM's Answer is Incorrect and Symbolic Reasoning Succeeds:

- **Direct Reasoning Errors**: The LLM correctly identifies the count but fails in its logical comparison, leading to an incorrect conclusion.
- **Symbolic Reasoning Accuracy**: SQL-based reasoning explicitly performs the correct comparison and produces an unambiguous result.
- **Scalability and Consistency**: SQL-based methods remain reliable as data size and complexity grow, unlike manual reasoning.

Conclusion: Symbolic SQL reasoning eliminates errors inherent in manual reasoning methods like Chain-of-Thought, ensuring precise and reliable results.

8.1.3 Example 3:

Q. In which city did Yohan Blake win his first medal?

Step 1: Start with the infobox table of Yohan Blake's medals.

Yohan Blake's Medal Record:		
Olympic Games		
Medal	**Year**	**Event**
Gold	2012 London	4×100 m relay
Gold	2016 Rio de Janeiro	4×100 m relay
Silver	2012 London	100 m
Silver	2012 London	200 m
World Championships		
Gold	2011 Daegu	100 m
Gold	2011 Daegu	4×100 m relay
Commonwealth Games		
Bronze	2018 Gold Coast	100 m
Bronze	2018 Gold Coast	4×100 m relay
World Relays		
Gold	2014 Bahamas	4×100 m
Gold	2014 Bahamas	4×200 m
Bronze	2017 Bahamas	4×200 m
World Junior Championships		
Gold	2006 Beijing	4×100 m relay
Silver	2008 Bydgoszcz	4×100 m relay
Bronze	2006 Beijing	100 m
Pan American Junior Championships		
Silver	2007 São Paulo	100 m
Bronze	2007 São Paulo	4×400 m relay
CAC Junior Championships (U20)		
Gold	2006 Port of Spain	100 m
Gold	2006 Port of Spain	200 m
Gold	2006 Port of Spain	4×100 m relay
CARIFTA Games		
Gold	2006 Les Abymes	200 m
Gold	2006 Les Abymes	4×100 m relay
Gold	2007 Providenciales	100 m
Gold	2007 Providenciales	4×100 m relay
Gold	2008 Basseterre	100 m
CARIFTA Games		
Gold	2005 Bacolet	100 m
Gold	2005 Bacolet	200 m
Continental Cup		
Gold	2018 Ostrava	4×100 m

Step 2: Transform the data into a relational schema and organize it into structured database tables for efficient querying (similar to Step 2 in previous examples).

Step 3: Write the SQL Query
The following query retrieves the location where Yohan Blake won his first medal:

```sql
SELECT DISTINCT m.location
FROM Medal m
JOIN Format f ON m.format_id = f.format_id
JOIN Tournament t ON f.tournament_id = t.
    tournament_id
JOIN Athlete a ON t.athlete_id = a.athlete_id
WHERE a.name = 'Yohan Blake'
  AND m.year = (
    SELECT MIN(m2.year)
    FROM Medal m2
    JOIN Format f2 ON m2.format_id = f2.format_id
    JOIN Tournament t2 ON f2.tournament_id = t2.
        tournament_id
    JOIN Athlete a2 ON t2.athlete_id = a2.
        athlete_id
    WHERE a2.name = 'Yohan Blake'
  );
```

Step 4: Execute the Query
The query outputs the location where Yohan Blake won his first medal.
Final Result: Bacolet

Direct Reasoning with Chain-of-Thought (COT):
Using manual reasoning, the LLM incorrectly identifies the location as Beijing:

```
Year 2006:
- Gold: 4x100 m relay (World Junior Championships, Beijing)
- Bronze: 100 m (World Junior Championships, Beijing)
Conclusion: First medal location is Beijing.
```

LLM's Answer: Beijing. In 2006, Yohan Blake won his first medal at the World Junior Championships in Beijing, where he secured a Gold in the 4x100 m relay and a Bronze in the 100 m.

Why the LLM's Answer is Incorrect and Symbolic Reasoning Succeeds:

- **Direct Reasoning Errors**: The LLM overlooks earlier results from 2005 in the CARIFTA Games held in Bacolet, where Yohan Blake won two Gold medals.
- **Symbolic Reasoning Accuracy**: SQL explicitly finds the minimum year and correctly identifies the location associated with the first medal.
- **Consistency and Scalability**: Symbolic SQL reasoning reliably handles large, complex medal records without omission or error.

Conclusion: Symbolic SQL reasoning eliminates the errors inherent in Chain-of-Thought reasoning, ensuring accurate and reliable results.

8.1.4 Example 4:

Q. How many medals did Mayu Matsumoto win in her twenties?

Step 1: Start with the infobox table of Mayu Matsumoto's medals.

Mayu Matsumoto		
Personal information		
Born	7 August 1995 (age 29)	
World Championships		
G	2018 Nanjing	Women's doubles
G	2019 Basel	Women's doubles
B	2021 Huelva	Women's doubles
B	2022 Tokyo	Women's doubles
Sudirman Cup		
S	2019 Nanning	Mixed team
S	2021 Vantaa	Mixed team
B	2023 Suzhou	Mixed team
Uber Cup		
S	2020 Aarhus	Women's team
B	2022 Bangkok	Women's team
Asian Championships		
S	2019 Wuhan	Women's doubles
B	2023 Dubai	Women's doubles
Asia Team Championships		
G	2020 Manila	Women's team
Asian Junior Championships		
B	2013 Kota Kinabalu	Mixed team

Figure 3: Mayu Matsumoto's Medal Infobox

Step 2: Transform the data into a relational schema and organize it into structured database tables for efficient querying. (similar to Step 2 in previous examples).

