GraphOTTER: Evolving LLM-based Graph Reasoning for Complex Table Question Answering

Qianlong Li[♠♡], **Chen Huang**[♠♡], **Shuai Li**[♠♡]
Yuanxin Xiang[♠♡], **Deng Xiong**[♣], **Wenqiang Lei**[♠♡*]
♠ Sichuan University ♣ Stevens Institute of Technology
♡ Engineering Research Center of Machine Learning and Industry Intelligence,
Ministry of Education, China
{liqianlong0922, huangc.scu, lsbeatsleo, whutxyx}@gmail.com
dxiong@stevens.edu wenqianglei@scu.edu.cn

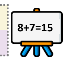
Abstract

Complex Table Question Answering involves providing accurate answers to specific questions based on intricate tables that exhibit complex layouts and flexible header locations. Despite considerable progress having been made in the LLM era, the reasoning processes of existing methods are often implicit, feeding the entire table into prompts, making it difficult to effectively filter out irrelevant information in the table. To this end, we propose GraphOTTER that explicitly establishes the reasoning process to pinpoint the correct answers. In particular, GraphOTTER leverages a graph-based representation, transforming the complex table into an undirected graph. It then conducts step-by-step reasoning on the graph, with each step guided by a set of pre-defined intermediate reasoning actions. As such, it constructs a clear reasoning path and effectively identifies the answer to a given question. Comprehensive experiments on two benchmark datasets and two LLM backbones demonstrate the effectiveness of GraphOTTER. Further analysis indicates that its success may be attributed to the ability to efficiently filter out irrelevant information, thereby focusing the reasoning process on the most pertinent data. Our code and experimental datasets are available at `https://github.com/JDing0521/GraphOTTER`.

1 Introduction

Complex tables in documents frequently employ advanced layouts like merged cells and flexible header locations (Zheng et al., 2023). This complexity facilitates the structured presentation of detailed information in a flexible manner, making them a common feature in financial reports (Zhu et al., 2021a; Chen et al., 2021) and other professional documents (Wang et al., 2021; Zhong et al., 2020). In this context, **Complex Table Question Answering** (QA) recently emerges as a crucial task

Club	Season	League			Total	
		Division	Apps	Goals	Apps	Goals
Gillingham	2007-08	League One	1	0	2	0
Stevenage	2010-11	League One	13	2	14	2
	2013-14	League One	45	6	48	6
	Total		58	8	62	8
Bristol City	2014-15	League One	46	7	51	7
	2015-16	Championship	41	1	41	1
	Total		87	8	92	8
Career Total			291	32	310	32



Question: How many goals did this player score in total for Bristol City and Stevenage in League One? **Answer:** 15

Figure 1: Example of complex table QA. Header cells are styled with a dashed background, while the reasoning process is highlighted using color.

(Zhao et al., 2023; Zheng et al., 2023; Zhang et al., 2024b), enabling the extraction of valuable information from complex tables. However, merged cells and nested structures contained in the complex tables introduce implicit semantic relationships between entities within cells, posing significant challenges for table structure understanding (Katsis et al., 2021; Cheng et al., 2022). Taking Figure 1 for example, a "total" cell represents scores across different divisions for the Bristol City, while for the Stevenage, it only encompasses scores from a single division.

To address the challenges, current research focuses on prompting large language models (LLMs) to to implicitly reason and derive answers based on carefully designed table representations. In particular, they propose transforming the table into more accessible formats such as tuples (Zhao et al., 2023), Markdown (Chen, 2022; Liu et al., 2024), and HTML (Zhang et al., 2024b). Some of these approaches also involve annotated table headers within the transformed representations to further improve the model's understanding of complex, hierarchical structures (Zhao et al., 2023). Subsequently, these techniques integrate the transformed table representations into carefully designed prompts to unlock the reasoning potential of LLMs (Cheng et al., 2022; Zhao et al., 2023; Zhang et al., 2024b),

[*]Corresponding author.

Proceedings of the 31st International Conference on Computational Linguistics, pages 5486–5506
January 19–24, 2025.

thereby enabling them to pinpoint accurate answers within complex tables.

However, the reasoning processes employed by existing approaches are frequently embedded within simple, free-form prompts, which struggles to offer a structured way to represent the intricate relationships among data in different (merged) columns and rows. As illustrated in Figure 1, identifying the correct answer typically involves focusing on a limited number of cells within the table (highlighted in color). In such scenarios, an implicit reasoning process that relies on the entire table representation might not effectively filter out irrelevant information (marked in white). This limitation can significantly diminish the effectiveness of complex table question answering (QA) systems and reduce their practical utility when dealing with detailed and intricate tabular data.

To this end, we aim to explicitly establishing the reasoning process for complex table QA, thereby assisting LLMs in filtering out irrelevant information and enhancing the task effectiveness. In this paper, we propose a novel approach, called **GraphOTTER**, that evolves Graph reasoning for cOmplex Table quesTion answERing. In particular, GraphOTTER leverages a graph-based representation, transforming the complex table into an undirected graph where header and data cells are treated as nodes. It also introduces a set of intermediate reasoning actions on the graph, such as *VisitNode*, *GetSharedNeighbours*, and *AnswerQuestion*, which act as tools to facilitate navigation and reasoning within the graph structure. Building upon this, GraphOTTER prompts LLMs to conduct step-by-step reasoning on the graph. For each reasoning step, it selects the most appropriate action to guide the LLM towards the answer. To facilitate the selection, a reasoning trace is maintained to track visited nodes, enabling GraphOTTER to effectively steer the reasoning process and filter out irrelevant information. As such, it constructs a clear reasoning path and effectively identifies the answer.

