# TabBridge: Bridging Structure and Context for Accurate Table Reasoning

**Jeongwoo Lee[1], Eunsoo Lee[2], Jihie Kim[2,†]**

[1]Department of Electronics and Electrical Engineering, Dongguk University
[2]Department of Computer Science and Artificial Intelligence, Dongguk University
{jwlee0519, dmstn7432}@dgu.ac.kr, jihie.kim@dgu.edu

## Abstract

Table reasoning remains challenging for Large Language Models (LLMs) as it requires integrating structured tabular information with natural language questions. Previous SQL-based approaches rely on surface-level alignment between question keywords and column headers, often generating queries with spurious or missing column mappings. We introduce TabBridge, a framework that incorporates both structural and contextual information for accurate table reasoning. TabBridge first generates a unified textual representation called Table Specification (TabSpec), preserving the structural information through row and column analysis. In order to ensure accuracy and consistency, we also employ a reconstruction-based evaluation mechanism to verify and refine the generated TabSpec. TabSpec is subsequently used to generate SQL aligned with the contextual intent of the question, enabling accurate interpretation of column semantics that are often overlooked by previous approaches. Across three public benchmarks, TabBridge shows consistent improvements over previous SQL-based methods, achieving 73.94% accuracy on WikiTableQuestions (+5.3 pp over the previous state of the art). TabBridge also demonstrates robust performance across diverse LLM backbones, confirming its generalizability across model architectures. Our code is available at https://github.com/raylee0519/TabBridge.

## 1 Introduction

Tabular data are a standard format for storing structured information and are widely used in real-world applications including relational databases and spreadsheets. Table reasoning involves textual, numerical, and logical inference (Ye et al., 2023), making the understanding of tabular data a key challenge in natural language processing (Ruan et al., 2024). This capability underpins tasks such as table-based fact verification (Chen et al., 2019), and question answering (Pasupat and Liang, 2015; Nan et al., 2021).

Recent research has explored symbolic approaches that generate SQL from table–question pairs and execute SQL queries for reasoning. H-STAR (Abhyankar et al., 2025) and Tab-SQLify (Nahid and Rafiei, 2024) adopt the Text-to-SQL paradigm to translate natural language questions into executable SQL queries, enabling structured reasoning via explicit query operations. However, Text-to-SQL inherently relies on schema linking to map question terms to column headers based on lexical or semantic similarity, making these methods remain fundamentally schema-bound (Lei et al., 2020). Directly applying this paradigm to Table Reasoning makes it difficult to accurately bridge the table's structure and the contextual intent of the question.

As illustrated in Figure 1, when a question employs terminology that differs substantially from the column header, schema linking fails to establish the correct correspondence. For example, the term "appearances" in the question does not directly align with the header "Total Apps", resulting in a mismatch at the schema level. Resolving this case requires reasoning beyond header-level matching by considering the full table context, specifically identifying that the numeric clue (37) mentioned in the question corresponds to the cell value (37, 22) under the "Total Apps" column.

To address this limitation, we pose the following research question: *How can LLMs perform accurate table reasoning that integrates both contextual and structural information?*

We answer this with **TabBridge**, motivated by the Construction-Integration (CI) model (Kintsch, 1988), which views comprehension as a two-stage process that first constructs structural representations and then integrates them with contextual information to produce a coherent interpretation. TabBridge adopts a two-stage framework combin-



*Proceedings of the First Workshop on Structured Understanding, Retrieval, and Generation in the LLM Era (SURGeLLM 2026)*, pages 219–240
July 3, 2026 ©2026 Association for Computational Linguistics

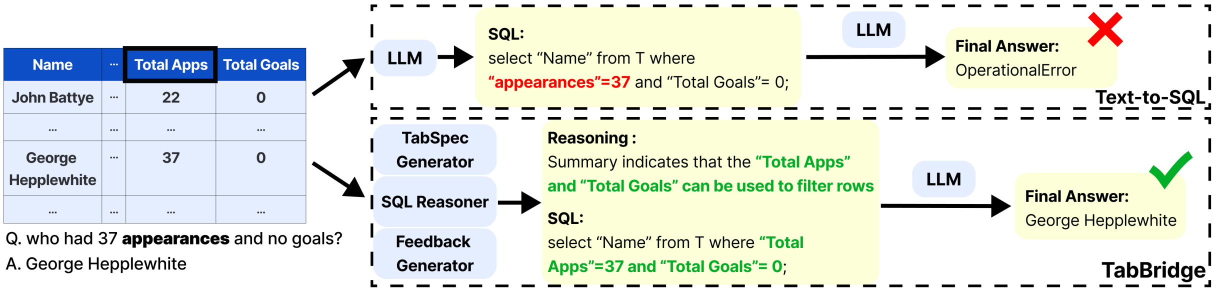


Figure 1: A representative limitation of Text-to-SQL-based table reasoning. When question terminology diverges from column headers, schema linking fails to bridge the table's structure and the contextual intent of the question. TabBridge constructs TabSpec to encode structural table information and establish accurate correspondence between question terms and column semantics.

ing Structural Encoding and Contextual Reasoning. The first stage, Structural Encoding, constructs a Table Specification (TabSpec) that explicitly preserves the table's structural information in a textual representation. The second stage, Contextual Reasoning, leverages the verified TabSpec to interpret questions in context and generate SQL aligned with both query intent and table structure.

We evaluate TabBridge on standard table reasoning benchmarks against previous SQL-based approaches. TabBridge achieves consistent improvements over existing baselines and substantially reduces overall error rates. In particular, TabBridge achieves 73.94% accuracy on WikiTableQuestions, improving by +5.3 pp over the previous state of the art. These results demonstrate that integrating structural encoding with contextual interpretation yields more accurate reasoning.

Our contributions are as follows:

1. We propose TabBridge, a two-stage framework bridging structural encoding and contextual reasoning.

2. We design a reconstruction-based evaluation to assess the structural fidelity of TabSpec.

3. We evaluate TabBridge on three public benchmarks, achieving 73.94% accuracy on WikiTableQuestions and improved semantic consistency in free-form reasoning.

## 2 Related Work

### 2.1 Semantic Parsing : Text-to-SQL

Text-to-SQL seeks to translate natural language questions into executable SQL queries by grounding the questions in database schemas encompassing tables, columns, and relations, a task that has seen significant progress since the introduction of WikiSQL (Zhong et al., 2017) and Spider (Yu et al., 2018). Subsequent studies have focused on improving the alignment between complex schema structures and natural language expressions through schema encoding, constraint-based decoding, and pretraining techniques (Guo et al., 2019; Bogin et al., 2019; Wang et al., 2019; Deng et al., 2021). However, the semantic mismatch between natural language queries and schema elements remains a persistent issue. Such misalignment frequently leads to incorrect column references or faulty conditions, producing inaccurate SQL queries (Qin et al., 2022; Huang et al., 2023). Although recent models have enhanced schema-level alignment, their ability to leverage content-level information derived from actual tables is still limited (Lei et al., 2020; Pi et al., 2022).

### 2.2 Table Reasoning

Early studies on table reasoning adopted an end-to-end paradigm, taking entire tables as input to directly generate answers (Yin et al., 2020; Herzig et al., 2020; Liu et al., 2021). While effective for simple queries, these approaches struggled with complex numerical computations and multi-step logical reasoning. The introduction of Chain-of-Thought (CoT) prompting (Tai et al., 2023; Chen et al., 2022) improved interpretability but these approaches relied heavily on token-level reasoning, limiting both numerical accuracy and scalability for long tables (Sui et al., 2024).

To address these limitations, Text-to-SQL based approaches have been introduced. SynTQA (Zhang et al., 2024b) converts natural language questions into executable SQL queries, while H-STAR (Abhyankar et al., 2025) and TabSQLify (Nahid and Rafiei, 2024) utilize SQL operations to process

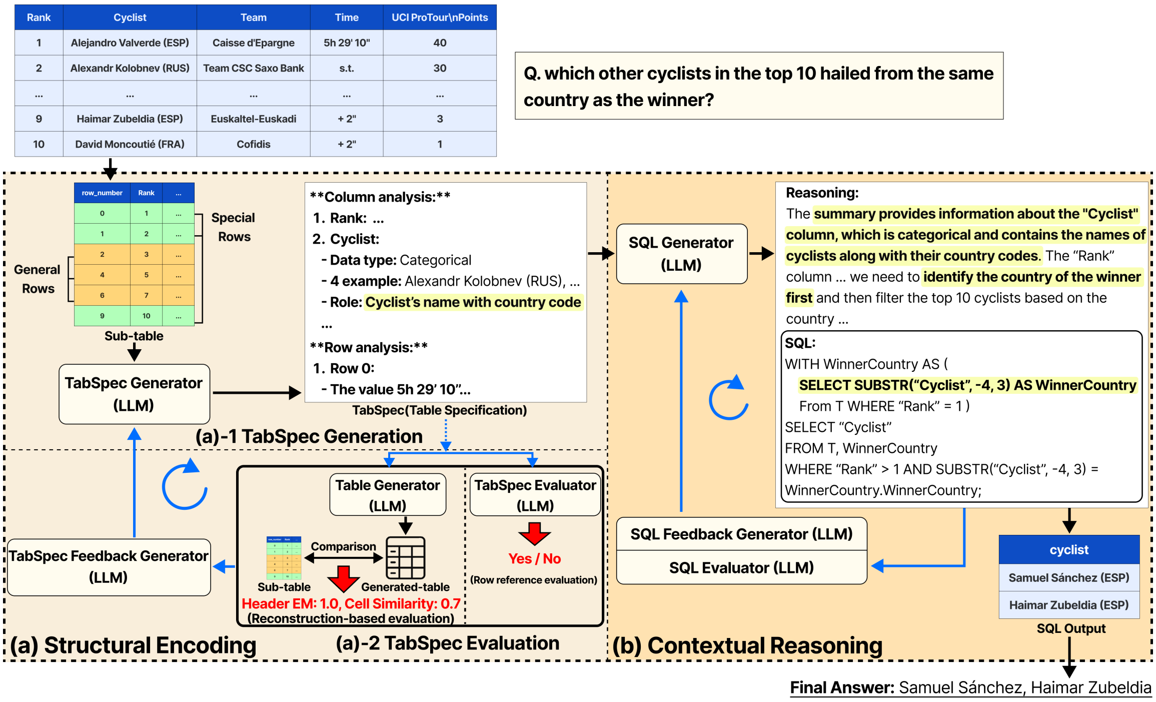


Figure 2: Overview of the TabBridge framework. **(a) Structural Encoding**: Sub-table generation extracts general and special rows, followed by TabSpec generation through column and row analysis. The generated TabSpec undergoes evaluation and refinement via a reconstruction-based evaluation and a row reference evaluation. **(b) Contextual Reasoning**: The refined TabSpec guides SQL generation, which is validated through SQL evaluation with iterative refinement for final answer derivation.