Step 3: Write the SQL Query
The following query calculates how many medals Mayu Matsumoto won between the ages of 20 and 29:

```sql
SELECT COUNT(m.medal_id) AS total_medals
FROM Medal m
JOIN Format f ON m.format_id = f.format_id
JOIN Tournament t ON f.tournament_id = t.
    tournament_id
JOIN Athlete a ON t.athlete_id = a.athlete_id
JOIN PersonalInformation pi ON a.athlete_id = pi.
    athlete_id
WHERE a.name = 'Mayu Matsumoto'
  AND (m.year - pi.birth_year) BETWEEN 20 AND 29;
```

Step 4: Execute the Query
The query calculates medals for Mayu Matsumoto won during her twenties (from 2015 to 2024, since she was born in 1995).

Final Result: 12

Direct Reasoning with Chain-of-Thought (COT):
Using manual reasoning, the LLM incorrectly calculates the total as follows:
Mayu Matsumoto, born in **1995**, was in her twenties from **2015 to 2024**. During this period:
World Championships: 2 Gold (2018, 2019), 1 Silver (2021), 2 Bronze (2021, 2022).
Sudirman Cup: 2 Silver (2019, 2021), 1 Bronze (2023).
Uber Cup: 1 Silver (2020), 1 Bronze (2022).
Asian Championships: 1 Silver (2019), 1 Bronze (2023).
Total (COT Reasoning): 10 medals.

Trends in the LLM's Reasoning:

- **Incorrect Medal Count for World Championships:** The LLM claims 2 Gold, 2 Silver, and 1 Bronze, but the infobox shows 2 Gold (2018, 2019), 1 Bronze (2022), and only 1 Silver (2021).
- **Overcounted/Undercounted Totals:** The total medals, when carefully counted, sum to 12, not 10:
  - **World Championships:** 2 Gold, 1 Silver, 1 Bronze (Total = 4).
  - **Sudirman Cup:** 2 Silver, 1 Bronze (Total = 3).
  - **Uber Cup:** 1 Silver, 1 Bronze (Total = 2).
  - **Asian Championships:** 1 Silver, 1 Bronze (Total = 2).
  - **Incorrectly Excluded 2020 Medal:** Asian Team Championships (2020, age 25) is excluded incorrectly.
  - **Correctly Excluded 2013 Medal:** Asian Junior Championships (2013, age 18) is excluded correctly.
- **Temporal Misinterpretation:** The LLM fails to count some of the medals in the 20-29 age range and fails to sum them accurately.

Symbolic Reasoning Accuracy:

- SQL precisely filters years between 2015 and 2024, ensuring only valid medals are counted.
- Symbolic reasoning eliminates human counting errors and temporal miscalculations.
- The result is accurate: **12 medals**.

Conclusion: The LLM's Chain-of-Thought reasoning undercounts Mayu Matsumoto's medals, providing an incorrect total of 10 due to miscounting and temporal errors. Symbolic SQL reasoning accurately identifies the correct total as 12 medals won during her twenties.

8.1.5 Example 5:

Q. How many times did Sandra Sánchez win a medal in the World Championships before 2021?

Step 1: Start with the infobox table of Sandra Sánchez's medals.

Sandra Sánchez

Summer Olympics		
G	2020 Tokyo	Individual kata
World Championships		
G	2018 Madrid	Individual kata
G	2021 Dubai	Individual kata
B	2016 Linz	Individual kata
European Championships		
G	2015 Istanbul	Individual kata
G	2016 Montpellier	Individual kata
G	2017 Kocaeli	Individual kata
G	2018 Novi Sad	Individual kata
G	2019 Guadalajara	Individual kata
G	2021 Poreč	Individual kata
G	2022 Gaziantep	Individual kata
European Games		
G	2015 Baku	Individual kata
G	2019 Minsk	Individual kata
World Beach Games		
G	2019 Doha	Individual kata
World Games		
G	2022 Birmingham	Individual kata
S	2017 Wrocław	Individual kata

Figure 4: Sandra Sánchez's Medal Infobox

Step 2: Transform the data into a relational schema and organize it into structured database tables for efficient querying (similar to Step 2 in previous examples).

Step 3: Write the SQL Query
The following query calculates how many medals Sandra Sánchez won in the World Championships before the year 2021:

```
SELECT COUNT(m.medal_id) AS total_medals
FROM Medal m
JOIN Format f ON m.format_id = f.format_id
JOIN Tournament t ON f.tournament_id = t.
    tournament_id
JOIN Athlete a ON t.athlete_id = a.athlete_id
WHERE a.name = 'Sandra Sánchez'
  AND t.name = 'World Championships'
  AND m.year < 2021;
```

Step 4: Execute the Query
The query outputs the total number of medals Sandra Sánchez won in the World Championships before 2021.

Final Result: 2

Direct Reasoning with Chain-of-Thought (COT):
The LLM incorrectly provides the following reasoning:
Sandra Sánchez won a Bronze medal in the World Championships in 2016, which is before 2021. Therefore, the answer is 1.

Errors in the LLM's Reasoning:

- **Missed Medal in 2018:** While the LLM identifies the 2016 Bronze medal, it fails to recognize the 2018 Gold medal in Madrid, which also occurred before 2021.
- **Incomplete Temporal Analysis:** The LLM does not account for all relevant years when performing temporal reasoning, leading to an undercount of medals.

Symbolic Reasoning Accuracy:

- SQL explicitly filters medals in the World Championships where the year is less than 2021.
- The query correctly identifies both the 2016 Bronze medal and the 2018 Gold medal, producing the accurate total of **2 medals**.
- Symbolic reasoning eliminates human oversight by systematically querying all relevant data within the temporal range.