To evaluate our effectiveness, we conduct comparative experiments with various baselines using two benchmark datasets and two LLM backbones[1]. Compared to baselines, GraphOTTER achieves a notable improvement in complex table question answering, exhibiting an average performance gain of +4.77% over the best baseline. Our in-depth analysis reveals that this success can be attributed to the explicit reasoning process, which reduce irrelevant information in the graph/table. Moreover, we experimentally show that combining graph representations with explicit reasoning creates a powerful synergy that harnesses the flexibility of the graph while mitigating its inherent complexity. This approach holds significant promise for accurately pinpointing answers from complex tables. To sum up, we claim the following contributions.

- We call attention to the importance of explicitly establishing the reasoning process for Complex Table QA. This promotes the task effectiveness and its real-world utility.
- We propose GraphOTTER, a novel LLM-based approach that evolves graph reasoning to construct a clear reasoning path and effectively identifies the answer to a given question.
- We conduct extensive experiments on benchmark datasets to show the superiority of GraphOTTER. We further analyze the potential reason for the effectiveness of graph reasoning.

2 Related work

We focus on evolving LLM-based graph reasoning for complex table QA task. We clarify our difference to existing methods as follows.

Representation for Complex Table. Unlike simple tables (Zhu et al., 2021b; Deng et al., 2022), which can be readily processed, the complex structures of real-world tables necessitate the development of more sophisticated representations to facilitate effective model comprehension. This includes transforming table headers into hierarchy-aware logical form (Cheng et al., 2022) and converting the entire table into tuples (Zhao et al., 2023), Markdown (Chen, 2022; Liu et al., 2024), and HTML (Zhang et al., 2024b). However, these methods typically rely on annotated table headers or assume that headers are located only at the top or left side of the table. In contrast, GraphOTTER empowers the LLM to infer header cells among connected nodes in the graph, enhancing its adaptability to diverse table structures.

Reasoning for Complex Table QA. Complex table QA relies on the ability to reason with information directly from the table itself to derive accurate answers to specific questions. Currently, the reasoning processes of existing methods employed are often implicitly represented within free-form text prompts (Cheng et al., 2022; Zhao et al., 2023), fine-tuned table embeddings based on the GCN or

[1]Taking their costs and effectiveness into consideration.

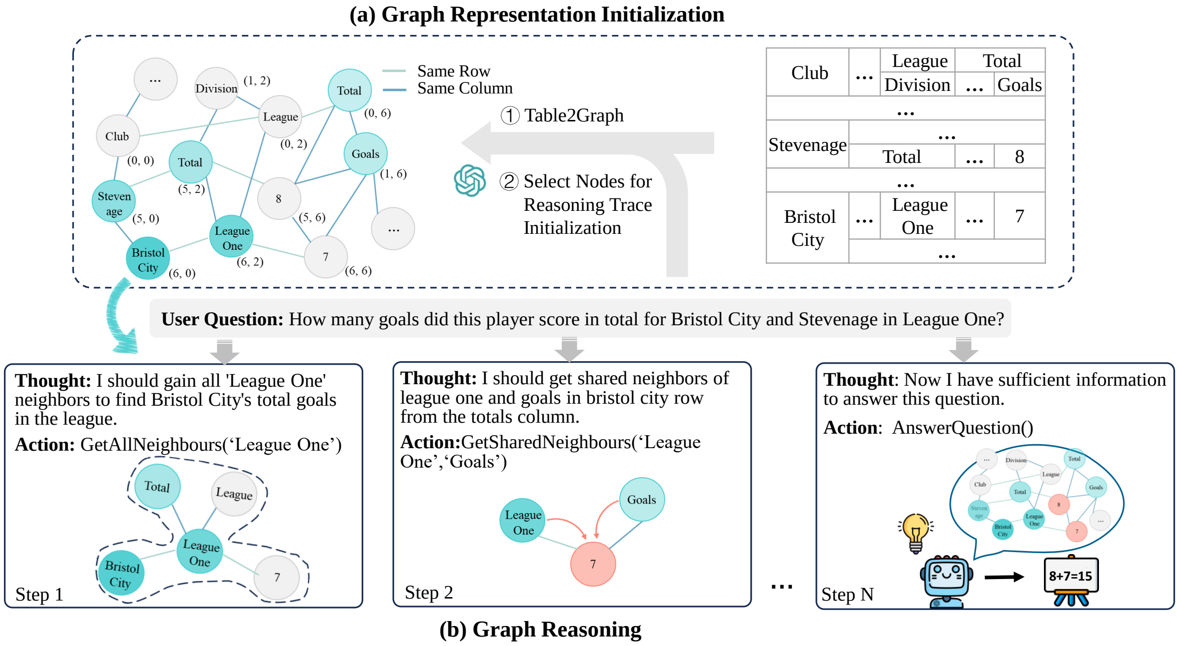


Figure 2: Overview of GraphOTTER. It first transforms the table into a graph representation (For clarity, this graph only shows some of the edges between nodes in the same row/column). Then, it establishes a step-by-step reasoning process on this graph, guided by a reasoning trace and a set of pre-defined intermediate actions (Note that the nodes in the reasoning trace are colored). These actions facilitate navigation and reasoning within the graph structure, leading to a more explicit and efficient approach for complex table QA.

transformer (Zheng et al., 2023; Jia et al., 2023), or code interpreter (Zhang et al., 2024b), which struggles to offer a structured way to represent the intricate relationships among data in different (merged) columns and rows. In this case, we argue that the implicit reasoning may be inadequate when being tasked with reasoning over complex tables. By explicitly representing the reasoning process, GraphOTTER overcomes the limitations, paving the way for more accurate complex table QA. Notably, GraphOTTER distinguishes itself from existing graph-based complex table QA methods, which typically rely on fine-tuning transformers and GNN-style models for table structure encoding (Zheng et al., 2023; Jia et al., 2023). GraphOTTER, in contrast, explicitly reasons through a series of intermediate steps on the graph via in-context learning. Also, GraphOTTER may be conceptually close to Chain-of-Table (Wang et al., 2024) yet vitally different due to the following reasons. While Chain-of-Table focuses on simple table understanding, employing multi-step tabular reasoning to form a chain of transformed tables, GraphOTTER leverages graph reasoning to address the complex table QA. See Table 2 for our effectiveness.