numerical queries and extract relevant sub-tables. Weaver (Khoja et al., 2025) combines SQL and LLM depending on the situation, complementing each other's limitations. Despite these advancements, such methods remain schema-bound, relying heavily on header information rather than actual table content. This reliance often leads to SQL queries that misinterpret question intent or exhibit semantic inconsistencies (Zhong et al., 2017; Yu et al., 2018; Wang et al., 2019).

# 3 Method

Motivated by the Construction-Integration (CI) model (Kintsch, 1988), which views comprehension as a two-stage process, TabBridge separates the construction of structural representations from their integration with contextual information. This separation ensures that structural fidelity is established before contextual interpretation begins, directly addressing the schema-bound limitations of Text-to-SQL approaches. Figure 2 illustrates the overview of the TabBridge framework. Section 3.1 describes the process of generating TabSpec through **Structural Encoding**. Section 3.2 explains how TabSpec is evaluated and refined. Section 3.3 describes how TabSpec supports **Contextual Reasoning** to produce SQL queries, reflecting both structural and contextual understanding.

## 3.1 Structural Encoding

Grounded in the CI model's construction stage, which emphasizes building explicit structural representations prior to contextual integration, the Structural Encoding process is designed to capture and preserve the structure of a table, regardless of its size. To achieve this, we first construct a sub-table composed of general rows and special rows. The encoded representation, referred to as the Table Specification (TabSpec), is then generated from the sub-table, as shown in Figure 2 (a).

**Sub-table Generation.** To consistently preserve structural information across tables of varying sizes, we extract two complementary row types from the original table. General rows reflect the dominant column-wise distribution, while special rows exhibit structural divergence, such as totals, anchors, or section dividers. To capture this distinction, we compute pairwise Euclidean distances across all rows, selecting the $k = 3$ rows with the lowest average pairwise distance as general rows and those with the highest as special rows, where the optimal value of $k$ is validated through an ablation study (Section A.1).

**TabSpec Generation.** The TabSpec Generator (LLM) receives the generated sub-table as input and performs **(1) column analysis** and **(2) row analysis** to construct a text-based TabSpec that encodes structural information.

(1) **Column analysis** encodes each column's data type (numeric, categorical, date, text, mixed), role (identifier, metric, reference), and representative examples. For example, as shown in Figure 2 (a), the *Cyclist* column is recognized as categorical, representing cyclist names that include country codes such as "Alexandr Kolobnev (RUS)". (2) **Row analysis** identifies and retains special rows that exhibit structural divergence from the general row distribution, along with their classification rationale. Since SQL generation operates at the column level, explicitly identifying irregular rows provides guidance for accurate question answering grounded in the table structure. For instance, as shown in the sub-table in Figure 2 (a), rows 0, 1, and 9 are identified as special rows during sub-table generation. The TabSpec Generator records row 0 with the actual time value "5h 29' 10”" in the *Time* column and notes that other rows (e.g., s.t., +2) are defined relative to it. This identification prevents the model from treating relative expressions such as s.t. and +2 as independent values, thereby ensuring accurate extraction of the reference value in the *Time* column.

## 3.2 TabSpec Evaluation and Refinement

In this section, we describe the evaluation and refinement of the structural information encoded in the TabSpec generated in Section 3.1. We employ a row reference evaluation to confirm accurate representation of special rows, and a reconstruction-based evaluation to verify the overall structural consistency. Figure 3 illustrates the detailed process of TabSpec evaluation and refinement.

**Row reference evaluation.** As illustrated in Figure 3 (a), we assess two factors to evaluate the correctness of the row analysis : **(1) Special row identification** evaluates whether the rows specified in the row analysis are accurately described, ensuring that even correctly identified rows are not accompanied by incorrect explanations. **(2) Accuracy of row classification** evaluates whether unnecessary rows are included in the row analysis. This step examines rows initially identified as

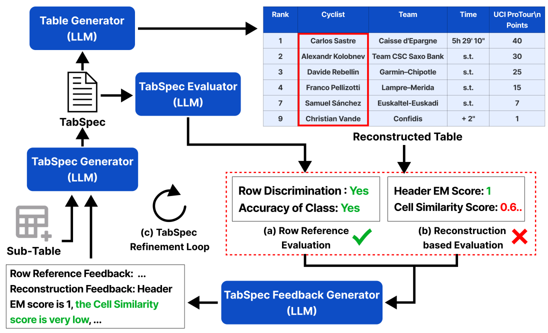


Figure 3: Details in TabSpec Evaluation and Refinement

special based on sub-table construction, ensuring that only structurally significant special rows are retained and that general rows are not misclassified. Prior work shows that pairwise comparison yields higher accuracy in LLM-based evaluation (Gu et al., 2025). Accordingly, we restrict row reference evaluation to binary decisions (yes/no) to ensure consistency and reproducibility.

**Reconstruction-based evaluation.** A faithful TabSpec should contain sufficient structural information to recover the original table. Based on this principle, we reconstruct a table from the generated TabSpec using the table generation model from Map&Make (Ahuja et al., 2025), and compare it against the original sub-table to quantitatively assess structural fidelity.

Two quantitative metrics are employed to jointly assess structural fidelity at both the column and cell levels, as shown in Figure 3 (b). **(1) Header EM (Exact Match)** measures whether the reconstructed column names exactly match the original ones, evaluating column-level alignment accuracy and ensuring that correct columns can be referenced without errors during subsequent SQL generation and execution. **(2) Cell Similarity** calculates the BERTScore between corresponding cell values, capturing semantically equivalent expressions across different surface forms and evaluating whether the TabSpec preserves the underlying data distribution.

**TabSpec Refinement Loop.** When a TabSpec does not satisfy either the reconstruction-based or the row reference evaluation, an LLM-generated feedback signal, together with the original sub-table, is fed back to the TabSpec Generator to improve the specification. For instance, as illustrated in Figure 3 (c), if the TabSpec omits that the *Cyclist* column contains country codes, the reconstructed table also lacks these values, producing a low cell similarity score. The evaluator reports this discrepancy and score as feedback, prompting the generator to revise the TabSpec accordingly. This refinement loop continues until all criteria are satisfied, with a maximum of $n = 2$ iterations to prevent excessive computation, where the optimal limit is validated through an ablation study (Section A.2).

### 3.3 Contextual Reasoning

Following the CI model's integration stage, this section describes how TabSpec is leveraged to generate SQL queries that align with both the structural information of the table and the contextual intent of the question. For this, we introduce the SQL evaluation, consisting of Question Alignment and TabSpec Alignment, which verify whether the generated SQL reflects both the contextual intent of the question and the table structure, as illustrated in Figure 2 (b).

**SQL Generation.** The SQL generator takes as input the question $q$, the original table $T$, and the TabSpec, and produces an SQL. For example, when a question requires reasoning over the *country* dimension, the model refers to the **column analysis** (Section 3.1) in the TabSpec, recognizing that the *Cyclist* column contains embedded country codes. Through this contextual linkage, the model correctly interprets *Cyclist* as a categorical column associated with the country and generates the SQL query that successfully retrieves cyclists from the same country (e.g., "ESP").

**SQL Evaluation.** The SQL Evaluator compares the generated SQL against both the TabSpec and the given question to ensure structural and contextual consistency. This evaluation consists of two components : **(1) Question Alignment** verifies that the selected columns are consistent with the question's context and that special rows identified in the TabSpec are properly referenced when contextually relevant. **(2) TabSpec Alignment** verifies whether the SQL query introduces structural elements undefined in the TabSpec and whether operations conform to the specified data types (e.g., not treating categorical values as numeric). For instance, if the TabSpec specifies that the *Cyclist* column contains country codes but the SQL query ignores the country constraint, the TabSpec Alignment component detects this mismatch.

**SQL Refinement Loop.** If the SQL fails either evaluation, the evaluation results are provided as feedback and the SQL is regenerated to improve its structural and contextual consistency. This refinement loop continues until all criteria are satisfied, with a maximum of $n = 2$ iterations to prevent excessive computation, where the optimal limit is validated through an ablation study (Section A.2).

## 4 Experiments

### 4.1 Experimental Setup

**Preprocessing.** The raw tables were converted into pandas DataFrames with unique headers and standardized numeric and date formats, without removing or aggregating any rows during the process. Each table was then serialized into a Markdown-style linearized format to preserve structural alignment within text-based prompts.