Conclusion: The LLM's Chain-of-Thought reasoning incorrectly counts only **1 medal** due to missed temporal filtering. Symbolic SQL reasoning, by explicitly querying for medals before 2021, produces the correct result: 2 medals.

8.2 Result Analysis for all models:

8.2.1 Analysis for GPT-4o:

From Table 8, comparing **SQL Adaptive** with **Direct Adaptive** across key aspects, we observe:

- **Counterfactual Gap:** For **Table - Adaptive** (EMS), the gap between Original (**54.92**) and CounterFact (**41.70**) is **13.22**. For **SQL Schema - Adaptive** (EMS), the gap reduces to **1.02**, indicating improved robustness to counterfactual data.

- **Scalability to Table Size:** For **Table - Adaptive** (EMS), the gap between Large (**48.88**) and Small (**73.92**) tables is **25.04**. For **SQL Schema - Adaptive** (EMS), the gap decreases significantly to **1.07**, demonstrating better scalability to large table sizes.
- **Question Complexity:** For **Table - Adaptive** (EMS), the performance on Easy, Medium, and Hard questions is **74.38**, **63.91**, and **54.17**, respectively. For **SQL Schema - Adaptive** (EMS), the performance improves to **79.83** (Easy), **73.57** (Medium), and **66.32** (Hard), showing better handling of increasing question complexity.
- **Adaptive Few-Shot Effectiveness:** For Original data, **SQL Schema - Adaptive** (EMS: **68.04**) outperforms **Table - Adaptive** (EMS: **54.92**), highlighting the benefit of adaptive prompting in achieving higher accuracy.

These results demonstrate that **SQL Adaptive** consistently outperforms **Direct Adaptive** by reducing the counterfactual gap, improving scalability to large tables, and enhancing performance across question complexities.

8.2.2 Analysis for GPT-4o Mini:

From Table 9, comparing **SQL Adaptive** with **Table Adaptive** across key aspects, we observe:

- **Counterfactual Gap:** For **Table - Adaptive** (EMS), the gap between Original (**48.79**) and CounterFact (**35.48**) is **13.31**. For **SQL Schema - Adaptive** (EMS), the gap reduces significantly to **3.65** (Original: **68.69**, CounterFact: **65.24**), indicating improved robustness to counterfactual data.
- **Scalability to Table Size:** For **Table - Adaptive** (EMS), the gap between Large (**39.96**) and Small (**64.80**) tables is **24.84**. For **SQL Schema - Adaptive** (EMS), the gap reduces to **3.30** (Large: **65.43**, Small: **68.77**), demonstrating better scalability to large table sizes.
- **Question Complexity:** For **Table - Adaptive** (EMS), the scores for Easy, Medium, and Hard questions are **63.16**, **52.07**, and **44.49**, respectively. For **SQL Schema - Adaptive** (EMS), the performance improves to **76.87** (Easy), **70.41** (Medium), and **63.13** (Hard), showcasing better handling of increasing question complexity.
- **Adaptive Few-Shot Effectiveness:** For Original data, **SQL Schema - Adaptive** (EMS: **68.69**) significantly outperforms **Table - Adaptive** (EMS: **48.79**), highlighting the superior accuracy achieved with symbolic reasoning and adaptive prompting.

These results clearly show that **SQL Adaptive** consistently outperforms **Table Adaptive**, with smaller counterfactual and table size gaps, and better performance across question complexity levels.

8.2.3 Analysis for Gemini 1.5 Flash:

From Table 10, comparing **SQL Adaptive** with **Table Adaptive**, we observe:

- **Counterfactual Gap:** For **Table - Adaptive** (EMS), the gap between Original (**52.90**) and CounterFact (**42.91**) is **9.99**. In comparison, for **SQL Schema - Adaptive** (EMS), the gap is reduced to **2.78** (Original: **65.49**, CounterFact: **62.71**), indicating significantly improved robustness to counterfactual data.
- **Scalability to Table Size:** For **Table - Adaptive** (EMS), the gap between Large (**41.02**) and Small (**66.25**) tables is **25.23**. For **SQL Schema - Adaptive** (EMS), the gap reduces to **4.23** (Large: **69.30**, Small: **73.53**), showcasing SQL's superior handling of larger tables.
- **Question Complexity:** For **Table - Adaptive** (EMS), the scores for Easy, Medium, and Hard questions are **65.76**, **55.95**, and **45.92**, respectively. For **SQL Schema - Adaptive** (EMS), the scores improve to **76.26** (Easy), **73.72** (Medium), and **63.12** (Hard), highlighting better performance as question complexity increases.
- **Adaptive Few-Shot Effectiveness:** For Original data, **SQL Schema - Adaptive** (EMS: **65.49**) outperforms **Table - Adaptive** (EMS: **52.90**), demonstrating the effectiveness of adaptive few-shot prompting with symbolic reasoning.

These observations show that **SQL Adaptive** significantly reduces the counterfactual gap, scales better with large tables, and consistently achieves higher accuracy across all question complexities compared to **Table Adaptive**.

8.2.4 Analysis for Gemini 1.5 Pro:

From Table 11, comparing **SQL Adaptive** with **Table Adaptive**, we observe:

- **Counterfactual Gap:** For **Table - Adaptive** (EMS), the gap between Original (**53.48**) and CounterFact (**44.19**) is **9.29**. For **SQL Schema - Adaptive** (EMS), the gap is reduced to **0.16** (Original: **65.29**, CounterFact: **65.13**), showcasing excellent robustness to counterfactual data.