LLM-based Graph Reasoning. LLM-based graph reasoning involves utilizing LLMs to analyze and interpret the graph, extracting new insights or relationships based on the graph's structure and properties (Agrawal et al., 2024). This process typically entails a multi-step approach: identifying the required information, searching for relevant data within the graph, and generating an answer based on the retrieved information (Ren et al., 2024; Shang and Huang, 2024). To enhance reasoning capabilities, some approaches further integrate external tools like retrievers (Sun et al., 2023; Wang et al., 2023; Jiang et al., 2024; Wu et al., 2023). In this paper, we evolving the LLM-based graph reasoning to solve the complex table QA task, which guides the reasoning process in a step-by-step manner, avoiding irrelevant information.

3 GraphOTTER for Complex Table QA

Task Formulation. Given a complex table T and a user question Q, we transform the table into a graph $G = \{\mathcal{N}, \mathcal{E}, \mathcal{V}\}$, where $\mathcal{N}$, $\mathcal{E}$ and $\mathcal{V}$ represent the sets of nodes, edges, and corresponding cell values, respectively. Leveraging this graph representation, we conduct step-by-step reasoning to pinpoint the correct answer A. In this process, we dynamically maintain a reasoning trace Tr, represented as a subgraph that consists of the visited nodes.

3.1 Method Overview

GraphOTTER aims to establish the step-by-step reasoning process for complex table QA. As depicted in Figure 2, at each reasoning step, GraphOTTER prompts the LLM to select a reasoning action from a pre-defined set of intermediate reasoning actions $\mathcal{R}$ (detailed in Table 1). To guide this process, GraphOTTER introduces a reasoning trace Tr that tracks the nodes visited during reasoning, which further contributes to the selection of the next intermediate action. Finally, the process terminates when GraphOTTER selects the *AnswerQuestion* action, producing the final answer. See the pseudo code in Algorithm 1 and the case study in Appendix C for better understanding.

3.2 Graph Representation Initialization

GraphOTTER transforms the input table T into an undirected graph G, where edges connect cells within the same row or column. In particular, each node $\mathcal{N}_i \in \mathcal{N}$ is denoted as a triple $(Rid, Cid, \mathcal{V}_i)$, where Rid and Cid are the row and column indices in the table. Notably, for merged cells, their Rid or/and Cid values are represented as sets containing the indices of the corresponding split cells. Moreover, each edge $\mathcal{E}_{ij} \in \mathcal{E}$ is denoted as a triple $(\mathcal{N}_i, \mathcal{N}_j, M_{ij})$, where M_{ij} indicates whether the two connected nodes are in the same row or the same column in the table. Notably, both header and data cells are treated as nodes without distinction. This approach eliminates the reliance on pre-defined headers, unlike existing methods that require pre-annotated headers or assume headers are located only at the top or left side of the table (Wang et al., 2024; Liu et al., 2024; Ye et al., 2023; Cheng et al., 2022). This enhances GraphOTTER's adaptability to diverse table structures.

To initiate the reasoning process, GraphOTTER instructs the LLM to select a small set of nodes relevant to the user question Q as the initial set for initializing the reasoning trace Tr. For implementations, we initially filter up to eight most relevant cells, with the sensitivity analysis being explored in Section 4.4.

3.3 Graph Reasoning for Complex Table QA

GraphOTTER implements a **Thought-Action-Update** process for step-by-step graph reasoning. To guide the reasoning process, it further introduces a reasoning trace, Tr, which tracks visited nodes during reasoning. Specifically, at each step,

Reasoning Actions	Description
VisitNode	Go to the specified node on the graph.
GetAllNeighbours	Get all neighbors of the specified node.
GetSharedNeighbours	Get all shared neighbors between two specified nodes.
AnswerQuestion	Answer the user question.

Table 1: Reasoning actions on the graph. They are simple yet flexible, encompassing common operations required for reasoning on the graph.

GraphOTTER prompts the LLM to first think about the current reasoning state based on the current reasoning trace and other relevant arguments. Based on this thought process, GraphOTTER selects an action from the reasoning action set $\mathcal{R}$, as outlined in Table 1. Finally, the trace is updated after GraphOTTER executes the chosen action, which in turn influences the subsequent reasoning state.

Thought. GraphOTTER gathers scattered information to establish its current reasoning state, considering not only the global information provided by the entire graph but also local information specific to the input question. This local information includes the outputs of the previous *Thought* and *Action* step, the historical reasoning trace, which helps distill key insights from past reasoning steps, allowing GraphOTTER to focus on important details, and self-inferred semantic relationships between connected nodes, leveraging the LLM's semantic understanding capabilities to identify header cells.

