SMART: Semantic Header Flattening and Pseudo-Code-Style Reasoning for LLM-based Complex Table Question Answering

Anonymous ACL submission

Abstract

Complex table question answering (TQA) remains challenging, as real-world table, usually designed for human readability with multi-level headers and fragmented hierarchical semantics, largely hindering large language models (LLMs) from accurately aligning conditions, attributes, and values during reasoning. Existing approaches typically rely on handcrafted table linearization or prompts, forcing LLMs to infer header hierarchies, which frequently leads to brittle reasoning and hallucinations. To this end, we propose **SMART**, a unified framework that explicitly decouples table structure understanding from reasoning execution. SMART consists of three components: Semantic Header Flattening for converting multi-level headers into explicit single-level descriptors, Global Understanding for capturing holistic table–question semantics, and Pseudo-Code-Style Reasoning for structured, step-by-step inference with external validation. Extensive experiments on multiple benchmarks demonstrate that SMART substantially improves both the accuracy and robustness of complex TQA, achieving state-of-the-art performance. The code is available at https://anonymous.4open.science/r/SMART-85FA

1 Introduction

Table question answering (TQA) has increasingly become a core nature language processing (NLP) task (Zhao et al., 2023; Zheng et al., 2023; Zhang et al., 2024c), enabling various applications like data analytics, decision support, and information retrieval. Generally, real-world tables frequently feature multi-level headers with hierarchical layered semantics, posing significant challenges for TQA beyond flat-table reasoning (Wang et al., 2021; Cheng et al., 2022). These latent structures are difficult to interpret directly, and naive table-based reasoning often yields inconsistent or erroneous answers. Consequently, explicitly modeling hierarchical header semantics is crucial for enabling reliable reasoning in complex TQA.

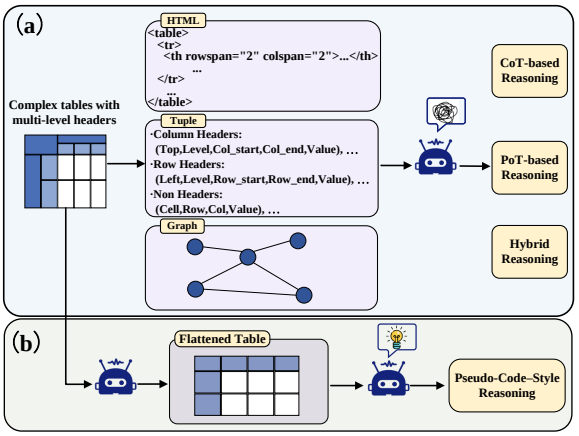

Figure 1: Comparison of complex table representation methods and TQA reasoning paradigms: (a) existing approaches; (b) SMART.

With the rapid advancement of large language models (LLMs) (Brown et al., 2020; Chung et al., 2024; Welsby and Cheung, 2023; Achiam et al., 2023), most studies in table question answering primarily focus on prompt strategies to guide LLMs toward reliable reasoning. Indeed, prior TQA works mostly process flat-tables, with limited work dedicated to complex tables containing multi-level headers. As illustrated in Figure 1(a), these approaches merely convert hierarchical header structures into tuples(Zhao et al., 2023), HTML(Zhang et al., 2024c) or graphs (Li et al., 2025) for representation, yet leave implicit semantics and residual hierarchy that impede reasoning.

Beyond representation, the reasoning paradigm introduces further complexities, posing additional challenges for TQA. For example, (1)Chain-of-Thought (CoT) methods (Wei et al., 2022; Zhao et al., 2023) yields nature-language intermediate steps for LLMs synergy, but lack structural constraints and verifiability, easily causing unstable or hallucinated reasoning; (2)Program-of-Thought

(PoT) methods (Chen et al., 2022; Zhang et al., 2024c) translate reasoning into programmatic form (e.g., Python or SQL), enabling deterministic computation and step-level validation. Yet this translation is fragile: minor (e.g., code/mapping) errors often cause total failure, even disadvantaging models if with limited code-generation ability; and (3)Hybrid CoT–PoT approaches (Yang et al., 2025; Liu et al., 2024) attempt to combine both paradigms, but rely on multi-path cross-modal reasoning, easily causing high substantial computational overhead and instability due to cross-mode inconsistencies.

Thereby, an effective and widely-adopted solution is to explicitly model hierarchical header semantics while enabling reliable and verifiable reasoning. To this end, we introduce SeMantic header flAttening and pseudo-code-style Reasoning for LLM-based complex Table question answering (**SMART**). SMART addresses challenges through three complementary steps. First, Semantic Header Flattening systematically transforms hierarchical headers into single-level natural language descriptors optimized for reasoning tasks. Next, Global Understanding generates a concise semantic summary of the table and question, providing high-level guidance for downstream reasoning. Finally, Pseudo-Code–Style Reasoning enables interpretable multi-step inference by composing a restricted set of atomic operations in pseudo-code format, with each step externally validated for intermediate computations and sorting, helping to mitigate numerical hallucinations. In summary, our contributions are as follows:

- We propose Semantic Header Flattening and Global Understanding to transform complex tables into a reasoning-friendly format and provide guidance for multi-step reasoning.
- We introduce a Pseudo-Code–Style Reasoning paradigm and incorporate external verification to mitigate numerical and sorting hallucinations.
- Extensive experiments on HiTab, AIT-QA, and WikiTableQA demonstrate that SMART consistently achieves state-of-the-art performance in complex table question answering.

2 Related work

2.1 Complex Table Representation

Representing complex tables with multi-level headers has been widely studied as a prerequisite for structured table understanding(Cheng et al., 2022; Katsis et al., 2021). TableParser (Zhao et al., 2023) represents header and non-header cells using five-tuples and four-tuples, respectively, incorporating hierarchical and positional information. E^5 (Zhang et al., 2024c) serializes complex tables into HTML and prompts the LLM to generate a textual explanation of the header hierarchy, providing structural cues for interpreting multi-level headers. GraphOTTER (Li et al., 2025) models table as an undirected cell graph with edges connecting cells in the same row or column, and serializes this graph into a prompt to expose structural relations without differentiating headers from data cells explicitly. Despite their effectiveness, these approaches still fail to fully expose the structural semantics inherent in multi-level headers, limiting their suitability for subsequent reasoning.