- **Scalability to Table Size:** For **Table - Adaptive** (EMS), the gap between Large (**41.86**) and Small (**67.27**) tables is **25.41**. For **SQL Schema - Adaptive** (EMS), the gap reduces significantly to **2.88** (Large: **72.43**, Small: **75.31**), demonstrating better scalability with large tables.

- **Question Complexity:** For **Table - Adaptive** (EMS), the scores for Easy, Medium, and Hard questions are **66.26**, **56.47**, and **46.74**, respectively. For **SQL Schema - Adaptive** (EMS), the scores improve to **75.86** (Easy), **71.47** (Medium), and **59.24** (Hard), highlighting superior handling of increasing question complexity.

- **Adaptive Few-Shot Effectiveness:** For Original data, **SQL Schema - Adaptive** (EMS: **65.29**) significantly outperforms **Table - Adaptive** (EMS: **53.48**), demonstrating the clear benefits of symbolic reasoning combined with adaptive few-shot prompting.

These results clearly highlight that **SQL Adaptive** consistently reduces counterfactual gaps, scales better with table size, and improves performance across question complexities compared to **Table Adaptive**.

8.2.5 Analysis for Llama 3.1 70B:

From Table 12, comparing **SQL Adaptive** with **Table Adaptive**, we observe:

- **Counterfactual Gap:** For **Table - Adaptive** (EMS), the gap between Original (**53.63**) and CounterFact (**39.20**) is **14.43**. For **SQL Schema - Adaptive** (EMS), the gap reduces to **0.55** (Original: **64.36**, CounterFact: **63.81**). SQL Adaptive is clearly more robust to counterfactual data.

- **Scalability to Table Size:** For **Table - Adaptive** (EMS), the gap between Large (**46.65**) and Small (**68.07**) tables is **21.42**. For **SQL Schema - Adaptive** (EMS), the gap remains smaller at **1.98** (Large: **66.91**, Small: **68.89**), showing better performance scalability with table size.

- **Question Complexity:** For **Table - Adaptive** (EMS), the scores for Easy, Medium, and Hard questions are **69.40**, **59.76**, and **52.85**, respectively. For **SQL Schema - Adaptive** (EMS), the scores are **77.19** (Easy), **67.65** (Medium), and **60.92** (Hard), showing a consistent improvement across complexities.

- **Adaptive Few-Shot Effectiveness:** For Original data, **SQL Schema - Adaptive** (EMS: **64.36**) performs significantly better than **Table - Adaptive** (EMS: **53.63**), confirming the benefit of symbolic reasoning with adaptive few-shot prompting.

Overall, **SQL Adaptive** demonstrates clear improvements over **Table Adaptive** in counterfactual robustness, scalability to table size, and performance across question complexity levels. The observed gaps in Table Adaptive remain substantial, especially for counterfactual and large table scenarios.

8.2.6 Analysis for Mixtral 8x7B:

From Table 13, comparing **SQL Adaptive** with **Table Adaptive**, we observe:

- **Counterfactual Gap:** For **Table - Adaptive** (EMS), the gap between Original (**37.54**) and CounterFact (**30.62**) is **6.92**. For **SQL Schema - Adaptive** (EMS), the gap remains comparable at **4.16** (Original: **25.09**, CounterFact: **20.89**). Here, SQL Adaptive does not demonstrate a significant improvement.

- **Scalability to Table Size:** For **Table - Adaptive** (EMS), the gap between Large (**34.94**) and Small (**47.72**) tables is **12.78**. For **SQL Schema - Adaptive** (EMS), the gap is still notable at **7.03** (Large: **24.54**, Small: **33.57**). While smaller, it indicates that SQL Adaptive does not scale as effectively here.

- **Question Complexity:** For **Table - Adaptive** (EMS), the scores for Easy, Medium, and Hard questions are **50.96**, **38.46**, and **35.74**, respectively. For **SQL Schema - Adaptive** (EMS), the scores are lower at **26.78** (Easy), **46.55** (Medium), and **21.56** (Hard). SQL Adaptive fails to outperform Table Adaptive for Easy and Hard questions, only improving slightly on Medium questions.

- **Adaptive Few-Shot Effectiveness:** For Original data, **SQL Schema - Adaptive** (EMS: **25.09**) is significantly lower than **Table - Adaptive** (EMS: **37.54**), indicating poor performance overall.

Overall, Table Adaptive clearly outperforms SQL Adaptive in most metrics for Mixtral 8x7B. SQL Adaptive struggles with counterfactual robustness, scalability to table size, and performance across question complexities.

8.2.7 Analysis for SQL Coder 70B:

Since this is a **code-based model**, we only evaluate baselines related to **code generation** and exclude text generation baselines. From Table 14, we observe the following for **SQL Schema**:

- **Counterfactual Gap:** For **SQL Static** (EMS), the gap between Original (**51.90**) and CounterFact (**48.64**) is **3.26**. For **SQL Adaptive** (EMS), the gap reduces to **2.38** (Original: **55.88**, CounterFact: **53.50**), demonstrating improved robustness with adaptive few-shot prompting.

- **Scalability to Table Size:** For **SQL Static** (EMS), the gap between Large (**62.28**) and Small (**59.53**) tables is **2.75**. For **SQL Adaptive** (EMS), the gap is slightly larger at **4.22** (Large: **63.17**, Small: **58.95**), showing minor regression in scalability.

- **Question Complexity:** For **SQL Static** (EMS), the scores for Easy, Medium, and Hard questions are **75.82**, **43.39**, and **50.63**, respectively. For **SQL Adaptive** (EMS), the scores improve for Medium (**58.38**) and Hard (**51.74**) questions but slightly decrease for Easy (**63.93**), indicating uneven performance gains.