**Benchmark Datasets.** We evaluate TabBridge on three public benchmarks covering complementary aspects of table reasoning. **(1) WikiTableQuestions (WTQ)** (Pasupat and Liang, 2015) is a large-scale question answering benchmark requiring filtering, aggregation, and comparison over Wikipedia tables (4,344 test pairs). **(2) TabFact** (Chen et al., 2019) is a fact verification benchmark where each statement is labeled as entailed or refuted by the table (2,024 test statements across 298 tables). **(3) FeTaQA** (Nan et al., 2021) is a free-form question answering benchmark requiring natural language answers grounded in tabular content (2,003 test samples).

**Baseline Methods.** We compare against two categories of methods: (1) Non-SQL-based reasoning, which includes End-to-End QA , Few-shot QA, and TableCoT (Jin and Lu, 2023); (2) SQL-based and SQL-assisted reasoning, which includes Text-to-SQL, NormTab (Nahid and Rafiei, 2025), DATER (Ye et al., 2023), ReAcTable (Zhang et al., 2024c), ALTER (Zhang et al., 2024a), H-STAR (Abhyankar et al., 2025), and TabSQLify (Nahid and Rafiei, 2024). Through this comparison, we analyze how TabBridge improves over existing SQL-centric approaches in terms of accuracy, interpretability, and structural consistency.

**Evaluation Metrics.** We adopt task-specific evaluation metrics to measure both the precision and faithfulness of reasoning. For WikiTableQuestions, we use Exact Match (EM), which assesses whether the SQL execution result exactly matches the gold-standard answer. For TabFact, we adopt binary classification accuracy, as the task requires determining whether a given statement is entailed or refuted by the table. For FeTaQA, we evaluate generated free-form answers using ROUGE-1, ROUGE-2, and ROUGE-L (Lin, 2004), which measure the degree of lexical overlap with reference answers.

## 5 Results

### 5.1 Main Results

Table 1 reports performance on both WikiTableQuestions and TabFact. TabBridge uses the ($k$/$k$) sub-table setting, where $k$ general and $k$ special rows are selected. We evaluate TabBridge against general reasoning baselines and SQL-based methods. On WikiTableQuestions, TabBridge (3/3) reaches 73.94% and TabBridge (4/4) reaches 71.53%, outperforming NormTab (TabSQLify) (68.63%) and H-STAR (Text+SQL) (68.62%). On TabFact, TabBridge (3/3) achieves 83.40% and TabBridge (4/4) achieves 80.48%, remaining competitive with strong baselines such as TabSQLify (85.03%) and H-STAR (Text+SQL) (86.51%), both of which incorporate text-based interpretation alongside SQL execution.

These results show that integrating structural and contextual information during SQL generation enhances both condition interpretation and logical reasoning, leading to consistent gains across table reasoning benchmarks. A more detailed analysis of performance variation according to the number of selected rows is presented in Section A.1.

### 5.2 Results in FeTaQA

Table 2 presents the FeTaQA results, which evaluate free-form answer generation using ROUGE metrics. TabBridge achieved ROUGE-1 of 0.70, ROUGE-2 of 0.45, and ROUGE-L of 0.57, showing performance competitive with recent table reasoning models. In particular, the column role and data type information encoded in TabSpec contributes to accurately referencing relevant columns during free-form answer generation, leading to semantically consistent descriptions. These results demonstrate that TabBridge is effective not only for structured reasoning but also for free-form reasoning.

| Method | TabFact (%) | WikiTQ (%) |
|---|---|---|
| End-to-End QA | 70.45 | 51.84 |
| Few-shot QA | 71.54 | 52.56 |
| TableCoT | 73.10 | 52.40 |
| H-STAR(text) | 79.43 | 61.47 |
| Text-to-SQL | 52.05 | 42.03 |
| NormTab + SQL | 68.90 | 61.20 |
| Dater | 78.01 | 52.90 |
| ReAcTable | 73.10 | 52.50 |
| ALTER | 84.30 | 67.40 |
| H-STAR(SQL) | 58.50 | 46.09 |
| H-STAR(Text+SQL) | **86.51** | 68.62 |
| TabSQLify | 85.03 | 64.70 |
| NormTab + TabSQLify | – | 69.56 |
| **TabBridge (4/4)** | 80.48 | 71.53 |
| **TabBridge (3/3)** | 83.40 | **73.94** |

Table 1: Performance comparison on TabFact and WikiTableQuestions. Results are reported as accuracy (%).

| Method | ROUGE-1 | ROUGE-2 | ROUGE-L |
|---|---|---|---|
| TableCoT | 0.62 | 0.39 | 0.51 |
| Dater | 0.66 | 0.45 | 0.56 |
| ReAcTable | **0.71** | **0.46** | **0.61** |
| TabSQLify | 0.58 | 0.35 | 0.48 |
| **TabBridge** | 0.70 | 0.45 | 0.57 |

Table 2: FeTaQA results evaluated with ROUGE metrics. TabBridge achieves performance comparable to state-of-the-art methods.

### 5.3 Evaluation Across LLMs

We further evaluate TabBridge under diverse LLM backbones and parameter scales, including GPT-3.5-turbo (Brown et al., 2020), LLaMA2, LLaMA3.3 (Touvron et al., 2023), Qwen2.5 (Qwen et al., 2025), and Gemma (Team et al., 2024). As summarized in Table 3, TabBridge achieves strong performance across most model families, with consistent gains over TabSQLify, NormTab, and H-STAR in the majority of settings. Performance gains are consistent across most model families and notably persist even with smaller models, indicating that the method's benefits are largely independent of architecture and scale. These results demonstrate that TabBridge serves as a model-agnostic preprocessing framework that enhances SQL reasoning through structurally grounded contextual representations.

| Method | GPT-3.5 Turbo | Qwen2.5 7B | Qwen2.5 14B | Qwen2.5 72B | LLaMA3.3 70B | Gemma2 9B | Gemma2 27B | LLaMA2 7B | LLaMA2 13B | LLaMA2 70B |
|---|---|---|---|---|---|---|---|---|---|---|
| TabSQLify | 64.70 | 69.57 | 70.63 | 74.36 | 79.21 | 57.57 | 64.62 | 25.37 | 36.67 | 55.57 |
| NormTab(SQL) | 61.20 | 53.26 | 55.80 | 65.30 | 66.97 | 54.39 | 62.08 | 21.65 | 37.54 | 51.27 |
| NormTab(TabSQLify) | 68.63 | 57.60 | 60.70 | 71.32 | 72.43 | 59.99 | 69.99 | 33.88 | 42.86 | 55.87 |
| H-STAR | 69.56 | 35.45 | 40.19 | 59.01 | 61.83 | 42.54 | 58.68 | 15.88 | 23.51 | 38.58 |
| **TabBridge** | **73.94** | **71.00** | **74.26** | **77.10** | **82.59** | **59.01** | **69.75** | **41.87** | **46.21** | **60.09** |

Table 3: WTQ results grouped by LLM family with sub-model sizes. Scores are reported as percentages. TabBridge is bolded. NormTab data are set using Targeted.

### 5.4 Analysis of Prior Error Cases

We conduct an error analysis on 100 error cases previously reported by TabSQLify. Errors are categorized into four types: **(1) Missing Rows/Columns**, where rows or columns are omitted during table manipulation; **(2) Incorrect Annotation**, where dataset instances are mislabeled despite correct model outputs; **(3) Incorrect Reasoning**, where the model produces incorrect answers; and **(4) Correct Answers**, where the generated answer matches the ground truth.

As illustrated in Figure 4, unlike sub-table methods, TabBridge reasons over the complete table without discarding any rows from the original table, and thus incurs no Missing Rows/Columns errors. In contrast, TabSQLify and H-STAR recorded 62 and 20 such cases, respectively, confirming that preserving full table structure reduces information loss and leads to more accurate reasoning.

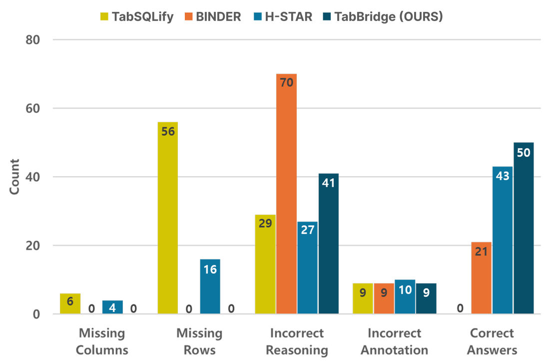


Figure 4: Error analysis on 100 error cases provided by TabSQLify

### 5.5 Component Analysis

To verify the necessity of each component, we progressively remove key modules and evaluate on WikiTableQuestions. As shown in Table 4, the complete TabBridge achieves 73.94% accuracy. Removing SQL Evaluation & Refinement reduces performance to 67.29%, and further removing TabSpec Evaluation & Refinement lowers it to 64.52%. Eliminating TabSpec Generation yields 58.76%, confirming that each component contributes cumulatively to the overall performance.

| Method | WikiTQ |
|---|---|
| **TabBridge** | **73.94** |
| w/o SQL Evaluation & Refinement | 67.29 |
| w/o TabSpec Evaluation & Refinement | 64.52 |
| w/o TabSpec Generation | 58.76 |

Table 4: Component analysis on WikiTableQuestions. Each row removes a key module to assess its contribution.

### 5.6 Efficiency Analysis

To analyze computational efficiency, we measure average token consumption and runtime per sample across 50 randomly selected WikiTableQuestions. As shown in Table 5, token usage and runtime increase consistently with the number of TabSpec iterations, where (1-iter) applies a single SQL refinement loop, (2-iter) incorporates one TabSpec refinement and one SQL refinement, and (3-iter) performs two TabSpec refinement rounds and one SQL refinement. Token consumption increases substantially with additional TabSpec refinement rounds, as each round requires multiple sequential LLM invocations for evaluation and feedback generation, resulting in cumulative latency.