Action. We guide GraphOTTER to select the subsequent intermediate action from the action set R based on the reasoning state. As outlined in Table 1, we have designed a simple yet flexible set of reasoning actions encompassing common operations required for reasoning on the graph. These actions, acting as tools, enhance GraphOTTER's reasoning performance. In particular, Action *VisitNode* facilitates node selection and traversal within the graph. In scenarios where multiple nodes share the same value, GraphOTTER presents these nodes to the LLM for selection, accompanied by structural information (Cid and Rid) to enable informed decision-making. Furthermore, actions *GetAllNeighbours* and *GetSharedNeighbours* empower GraphOTTER to focus on local information and analyze the surrounding neighbors of selected node(s). We also incorporate the action of answering questions within this set. This aligns with the concept of learning to defer (Madras et al., 2018; Tailor et al., 2024), which promotes GraphOTTER to automatically evaluate the need for additional information before answering, rather than relying on heuristic rules as in (Wang et al., 2024).

Notably, unlike implicit reasoning where the entire table is embedded in a prompt (Cheng et al., 2022; Zhao et al., 2023; Zhang et al., 2024b), our method explicitly establishes the reasoning path via executing these actions, which could effectively filter out irrelevant information and guide reasoning towards a more accurate answer, a topic we will analysis in Section 4.3.

Update. The reasoning trace Tr is updated by appending the new visited node after GraphOTTER executes the *VisitNode* action. The newly added node establish edges with previously visited nodes if they share common neighbors or are directly connected in the original graph. This process effectively updates the local information for the subsequent *Thought* step. In this way, GraphOTTER forms a iterative reasoning on the graph to pinpoint the correct answer.

Algorithm 1 Pseudo code of GraphOTTER

```
Input: (T, Q) is a complex table-question pair.
  ▷ See Section 3.2 for details.
  Transform the table T into the graph G.
  Initialize reasoning trace Tr based on G.
  ▷ Reasoning on the graph, see Section 3.3.
  repeat
      Update reasoning state via Thought based
      on reasoning trace Tr
      Reason one step via Action based the state
      Modify reasoning trace Tr via Update
  until AnswerQuestion is selected
  ▷ See Section 3.4 for details.
  Answer generation via in-context learning.
```

3.4 Answer Generation

When the *AnswerQuestion* action is selected, we instruct the LLM via CoT prompts to produce the final answer to the user question based on the question and current local reasoning state from *Thought* step, instead of using the whole graph.

4 Experiments

To evaluate our effectiveness, we conduct comparative experiments with various baselines using two benchmark datasets and two LLM backbones. In particular, we study how effective is our GraphOTTER compare to existing complex table QA methods (Section 4.2). Furthermore, we comprehensively analyze the advantages and characteristics of GraphOTTER in Section 4.3 and 4.4. For better understanding the reasoning process of GraphOTTER, we report case studies in Appendix C due to the space limit. Finally, our code and experimental datasets are available at `https://github.com/JDing0521/GraphOTTER`.

4.1 Experimental Setups

Dataset. Following Zhao et al. (2023), we use two benchmark datasets: Hitab(Cheng et al., 2022) and AIT-QA(Katsis et al., 2021). These two datasets contain a large number of complex tables collected from industrial documents, which not only provide larger-scale tables but also reflect the real data analysis scenarios in the industry. Statistics of the datasets can be found in Appendix B.

Baselines. Current approaches to complex table QA rely on predefined table headers and implicit reasoning processes. These methods can be broadly categorized into two types: *One-shot Implicit Reasoning* and *Iterative Implicit Reasoning*, which prompts LLMs in a iterative manner. We also include the only existing method that employs *iterative explicit reasoning* as a baseline.

- **One-shot Implicit Reasoning**. This category encompasses TableReasoner (Chen, 2022) and TableParser (Zhao et al., 2023), which leverage carefully crafted (CoT) prompts and table representations based on Markdown or tuples.

- **Iterative Implicit Reasoning**. ReAct (Yao et al., 2023) utilizes HTML-based table representations and guides LLMs in generating (Thought, Act, Observation) tuples. E^5(Zhang et al., 2024b) builds upon ReAct, further incorporating self-described table structure information and a code interpreter to enhance the action generation process. MIX-SC (Liu et al., 2024) employs multiple reasoning attempts for self-consistency, including carefully designed DataFrame-based representation and ReAct-style iterative reasoning using Python interpreter and in-context learning.

- **Iterative Explicit Reasoning**. This involves Chain-of-Table (Wang et al., 2024). While it is designed for simple tables, it stands out as the only iterative explicit reasoning method. It utilizes multi-step reasoning on the table, forming a chain of transformed tables through tabular operations, rather than our graph reasoning.

Evaluation Metrics. Following previous research (Zheng et al., 2023; Chen, 2022; Zhang et al., 2024b; Liu et al., 2024; Zhao et al., 2023), we employ a multifaceted evaluation approach. Firstly, we utilize Exact Match (EM) to quantify the precision of model outputs against the ground truth. Furthermore, we leverage a LLM-based evaluator (LLM Eval) to assess the accuracy of predictions. The detailed implementation is offered in Appendix A.2.

Implementation Details. Considering the financial constraints, we experiment on two LLM backbones for all methods: Gemini-1.5[2] and Qwen2[3]. To ensure reproducibility, we maintain consistent temperature and random seed settings for both backbones (where applicable). For implementations of baselines, we rigorously adhered to their official GitHub code and original paper prompts for their implementation. For GraphOTTER, to initiate the reasoning process, we selected up to eight of the most relevant cells for all datasets. Further implementation details regarding baselines and GraphOTTER are presented in Appendix A.

4.2 Main Results

Table 2 and Table 3 present an overview of the results across two benchmark datasets and LLM backbones. These results suggest that GraphOTTER stands out as the leading method for complex table QA. We provide further analysis below.