- **Overall Accuracy:** For Original data, **SQL Adaptive** (EMS: **55.88**) outperforms **SQL Static** (EMS: **51.90**), highlighting the effectiveness of adaptive few-shot prompting for code-specific tasks.

Overall, SQL Adaptive demonstrates improved robustness and accuracy compared to SQL Static, particularly on counterfactual and medium-complexity queries, with some inconsistencies in scalability and easy question performance.

8.2.8 Analysis for Code Llama 70B:

Since this is a **code-based model**, we only evaluate baselines related to **code generation** and exclude text generation baselines. From Table 15, we observe the following for **SQL Schema**:

- **Counterfactual Gap:** For **SQL Static** (EMS), the gap between Original (**15.84**) and CounterFact (**32.62**) is substantial at **16.78**, indicating performance degradation. For **SQL Adaptive** (EMS), the gap reduces to **16.53** (Original: **23.53**, CounterFact: **40.06**). While there is slight improvement, the gap remains significant.

- **Scalability to Table Size:** For **SQL Static** (EMS), the gap between Large (**29.82**) and Small (**41.64**) tables is **11.82**. For **SQL Adaptive** (EMS), the gap decreases to **10.53** (Large: **37.61**, Small: **48.14**), indicating modest improvements in handling table size.

- **Question Complexity:** For **SQL Static** (EMS), the scores for Easy, Medium, and Hard questions are **53.42**, **41.62**, and **38.94**, respectively. For **SQL Adaptive** (EMS), the scores improve across all complexities to **65.16** (Easy), **50.89** (Medium), and **40.61** (Hard), showing clear improvements, particularly for Easy and Medium questions.

- **Overall Accuracy:** For Original data, **SQL Adaptive** (EMS: **23.53**) outperforms **SQL Static** (EMS: **15.84**), demonstrating the benefits of adaptive few-shot prompting for overall accuracy.

Overall, **SQL Adaptive** shows moderate improvements over **SQL Static**, particularly in handling table size and question complexities, though counterfactual robustness remains a challenge.

Output	Context	Adaptive	Metric	Results Across Categories						
				Original	**CounterFact**	**Large**	**Small**	**Easy**	**Medium**	**Hard**
Text	-	-	REMS	23.91	14.12	23.84	25.79	28.67	25.55	23.83
			EMS	22.96	12.92	23.05	24.68	27.26	24.06	22.35
	Table	zero shots	REMS	52.14	41.18	45.24	67.81	72.34	59.43	52.9
			EMS	49.7	39.29	43.31	64.56	69.86	57.2	49.02
		Static	REMS	56.16	43.96	48.96	73.57	73.45	65.39	56.97
			EMS	53.95	41.91	46.84	71.11	71.18	63.12	53.35
		Adaptive	REMS	57.51	43.93	51.41	76.97	76.89	66.00	58.00
			EMS	54.92	41.70	48.88	73.92	74.38	63.91	54.17
SQL	Schema	zero shots	REMS	49.4	47.41	55.99	58.58	63.53	65.20	45.46
			EMS	47.27	45.27	54.09	56.14	61.21	63.12	42.02
		Static	REMS	62.90	61.65	72.01	73.67	80.96	76.27	64.69
			EMS	61.36	60.40	70.82	71.93	78.58	75.54	62.62
		Adaptive	REMS	68.41	67.42	73.42	72.80	79.94	74.55	66.97
			EMS	68.04	67.02	73.23	72.16	79.83	73.57	66.32

Table 8: Evaluation Results across multiple datasets for GPT-4o

Output	Context	Few Shots	Metric	Results Across Categories						
				Original	**CounterFact**	**Large**	**Small**	**Easy**	**Medium**	**Hard**
Text	-	-	REMS	21.48	13.90	20.21	22.75	26.08	22.30	19.03
			EMS	20.24	13.02	19.70	21.87	24.84	20.51	17.40
	Table	zero shots	REMS	49.59	33.52	40.18	63.60	62.07	47.53	43.62
			EMS	47.23	31.04	37.73	60.47	59.19	44.58	39.55
		Static	REMS	49.94	36.94	41.33	66.45	64.23	51.13	47.13
			EMS	47.58	34.91	39.03	63.63	61.84	48.32	43.36
		Adaptive	REMS	51.13	38.37	42.43	67.66	66.10	54.81	48.63
			EMS	48.79	35.48	39.96	64.80	63.16	52.07	44.49
SQL	Schema	zero shots	REMS	39.93	41.28	38.76	48.50	57.45	50.88	34.21
			EMS	38.24	39.63	37.36	46.67	55.30	49.51	31.41
		Static	REMS	57.44	51.18	53.94	66.52	77.57	65.29	57.04
			EMS	56.57	50.36	53.16	65.73	75.93	65.29	56.33
		Adaptive	REMS	68.97	65.40	65.70	69.20	76.98	71.07	63.93
			EMS	68.69	65.24	65.43	68.77	76.87	70.41	63.13