## 6 Conclusion

In this paper, we propose TabBridge, a framework that addresses the schema-bound limitations of Text-to-SQL approaches by integrating structural encoding and contextual reasoning. TabBridge employs a reconstruction-based evaluation loop that

| Method | Token | Runtime (s) |
|---|---|---|
| NormTab + SQL | 3,757 | 16.2 |
| NormTab + TabSQLify | 5,720 | 20.7 |
| H-STAR | 9,768 | 42.1 |
| TabBridge (1-iter) | 3,328 | 15.3 |
| TabBridge (2-iter) | 6,493 | 38.4 |
| TabBridge (3-iter) | 10,482 | 58.6 |

Table 5: Efficiency comparison of table reasoning systems averaged over 50 randomly sampled WikiTableQuestions test instances.

verifies and refines the Table Specification (TabSpec) before SQL generation, ensuring structural fidelity. With the refined TabSpec, TabBridge performs contextual reasoning by combining the complete structural understanding of the table with the question to generate accurate SQL. Experimental results show that TabBridge achieves 73.94% on WikiTableQuestions (+5.3 pp over the previous state of the art) and demonstrates consistent improvements across diverse LLM backbones, confirming its generalizability across model architectures. In future work, we plan to explore extending TabSpec to multi-turn reasoning and domain-specific tables.

## Limitations

While TabBridge achieves significant performance gains, we identify several limitations that we aim to address in future work. First, the multi-stage pipeline, which comprises TabSpec generation, verification, and iterative SQL refinement, incurs higher latency and token consumption compared to single-pass methods. This trade-off between reasoning depth and practical efficiency may affect deployment in resource-constrained settings. To mitigate this, future research will explore caching strategies for frequently accessed table structures and distillation techniques to streamline the agentic workflow into a more efficient model. Furthermore, while our current framework utilizes a fixed sampling of general and special rows based on its empirical effectiveness for the evaluated benchmarks, this strategy may face scalability challenges with massive or extremely heterogeneous tables. Extending our approach to dynamic, content-aware sampling and evaluating it across even more complex schemas, such as hierarchical or multi-table databases, will be crucial for maintaining generalizability in diverse real-world scenarios.

## 7 Acknowledgments

This research was supported by the MSIT(Ministry of Science and ICT), Korea, under the ITRC(Information Technology Research Center) support program(IITP-2026-RS-2020-II201789), and the Artificial Intelligence Convergence Innovation Human Resources Development(IITP-2026-RS-2023-00254592) supervised by the IITP(Institute for Information & Communications Technology Planning & Evaluation).

## References

Nikhil Abhyankar, Vivek Gupta, Dan Roth, and Chandan K. Reddy. 2025. H-STAR: LLM-driven hybrid SQL-text adaptive reasoning on tables. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 8841–8863, Albuquerque, New Mexico. Association for Computational Linguistics.

Naman Ahuja, Fenil Bardoliya, Chitta Baral, and Vivek Gupta. 2025. Map&make: Schema guided text to table generation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 30249–30262.

Ben Bogin, Matt Gardner, and Jonathan Berant. 2019. Representing schema structure with graph neural networks for text-to-sql parsing. *Preprint*, arXiv:1905.06241.

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, and 12 others. 2020. Language models are few-shot learners. *Preprint*, arXiv:2005.14165.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. 2022. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*.

Wenhu Chen, Hongmin Wang, Jianshu Chen, Yunkai Zhang, Hong Wang, Shiyang Li, Xiyou Zhou, and William Yang Wang. 2019. Tabfact : A large-scale dataset for table-based fact verification. In *International Conference on Learning Representations (ICLR)*, Addis Ababa, Ethiopia.

Xiang Deng, Ahmed Hassan Awadallah, Christopher Meek, Oleksandr Polozov, Huan Sun, and Matthew Richardson. 2021. Structure-grounded pretraining for text-to-sql. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, page 1337–1350. Association for Computational Linguistics.

Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, Saizhuo Wang, Kun Zhang, Yuanzhuo Wang, Wen Gao, Lionel Ni, and Jian Guo. 2025. A survey on llm-as-a-judge. *Preprint*, arXiv:2411.15594.

Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019. Towards complex text-to-sql in cross-domain database with intermediate representation. In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pages 4524–4535.

Jonathan Herzig, Paweł Kasper 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*.

Zezhou Huang, Pavan Kalyan Damalapati, and Eugene Wu. 2023. Data ambiguity strikes back: How documentation improves gpt's text-to-sql. *arXiv preprint arXiv:2310.18742*.

Ziqi Jin and Wei Lu. 2023. Tab-CoT: Zero-shot tabular chain of thought. In *Findings of the Association for Computational Linguistics: ACL 2023*.

Rohit Khoja, Devanshu Gupta, Yanjie Fu, Dan Roth, and Vivek Gupta. 2025. Weaver: Interweaving sql and llm for table reasoning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 28270–28296.

Walter Kintsch. 1988. The role of knowledge in discourse comprehension: a construction-integration model. *Psychology review*, 95(2):163.

Wenqiang Lei, Weixin Wang, Zhixin Ma, Tian Gan, Wei Lu, Min-Yen Kan, and Tat-Seng Chua. 2020. Re-examining the role of schema linking in text-to-SQL. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6990–6996, Online. Association for Computational Linguistics.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out: Proceedings of the ACL-04 Workshop*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

Qian Liu, Bei Chen, Jiaqi Guo, Morteza Ziyadi, Zeqi Lin, Weizhu Chen, and Jian-Guang Lou. 2021. TAPEX: Table pre-training via learning a neural SQL executor. In *The International Conference on Learning Representations (ICLR)*.

Md Mahadi Hasan Nahid and Davood Rafiei. 2024. Tabsqlify: Enhancing reasoning capabilities of llms through table decomposition. *Preprint*, arXiv:2404.10150.

Md Mahadi Hasan Nahid and Davood Rafiei. 2025. Normtab: Improving symbolic reasoning in llms through tabular data normalization. *Preprint*, arXiv:2406.17961.

Linyong Nan, Chiachun Hsieh, Ziming Mao, Xi Victoria Lin, Neha Verma, Rui Zhang, Wojciech Kryściński, Nick Schoelkopf, Riley Kong, Xiangru Tang, Murori Mutuma, Ben Rosand, Isabel Trindade, Renusree Bandaru, Jacob Cunningham, Caiming Xiong, and Dragomir Radev. 2021. Fetaqa: Free-form table question answering. *Preprint*, arXiv:2104.00369.

Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables. *Preprint*, arXiv:1508.00305.

Xinyu Pi, Bing Wang, Yan Gao, Jiaqi Guo, Zhoujun Li, and Jian-Guang Lou. 2022. Towards robustness of text-to-SQL models against natural and realistic adversarial table perturbation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1969–1981, Dublin, Ireland. Association for Computational Linguistics.

Bowen Qin, Binyuan Hui, Lihan Wang, Min Yang, Jinyang Li, Binhua Li, Ruiying Geng, Rongyu Cao, Jian Sun, Luo Si, Fei Huang, and Yongbin Li. 2022. A survey on text-to-sql parsing: Concepts, methods, and future directions. *Preprint*, arXiv:2208.13629.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, and 25 others. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

Yucheng Ruan, Xiang Lan, Jingying Ma, Yizhi Dong, Kai He, and Mengling Feng. 2024. Language modeling on tabular data: A survey of foundations, techniques and evolution. *arXiv preprint arXiv:2408.10548*.

Yuan Sui, Mengyu Zhou, Mingjie Zhou, Shi Han, and Dongmei Zhang. 2024. Table meets llm: Can large language models understand structured table data? a benchmark and empirical study. *Preprint*, arXiv:2305.13062.

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

Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, and 1 others. 2024. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. Llama: Open and efficient foundation language models. *Preprint*, arXiv:2302.13971.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2019. Rat-sql: Relation-aware schema encoding and linking for text-to-sql parsers. In *Proceedings of the 58th annual meeting of the association for computational linguistics*, pages 7567–7578.

Yunhu Ye, Binyuan Hui, Min Yang, Binhua Li, Fei Huang, and Yongbin Li. 2023. 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*, pages 174–184.

Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. Tabert: Pretraining for joint understanding of textual and tabular data. In *Proceedings of the 58th annual meeting of the association for computational linguistics*, pages 8413–8426.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shan Roman, Zilin Wang, and Dragomir Radev. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921, Brussels, Belgium. Association for Computational Linguistics.

Han Zhang, Yuheng Ma, and Hanfang Yang. 2024a. Alter: Augmentation for large-table-based reasoning. *Preprint*, arXiv:2407.03061.

Siyue Zhang, Anh Tuan Luu, and Chen Zhao. 2024b. Syntqa: Synergistic table-based question answering via mixture of text-to-sql and e2e tqa. *Preprint*, arXiv:2409.16682.

Zihan Zhang, Mengxuan Fan, Jajhyung Park, Yelong Shen, Nan Duan, Ruoxi Jia, and Chen-Yu Lee. 2024c. ReAcTable: Enhancing tabular reasoning with reasoning-acting synergy. In *The Twelfth International Conference on Learning Representations (ICLR)*.

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

# A Ablation Study

This section presents ablation studies designed to examine the effect of row selection during sub-table generation and the optimal number of refinement loops in SQL and TabSpec generation. The experiments were conducted on 120 samples randomly selected from the WikiTableQuestions dataset.