***How effective is GraphOTTER?* – It consistently outperforms all baselines across various datasets and LLM backbones, demonstrating its superiority in complex table QA**. As illustrated in Table 2, while the MIX-SC method demonstrated improved performance by concurrently incorporating multiple inference techniques, including direct prompts and ReAct, and employing self-consistency for aggregation, its results still lag behind our proposed approach. Notably, our method surpasses MIX-SC, the best baseline, by an average of +9.07% and +1.63% on datasets Hitab and AIT-QA, respectively, highlighting the significant advantage of our model. Notably, such advantage contributes to our explicit graph reasoning, a topic we will delve into later in Section 4.3.

***How well does GraphOTTER handle different user questions?* – It achieves comparable and even superior performance**. Our model, GraphOTTER, employs a graph representation where data and header cells are treated equally. This necessitates the model to infer the headers of connected nodes within the graph during the Thought step. To assess the effectiveness of this self-inferred header mechanism, we analyze performance on both header-related and header-unrelated user questions. Note that these questions are annotated within the AIT-QA dataset. As illustrated in Table 3, GraphOTTER achieves promising results on both question types, highlighting the efficacy of our self-inferred headers. Further ablation in Section 4.3 further supports this observation. In our opinion, this performance can be attributed to the powerful semantic understanding capabilities of LLMs, enhancing the GraphOTTER's adaptability to diverse table structures.

4.3 Ablation & In-depth Analysis

We delve into ablation studies and detailed analysis to investigate our strengths, where two ablations are involved. We provide the results of Gemini in Table 4 and draw the following observations.

- *GraphOTTER w/ known headers* is provided with ground truth table header information, eliminating the need for the LLM to infer headers.
- *GraphOTTER w/ implicit reasoning* utilizes the one-shot implicit reasoning approach, directly prompting the LLM to generate the answer based on graph-based representation, rather than performing explicit graph reasoning. We refer to the TableReasoner[4] prompts by creating a graph-representation variant.

***Why the graph reasoning is effective?* – The success of explicit graph reasoning may be attributed to its ability to effectively filter out irrelevant information**. Compared to GraphOTTER with its variant utilizing the same graph representation but employing the implicit reasoning, GraphOTTER demonstrates significant improvements in both datasets. This advantage is attributed to instructing the LLM to answer the question using a specific reasoning trace rather than the entire graph/table. This effectively filters out irrelevant information from the raw table. Further analysis, as shown in Figure 3, reveals that GraphOTTER utilizes a smaller number of nodes compared to other methods when generating answers. These

[2] Gemini-1.5-flash-latest

[3] Qwen2-72B-Instruct

[4] We opted to avoid using iterative implicit reasoning methods as they designed table-specified actions for ReAct-style iterations.

Reasoning Type	Methods	Gemini 1.5				Qwen2			
		Hitab		AIT-QA		Hitab		AIT-QA	
		EM	LLM Eval	EM	LLM Eval	EM	LLM Eval	EM	LLM Eval
One-shot Implicit	TableReasoner(Chen, 2022)	44.19	68.18	65.40	77.38	56.41	75.68	76.29	88.83
	TableParser(Zhao et al., 2023)	57.51	68.94	63.22	78.20	44.57	69.76	64.85	83.11
Iterative Implicit	ReAct(Yao et al., 2023)	43.88	47.60	47.14	50.41	54.73	57.89	68.12	72.75
	E5(Zhang et al., 2024b)	57.26	62.44	59.40	64.31	43.56	47.16	56.40	59.13
	MIX-SC(Liu et al., 2024)	62.41	68.99	69.95	74.86	73.42	77.08	84.20	91.83
Iterative Explicit	Chain-of-Table(Wang et al., 2024)	25.82	36.68	18.53	23.16	44.26	62.69	49.32	61.04
	GraphOTTER (*Ours*)	**67.76**	**69.66**	**81.47**	**82.56**	**73.74**	**77.37**	**88.28**	**92.64**

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

Model	Gemini 1.5		Qwen2	
	HRQ	HUQ	HRQ	HUQ
One-shot Implicit Reasoning				
TableReasoner	71.05	63.92	81.58	90.72
TableParser	**77.63**	78.35	80.26	83.85
Iterative Implicit Reasoning				
ReAct	44.74	51.89	60.53	75.95
E5	63.16	64.60	53.95	60.48
MIX-SC	59.21	78.97	**84.21**	93.81
Iterative Explicit Reasoning				
Chain-of-Table	28.95	21.65	57.89	61.86
GraphOTTER(*Ours*)	69.74	**85.91**	**84.21**	**94.85**

Table 3: Detailed evaluation on different user questions, categorized by whether their answers are related to the table headers or not. "HRQ" represents "Header-related questions", and "HUQ" represents "Header-unrelated questions". Here, LLM Eval is used. For results on EM, refer to Table 5 in appendix.

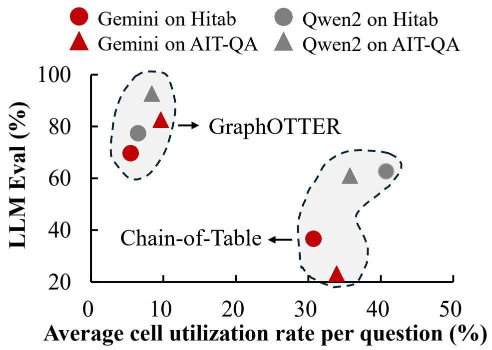


Figure 3: Average percentage of cells/nodes required for generating the final answer. We report results on explicit reasoning methods, while for implicit reasoning based methods, they use all the cells in the whole table.

nodes represent the reasoning trace produced by the model's step-by-step reasoning on the graph. Based on our calculation, on average, 93.38% of these nodes directly contain the correct answer, demonstrating the efficacy of explicit graph reasoning in eliminating irrelevant information and improving the accuracy of LLM responses.