Table 9: Evaluation Results for GPT-4o Mini across multiple datasets

Output	Context	Few Shots	Metric	Results Across Categories						
				Original	**CounterFact**	**Large**	**Small**	**Easy**	**Medium**	**Hard**
Text	-	-	REMS	22.11	15.02	23.88	21.74	27.81	23.23	21.59
			EMS	18.69	11.76	19.49	18.93	24.03	19.41	17.30
	Table	zero shots	REMS	55.17	45.42	42.71	69.88	68.78	56.61	51.18
			EMS	48.79	37.61	34.96	61.11	59.15	48.82	39.00
		Static	REMS	57.46	47.20	46.44	71.89	73.76	61.94	54.99
			EMS	50.00	39.08	38.35	62.35	63.16	53.53	42.96
		Adaptive	REMS	59.09	48.25	47.93	73.17	76.28	64.94	57.83
			EMS	52.90	42.91	41.02	66.25	65.76	55.95	45.92
SQL	Schema	zero shots	REMS	47.27	42.56	46.23	56.45	63.49	59.74	42.09
			EMS	39.34	33.45	44.29	53.65	52.96	52.48	34.59
		Static	REMS	66.43	62.38	71.39	77.20	88.17	79.22	65.04
			EMS	57.93	54.43	63.13	70.89	79.54	68.96	58.42
		Adaptive	REMS	72.91	71.67	78.18	81.71	87.06	80.80	74.04
			EMS	65.49	62.71	69.30	73.53	76.26	73.72	63.12

Table 10: Evaluation Results for Gemini 1.5 Flash across multiple datasets.

Output	Context	Few Shots	Metric	Results Across Categories						
				Original	CounterFact	Large	Small	Easy	Medium	Hard
Text	-	-	REMS	23.37	15.88	24.20	24.08	29.15	26.45	23.38
			EMS	19.90	13.45	21.19	20.99	24.94	21.76	18.77
	Table	zero shots	REMS	54.94	45.40	46.34	69.74	73.26	60.71	57.43
			EMS	48.06	37.39	38.98	61.11	62.59	53.24	45.60
		Static	REMS	59.01	46.87	52.42	72.93	75.72	65.10	60.28
			EMS	52.91	39.92	43.86	65.02	65.79	58.53	50.00
		Adaptive	REMS	60.23	49.04	48.79	73.98	78.04	66.43	59.28
			EMS	53.48	44.19	41.86	67.27	66.26	56.47	46.74
SQL	Schema	zero shots	REMS	49.24	43.56	48.21	57.62	64.42	61.10	43.00
			EMS	41.32	35.31	45.87	55.11	54.78	53.79	35.96
		Static	REMS	67.76	63.63	76.80	82.82	89.33	80.81	66.42
			EMS	59.08	55.52	71.32	77.41	80.86	70.33	59.59
		Adaptive	REMS	73.04	73.58	81.94	84.38	87.31	80.01	71.43
			EMS	65.29	65.13	72.43	75.31	75.86	71.47	59.24

Table 11: Evaluation Results for Gemini 1.5 Pro across multiple datasets.

Output	Context	Few Shots	Metric	Results Across Categories						
				Original	CounterFact	Large	Small	Easy	Medium	Hard
Text	-	-	REMS	17.46	12.74	20.89	18.09	22.31	17.93	17.18
			EMS	16.61	12.16	19.89	17.31	21.31	16.96	16.27
	Table	zero shots	REMS	54.63	39.93	46.10	64.62	70.10	58.31	53.37
			EMS	52.60	37.48	44.05	62.57	67.49	56.21	50.35
		Static	REMS	64.33	48.44	57.27	75.40	79.04	70.23	60.98
			EMS	62.46	46.21	55.58	73.68	76.91	67.46	57.44
		Adaptive	REMS	55.73	41.29	48.48	70.56	71.35	62.54	56.43
			EMS	53.63	39.20	46.65	68.07	69.40	59.76	52.85
SQL	Schema	zero shots	REMS	34.77	31.52	37.08	41.36	45.83	40.81	29.74
			EMS	33.56	30.47	36.06	39.77	44.26	39.65	27.54
		Static	REMS	54.59	47.31	55.38	63.12	68.51	60.83	56.15
			EMS	53.81	46.64	54.65	62.22	67.21	60.55	55.63
		Adaptive	REMS	64.61	64.04	67.40	69.54	77.42	68.01	61.78
			EMS	64.36	63.81	66.91	68.89	77.19	67.65	60.92

Table 12: Evaluation Results for Llama 3.1 70B across multiple datasets.

Output	Context	Few Shots	Metric	Results Across Categories						
				Original	CounterFact	Large	Small	Easy	Medium	Hard
Text	-	-	REMS	17.53	13.52	21.71	20.31	20.38	20.06	19.95
			EMS	15.92	12.73	20.82	19.53	19.26	17.75	18.08
	Table	zero shots	REMS	40.26	32.62	37.58	56.55	54.09	46.27	38.65
			EMS	38.06	30.47	35.50	53.33	51.78	43.79	35.05
		Static	REMS	50.92	42.31	43.17	68.13	64.75	55.80	49.06
			EMS	48.79	40.20	40.89	65.03	62.30	53.25	45.62
		Adaptive	REMS	39.98	33.23	37.68	50.69	53.84	41.02	40.33
			EMS	37.54	30.62	34.94	47.72	50.96	38.46	35.74
SQL	Schema	zero shots	REMS	20.35	15.07	19.30	27.74	21.65	27.19	22.43
			EMS	19.90	14.74	18.96	26.55	20.63	27.02	21.56
		Static	REMS	47.42	41.38	48.78	54.63	69.46	46.22	42.45
			EMS	46.02	40.20	47.96	52.98	67.35	45.36	40.61
		Adaptive	REMS	25.45	21.26	24.72	33.96	26.91	47.24	22.20
			EMS	25.09	20.89	24.54	33.57	26.78	46.55	21.56

Table 13: Evaluation Results for Mixtral 8x7B across multiple datasets.