## A.1 Effect of Row Selection in Sub-table Generation.

The table 6 shows that the 3/3 configuration (three general and three special rows) achieved the highest accuracy of 76.67%, outperforming both smaller and larger sub-tables as well as the full table. Using fewer rows (2/2) failed to capture enough structural variation, while including more rows (4/4, 5/5) introduced noise that degraded performance. The full table setting yields the lowest accuracy, as structural cues were diluted across excessive context. These findings indicate that a balanced selection of three representative rows and three special rows provides the optimal trade-off, preserving both regularity and variation while minimizing noise in table reasoning. The current approach uses a fixed value of k across all tables; dynamically adjusting k based on table size remains a promising direction for future work. Detailed results for each row selection are shown in Figure5, 6.

| Setting | Correct / Total | Accuracy |
|---|---|---|
| 2/2 | 74/120 | 61.67% |
| 3/3 | 92/120 | 76.67% |
| 4/4 | 87/120 | 68.33% |
| 5/5 | 78/120 | 65.00% |
| Full Table | 68/120 | 56.67% |

Table 6: Ablation study on row selection for sub-table generation

## A.2 Effect of Refinement Loops

We examine the impact of iterative refinement on both the TabSpec and SQL evaluation stages by varying the number of loop iterations to identify the configuration that balances accuracy and computational cost.

**TabSpec Refinement Loops.** As shown in Table 7, accuracy improves progressively with the number of iterations, reaching 77.50% at the third loop. Given the marginal improvement between the second and third loops, we adopt the second-loop configuration in the main experiments.

| Iteration | Correct / Total | Accuracy |
|---|---|---|
| 0 | 78/120 | 65.00% |
| 1 | 87/120 | 72.50% |
| 2 | 92/120 | 76.67% |
| 3 | 93/120 | **77.50%** |

Table 7: Ablation study on TabSpec refinement loops

**SQL Refinement Loops.** As shown in Table 8, accuracy peaks at 76.67% in the second loop and slightly declines thereafter, suggesting that two refinement iterations provide the optimal balance between correction and stability in SQL reasoning.

| Iteration | Correct / Total | Accuracy |
|---|---|---|
| 0 | 76/120 | 63.33% |
| 1 | 86/120 | 71.66% |
| 2 | 92/120 | **76.67%** |
| 3 | 90/120 | 75.00% |

Table 8: Ablation study on SQL refinement loops

## B Example Figures

We include example inputs and corresponding outputs used in the ablation study to provide qualitative illustration of each configuration's reasoning behavior. Figures 5 ~10 are presented in this section.

## C Example Prompts

We provide the prompts used in our experiments. Figures 11 ~22 present the full prompt templates employed across the reasoning stages.

—

**Comparison of Linux distributions**

| Distribution | x86 | x86-64 | ia64 | ppc64 | ... | alpha | m68k |
|---|---|---|---|---|---|---|---|
| Arch Linux | Yes (>=i686) | Yes | No | No | ... | No | No |
| BackTrack | Yes | Yes | No | No | ... | No | No |
| Bodhi Linux | Yes | Yes | No | No | ... | No | No |
| BOSS Linux | Yes | Yes | No | Yes | ... | No | No |
| ... | ... | ... | ... | ... | ... | ... | ... |
| Yellow Dog Linux | No | No | No | Yes | ... | No | No |
| Distribution | x86 | x86-64 | ia64 | ppc64 | ... | alpha | m68k ✗ |

Question: how mant distributors are on the data table

2×2 Sub-Table

Generated Reasoning SQL:
To determine the number of unique distributors in the table, we need to consider the "Distribution" column, which is categorical and serves as the identifier for each distribution. We should count the distinct values in this column to find the total number of unique distributors.

Generated Final SQL:
SELECT COUNT(DISTINCT "Distribution") AS unique_distributors FROM T;

Final Answer: 35 ✗

3×3 Sub-Table **(Ours)**

Generated Reasoning SQL:
To determine the number of distributors in the data table, we need to consider the "Distribution" column, which contains categorical values representing different Linux distributions. We should count the distinct values in this column to find the total number of unique distributors present in the table. Special information such as row 34, which contains the placeholder "Distribution" instead of a specific distribution name, should be excluded from the count.

Generated Final SQL:
SELECT COUNT(DISTINCT "Distribution") AS distributor_count FROM T WHERE "Distribution" NOT LIKE '%Distribution%';

Final Answer: 34 ✓

4×4 Sub-Table

Generated Reasoning SQL:
To find the number of distributors in the table, we need to consider the "Distribution" column, which contains categorical values identifying each Linux distribution. We should count the distinct values in this column to find the total number of unique distributors present in the table.

Generated Final SQL:
SELECT COUNT(DISTINCT "Distribution") AS distributor_count FROM T;

Final Answer: 35 ✗

5×5 Sub-Table

Generated Reasoning SQL:
To determine the number of distributors present in the dataset, we need to consider the "Distribution" column, which is categorical and serves as the primary identifier for each distribution. We should count the distinct values in this column to find the total number of distributors.

Generated Final SQL:
SELECT COUNT(DISTINCT "Distribution") AS unique_distributors FROM T;

Final Answer: 35 ✗

Full Table

Generated Reasoning SQL:
To find the total number of unique distributors, we need to consider the "Distribution" column, which contains categorical values representing different Linux distributions. We should count the distinct values in this column to determine the number of unique distributors in the table.

Generated Final SQL:
SELECT COUNT(DISTINCT "Distribution") AS unique_distributors FROM T;

Final Answer: 35 ✗

Figure 5: Example of the effect of varying the number of rows in the sub-table (2/2, 3/3, 4/4, 5/5, Full Table) on SQL reasoning. The 3/3 setting achieved the best performance by balancing general and special rows, while other configurations introduced either insufficient information or excessive noise.

**1993 German motorcycle Grand Prix**

| Pos | Rider | Manufactuer | Time/Retired | Points |
|---|---|---|---|---|
| 1 | Doriano Romboni | Honda | 33:53.776 | 25 |
| 2 | Loris Capirossi | Honda | +0.090 | 20 |
| 3 | Helmut Bradl | Honda | +0.384 | 16 |
| ... | ... | ... | ... | ... |
| Ret | Wilco Zeelenberg | Aprilia | Retirement | ... |
| Ret | Jean-Pierre Jeandat | Aprilia | Retirement | nan |
| DNS | Nobuatsu Aoki | Honda | Did not start | nan |

Question: how many of motorcycles in the top 10 placers were manufactured by honda?

2×2 Sub-Table

Generated Reasoning SQL:
To determine the number of motorcycles in the top 10 placers that were manufactured by Honda, we need to consider the "Pos" column to identify the top 10 positions and the "Manufacturer" column to check the manufacturer of the motorcycles. We should exclude atypical information marked as "Ret" in the "Pos" column as they represent retirements and not actual positions in the race.

Generated Final SQL:
SELECT COUNT("Manufactuer") FROM T WHERE "Pos" NOT LIKE 'Ret' AND "Pos" <= 10 AND "Manufactuer" = 'Honda';

Final Answer: 2 ✗

3×3 Sub-Table **(Ours)**

Generated Reasoning SQL:
To answer this question, we need to consider the rows corresponding to the top 10 placers based on the "Pos" column. We will filter out rows where "Pos" is a Character value from 1 to 10 and then count the occurrences where the "Manufacturer" is Honda.

Generated Final SQL:
SELECT COUNT("Manufactuer") FROM T WHERE "Pos" IN ('1', '2', '3', '4', '5', '6', '7', '8', '9', '10') AND "Manufactuer" = 'Honda';

Final Answer: 6 ✓

4×4 Sub-Table

Generated Reasoning SQL:
To determine the number of motorcycles in the top 10 placers that were manufactured by Honda, we need to consider the "Pos" column to identify the top 10 positions and the "Manufacturer" column to check the manufacturer. We should exclude atypical rows that do not represent actual placers in the race.

Generated Final SQL:
SELECT COUNT("row_number") FROM T WHERE "Pos" <= 10 AND "Manufacturer" = 'Honda' AND "Pos" NOT IN ('Ret', 'DNS') AND "Manufacturer" NOT IN ('Suzuki');

Final Answer: 0 ✗

5×5 Sub-Table

Generated Reasoning SQL:
To determine the number of motorcycles in the top 10 placers that were manufactured by Honda, we need to consider the "Pos" column to identify the top 10 positions and the "Manufacturer" column to check if the manufacturer is Honda. We should exclude atypical rows such as retirements or rows with significantly irregular time values, as they do not represent typical race outcomes.

Generated Final SQL:
SELECT COUNT("row_number") FROM T WHERE "Pos" <= 10 AND "Manufacturer" = 'Honda' AND "Time/Retired" NOT LIKE '%Retirement%' AND "Time/Retired" NOT LIKE '%:%';

Final Answer: 0 ✗

Full Table

Generated Reasoning SQL:
To determine the number of motorcycles in the top 10 placers that were manufactured by Honda, we need to consider the "Manufacturer" column to identify Honda motorcycles and the "Pos" column to filter the top 10 positions. We should count the rows where the manufacturer is Honda and the position is within the top 10.