2.2 Reasoning for Table QA

Driven by recent advances in LLM-based reasoning techniques (Zhang et al., 2024a; Sahoo et al., 2024), current approaches to TQA predominantly follow several representative reasoning paradigms. Chain-of-Thought (CoT)(Wei et al., 2022) reasoning leverages LLMs to generate intermediate natural-language steps, providing strong interpretability and flexibility. For instance, TableParser (Zhao et al., 2023) performs step-wise reasoning over tuple-represented cells, while Table-Critic (Yu et al., 2025) employs a multi-agent framework to refine and correct step-by-step CoT reasoning for TQA. However, purely textual CoT lacks explicit structural grounding, making it prone to numerical hallucinations and insufficient for complex reasoning tasks. Program-of-Thought (PoT)(Chen et al., 2022) reasoning formulates the reasoning process as a sequence of executable programs (e.g., SQL or Python) that implement structured, deterministic multi-step operations over table data (Ye et al., 2023; Cheng et al., 2023; Zhang et al., 2024c,b; Mao et al., 2024). This approach ensures high precision and verifiable reasoning, but it depends on strict schema alignment and is fragile to errors in program generation. Hybrid reasoning combines CoT and PoT paradigms to balance flexibility and reliability. MIX-SC (Liu et al., 2024) fuses outputs from Python agents and textual chains via self-consistency, whereas TIDE (Yang et al., 2025) executes both CoT-based Direct-Prompting and PoT-based Agent modes before merging results. While hybrid reasoning improves accuracy, it incurs ad-

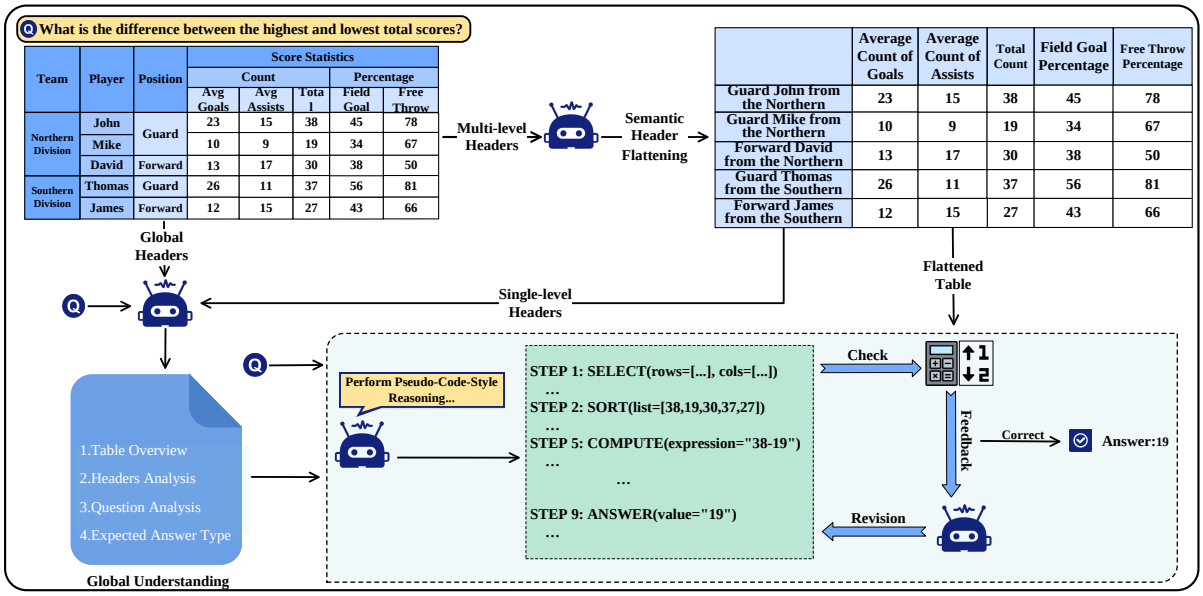

Figure 2: Overview of the proposed SMART framework. The framework flattens hierarchical table headers into single-level headers, constructs a global guidance text, and performs Pseudo-Code–Style Reasoning with external validation of intermediate computations and sorting.

ditional reasoning overhead. These observations suggest the need for a reasoning framework that strikes a balance between the flexibility of CoT and the reliability of PoT, enabling both interpretable and verifiable multi-step inference over complex tables.

3 Method

3.1 Overview

As mentioned in Figure 2, we introduce SMART, an efficient framework for TQA designed for complex tables. First, Semantic Header Flattening leverages the linguistic capacity of LLMs to convert hierarchical headers into a coherent layer of natural-language descriptors, producing a semantically normalized table with a single-level header. Based on the flattened headers and the question, the Global Understanding module generates a concise table question semantic summary to guide reasoning. Finally, Pseudo-Code–Style Reasoning performs structured multi-step inference, with a lightweight external calculator and sorter verifying computations and sorting to mitigate numerical hallucinations.

3.2 Task Description

The complex table question answering task involves answering a question Q over a table T using a model M, guided by a prompt P. The table may contain hierarchical headers, merged cells, and diverse data types, while the question may require multi-step reasoning, aggregation, or comparison. Formally, the task is defined as:

$$A = M(T, Q, P) \tag{1}$$

where A denotes the answer predicted by the model.

3.3 Semantic Header Flattening

The Semantic Header Flattening module converts each multi-level row or column header into a single natural-language descriptor. We first identify the global column headers, defined as any header that spans all columns (e.g., *Score Statistics* in Fig. 2). Each global header is transformed into a column-level description—"`The column is about score statistics.`"—which is incorporated as part of the table caption for downstream reasoning.

For the remaining multi-level column headers, we perform semantic flattening by prompting an LLM to map each hierarchical header path $\{h^{(1)}, h^{(2)}, \ldots, h^{(k)}\}$ into a single natural language descriptor:

$$h^{\text{flat}} = \text{LLM}(h^{(1)}, h^{(2)}, \ldots, h^{(k)}) \tag{2}$$

The prompts are explicitly designed to guide the model in interpreting the hierarchical relationships among the headers, thereby capturing their underlying structural semantics and converting them into explicit natural language descriptions. Row headers are processed analogously.

After semantic flattening, all hierarchical row and column headers are replaced with their corresponding single-level descriptors, resulting in a semantically normalized table representation, denoted as $\mathbf{T}^{\text{flat}}$. This flattened table serves as the input to the subsequent Pseudo-Code-Style Reasoning module, ensuring that multi-step inference is performed over explicitly represented header semantics on a table with simple, single-level headers. Detailed prompt templates are provided in Appendix A.1.

3.4 Global Understanding

Motivated by the human habit of first forming a holistic understanding of a table before answering, we design a Global Understanding module that generates a concise textual summary of high-level cues based on both the table and question. The summary contains four components:

(1) Table Overview. A brief summary of the table's thematic content derived from the caption and global headers.

(2) Header Analysis. A short interpretation of the column and row headers, including their semantic roles, measurement meanings, and inherent logical relations. Aggregated or summative headers are highlighted, as they may directly imply or simplify reasoning.

(3) Question Analysis. An interpretation of the question intent and its linkage to the table data.

(4) Expected Answer Type. A concise indication of the expected answer type to guide the formatting of the downstream reasoning output.

By providing high-level semantic cues, this description strengthens the LLM's understanding of the QA scenario and lowers the complexity of the following step-wise reasoning. Detailed prompt templates are provided in Appendix A.2.