Output	Context	Few Shots	Metric	Results Across Categories						
				Original	**CounterFact**	**Large**	**Small**	**Easy**	**Medium**	**Hard**
SQL	Schema	zero shots	REMS	19.59	18.32	20.56	27.60	29.05	20.28	17.29
			EMS	18.17	17.02	19.46	26.08	26.78	19.72	15.72
		Static	REMS	53.20	49.89	63.43	61.01	78.32	43.39	51.88
			EMS	51.90	48.64	62.28	59.53	75.82	43.39	50.63
		Adaptive	REMS	57.10	55.03	64.00	60.55	65.02	60.13	54.07
			EMS	55.88	53.50	63.17	58.95	63.93	58.38	51.74

Table 14: Evaluation Results for SQL Coder 70B across multiple datasets

Output	Context	Few Shots	Metric	Results Across Categories						
				Original	**CounterFact**	**Large**	**Small**	**Easy**	**Medium**	**Hard**
SQL	Schema	0	REMS	12.37	19.77	13.76	21.31	33.81	25.98	19.19
			EMS	12.22	19.46	13.76	21.00	33.47	25.64	18.64
		Static	REMS	16.53	33.29	29.82	42.04	54.13	41.62	39.09
			EMS	15.84	32.62	29.82	41.64	53.42	41.62	38.94
		Adaptive	REMS	24.02	40.73	38.00	48.56	65.56	51.78	41.63
			EMS	23.53	40.06	37.61	48.14	65.16	50.89	40.61

Table 15: Evaluation Results for Code Llama across multiple datasets

9 SQL Code Generation Prompt

Task Instruction:

You will be given a question and your task is to provide the SQL logic to answer a natural language question based on the provided schema. Few Examples of the task will be provided below. Assume that all the data is already inserted into the database.

1. Table Schemas:

```
CREATE TABLE Athlete (
    athlete_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL
);
CREATE TABLE Tournament (
    tournament_id INT AUTO_INCREMENT PRIMARY KEY,
    athlete_id INT,
    name VARCHAR(100) NOT NULL,
    FOREIGN KEY (athlete_id) REFERENCES Athlete(athlete_id)
);
CREATE TABLE Format (
    format_id INT AUTO_INCREMENT PRIMARY KEY,
    tournament_id INT,
    name VARCHAR(100) NOT NULL,
    FOREIGN KEY (tournament_id) REFERENCES Tournament(tournament_id)
);
CREATE TABLE Medal (
    medal_id INT AUTO_INCREMENT PRIMARY KEY,
    format_id INT,
    type VARCHAR(50) NOT NULL,
    year INT,
    location VARCHAR(100) NOT NULL,
    FOREIGN KEY (format_id) REFERENCES Format(format_id)
);
CREATE TABLE PersonalInformation (
    info_id INT AUTO_INCREMENT PRIMARY KEY,
    athlete_id INT,
    birth_year INT,
    birth_month INT,
    birth_day INT,
    FOREIGN KEY (athlete_id) REFERENCES Athlete(athlete_id)
);
```

2. Table Descriptions:

```
describe athlete;
+------------+--------------+------+-----+---------+----------------+
| Field      | Type         | Null | Key | Default | Extra          |
+------------+--------------+------+-----+---------+----------------+
| athlete_id | int(11)      | NO   | PRI | NULL    | auto_increment |
| name       | varchar(100) | NO   |     | NULL    |                |
+------------+--------------+------+-----+---------+----------------+
describe personalinformation;
+-------------+---------+------+-----+---------+----------------+
| Field       | Type    | Null | Key | Default | Extra          |
+-------------+---------+------+-----+---------+----------------+
| info_id     | int(11) | NO   | PRI | NULL    | auto_increment |
| athlete_id  | int(11) | YES  | MUL | NULL    |                |
| birth_year  | int(11) | YES  |     | NULL    |                |
| birth_month | int(11) | YES  |     | NULL    |                |
| birth_day   | int(11) | YES  |     | NULL    |                |
+-------------+---------+------+-----+---------+----------------+
describe tournament;
+---------------+--------------+------+-----+---------+----------------+
| Field         | Type         | Null | Key | Default | Extra          |
+---------------+--------------+------+-----+---------+----------------+
| tournament_id | int(11)      | NO   | PRI | NULL    | auto_increment |
| athlete_id    | int(11)      | YES  | MUL | NULL    |                |
| name          | varchar(100) | NO   |     | NULL    |                |
+---------------+--------------+------+-----+---------+----------------+
describe format;
+---------------+--------------+------+-----+---------+----------------+
| Field         | Type         | Null | Key | Default | Extra          |
+---------------+--------------+------+-----+---------+----------------+
| format_id     | int(11)      | NO   | PRI | NULL    | auto_increment |
| tournament_id | int(11)      | YES  | MUL | NULL    |                |
| name          | varchar(100) | NO   |     | NULL    |                |
```

```
+---------------+--------------+------+-----+---------+----------------+
describe medal;
+-----------+-------------+------+-----+---------+----------------+
| Field     | Type         | Null | Key | Default | Extra          |
+-----------+-------------+------+-----+---------+----------------+
| medal_id  | int(11)      | NO   | PRI | NULL    | auto_increment |
| format_id | int(11)      | YES  | MUL | NULL    |                |
| type      | varchar(50)  | NO   |     | NULL    |                |
| year      | int(11)      | YES  |     | NULL    |                |
| location  | varchar(100) | NO   |     | NULL    |                |
+-----------+-------------+------+-----+---------+----------------+
```