Generated Final SQL:
SELECT COUNT("Manufacturer") FROM T WHERE "Manufacturer" = 'Honda' AND "Pos" <= 10;

Final Answer: 0 ✗

Figure 6: Example of the effect of varying the number of rows in the sub-table on SQL reasoning. The 3/3 setting achieved the best performance by balancing general and special rows, while other configurations introduced either insufficient information or excessive noise.

```
=== Sub Table Preview ===
| row_number | Description Losses | 1939/40 | 1940/41 | 1941/42 | 1942/43 | 1943/44 | 1944/45 | Total |
|-------------:|:-------------------------|----------:|----------:|----------:|----------:|----------:|----------:|--------:|
| 0 | Direct War Losses | 360000 | nan | nan | nan | nan | 183000 | 543000 |
| 1 | Murdered | 75000 | 100000 | 116000 | 133000 | 82000 | nan | 506000 |
| 2 | Deaths In Prisons & Camps | 69000 | 210000 | 220000 | 266000 | 381000 | nan | 1146000 |
| 4 | Murdered in Eastern Regions | nan | nan | nan | nan | nan | 100000 | 100000 |
| 7 | Deaths other countries | nan | nan | nan | nan | nan | nan | 2000 |
| 9 | Total | 504000 | 352000 | 407000 | 541000 | 681000 | 270000 | 2770000 |
```

Sub Table

```
=== TabSpec (Table Specification) ===
**Column Analysis:**

1. **Description Losses:**
  - Data type: Categorical
  - 4 representative example values: Deaths In Prisons & Camps, Deaths Outside of Prisons & Camps, Murdered in Eastern Regions, Deaths other countries
  - Role of the column: specifies the type or category

2. **1939/40**
  - Data type: Numeric
  - Representative example values: 360000, 75000, 69000, nan
  - Role: Represents the number of deaths recorded in the year 1939/40.

8. **Total:**
  - Data type: Numeric
  - Representative example values: 543000, 506000, 1146000, 473000
  - Role: Provides the cumulative total of deaths across all recorded years for each loss category.

**Row Analysis:**
1. **Row 7 (Murdered in Eastern Regions):**
  - The value for 1944/45 is 100,000, which is significantly higher compared to the other years. This makes the row an outlier as it deviates from the pattern of missing data in the other years.
```

Generated TabSpec

```
"row_analysis_descrimination": {
   "result": "no",
   "reasoning": "Row 9 (Total) should be identified as special row because it aggregates all other rows and follows a distinct structural role." },
"accuracy_of_classification": {
   "result": "no",
   "reasoning": "Row 7 (Murdered in Eastern Regions) was labeled as special row due to its single non-missing value, but this pattern simply reflects incomplete data rather than an abnormal or structurally distinct case." },
"overall_assessment": "Row 9 (Total) should have been flagged as the special row summarizing all other rows, whereas Row 7 was wrongly classified despite representing missing data rather than structural irregularity.",
"all_criteria_passed": false,
"missing_row_analysis": ["Row 9"]
```

TabSpec Evaluation

```
"Row Analysis FeedBack": "The analysis wrongly flagged Row 7 as special row due to missing values, which are normal. The true special row is Row 9, the 'Total' row that serves as an aggregate summary. The model should focus on detecting structural roles like summary or total rows."
"Header FeedBack": "The analysis treated all yearly columns and the 'Total' column as a single combined header instead of recognizing them as separate measure columns. Each year (1939/40 to 1944/45) and the 'Total' should be extracted as distinct headers representing individual numeric attributes.
```

TabSpec Feedback

```
=== TabSpec (Table Specification) ===
**Column Analysis:**

1. **Description Losses:** ...

**Row Analysis:**
1. **Row 9 (Total):**
  - This row is an aggregate summary representing cumulative totals across all categories, making it an atypical summary row rather than a regular data entry.
```

TabSpec Refinement

Figure 7: An example of TabSpec generation and refinement. The initial TabSpec incorrectly identifies Row 7 as a special row due to missing values, while failing to detect Row 9 (Total) as an aggregate summary row. The TabSpec Evaluator detects this misclassification, and the subsequent refinement corrects the row analysis by properly identifying Row 9 (Total) as a special row.

```
=== Sub Table Preview ===
| row_number | Name | City | Hospital beds | Operating rooms | Total | Trauma designation | Affiliation | Notes |
|-------:|:----------------------------------------|:-----------------|-----------:|---------:|-------:|:------------|:-------|:----------------------------------------|
| 8 | Asheville Specialty Hospital | Asheville | 34 | 0 | 34 | - | - | - |
| 25 | Carolinas Medical Center | Charlotte | 874 | 51 | 925 | Level I | CHS | Secondary campus of the University of North Carolina School of Medicine |
| 28 | Carolinas Specialty Hospital | Charlotte | 40 | 0 | 40 | - | - | - |
| 31 | Central Carolina Hospital | Sanford | 137 | 8 | 145 | - | Tenet | - |
| 83 | North Carolina Specialty Hospital | Durham | 18 | 4 | 22 | - | - | - |
| 117 | Vidant Medical Center | Greenville | 861 | 35 | 896 | Level I | Vidant | Primary teaching hospital of East Carolina University Brody School of Medicine |
```

Sub Table

```
=== TabSpec (Table Specification) ===
**Column Analysis:**

1. **Name**
  - Data type: Categorical
  - Representative values: Carolinas Medical Center, Carolinas Specialty Hospital, Central Carolina Hospital, North Carolina Specialty Hospital
  - Role: Identifies each hospital by its official institutional name

2. **City**
  - Data type: Categorical
  - Representative values: Charlotte, Sanford, Durham, Greenville
  - Role: Specifies the city in which the hospital operates.
...

8. **Notes**
  - Data type: Categorical (Textual)
  - Representative values: Secondary campus of the University of North Carolina School of Medicine, -, -, Primary teaching hospital of East Carolina University Brody School of Medicine
  - Role: Provides supplementary institutional information such as academic or operational details.

**Row Analysis:**
No special row
```

Generated TabSpec

```
"row_analysis_descrimination": {
   "result": "yes",
   "reasoning": "The generated analysis correctly determined that there are no valid atypical rows beyond the provided candidates." },
"accuracy_of_classification": {
   "result": "yes",
   "reasoning": "The classification aligns with the given criteria: no abnormal patterns, formatting irregularities, or aggregate summaries are present." },
"overall_assessment": "All evaluation criteria satisfied — the generated Row Analysis is complete, accurate, and correctly concludes that no atypical rows exist.",
"all_criteria_passed": yes,
"missing_row_analysis": [ ]
```

TabSpec Evaluation

Figure 8: An example of TabSpec generation that passes evaluation. The TabSpec correctly identifies the special rows and column roles, and the TabSpec Evaluator confirms that all criteria are satisfied without requiring further refinement.

=== Question ===
what is the total number of deaths outside of prisons and camps?

Reasoning:
The summary indicates that the "Description Losses" column contains categorical values, ...
To calculate the total number of deaths outside of prisons and camps, we need to sum the values in the respective columns where the "Description Losses" is equal to "Deaths Outside of Prisons & Camps".

Generated Final SQL:
select sum(cast("1939/40" as integer) + cast("1940/41" as integer) + cast("1941/42" as integer) + cast("1942/43" as integer) + cast("1943/44" as integer) + cast("1944/45" as integer)) as total_deaths from T;

**Generated SQL**

"Question Alignment": {
"result": "no",
"reasoning": "The question asks for the total number of deaths outside of prisons and camps. Since a 'Total' column already exists, the query should have directly used it instead of manually summing all yearly columns. The omission of the 'Total' column leads to redundancy and misalignment with the intended role of columns." },

"TabSpec Alignment": {
"result": "no",
"reasoning": "The reasoning step states that the query should filter rows where 'Description Losses' = 'Deaths Outside of Prisons & Camps', but the actual SQL omits this WHERE clause. This makes the SQL inconsistent with its own reasoning and the intended semantics." },

"overall_assessment": "The SQL fails to align with the question and reasoning. It ignores the existing 'Total' column, lacks the required filtering condition, and diverges from the described reasoning process. While data type handling and 'Row Analysis' usage are correct, the overall query is not faithful or role-appropriate.",
"all_criteria_passed": false,
"issues": [ "Missing WHERE condition", "'Total' column not utilized", "Reasoning-SQL inconsistency" ] }

**SQL Evaluation**

"Question Alignment Feedback": "Use the 'Total' column to answer total-based questions instead of summing individual year columns. Focus on columns whose roles directly express the quantity asked in the question.",

"TabSpec Alignment Feedback": "Make sure to include a WHERE clause filtering rows where 'Description Losses' = 'Deaths Outside of Prisons & Camps', as defined in the schema. Use exact column names from the SS to maintain alignment with the schema."

**SQL Feedback**

Reasoning:
The summary indicates that "Description Losses" is a categorical column describing different types of deaths, and "Total" is a numeric column already representing the aggregated value across years.
To find the total number of deaths outside of prisons and camps, we simply need to select the value in the "Total" column for the row where "Description Losses" equals 'Deaths Outside of Prisons & Camps'.

SQL:
select "Total" as total_deaths FROM T WHERE "Description Losses" = 'Deaths Outside of Prisons & Camps';

**SQL Refinement**

Figure 9: An example of SQL generation and refinement. The initial SQL fails to return the correct total as intended by the question, and is flagged by both Question Alignment and TabSpec Alignment evaluations for inconsistency with the TabSpec. The subsequent refinement corrects the query to properly address the question's intent and align with the TabSpec.

=== Question ===
how many rooms total are at cone health?

Reasoning:
To determine the total number of rooms at Cone Health, we need to sum the values in the "Operating rooms" column where the "Affiliation" is equal to 'Cone'. We will consider all rows with the affiliation 'Cone' to calculate the total number of rooms at Cone Health.

Generated Final SQL:
SELECT SUM("Operating rooms") AS total_rooms_at_cone_health FROM T WHERE "Affiliation" = 'Cone';

**Generated SQL**

"Question Alignment": {
"result": "no",
"reasoning": "The question refers to the total number of rooms at a specific facility named 'Cone Health'. The refined SQL correctly uses the 'Total' measure but filters by 'Affiliation = Cone', which aggregates data across all hospitals under the Cone network instead of isolating the Cone Health facility." },
"TabSpec Alignment": { "result": "yes", ... },
"overall_assessment": "The SQL misunderstands both the target metric and the identifying condition. To answer the question correctly, select Total where Name equals Cone Health (optionally summing in case of duplicates).
For example: SELECT SUM(Total) AS total_rooms_at_cone_health FROM T WHERE Name = 'Cone Health';",
"all_criteria_passed": false,
"issues": [ "Wrong metric: Operating rooms used instead of Total" ] }

**SQL Evaluation**

"Question Alignment Feedback": "When the question targets a specific hospital, use the 'Name' column (facility identifier) instead of the broader 'Affiliation' field. The measure should remain 'Total' to represent the number of rooms. Avoid network-wide filters unless explicitly asked for aggregate results."