3.5 Pseudo-Code-Style Reasoning

To ensure both interpretability and reliability in multi-step table reasoning, we design a Pseudo-Code–Style Reasoning paradigm that integrates the complementary strengths of natural language and programmatic reasoning. This paradigm allows the model to articulate intermediate steps in natural language, preserving flexibility and expressiveness, while structuring the inference process in a pseudo-code format that supports stepwise verification. Formally, given a semantically flattened table $\mathbf{T}^{\text{flat}}$ and a question Q, a single reasoning instance

Pseudo Function Template:

STEP <number>: <Function>(<parameter if any>)
description: <Natural language explanation>
formula: <List of formulas, [] if none>
output: <Result>

Figure 3: Template of pseudo operation functions for each step in the reasoning process.

is modeled as a sequence of pseudo-code–style operation functions:

$$\text{Reasoning} = f_n \circ \cdots \circ f_1(Q, \mathbf{T}^{\text{flat}}), \quad f_i \in \mathcal{F} \quad (3)$$

where $\mathcal{F} = \{\text{Select}, \text{Compute}, \text{Sort}, \text{Infer}, \text{Answer}\}$ denotes the set of allowed operations. The internal structure of each operation function is shown in Figure 3 and consists of three components: a textual description, a formula, and an output. This design preserves interpretability while imposing structural constraints to enable controlled and verifiable multi-step inference.

Leveraging this structured pseudo-code design, we address a key challenge in LLM-based reasoning: numerical hallucinations. Formally, a pseudo-code–style reasoning trace is represented as an ordered sequence $\mathcal{R} = \langle s_1, s_2, \ldots, s_N \rangle$, where each step s_i corresponds to an atomic operation selected from the predefined operation set. The LLM first generates a complete reasoning trace, in which the Compute and Sort operations produce arithmetic expressions and ordered lists in a standardized format. The Compute function supports basic arithmetic operations, including addition, subtraction, multiplication, and division, and also allows the *len* function for counting elements. The Sort function specifies the generation of numerically ordered lists according to the reasoning requirements. After a full reasoning trace is generated, the intermediate formulas and outputs from all Compute and Sort steps are extracted using pattern-matching procedures and subsequently verified by external validator. We formalize the verification process as follows: $V(\mathcal{R}) \rightarrow \{0, 1\}$, which returns 1 if and only if all Compute and Sort steps in $\mathcal{R}$ are consistent with the external validator. If $V(\mathcal{R}) = 0$, the validator identifies the earliest failing step:

$$s_j = \arg\min_i \mathbb{I}[s_i \text{ fails verification}] \quad (4)$$

Then validator generates a concise repair hint localized to that step (e.g., "`Step 2: [3,1,2] sorting`

Algorithm 1 Pseudo-Code–Style Reasoning

Input: Flattened table $\mathbf{T}^{\text{flat}}$, Question Q, Global Understanding G, LLM M, Max corrections K
Output: Answer A, Reasoning trace $\mathcal{R}$
$\mathcal{R} \leftarrow M.\text{reason}(\mathbf{T}^{\text{flat}}, Q, G)$ ▷ Generate initial reasoning trace
$attempts \leftarrow 0$
$(error, hint) \leftarrow V(\mathcal{R})$ ▷ Validate and locate first failing step
while $error \neq \text{None} \wedge attempts < K$ **do**
    $\mathcal{R} \leftarrow M.\text{revise}(\mathcal{R}, hint)$
    $(error, hint) \leftarrow V(\mathcal{R})$
    $attempts \leftarrow attempts + 1$
end while
$A \leftarrow \text{ExtractAnswer}(\mathcal{R})$
return $A, \mathcal{R}$

`error, correct ascending is [1,2,3]`"). This hint is then fed back to the LLM, which performs a local correction starting from s_j and regenerates all subsequent steps. The resulting check–revision loop iterates until $V(\mathcal{R}) = 1$ or a predefined maximum number of correction attempts is reached. Once validated, the final reasoning trace $\mathcal{R}$ and the corresponding answer A are returned.

The overall reasoning process is summarized in Algorithm 1, with additional prompt and implementation details provided in Appendix A.3.

4 Experiments

4.1 Setup

Benchmarks,Baselines,and Metrics We evaluate our framework on two complex table question answering benchmarks, HiTab(Cheng et al., 2022) and AIT-QA(Katsis et al., 2021), which feature heterogeneous layouts and multi-level table headers. To isolate and assess the effectiveness of the proposed Pseudo-Code–Style Reasoning paradigm, we additionally conduct experiments on WikiTableQA(Pasupat and Liang, 2015), a flattened table question answering benchmark that focuses on complex questions. Dataset statistics are provided in Table 1.

We compare our method against three categories of baselines. The first category consists of complex table question answering approaches(Zhao et al., 2023; Zhang et al., 2024c; Li et al., 2025), which are designed to reason directly over multi-level and hierarchical tables. These baselines evaluate the overall effectiveness of our framework. The second category includes different reasoning-paradigm-based methods, covering COT-Based, POT-Based, and Hybrid approaches(Chen, 2022; Zhao et al., 2023; Li et al., 2025; Zhang et al., 2024c; Wang et al., 2024; Liu et al., 2024). This allows us to assess the advantages of the Pseudo-Code–Style Reasoning paradigm over alternative reasoning approaches. The third category consists of methods(Cheng et al., 2023; Ye et al., 2023; Nahid and Rafiei, 2024; Wang et al., 2024; Mao et al., 2024) designed to tackle complex questions over flattened tables, enabling an evaluation of our Pseudo-Code–Style paradigm's ability to reason over complex queries.

Dataset	Tables	QA Pairs	Domain
HiTab	538	1,584	Open domain
AIT-QA	116	515	Airline industry
WikiTableQA	421	4,344	Wikipedia

Table 1: Statistics of benchmark datasets used for TQA.

Following prior work(Chen, 2022; Zhao et al., 2023; Zhang et al., 2024c; Li et al., 2025), we adopt **Exact Match (EM)** and an **LLM-based Evaluator (LLM Eval)** to assess prediction accuracy. Implementation details are provided in Appendix B.1.

Implementations Considering computational cost, we select two widely adopted LLM backbones for experiments, Qwen2[1] and LLaMA3.1[2]. To ensure fair and reproducible comparisons, we adopt consistent experimental settings, including a temperature of 0.1, a maximum of 3 Pseudo-Code–Style Reasoning iterations, and a maximum output token length of 4096 to accommodate complete multi-step Pseudo-Code-Style Reasoning. All other model parameters are kept at their default values. Baseline methods are implemented following their official repositories and the prompt designs described in the original papers. Detailed parameter settings and additional analysis are provided in Appendix B.2.