3. Example Data:

```
Athlete Table
+------------+----------------+
| athlete_id | name           |
+------------+----------------+
|         50 | Carolina Marín |
+------------+----------------+
PersonalInformation Table
+---------+------------+------------+-------------+-----------+
| info_id | athlete_id | birth_year | birth_month | birth_day |
+---------+------------+------------+-------------+-----------+
|      40 |         50 |       1993 |           6 |        15 |
+---------+------------+------------+-------------+-----------+
Tournament Table
+---------------+------------+------------------------------+
| tournament_id | athlete_id | name                         |
+---------------+------------+------------------------------+
|           281 |         50 | Olympic Games                |
|           282 |         50 | World Championships          |
|           285 |         50 | European Women               |
+---------------+------------+------------------------------+
Format Table
+-----------+---------------+-----------------+
| format_id | tournament_id | name            |
+-----------+---------------+-----------------+
|       392 |           281 | Women's singles |
|       393 |           282 | Women's singles |
|       396 |           285 | Women's team    |
+-----------+---------------+-----------------+
Medal Table
+----------+-----------+-------------+------+----------------+
| medal_id | format_id | type        | year | location       |
+----------+-----------+-------------+------+----------------+
|      692 |       392 | MedalGold   | 2016 | Rio de Janeiro |
|      696 |       393 | MedalSilver | 2023 | Copenhagen     |
|      706 |       396 | MedalBronze | 2016 | Kazan          |
+----------+-----------+-------------+------+----------------+
```

Example 1:
Question: Which tournament(s) has Zhang Jike won the most Medals in?

```sql
WITH medal_counts AS (
    SELECT t.name AS tournament_name, m.year, COUNT(m.medal_id) AS total_medal_count
    FROM Medal m
    JOIN Format f ON m.format_id = f.format_id
    JOIN Tournament t ON f.tournament_id = t.tournament_id
    JOIN Athlete a ON t.athlete_id = a.athlete_id
    WHERE a.name = 'Zhang Jike'
    GROUP BY t.name, m.year
)
SELECT tournament_name, year
FROM medal_counts
WHERE total_medal_count = (
    SELECT MAX(total_medal_count)
    FROM medal_counts
);
```

Example 2:
Question: In which year(s) did Seo Seung-jae win medals in the Asian Junior Championships?

```sql
SELECT DISTINCT m.year
FROM Medal m
JOIN Format f ON m.format_id = f.format_id
JOIN Tournament t ON f.tournament_id = t.tournament_id
JOIN Athlete a ON t.athlete_id = a.athlete_id
WHERE a.name = 'Seo Seung-jae'
  AND t.name = 'Asian Junior Championships';
```

Example 3:
Question: Which was the most current medal win for Dola Banerjee?

```sql
SELECT m.type, m.year, m.location, f.name AS format_name, t.name AS tournament_name
FROM Medal m
JOIN Format f ON m.format_id = f.format_id
JOIN Tournament t ON f.tournament_id = t.tournament_id
JOIN Athlete a ON t.athlete_id = a.athlete_id
WHERE a.name = 'Dola Banerjee'
  AND m.year = (
    SELECT MAX(m2.year)
    FROM Medal m2
    JOIN Format f2 ON m2.format_id = f2.format_id
    JOIN Tournament t2 ON f2.tournament_id = t2.tournament_id
    JOIN Athlete a2 ON t2.athlete_id = a2.athlete_id
    WHERE a2.name = 'Dola Banerjee'
  );
```

Example 4:
Question: How many international medals did Rawinda Prajongjai win in 2023?

```sql
SELECT COUNT(m.medal_id) AS total_medals
FROM Medal m
JOIN Format f ON m.format_id = f.format_id
JOIN Tournament t ON f.tournament_id = t.tournament_id
JOIN Athlete a ON t.athlete_id = a.athlete_id
WHERE a.name = 'Rawinda Prajongjai'
  AND m.year = 2023;
```

Example 5:
Question: In which year(s) did Huang Dongping win the highest number of medals during their career?

```sql
SELECT m.year
FROM Medal m
JOIN Format f ON m.format_id = f.format_id
JOIN Tournament t ON f.tournament_id = t.tournament_id
JOIN Athlete a ON t.athlete_id = a.athlete_id
WHERE a.name = 'Huang Dongping'
GROUP BY m.year
ORDER BY COUNT(m.medal_id) DESC
LIMIT 1;
```

Example 6:
Question: In which year(s) did Tomokazu Harimoto win the lowest number of medals during their career?

```sql
SELECT m.year
FROM Medal m
JOIN Format f ON m.format_id = f.format_id
JOIN Tournament t ON f.tournament_id = t.tournament_id
JOIN Athlete a ON t.athlete_id = a.athlete_id
WHERE a.name = 'Tomokazu Harimoto'
GROUP BY m.year
HAVING COUNT(m.medal_id) = (
    SELECT MIN(medal_count)
    FROM (
        SELECT COUNT(m2.medal_id) AS medal_count
        FROM Medal m2
        JOIN Format f2 ON m2.format_id = f2.format_id
        JOIN Tournament t2 ON f2.tournament_id = t2.tournament_id
        JOIN Athlete a2 ON t2.athlete_id = a2.athlete_id
        WHERE a2.name = 'Tomokazu Harimoto'
        GROUP BY m2.year
    ) AS yearly_medal_counts
);
```

Instructions for Writing Queries:

1. If a question can have multiple answers, do not limit the response to only one. Instead, output all possible answers.
2. Use the column names as specified in the schema to find the necessary parameters for the query.
3. An event is a combination of Tournament, Format, and the corresponding year.
4. There are three types of medals in the Medal Table: MedalGold, MedalSilver, MedalBronze.