***How effective is the graph-based table representation?* – Combining graphs with explicit reasoning creates a powerful synergy**. While graph representations offer significant potential, their inherent complexity (e.g., richer node relationships than tables) compared to tables can pose challenges for implicit reasoning.

LLMs processing graphs may have difficulty distinguishing key connections from irrelevant ones, resulting in inaccuracies (Refer to *TableReasoner* and the *implicit reasoning variant of GraphOTTER* in Table 4). However, combining graphs with explicit reasoning methods can alleviate these issues (cf. our results).

Explicit reasoning on the graph enables systematic exploration and identification of key connections and inferences from its structure. This targeted approach may reduce the risk of noise or irrelevant relationships impacting results, ultimately enhancing the accuracy and reliability of the inferences. Thus, combining graphs with explicit reasoning creates a powerful synergy that harnesses the flexibility of graphs while mitigating their inherent complexity, which holds immense promise for extracting correct answer from complex table.

***How effective is the self-inferred table headers?* – The self-inferred table headers demonstrates high availability and their effectiveness closely resemble that of using ground truth**. Through controlled variable experiments, we confirm the effectiveness of the self-inferred table header information, demonstrating results of GraphOTTER comparable to those obtained using GraphOTTER *w/ known headers*.

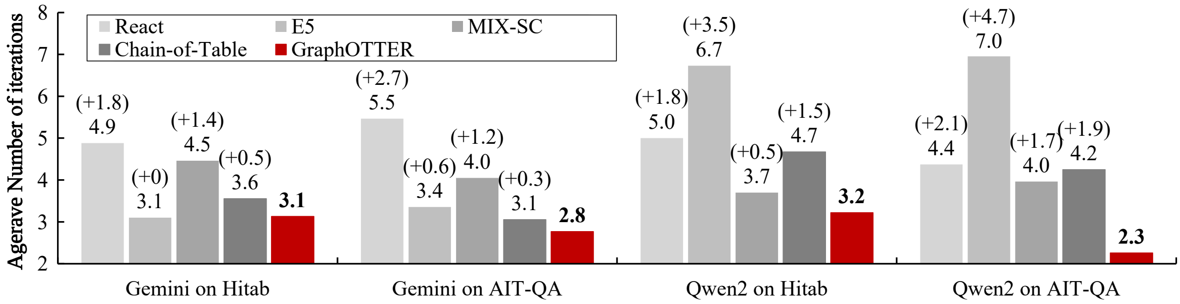


Figure 4: Illustration of the iteration efficiency of different methods. GraphOTTER achieves superior performance while requiring fewer iterations.

Model	HiTab		AIT-QA	
	EM	LLM Eval	EM	LLM Eval
TableReasoner	44.19	68.18	76.29	77.38
Chain-of-Table	25.82	36.68	18.53	23.16
GraphOTTER	67.76	69.66	81.47	82.56
w/ known headers	**68.46**	**70.55**	**82.29**	**82.83**
w/ implicit reasoning	27.97	67.05	45.78	79.84

Table 4: Ablation studies using Gemini. Combining the graph with explicit reasoning creates a powerful synergy for complex table QA. The GraphOTTER-inferred table headers demonstrates high availability.

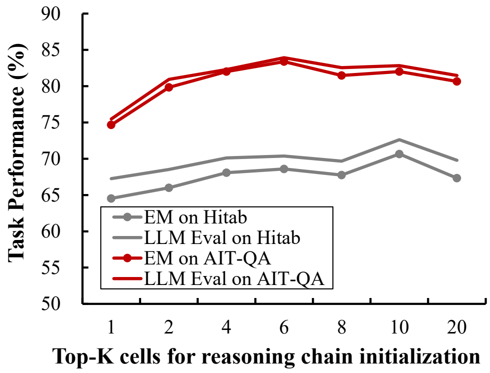


Figure 5: Impact of reasoning trace initialization (i.e., tuning top-k relevant cells to initialize the trace). Proper reasoning trace initialization ensures that GraphOTTER begins its reasoning with enough information without being overwhelmed by irrelevant details.

4.4 Characteristics of GraphOTTER

This section aims to uncover characteristics of GraphOTTER in terms of iteration efficiency and hyper-parameter sensitivity within reasoning trace initialization. Detailed observations are as follows.

How is the iteration efficiency of GraphOTTER? **– GraphOTTER achieves superior performance while requiring fewer iterations**. Figure 4 presents the average number of iterations per user question for GraphOTTER and iterative reasoning-based baselines across two datasets. The results demonstrate that GraphOTTER consistently requires fewer iterations while achieving higher QA performance (cf. Table 2). Importantly, ReAct and E5 exhibit a higher number of iterations across different backbones, primarily due to errors in the LLM-generated Python code, leading to a recursive cycle of erroneous corrections. Chain-of-Table, designed for simpler table understanding, struggles to model complex relationships between multiple cells, necessitating additional iterations to retrieve comprehensive information. In contrast, GraphOTTER's graph-based representation and step-by-step reasoning capabilities enable it to converge on accurate answers with fewer iterations.

What is the impact of reasoning trace initialization? **– Proper reasoning trace initialization ensures that GraphOTTER begins its reasoning with enough information to make accurate inferences without being overwhelmed by irrelevant details**. Considering GraphOTTER instructs the LLM to select a small set of nodes relevant to the question Q as the initial set for the reasoning trace, we analyze the impact of the number of these initial nodes on GraphOTTER's performance, as shown in Figure 5. The results indicate that performance suffers when the number of nodes is either too small or too large. With too few nodes, GraphOTTER may lack sufficient information, ultimately degrading overall performance. Conversely, excessive nodes may introduce irrelevant information, interfering with the LLM's reasoning process and negatively impacting performance. A potential solution to these issues could involve equipping GraphOTTER with the ability to trace back and try new reasoning paths on the graph. However, given our focus on the initial development of graph reasoning for complex table QA, we leave this for future research. Empirically, we recommend using at most 4 to 8 initial nodes, and all experiments in this paper choose up to 8 nodes without further tuning.