5 Experiment Results

5.1 Main Results

Table 2 compares the performance of Qwen2 and LLaMA3.1 backbones on HiTab and AIT-QA. SMART consistently outperforms all baselines

[1]Qwen2-72B-Instruct
[2]LLaMA3.1-70B-Instruct

Methods	Qwen2				LLaMA3.1			
	HiTab		AIT-QA		HiTab		AIT-QA	
	EM	LLM Eval	EM	LLM Eval	EM	LLM Eval	EM	LLM Eval
TableParser(Zhao et al., 2023)	44.57	69.76	59.26	78.75	52.46	64.20	74.07	75.44
E^5 (Zhang et al., 2024c)	43.56	47.16	62.96	64.91	65.53	67.23	79.53	81.29
GraphOTTER(Li et al., 2025)	73.74	77.37	87.37	87.57	75.18	77.71	86.38	86.77
Ours	**73.78**	**82.84**	**89.13**	**89.51**	**76.76**	**83.53**	**89.30**	**89.49**

Table 2: Overall evaluation on complex table QA. Our stands out as the most effective method for this task across two benchmark datasets and LLM backbones.

across both EM and LLM Eval metrics. On Qwen2, it achieves a 5.47% gain in LLM Eval on HiTab and 1.76%/1.94% improvements on AIT-QA for EM and LLM Eval, respectively. Results on LLaMA further demonstrate SMART's robustness, with gains of 1.58% (EM) and 5.82% (LLM Eval) on HiTab, and 2.92%/2.72% on AIT-QA, showing its effectiveness on complex tables.

A detailed analysis reveals a divergence between EM and LLM Eval metrics. On HiTab, the improvements in LLM Eval are more pronounced, reflecting that SMART's multi-step reasoning often yields semantically correct answers that do not exactly match the reference labels. This discrepancy arises naturally from the QA process, where intermediate reasoning steps, aggregations, or computations can lead to minor variations in the final output. In contrast, on AIT-QA, performance gains are more consistent across EM and LLM Eval, suggesting that SMART effectively produces both exact matches and semantically accurate answers within the structured context of airline tables.

As shown in Figure 4, we evaluate SMART on the four official subsets of the AIT-QA dataset: Table-driven, KPI-driven, Row header hierarchy, and No row header hierarchy. SMART consistently outperforms all baselines across these subsets. Performance on Table-driven and KPI-driven questions demonstrates its ability to handle diverse question types, while results on Row header hierarchy questions indicate that it effectively handles questions requiring hierarchical row header reasoning. Overall, these results confirm SMART's robustness across varying question types and reasoning requirements in tables.

Table 3 presents the performance of different reasoning paradigms on HiTab using the Qwen2 backbone. Compared with other reasoning paradigms, our Pseudo-Code-Style approach substantially outperforms these baselines. This improvement arises

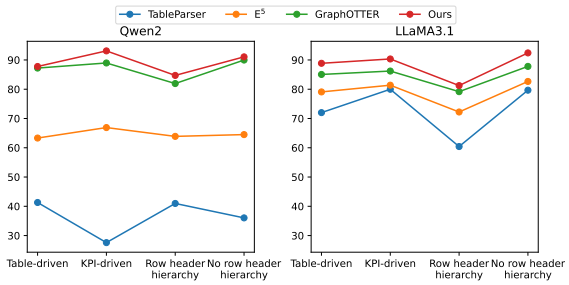

Figure 4: Comparison of EM performance across different subsets of the AIT-QA dataset.

from its structured reasoning paradigm: complex table reasoning is decomposed into explicit, step-by-step operations, and the reasoning actions and output format are strictly constrained. By guiding the model to follow a well-defined reasoning flow, this approach mitigates errors caused by ambiguous or uncoordinated inference steps, while preserving the large language model's strength in natural-language reasoning. Consequently, Pseudo-Code-Style Reasoning enables more accurate and semantically consistent answers, effectively addressing the challenges posed by complex questions.

Table 4 reports the performance of SMART on the WikiTableQA benchmark using the LLaMA3.1 backbone. Our method achieves an EM score of 71.69%, showing consistent improvements over representative mainstream methods. This result shows that Pseudo-Code–Style Reasoning is effective for complex multi-step table question answering. By structuring the reasoning process, it enables more accurate answers to complex queries.

Overall, these observations indicate that SMART provides a robust reasoning framework capable of handling complex tables and complex question answering scenarios.

5.2 Ablation Study

To assess the contributions of individual components, we conduct ablation studies on modules that

Reasoning Type	Methods	EM	LLM Eval
COT-Based	TableReasoner(2022)	56.41	75.68
	TableParser(2023)	44.57	69.76
	GraphOTTER(2025)	73.74	77.37
POT-Based	E^5(2024c)	43.56	47.16
Hybrid-Based	Chain-of-Table(2024)	44.26	62.69
	MIX-SC(2024)	73.42	77.08
Pseudo-Code-Style	Ours	**73.78**	**82.84**

Table 3: Performance comparison of different reasoning methods on HiTab using the QWen2.

Method	EM
Binder(Cheng et al., 2023)	50.51
Dater(Ye et al., 2023)	43.53
TabSQLify(Nahid and Rafiei, 2024)	55.78
Chain-of-Table(Wang et al., 2024)	62.22
PoTable(Mao et al., 2024)	65.56
Ours	**71.69**

Table 4: Performance comparison of SMART and mainstream TQA methods on the flattened-table dataset WikiTableQA.

Method	EM	LLM Eval
SMART	**73.78**	**82.84**
w/o Global Understanding	69.03	76.31
w/o Numerical Validation	72.97	82.07

Table 5: Ablation results on the HiTab dataset using Qwen2.

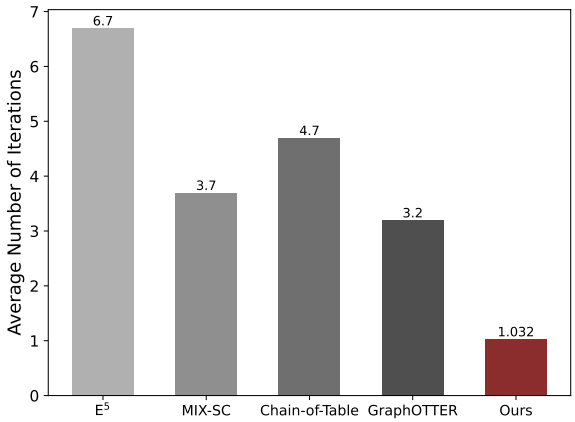

Figure 5: Average number of iterations for various mainstream iterative reasoning methods, evaluated using the Qwen2 model on the HiTab dataset.

directly affect the inference process, specifically focusing on the Global Understanding module and the Pseudo-Code–Style Reasoning scheme. The Semantic Header Flattening module, being essential for transforming complex tables into a format suitable for reasoning, is retained in all experiments and thus is not included in the ablation.

As shown in Table 5, the full SMART model achieves the best performance on both EM and LLM-based evaluation metrics, demonstrating the effectiveness of the complete system. Removing the Global Understanding module results in a substantial drop in performance, indicating that precomprehending high-level table cues and question context is crucial for accurate reasoning. Excluding numerical validation in the Pseudo-Code–Style Reasoning step also degrades both EM and LLM Eval scores, confirming that external verification of arithmetic and sorting operations significantly enhances reasoning reliability. These results highlight the complementary contributions of each component and validate the design choices of SMART.