**SQL Feedback**

Reasoning:
To account for potential naming variations (e.g., "Cone Health System," "Cone Hospital"), the query expands the filter to include the entire Cone network by affiliation. While this seems safer, it mistakenly sums multiple facilities instead of isolating the single "Cone Health" record — resulting in semantic overreach.

SQL:
select sum("Total") as total_rooms_at_cone_health from T where "Affiliation" = 'Cone';

**SQL Refinement**

Figure 10: An example of SQL generation and refinement where only Question Alignment fails. The initial SQL does not correctly address the question's intent, and is flagged solely by the Question Alignment evaluation. The subsequent refinement corrects the query to properly align with the question while TabSpec Alignment is already satisfied.

You are a strict table analysis assistant.
Here is a partial preview of a table in markdown format:

{table_markdown}

Additionally, the following ROW INDICES were identified as ATYPICAL ANALYSIS by statistical similarity analysis:
{json.dumps(special rows, ensure_ascii=False)}

**Important Rules:**
- Not all candidate row analysis are truly special. Some are just representative data rows.
- Do not treat empty or missing cells as distinctive values. They are simple missing data and must not be treated as distinctive special row.

**Your task:**
1. For every column, describe:
  - Data type (numeric, categorical, etc.)
  - 4 representative example values
  - Role of the column (identifier, descriptor, measure, etc.)

2. Row Analysis:
- For each candidate row, explicitly mention the exact values that make it distinctive (e.g., "National Cup = Semifinals", "Reg. Season = 5th?").
- If these values are clearly different from the majority of the table, explain why this makes the row an special row.
- Focus only on candidate rows; do not discuss rows outside the candidate list.

Figure 11: Prompt for TabSpec Generation.

You are an expert at extracting table schemas from TabSpec (Table Specification) and titles.

***TASK***:
Given a table title and TabSpec, extract the table name and column headers to create the table structure.

***INSTRUCTIONS***:
1. Use the table title as the table name
2. Extract column names from "### Column Analysis" section
3. Maintain exact column names and order as specified

***REASONING STEPS***:

**Step 1 - Extract Table Name**:
Use the provided table title as the table name.

**Step 2 - Extract Column Headers**:
From "### Column Analysis:" section, identify all column names:
- Look for patterns like "1. **Column Name:**"
- Extract exact column names from bold headers
- Maintain order as listed in the TabSpec

**Step 3 - Determine Row Structure**:
Based on the column information and examples, determine:
- What entities will be the rows (e.g., states, players, teams)
- ALWAYS set estimated_rows to exactly 6
- Primary identifier for each row

***OUTPUT FORMAT***:
```
{
  "table_name": "<Table Title>",
  "column_headers": [
    "<Column 1>",
    "<Column 2>",
    "<Column 3>",
    ...
  ],
  "estimated_rows": 6
}
```

Figure 12: Table Generator prompt for constructing the structural schema used in table regeneration.

```
You are an expert at converting TabSpec (Table Specification) into detailed table generation instructions.

***TASK***:
Given a TabSpec, convert descriptive information into specific instructions for table data generation.

***INSTRUCTIONS***:
1. Extract data patterns from "### Column Analysis" section
2. Convert "### Atypical Analysis" into special row instructions
3. Create realistic data generation rules based on representative values

***REASONING STEPS***:

**Step 1 - Analysis Column Information**:
- Extract representative values and data types for each column
- Create data generation patterns based on "### Column Analysis:" section

**Step 2 - Extract Unique Features**:
- Identify distinctive rows from "### Atypical Analysis"
- Convert into special handling instructions

**Step 3 - Define Generation Rules**:
- Create column relationship rules based on roles
- Establish formatting guidelines from representative values
- Extract concrete examples for realistic data generation

***OUTPUT FORMAT***:
{
  "data_generation_instructions": [
    "Generate [column] following pattern [examples]",
    "Generate [column] with [data type] values like [examples]"
  ],
  "row_analysis_instructions": [
    "Include [special characteristic] for distinctive rows"
  ],
  "column_relationship_rules": [
    "[Column A] should relate to [Column B] based on [role]"
  ],
  "data_patterns": [
   "[Key patterns observed]"
 ]
}
```

Figure 13: Table Generator prompt for organizing cell values to reflect TabSpec information into the reconstructed table.

```
You are an expert at generating realistic tabular data. Your task is to create a complete table using the
provided schema and instructions.

***TASK***:
**Given**:
*Schema*: JSON object with table name, column headers, and row information
*Instructions*: Detailed instructions for data generation, relationships, and special requirements

**Your goal**:
Generate a complete table by systematically filling each cell according to the provided schema and
instructions.

***REASONING STEPS***:
Follow the algorithm step by step:

**ALGORITHM**
Step 1: Initialize Table Structure
  Create empty table with column headers from schema
  Create 6 rows based on schema

Step 2: Generate Data Following Instructions
  For each data_generation_instruction:
    Generate realistic values using representative values from Column Analysis
    Apply column_relationship_rules for consistency

Step 3: Complete Table and Apply Special Handling
  Fill any remaining empty cells with consistent data
  Apply row_analysis_instructions for distinctive rows
  Ensure all relationship rules are satisfied

**DETAILED PROCESS**:

**Step 1 - Initialize**:
- Create table with column headers from schema
- Set up 6 empty rows

**Step 2 - Generate Data & Fill Each Cell**:
- Follow data_generation_instructions for each column and fill each cell
- Apply column_relationship_rules

**Step 3 - Finalize**:
- Complete any empty cells
- Handle row analysis from row_analysis_instructions
- Verify consistency

***REQUIREMENTS***:
1. Generate EXACTLY 6 data rows using representative values from Column Analysis
2. Follow column_relationship_rules for data consistency
3. Apply row_analysis_instructions for distinctive rows

***FINAL OUTPUT***:
Generate EXACTLY 6 data rows following the schema and instructions.
Use "|" to separate cells and maintain proper markdown table formatting.

Process the table generation step by step, then provide the final table in this format:

### <TABLE NAME>
| Column 1 | Column 2 | Column 3 | ... |
|----------|----------|----------|-----|
| Row 1 Data | Row 1 Data | Row 1 Data | ... |
| Row 2 Data | Row 2 Data | Row 2 Data | ... |
| ... (exactly 6 rows) |
```

Figure 14: Table Generator prompt for reconstructing the table based on the TabSpec information.

```
You are an expert at evaluating Row Analysis in TabSpec (Table Specification). Your task is to assess the
quality and completeness of Row Analysis identification.

***INPUT***:
**Table:** {evidence_sub_table}
**Candidate of Row Analysis:** {json.dumps(row_analysis, ensure_ascii=False)}
**Generated Row Analysis:** {row_analysis_analysis}

***TASK***:
Evaluate the generated Row Analysis against the given table data and Candidate of Row Analysis to
determine if all Row Analysis have been properly identified.

***Criteria for Row Analysis***
- Aggregate rows (totals, summaries, averages) must be flagged as row analysis
- Capture rows with abnormal patterns, values, or formats

***EVALUATION CRITERIA***:
**1. Row Analysis Discrimination**
- Check whether any rows that should be labeled as special row are not labeled, from the candidate list.
- Confirm that the evaluation follows the Criteria for Row Analysis.
- Row Analysis determination must follow consistent principles:
  - Aggregate rows (totals, summaries, averages) should be flagged as special row when they exist.
  - Rows that deviate strongly from the overall distribution, or that use a different format/notation compared
to the majority, should also be considered special row (e.g., 1200 vs. 1.2k, 05:23:10 vs. 7h 15m, £12.00 vs. 12
USD, 2024-09-01 vs. 09/01/2024).
  - Missing values (nan) are common and should not automatically be treated as special row; only when rare
may they be considered anomalies.
  - A row should not be flagged solely because of its magnitude if its role or semantics explain the
difference.
- Ensure that ordinary rows are not incorrectly flagged as special row (avoid false positives).

**2. Accuracy of Classification**
- Evaluate wheter the row labeled as Row Analysis are indeed special row
- Accurately assess whether ambiguous markings ("?", "5th?", nan) are common or specific based on
frequency of occurrence
- Confirm that the evaluation follows the Criteria for Row Analysis.

***OUTPUT FORMAT***:
{{
  "row_analysis_discrimination": {{
    "result": "yes" | "no",
    "reasoning": "<Detailed explanation>",
    "issues": ["<Missing Row Analysis 1>", "<Missing Row Analysis 2>"]
  }},
  "accuracy_of_classification": {{
    "result": "yes" | "no",
    "reasoning": "<Detailed explanation>",
    "issues": ["<Misclassification 1>", "<Misclassification 2>"]
  }},
  "overall_assessment": "<Summary>",
  "all_criteria_passed": true | false,
  "missing_row_analysis": ["<Row analysis 1>", "<Row analysis 2>", "<None>"],
}}
```

Figure 15: TabSpec Evaluator prompt for Row Analysis evaluation.

```
You are an expert data analyst providing constructive feedback to improve table analysis. Your task is to
provide specific, actionable feedback.

***TASK***:
Based on the evaluation results, provide detailed feedback to help improve the analysis.

***INPUT***:
**Sub Table:** {sub_tab}
**Generated Table:** {generated_table}
**EM Score:** {em_score}

***Guidelines:***
- Compare the original sub table headers with generated table headers
- Identify specific header discrepancies (missing, extra, or incorrectly named headers)
- Analyze how different the sub tables header, and generated tables header is.
- Provide feedback for proper header extraction from table, based on the previous failed case.
- Focus on what the model should pay attention to for improvement
- Consider both structural and semantic aspects of tables
- Remember that not all tables have spsecial row - uniform data is valid

**Output Format:**
{{
  "Header FeedBack": "<Detailed feedback on header extraction and matching issues and what the model
should focus on>"
}}

Provide feedback in JSON format using the appropriate output format based on the feedback type needed.
```