5 Conclusions

Closely revolving around the complex table QA, we introduce GraphOTTER, a novel method that explicitly establishes a reasoning process to pinpoint the correct answers. GraphOTTER converts the table into a graph and performs step-by-step reasoning with intermediate actions, leading to a clear and efficient path to the answer. Our extensive evaluation demonstrates that GraphOTTER outperforms existing implicit reasoning methods across various benchmark datasets and LLM backbones. This success can be attributed to GraphOTTER's ability to filter out irrelevant information, focusing the reasoning process on the most pertinent data. The powerful synergy between graphs and explicit reasoning harnesses the flexibility of graph representations while mitigating their inherent complexity. We believe this may offer immense promise for accurately extracting answers from complex tables.

Beyond handling complex table structures, addressing complex user queries is crucial for the real-world effectiveness of TableQA systems. Therefore, future work will extend this method to multi-turn conversation scenarios, enabling active exploration and clarification of complex user questions (Zhang et al., 2024a; Chen et al., 2023).

Limitations

Iterative reasoning can be more expensive than one-shot methods. Our method, like other iterative reasoning approaches, requires multiple interactions with the LLM to answer a single question, which can be more expensive than one-shot methods. While our method proves to be more efficient in terms of the number of iterations compared to other iterative methods (as shown in Figure 4), our future work will focus on combining one-shot and iterative reasoning techniques to minimize unnecessary interactions with the LLM. This combination will allow us to achieve a balance between cost-effectiveness and task performance.

More advanced reasoning actions on the graph. This paper introduces four simple yet flexible reasoning actions, demonstrating their effectiveness in our experiments. However, to further enhance the performance of graph reasoning, future research will explore the introduction of more advanced actions. For instance, extending the actions related to accessing neighbors of specified nodes to include n-hop neighbors could provide the model with a longer-term perspective. However, as this study focuses on introducing graph reasoning to complex table QA for the first time, the exploration of advanced actions will be deferred to our future research.

Potential sensitivity to prompts. Similar to all other studies on prompting LLMs (Cheng et al., 2022; Zhao et al., 2023; Zhang et al., 2024b), our evaluation results could be influenced by the prompts. The impact of prompt design and prompt optimization represents a significant area for further research within the field of LLMs. Exploring the influence of different prompts and identifying optimal ones are crucial for maximizing the performance and effectiveness of LLM-based methods.

Acknowledgements

This work was supported in part by the National Natural Science Foundation of China (No. 62272330); in part by the Fundamental Research Funds for the Central Universities (No. YJ202219); in part by the Science Fund for Creative Research Groups of Sichuan Province Natural Science Foundation (No. 2024NSFTD0035); in part by the National Major Scientific Instruments and Equipments Development Project of Natural Science Foundation of China under Grant (No. 62427820)

References

Vaibhav Adlakha, Parishad BehnamGhader, Xing Han Lu, Nicholas Meade, and Siva Reddy. 2023. Evaluating correctness and faithfulness of instruction-following models for question answering. *arXiv preprint arXiv:2307.16877*.

Palaash Agrawal, Shavak Vasania, and Cheston Tan. 2024. Can llms perform structured graph reasoning? *Preprint*, arXiv:2402.01805.

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

Yue Chen, Dingnan Jin, Chen Huang, Jia Liu, and Wenqiang Lei. 2023. TRAVEL: Tag-aware conversational FAQ retrieval via reinforcement learning. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3861–3872, Singapore. Association for Computational Linguistics.

Zhiyu Chen, Wenhu Chen, Charese Smiley, Sameena Shah, Iana Borova, Dylan Langdon, Reema Moussa, Matt Beane, Ting-Hao Huang, Bryan Routledge, et al. 2021. Finqa: A dataset of numerical reasoning over financial data. *arXiv preprint arXiv:2109.00122*.

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.

Yang Deng, Wenqiang Lei, Wenxuan Zhang, Wai Lam, and Tat-Seng Chua. 2022. Pacific: Towards proactive conversational question answering over tabular and textual data in finance. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 6970–6984.

Ran Jia, Haoming Guo, Xiaoyuan Jin, Chao Yan, Lun Du, Xiaojun Ma, Tamara Stankovic, Marko Lozajic, Goran Zoranovic, Igor Ilic, et al. 2023. Getpt: Graph-enhanced general table pre-training with alternate attention network. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 941–950.

Jinhao Jiang, Kun Zhou, Wayne Xin Zhao, Yang Song, Chen Zhu, Hengshu Zhu, and Ji-Rong Wen. 2024. Kg-agent: An efficient autonomous agent framework for complex reasoning over knowledge graph. *arXiv preprint arXiv:2402.11163*.

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

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)*, pages 450–482, Mexico City, Mexico. Association for Computational Linguistics.

David Madras, Toni Pitassi, and Richard Zemel. 2018. Predict responsibly: improving fairness and accuracy by learning to defer. *Advances in neural information processing systems*, 31.

Xubin Ren, Jiabin Tang, Dawei Yin, Nitesh Chawla, and Chao Huang. 2024. A survey of large language models for graphs. *arXiv preprint arXiv:2405.08011*.

Wenbo Shang and Xin Huang. 2024. A survey of large language models on generative graph analytics: Query, learning, and applications. *arXiv preprint arXiv:2404.14809*.