5.3 Efficiency Analysis

We evaluate the reasoning efficiency of our method on the HiTab dataset by comparing it with several mainstream iterative reasoning approaches. In our framework, one iteration is defined as a complete reasoning cycle consisting of a Pseudo-Code–Style Reasoning, followed by formula extraction and verification, and a single correction reasoning if necessary. In our experiments, we empirically set the maximum number of iterations to 3; detailed analyses are provided in the Appendix B.2. For fair comparison, other methods follows the settings used in GraphOTTER(Li et al., 2025). As shown in Figure 5, our method achieves an average iteration count close to one, indicating substantially higher efficiency than existing iterative reasoning methods. This efficiency gain primarily stems from the design of our Pseudo-Code-Style Reasoning paradigm, which is able to complete most reasoning processes within a single iteration. Additional iterations are only triggered when numerical computation or sorting hallucinations are detected during verification, and such cases occur with relatively low frequency. As a result, our approach significantly reduces redundant reasoning iterations while maintaining reliable performance on complex TQA.

5.4 Case Study

To qualitatively analyze why our approach is better suited for TQA, we conduct a case study comparing our method with the strong baseline GraphOTTER.

Figure 6: Case Study. The first one is the reasoning process of the baseline method GraphOTTER, and the second one is our Pseudo-Code-Style Reasoning process

Based on the question and table shown in Figure 2, the reasoning processes of GraphOTTER and our method are illustrated in Figure 6.

As shown in Figure 6, GraphOTTER answers questions by constructing a task-specific subgraph that aggregates relevant information. This approach can omit critical evidence and struggles with multi-hop or compositional reasoning, where intermediate dependencies must be explicitly modeled, often leading to incomplete reasoning paths and unstable performance on complex TQA instances.

In contrast, our Pseudo-Code–Style Reasoning paradigm defines task-aligned reasoning actions executed in a structured and stepwise format, enabling systematic and interpretable multi-step inference. Each reasoning step is accompanied by a natural-language description, allowing LLMs to fully leverage their strengths in textual and logical reasoning. All generated computations and sorting expressions are then extracted and verified (red box), effectively mitigating errors from numerical, sorting, or intermediate reasoning hallucinations and enhancing the reliability and accuracy of the final answer. Overall, this case study illustrates that our method provides a more robust, TQA-oriented reasoning framework. Additional qualitative examples are provided in Appendix D.

6 Conclusion and Future work

We propose **SMART**, a framework for complex table question answering that integrates Semantic Header Flattening, Global Understanding, and Pseudo-Code–Style Reasoning. By converting hierarchical headers into explicit natural-language descriptors and structuring multi-step inference with external validation, SMART enhances both the reliability and effectiveness of TQA. Experiments across multiple datasets and metrics confirm its generalizability and compatibility with large language models, establishing SMART as a novel paradigm for complex table question answering.

Future work will explore structured reasoning paradigms for broader error detection and correction, aiming to mitigate numerical, logical, and multi-step inference errors while improving reliability and interpretability in complex table question answering.

Limitations

While SMART demonstrates strong performance on benchmark datasets, it has several limitations. First, the current verification mechanism mainly focuses on numerical and sorting errors, and does not directly address higher-level logical or textual reasoning mistakes. Second, SMART does not incorporate a dedicated mechanism for filtering potentially irrelevant table content. Extending the framework to handle these scenarios could further improve the reliability and generality of structured reasoning over complex tables.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and 1 others. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Wenhu Chen. 2022. Large language models are few (1)-shot table reasoners. *arXiv preprint arXiv:2210.06710*.

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*.

Zhoujun Cheng, Haoyu Dong, Zhiruo Wang, Ran Jia, Jiaqi Guo, Yan Gao, Shi Han, Jian-Guang Lou, and Dongmei Zhang. 2022. HiTab: A hierarchical table dataset for question answering and natural language generation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1094–1110, Dublin, Ireland. Association for Computational Linguistics.

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. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, and 1 others. 2024. Scaling instruction-finetuned language models. *Journal of Machine Learning Research*, 25(70):1–53.

Yannis Katsis, Saneem Chemmengath, Vishwajeet Kumar, Samarth Bharadwaj, Mustafa Canim, Michael Glass, Alfio Gliozzo, Feifei Pan, Jaydeep Sen, Karthik Sankaranarayanan, and 1 others. 2021. Ait-qa: Question answering dataset over complex tables in the airline industry. *arXiv preprint arXiv:2106.12944*.

Qianlong Li, Chen Huang, Shuai Li, Yuanxin Xiang, Deng Xiong, and Wenqiang Lei. 2025. Graphotter: Evolving llm-based graph reasoning for complex table question answering. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 5486–5506.

Tianyang Liu, Fei Wang, and Muhao Chen. 2024. Rethinking tabular data understanding with large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), NAACL 2024, Mexico City, Mexico, June 16-21, 2024*, pages 450–482. Association for Computational Linguistics.

Qingyang Mao, Qi Liu, Zhi Li, Mingyue Cheng, Zheng Zhang, and Rui Li. 2024. Potable: Programming standardly on table-based reasoning like a human analyst. *CoRR*, abs/2412.04272.

Md Mahadi Hasan Nahid and Davood Rafiei. 2024. TabSQLify: Enhancing reasoning capabilities of LLMs through table decomposition. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 5725–5737, Mexico City, Mexico. Association for Computational Linguistics.

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

Pranab Sahoo, Ayush Kumar Singh, Sriparna Saha, Vinija Jain, Samrat Mondal, and Aman Chadha. 2024. A systematic survey of prompt engineering in large language models: Techniques and applications. *arXiv preprint arXiv:2402.07927*.

Zhiruo Wang, Haoyu Dong, Ran Jia, Jia Li, Zhiyi Fu, Shi Han, and Dongmei Zhang. 2021. Tuta: Tree-based transformers for generally structured table pre-training. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 1780–1790.

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. 2024. Chain-of-table: Evolving tables in the reasoning chain for table understanding. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Philip Welsby and Bernard MY Cheung. 2023. Chatgpt.

Zhen Yang, Ziwei Du, Minghan Zhang, Wei Du, Jie Chen, Zhen Duan, and Shu Zhao. 2025. Triples as the key: Structuring makes decomposition and verification easier in llm-based tableqa. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.

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, SIGIR 2023, Taipei, Taiwan, July 23-27, 2023*, pages 174–184. ACM.

Peiying Yu, Guoxin Chen, and Jingjing Wang. 2025. Table-critic: A multi-agent framework for collaborative criticism and refinement in table reasoning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pages 17432–17451. Association for Computational Linguistics.

Yadong Zhang, Shaoguang Mao, Tao Ge, Xun Wang, Adrian de Wynter, Yan Xia, Wenshan Wu, Ting Song, Man Lan, and Furu Wei. 2024a. Llm as a mastermind: A survey of strategic reasoning with large language models. *arXiv preprint arXiv:2404.01230*.