Figure 16: TabSpec Feedback prompt for header evaluation, providing feedback when the header EM score does not meet the threshold.

```
You are an expert data analyst providing constructive feedback to improve table analysis. Your task is to
provide specific, actionable feedback.

***TASK***:
Based on the evaluation results, provide detailed feedback to help improve the analysis.

***INPUT***:
**Sub Table:** {sub_table}
**Generated Table:** {generated_table}
**BERT Score:** {bert_score}

***Guidelines:***
- Review the previous attempt's BERT Score and analyze how it differs from the expected performance.
- Compare the semantic meaning of original sub table content with generated table content
- Identify specific content discrepancies (missing data, incorrect values, format mismatches)
- Explain why certain content was misinterpreted or incorrectly generated
- Provide feedback for proper content extraction and representation, based on the previous failed case.
- Focus on what the model should pay attention to for improvement
- Consider both structural and semantic aspects of tables
- Remember that not all tables have special row - uniform data is valid

**Output Format:**
{{
  "Structure FeedBack": "<Detailed feedback on content similarity and semantic understanding issues and
what the model should focus on>"
}}

Provide feedback in JSON format using the appropriate output format based on the feedback type needed.
```

Figure 17: TabSpec Feedback prompt for structure evaluation, providing feedback when the cell similarity score does not meet the threshold.

```
You are an expert data analyst providing constructive feedback to improve table analysis. Your task is to
provide specific, actionable feedback.

***TASK***:
Based on the evaluation results, provide detailed feedback to help improve the analysis.

***INPUT***:
**Sub Table:** {sub_table}
**Generated Row Analysis:** {row_analysis}
**Evaluation Reasoning:** {evaluation_reasoning}

***Guidelines:***
- Analyze the evaluation reasoning to understand why the row analysis discrimination failed
- Point out rows incorrectly classified as row analysis when they are normal data points
- Identify rows that should have been flagged as row analysis but were missed
- Provide feedback on proper row analysis classification criteria, based on the previous failed case.
- Focus on what the model should pay attention to for improvement
- Consider both structural and semantic aspects of tables
- Remember that not all tables have special row - uniform data is valid

**Output Format:**
{{
  "Row Analysis FeedBack": "<Detailed feedback on row analysis discrimination issues and what the model
should focus on>"
}}

Provide feedback in JSON format using the appropriate output format based on the feedback type needed.
```

Figure 18: TabSpec Feedback prompt for Row Analysis evaluation, providing feedback when the Row Analysis criteria are not satisfied.

```
p_sql_wtq_with_tabspec = """
You will receive the **full table, the question, and the TabSpec**.
Generate SQL with a detailed explanation and the final query.
Exclude irrelevant rows (e.g., "Total", "Average", "World", "N/A", "?", "—") if the TabSpec marks them as such.

**Important Rule:**
- When using COUNT, you must **never use DISTINCT**.
- Whenever a calculation (e.g., SUM, AVG, MAX, MIN, arithmetic comparisons) is required, always CAST the
column values to INTEGER or FLOAT in SQL.
  Example: `SUM(CAST("Total Wins" AS INTEGER))`, `SUM(CAST("Total Wins" AS FLOAT))`,
- Never use `HAVING` for conditions on zero aggregrates
- All missing values in the table are stored as NULL
- When filtering categorical values that might appear as part of a longer string
  (e.g., "No playoff" vs "Champion (no playoff)"), always use a `LIKE '%value%'` condition instead of `=`.

Always **read the TabSpec first** and start with an explanation line beginning with 'Explanation:'.
- If the TabSpec labels a row as "ordinary", you must not exclude it.
- If the TabSpec marks a row as an special row and it is relevant to the question, explicitly exclude or handle
it in your SQL.
- In the explanation, explicitly mention which columns and values from the TabSpec are relevant to answer
the question.
- You must use the provided TabSpec carefully: if the TabSpec specifies data types (e.g., Numeric,
Categorical) or shows example values, use this information to decide how to construct conditions or
comparisons in SQL.
- Outlier notes in the TabSpec must also be considered: explain whether they should be excluded from the
SQL.
- Never ignore the TabSpec: every reasoning step must be grounded in it.

**If the TabSpec does not mention a column or entity explicitly, but the question specifies a clear condition
(e.g., a player name, a competition, a year), you must use that condition directly in the SQL filter based on
the most relevant column from the full table preview.**

After the explanation, on the next line, output the SQL query prefixed with 'SQL:'.
- Always wrap column names in double quotes (e.g., SELECT "column_name" FROM T).
- The SQL must directly follow from the reasoning in the Explanation section.

If the SQL alone cannot fully answer the question, state in the Explanation how the TabSpec information
should be used alongside SQL execution to derive the final answer.

Your final output must always include **both** the Explanation and the SQL, in that order.
```

Figure 19: Prompt for SQL Reasoning.

```
You are a strict SQL query evaluation expert. Your task is to evaluate the quality of a generated SQL query
based on specific criteria.

***INPUT***:
**Question:** {question}
**Table:** {table}
**Generated SQL:**: {generated_sql}
**TabSpec (Table Specification):** {tabspec}

***TASK***:
Evaluate the generated SQL query against the given question and TabSpec (Table Specification, Guideline
for table reasoning) based on the following criteria:

***EVALUATION CRITERIA***:
**Question Alignment**
1. Appropriate Role
- Query must return exactly the results intended by the question
- SQL query must select columns whose Roles, as defined in the TabSpec, are sufficient and appropriate for
deriving the result intended by the question.
- Logic flow and operations must align with the question's intent and the Roles of columns as defined in the
TabSpec.

2. Well Used Row Analysis
- Using a Row Analysis Information is not a essential, it should be used only if it is needed
- Row analysis should be used if they provide a more direct or accurate way to express the intended logic,
and omitted if they are irrelevant or would complicate the query unnecessarily.
- Consider whether aggregate or summary rows could provide shortcuts to complex calculations.
- Rows labeled as representative row should not be considered as row analysis and must not be flagged.
- If no row analysis are present, the result must explicitly be "yes".

**TabSpec Alignment**
1. Appropriate Data Type
- SQL query must use columns, values, and conditions consistently with their definitions in the TabSpec
- SQL syntax must be used correctly according to the data type
- Ensure proper handling of data types in comparisons, aggregations, and operations
- Example: categorical attributes should not be treated as numeric

2. Faithfulness
- SQL query must not introduce information that are not in the TabSpec
```

Figure 20: Prompt for SQL Evaluation.

```
You are an expert SQL mentor providing constructive feedback to improve SQL query generation. Your task is to analyze evaluation results and provide specific, actionable feedback for failed criteria.

***TASK***:
Based on the evaluation results of a generated SQL query, provide detailed feedback to help improve the query. Your feedback should be short, supportive, and instructional, focusing on how to act next time, not on what went wrong.

***INPUT***:
**Question:** {question}
**TabSpec (Table Specification):** {tabspec}
**Generated SQL:** {generated_sql}
**Evaluation Result:** {evaluation_result}

***FEEDBACK GUIDELINES***:
- For column selection:
  - Always ensure the selected columns directly represent what the question asks.
  - Focus on the column Roles that carry the information needed to answer the question.
  - Use only columns that contribute meaningfully to the query's goal.
- For row analysis:
  - Include special rows only when they simplify or clarify the query.
  - Review whether any aggregate/summary rows can be used logically.
  - Do not flag representative rows as special rows.

***OUTPUT FORMAT***:
{{
  "Appropriate Role Usage": "<Specific instructions on which columns to use/avoid based on their semantic roles>",
  "Proper Use of Atypical Analysis": "<Specific instructions on how to properly include/exclude/leverage row analysis>"
}}

***GUIDELINES***:
- Be specific and actionable in your feedback
- Focus on the failed criteria while considering the overall query quality
- Provide concrete SQL improvements, not just general advice
- Be constructive and educational in tone

Provide feedback in JSON format only.
```

Figure 21: SQL Refinement prompt for Question Alignment.

```
You are an expert SQL mentor providing constructive feedback to improve SQL query generation. Your task is to analyze evaluation results and provide specific, actionable feedback for failed criteria.

***TASK***:
Based on the evaluation results of a generated SQL query, provide detailed feedback to help improve the query. Your feedback should be short, supportive, and instructional, focusing on how to act next time, not on what went wrong.

***INPUT***:
**Question:** {question}
**TabSpec (Table Specification):** {tabspec}
**Generated SQL:** {generated_sql}
**Evaluation Result:** {evaluation_result}

***FEEDBACK GUIDELINES***:
- For data type handling:
  - Always match operations to the correct data type of each column.
  - Avoid treating categorical values as numeric or vice versa.
- For schema faithfulness:
  - Use only column names explicitly defined in the TabSpec.
  - Stick strictly to the provided schema when forming SQL logic.

***OUTPUT FORMAT***:
{{
  "Data Type Usage": "<Specific instructions on how to properly handle different data types in the query>",
  "Exact Column Name Feedback": "<Specific instructions on which exact column names from the TabSpec should be used instead>"
}}

***GUIDELINES***:
- Be specific and actionable in your feedback
- Focus on the failed criteria while considering the overall query quality
- Provide concrete SQL improvements, not just general advice
- Be constructive and educational in tone

Provide feedback in JSON format only.
```

Figure 22: SQL Refinement prompt for TabSpec Alignment.