Yunjia Zhang, Jordan Henkel, Avrilia Floratou, Joyce Cahoon, Shaleen Deep, and Jignesh M. Patel. 2024b. Reactable: Enhancing react for table question answering. *Proc. VLDB Endow.*, 17(8):1981–1994.

Zhehao Zhang, Yan Gao, and Jian-Guang Lou. 2024c. e5: Zero-shot hierarchical table analysis using augmented llms via explain, extract, execute, exhibit and extrapolate. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 1244–1258.

Bowen Zhao, Changkai Ji, Yuejie Zhang, Wen He, Yingwen Wang, Qing Wang, Rui Feng, and Xiaobo Zhang. 2023. Large language models are complex table parsers. *arXiv preprint arXiv:2312.11521*.

Mingyu Zheng, Yang Hao, Wenbin Jiang, Zheng Lin, Yajuan Lyu, QiaoQiao She, and Weiping Wang. 2023. IM-TQA: A Chinese table question answering dataset with implicit and multi-type table structures. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5074–5094, Toronto, Canada. Association for Computational Linguistics.

A Prompt

For all baselines, we used the official code from the original paper's GitHub repository or the official prompt from the original paper for implementation. The prompt design for SMART is as follows.

A.1 Header Flattening Prompt

We design a prompt to guide the LLM in capturing the hierarchical semantics of multi-level table headers and converting them into a single, reasoning-friendly natural language sentence. This ensures that the flattened output preserves the essential structural and semantic information required for downstream reasoning. The prompt is illustrated in Figure 7.

A.2 Global Understanding Prompt

We design a prompt to guide the LLM in producing a structured, high-level summary of a table and its associated question. By generating this textual summary, the model is provided with reasoning cues that facilitate stable and interpretable multi-step reasoning over complex tables (Prompt shown in Figure 8).

The Global Understanding is derived solely from the input table and question, serving as a summary-level reasoning guide without introducing external information.

A.3 Pseudo-Code-Style Reasoning Prompt

As illustrated in Figure 9, the prompt for Pseudo-Code–Style Reasoning is designed to provide detailed guidance for each operation, including its purpose, input and output format, and execution rules. This structured prompt ensures that the model generates consistent and interpretable reasoning steps for complex table question answering.

A.4 Reasoning Revise Prompt

When the external calculator or sorter detects an error in the reasoning process, it automatically generates a correction feedback to guide the model to revise its reasoning. Based on this feedback, the large language model is prompted to re-perform the reasoning with the identified mistakes explicitly addressed. The prompt used for reasoning correction is illustrated in the Figure 10.

A.5 LLM Eval

As shown in Figure 11, we adopt the LLM Eval prompt template from GraphOTTER to conduct our evaluation.

B Experimental Details

B.1 Metrics

While Exact Match is commonly used for evaluation, the flexible and open-ended nature of LLM outputs makes direct string matching or similarity-based metrics insufficient for reliably assessing both our method and the baselines (Zhao et al., 2023). Following prior work (Zhao et al., 2023; Li et al., 2025), we therefore adopt an LLM-based evaluator to verify the correctness of predictions generated by all methods. Specifically, we use the GPT-4.1 API as the evaluation backbone. The prompt template for LLM-based evaluation is provided in Appendix A.5.

B.2 Implementation Details

Model We locally deploy two LLM backbones, Qwen2-72B-Instruct and LLaMA3.1-70B-Instruct. For all experiments, the temperature is set to 0.1 to reduce randomness during generation, and the maximum number of output tokens is fixed at 4096. All models are executed in `bfloat16` precision. Unless otherwise specified, all remaining hyperparameters follow the default settings provided by the corresponding model implementations.

Dataset All datasets used in our experiments follow the original official splits and settings. A summary of dataset statistics is provided in Table 1, and the corresponding dataset files are available at `https://anonymous.4open.science/r/SMART-85FA`.

Iteration Analysis We analyze the effect of the maximum number of reasoning iterations in SMART. Experimental results indicate that enabling a single iteration already brings a substantial performance improvement over the non-iterative setting. Increasing the iteration number to two yields a marginal additional gain, while further increasing the number of iterations does not lead to noticeable improvements. This phenomenon can be explained by the error characteristics observed in Pseudo-Code–Style Reasoning. In most cases, the reasoning process involves one numerical hallucination, which can be effectively identified and corrected within a single repair iteration. When multiple iterations are triggered, they are typically caused by malformed computation expressions that violate predefined format constraints. In such cases, additional iterations fail to provide meaningful corrections and thus do not contribute to further performance gains. Based on this observation, we empirically set the maximum number of iterations to three, which is sufficient to cover rare corner cases while avoiding unnecessary computation and potential over-correction.

	Number of LLM calls per QA
Binder	50
Dater	100
TabSQLify	2
Chain-of-Table	≤ 25
PoTable	≤ 6
Ours	≤ 5

Table 6: Comparison of the number of LLM calls for SMART and mainstream TQA methods.

Moreover, we compare the number of LLM calls with other methods in Table 6. SMART achieves better performance with substantially fewer model invocations, demonstrating its practical effectiveness.

C Pseudo-Program Reasoning Actions

To facilitate structured, interpretable, and verifiable reasoning in complex table question answering, we introduce a set of five atomic reasoning actions that together form a Pseudo-Code-Style Reasoning chain. Each reasoning step invokes exactly one predefined action and strictly adheres to a fixed syntax, enabling deterministic parsing and external validation. The five actions are detailed below. The detailed prompt definition is illustrated in Figure 9.

C.1 SELECT

SELECT is responsible for retrieving relevant information from the table.

Purpose. This action grounds the reasoning process in explicit table evidence by selecting rows, columns, or individual cells prior to any inference or computation.

Arguments.

- `rows=<row_list>`: identifiers or indices of selected rows;
- `cols=<col_list>`: identifiers or indices of selected columns.

Behavior. The action extracts the values corresponding to the specified rows and columns. Selection may operate at different granularities, including entire rows, entire columns, or specific cells.

Output. A numeric value, a list of values, or a dictionary of selected values.

Constraints. No arithmetic or logical reasoning is allowed in this step. The `formula` field must be empty.

C.2 COMPUTE

COMPUTE performs explicit arithmetic operations over numeric values obtained from previous steps.

Purpose. This action ensures that numerical reasoning is executed in a fully constrained and machine-verifiable manner.

Arguments.

- `expression="<arithmetic_expression> = <result>"`.

Behavior. The left-hand side must be a machine-parseable arithmetic expression, and the right-hand side must be its computed result.

Allowed Operators. `+`, `-`, `*`, `/`. Mixed operations and parentheses are permitted. As well as the *len* function for counting elements. The format is specified as `len([...]) = <result>`.

Constraints. The expression must not contain text, units, commas, or explanations. Only numeric literals and arithmetic operators are allowed.

Output. A single numeric value corresponding to the computed result.

C.3 SORT

SORT orders a list of numeric values.

Purpose. This action supports comparison- and ranking-based reasoning by providing a deterministic sorting operation.

Arguments.

- `list_of_numbers=[...]`.

Behavior. The input list is sorted in *strict ascending order*.

Constraints. The input must be a pure list of numeric values. No filtering, aggregation, or inference is allowed.

Output. A list of numbers in ascending order.

C.4 INFER

INFER performs logical reasoning or summarizes intermediate results.

Purpose. This action enables high-level reasoning that cannot be reduced to simple selection or arithmetic, such as comparisons, condition checking, or intermediate conclusions.

Arguments.

- `inputs=[STEP_n, STEP_m, ...]`: references to previous reasoning steps.

Behavior. The action derives an intermediate conclusion or inferred value based on the outputs of prior steps.

Flexibility. INFER may be invoked at any point in the reasoning chain and can integrate multiple preceding steps.

Constraints. No new numerical computation should be introduced unless supported by prior **COMPUTE** steps. The `formula` field must be empty.

Output. An inferred value or intermediate conclusion.

C.5 ANSWER

ANSWER outputs the final result of the question.

Purpose. This action terminates the reasoning chain and returns the final answer.

Arguments.

- `value=<final_value>`.

Behavior. The action outputs the final value derived from all previous reasoning steps.

Constraints. No additional explanation, units, or symbols are allowed.

Output. The final answer only.

D Case Study

We present a complete reasoning example in Figure 12. First, the Semantic Header Flattening module converts the multi-level table headers into a markdown table with single-level row and column headers. Next, the Global Understanding module generates a textual summary that guides the subsequent Pseudo-Code–Style Reasoning. After the first reasoning, pattern matching identifies the formula in the Compute step and performs a verification, revealing a calculation error. A repair hint is then constructed to prompt the model to correct the reasoning. As illustrated in the Figure 12, the model successfully revises the erroneous computation and subsequent reasoning steps, ultimately producing the correct answer.

```
You are an expert in table understanding.

1.Task:
    - Convert multi-level headers into concise, natural-language single-level header
    phrases that capture the full hierarchical meaning. Each multi-level header is provided
     as a list  of hierarchical strings.

2.Input Format: multi_level_headers: List[List[str]]
    - Each inner list represents a multi-level header with hierarchical segments.
    - Example: [['Demographics', 'Age', '18-25'],['Income', 'Total', 'Median']]

3.Processing Rules:
    - Transform each multi-level header into a single, streamlined phrase that preserves
    all hierarchical meaning.
    - Identify key entities in the headers. Don't paraphrase or replace them unless
    necessary. You may insert prepositions or conjunctions to improve fluency.
    - Ensure every header is processed. Do not omit any header.

4.Output Format:
    - Do NOT include explanations.
    - Output a single line starting with "Headers:" followed by a list of phrases.
    - Example: Headers: ['Demographics age 18-25', 'Median total income']

Begin!
**Input**:
Headers: {Headers}

**Output**:
Headers: []
```

Figure 7: Prompt for Header Flattening

```
You are an expert in table understanding.
Given a table caption, top_headers,left_headers and a question, you must complete the
following analysis strictly following the specified format.

Task: Write a coherent text,describe the table structure as if explaining it to a reader.
1.Table Overview: Summarize what the table is about based on the caption.
2.Header Analysis:
  - For headers in top_headers and left_headers, briefly describe their meaning. If
  numerical measures appear (e.g., percent,population, rate, growth), indicate their meaning
  . Pay attention to headers that indicate total, all, entire region, etc.
  - Describe any natural logical relationships (e.g., time order, category groupings, region
  -subregion, measure vs. dimension) between headers.
  - Pay special attention to some summative or aggregated nodes(e.g., "all", "combine", "
  total", "sum", "average", "mean", "percent", "percentage", "proportion", "%", "probability
  ", "likelihood", etc.), as these headers help skip a lot of operations.
3.Question Analysis:
  - Carefully analyze what the question is asking (percentage, name, growth, decline, etc.),
   describe the relationship between it and the table data.
  - When the question asks about proportion or percentage, if data type in the table is
  already proportion or percentage, and does not need to be calculated.
4.Expected Answer Type:
  - Identify the type of expected answer, reminder the answer type so that a downstream
  reasoning module knows how to format its final answer.(e.g., A percent, A national name)

Begin!
**Input**:
1) Caption: {caption}
2) Top Headers: {top_headers}
3) Left Headers: {left_headers}
4) Question: {question}

**Output**: (a text)
```

Figure 8: Prompt for Global Understanding

```
You are an expert in Table Question Answering(TQA).
You will receive:
    Question
    TQA guidance
    Table - Markdown, first row = column headers, first column = row headers
Task:
    Follow TQA guidance, reason **step by step using a pseudo-program**, in which each step
    is a single action chosen from the following allowed actions:
    1. SELECT(rows=<row_list>, cols=<col_list>)
       - Select rows, columns, or cells from the table
       - formula: []
       - output: number, list, or dictionary of selected values

    2. COMPUTE(expression="<arithmetic_expression> = <result>")
        - Perform arithmetic computation in this step
        - The left side <arithmetic_expression> MUST be a **machine-parseable expression**
        - Allowed operators: +, -, *, /
        - Allowed function: len([...]) used for counting
        - MUST NOT contain text, commas, units, or explanations
        - Mixed operations ARE allowed: valid: "(12 - 2) * 3 / 2 = 15"
        - output MUST be the computed numeric result only

    3. SORT(list_of_numbers=[...])
       - Sort a list of numbers in **strict ascending order**
       - formula must be a pure list of numbers in ascending order
       - output: the sorted list

    4. INFER(inputs=[STEP_n, STEP_m, ...])
       - Perform logical inference or summarize previous steps
       - Can be used anywhere in the chain
       - formula: []
       - output: intermediate conclusion or inferred value

    5. ANSWER(value=<final_value>)
       - Output the final answer
       - formula: []
       - output: the final value only (no units, symbols, extra text)

Template: Each step must strictly follow this syntax:
    STEP <number>: <ACTION>(<arguments if any>)
        description: <one-line natural language explanation>
        formula: <list of formulas, [] if none>
        output: <result>

Rules:
    1. Each step must perform **exactly one action**.
    2. COMPUTE must contain exactly one equation of the form (no commas, units, or text): <
    arithmetic_expression> = <numeric_result>
    3. SORT formula must contain strictly ascending numbers list: [num1,num2,num3...]
    4. ANSWER output must be the final value only

Begin!
**Input**:
    Question: {question}
    TQA guidance:{global_understanding}
    Table: {table}

**Output**:(Do not output any other text)
    Step 1: SELECT(rows=..., cols=...)
    ...
    Step N: ANSWER(value=...)
```

Figure 9: Prompt for Pseudo-Code-Style Reasoning

```
You are an expert reasoning corrector for Table Question Answering (TQA).
You will receive:
    1.Question
    2.Table - Markdown table (first row = column headers, first column = row headers)
    3.Previous_Reasoning - a step-by-step pseudo-program using the specified STEP format
    4.Revise_Suggestions - natural-language comments describing what needs to be corrected.(
    formatting error|wrong sorting|arithmetic errors)
Reasoning format:
    Reason **step by step using a pseudo-program**, in which each step is a single action
    chosen from the following allowed actions:
    1. SELECT(rows=<row_list>, cols=<col_list>)
       - Select rows, columns, or cells from the table
       - formula: []
       - output: number, list, or dictionary of selected values

    2. COMPUTE(expression="<arithmetic_expression> = <result>")
        - Perform arithmetic computation in this step
        - The left side <arithmetic_expression> MUST be a **machine-parseable expression**
        - Allowed operators: +, -, *, /
        - Allowed function: len([...]) used for counting
        - MUST NOT contain text, commas, units, or explanations
        - Mixed operations ARE allowed: valid: "(12 - 2) * 3 / 2 = 15"
        - output MUST be the computed numeric result only

    3. SORT(list_of_numbers=[...])
       - Sort a list of numbers in **strict ascending order**
       - formula must be a pure list of numbers in ascending order
       - output: the sorted list

    4. INFER(inputs=[STEP_n, STEP_m, ...])
       - Perform logical inference or summarize previous steps
       - Can be used anywhere in the chain
       - formula: []
       - output: intermediate conclusion or inferred value

    5. ANSWER(value=<final_value>)
       - Output the final answer
       - formula: []
       - output: the final value only (no units, symbols, extra text)

    Each step must strictly follow this syntax:
        STEP <number>: <ACTION>(<arguments if any>)
            description: <one-line natural language explanation>
            formula: <list of formulas, [] if none>
            output: <result>

Rules:
    1. Each step must perform **exactly one action**.
    2. COMPUTE must contain exactly one equation of the form (no commas, units, or text): <
    arithmetic_expression> = <numeric_result>
    3. SORT formula must contain strictly ascending numbers list: [num1,num2,num3...]
    4. ANSWER output must be the final value only

Task:
    You must verify and correct the given pseudo-program based on the Revise_Suggestions.

Begin!
**Input**:
    Question: {question}
    Table: {caption}
    {table}
    Previous_Reasoning: {reason}
    Revise_Suggestions: {amend}

**Output**:
    Step 1: SELECT(rows=..., cols=...)
    ...
    Step N: ANSWER(value=...)
```

Figure 10: Prompt for Reasoning Revise

```
Instruction: You are CompareGPT, a machine to verify the correctness of predictions. Answer
 with only yes/no.

You are given a question, the corresponding ground-truth answer and a prediction from a
model. Compare the "Ground-truth Answer" and the "Prediction" to determine whether the
prediction correctly answers the question.
All information in the Ground-truth Answer must be present in the Prediction, including
numbers and dates.
You must answer "no" if there are any specific details in the Ground-truth Answer that are
not mentioned in the Prediction.
There should be no contradicting statements in the Prediction. The Prediction may contain
extra information. If the Prediction states something as a possibility, treat it as a
definitive answer.

Question: {question}
Ground-truth Answer: {ground_truth}
Prediction:  {response}

CompareGPT response:
```

Figure 11: LLM Evaluation Prompt

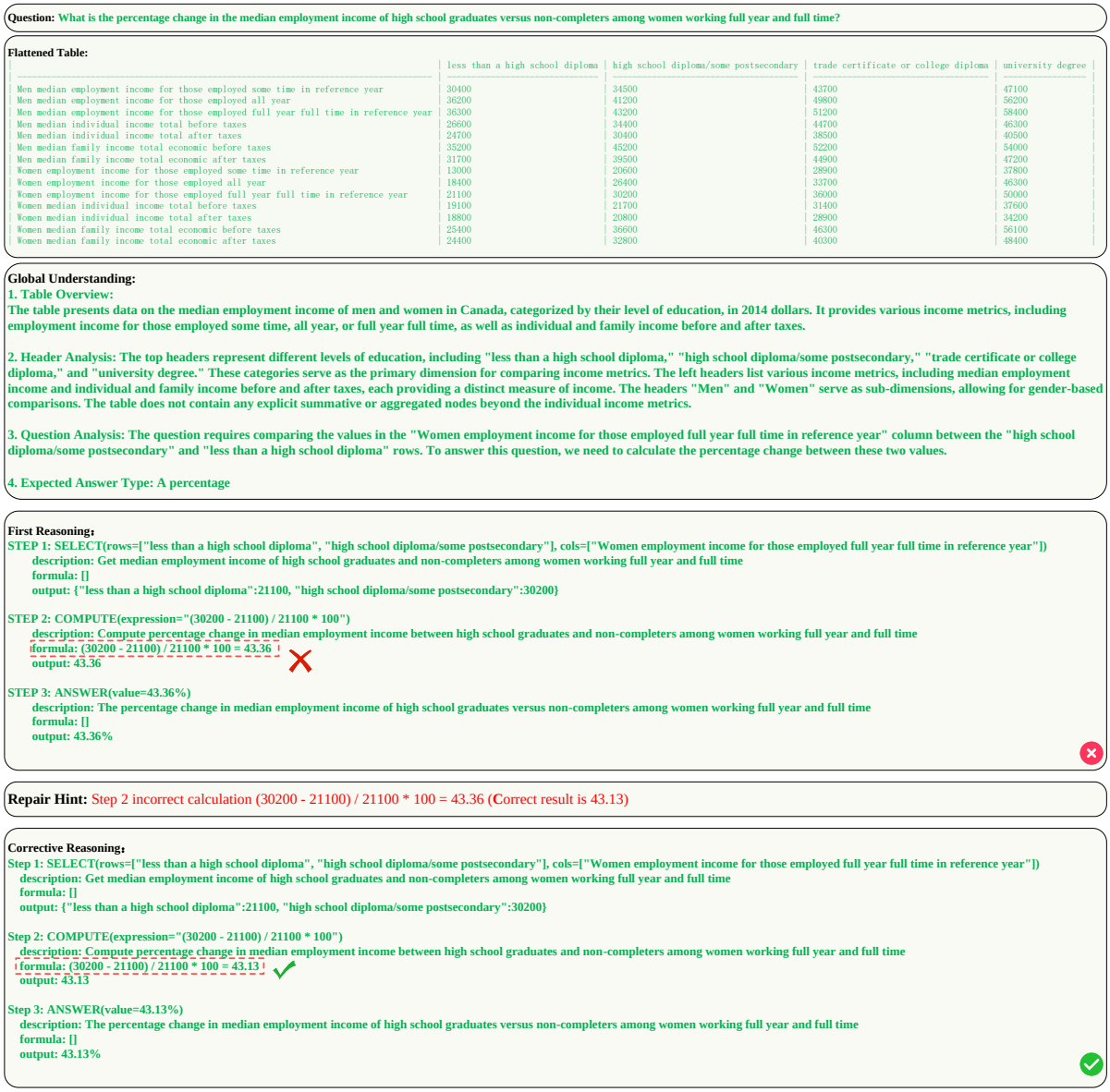

Figure 12: A complete example illustrating Qwen2's reasoning process on a HiTab table.