Jiashuo Sun, Chengjin Xu, Lumingyuan Tang, Saizhuo Wang, Chen Lin, Yeyun Gong, Heung-Yeung Shum, and Jian Guo. 2023. Think-on-graph: Deep and responsible reasoning of large language model with knowledge graph. *arXiv preprint arXiv:2307.07697*.

Dharmesh Tailor, Aditya Patra, Rajeev Verma, Putra Manggala, and Eric Nalisnick. 2024. Learning to defer to a population: A meta-learning approach. In *International Conference on Artificial Intelligence and Statistics*, pages 3475–3483. PMLR.

Jianing Wang, Qiushi Sun, Xiang Li, and Ming Gao. 2023. Boosting language models reasoning with chain-of-knowledge prompting. *arXiv preprint arXiv:2306.06427*.

Nancy XR Wang, Diwakar Mahajan, Marina Danilevsky, and Sara Rosenthal. 2021. Semeval-2021 task 9: Fact verification and evidence finding for tabular data in scientific documents (sem-tab-facts). *arXiv preprint arXiv:2105.13995*.

Zilong Wang, Hao Zhang, Chun-Liang Li, Julian Martin Eisenschlos, Vincent Perot, Zifeng Wang, Lesly Miculicich, Yasuhisa Fujii, Jingbo Shang, Chen-Yu Lee, et al. 2024. Chain-of-table: Evolving tables in the reasoning chain for table understanding. *arXiv preprint arXiv:2401.04398*.

Yike Wu, Nan Hu, Guilin Qi, Sheng Bi, Jie Ren, Anhuan Xie, and Wei Song. 2023. Retrieve-rewrite-answer: A kg-to-text enhanced llms framework for knowledge graph question answering. *arXiv preprint arXiv:2309.11206*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: synergizing reasoning and acting in language models (2022). *arXiv preprint arXiv:2210.03629*.

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.

Tong Zhang, Peixin Qin, Yang Deng, Chen Huang, Wenqiang Lei, Junhong Liu, Dingnan Jin, Hongru Liang, and Tat-Seng Chua. 2024a. CLAMBER: A benchmark of identifying and clarifying ambiguous information needs in large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10746–10766, Bangkok, Thailand. Association for Computational Linguistics.

Zhehao Zhang, Yan Gao, and Jian-Guang Lou. 2024b. 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.

Xu Zhong, Elaheh ShafieiBavani, and Antonio Jimeno Yepes. 2020. Image-based table recognition: data, model, and evaluation. In *European conference on computer vision*, pages 564–580. Springer.

Fengbin Zhu, Wenqiang Lei, Youcheng Huang, Chao Wang, Shuo Zhang, Jiancheng Lv, Fuli Feng, and Tat-Seng Chua. 2021a. TAT-QA: A question answering benchmark on a hybrid of tabular and textual content in finance. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3277–3287, Online. Association for Computational Linguistics.

Fengbin Zhu, Wenqiang Lei, Youcheng Huang, Chao Wang, Shuo Zhang, Jiancheng Lv, Fuli Feng, and Tat-Seng Chua. 2021b. Tat-qa: A question answering benchmark on a hybrid of tabular and textual content in finance. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Association for Computational Linguistics.

A Implementation Details

Considering the financial issue, we experimented with two LLM backbones for all methods: Gemini-1.5 (*gemini-1.5-flash-latest*) and Qwen2 (*Qwen2-72B-Instruct*). To guarantee reproducibility, we set the same temperature and the seed for both backbones (if applicable). Importantly, to mirror real-world scenarios, GraphOTTER is conducted without prior knowledge of the table headers. For baseline methods, cells located at the top or left of the table were assumed to be headers.

A.1 Implementation of Baselines

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. We refer to Appendix A.4.1 for prompts.

Importantly, given a complex table in a document, we cannot definitively determine the location of headers beforehand. The flexible nature of complex tables allows headers to appear in various positions, not solely at the top or left edges. This necessitates alternative solutions beyond manual header cell annotation, as it is not a practical approach (Zheng et al., 2023). To this end, we aim to simulate this real-world scenario by simply removing the header cell information in the prompts of the TableParser (Zhao et al., 2023).

A.2 Implementation of LLM-based Evaluator

While exact match scores are commonly employed in evaluation (Zheng et al., 2023), the flexible nature of LLM outputs makes direct alignment or similarity measures, such as exact match, inadequate for assessing our method and baselines (Zhao et al., 2023). Therefore, we employ an LLM as an evaluator, following the approach of Zhao et al. (2023); Adlakha et al. (2023), to verify the accuracy of predictions made by all methods. Specifically, we utilize Gemini 1.5 (*gemini-1.5-flash-latest*) as the backbone of the LLM evaluator, considering cost-effectiveness. The prompt template used for this evaluation is presented in Appendix A.4.3.

A.3 Implementation of GraphOTTER

A.3.1 Graph Representation Initialization

- **Transforming Table to Graph**. GraphOTTER transforms the input table into an undirected graph, where edges connect cells within the same row or column.

- **Reasoning Trace Initialization**. To filter out irrelevant information and initialize the reasoning trace, we guide the LLM to identify nodes relevant to the given user question, effectively constructing a subgraph. In our implementation, we employ gte-base (`https://huggingface.co/thenlper/gte-base`) to retrieve relevant nodes from the embedding space and the LLM itself to generate nodes through in-context learning. Finally, given the two sets of nodes obtained, we take the intersection and obtain at most 8 most relevant ones. See Figure 10 for prompts and the format of the reasoning trace is illustrated in Figure 14.

A.3.2 Graph Reasoning

We employ three processes to facilitate step-by-step reasoning over the graph via Thought-Action-Update. Prompts for the three-step process are provided in Figure 12 and Figure 11. During the reasoning, we defined four reasoning actions, acting as four functions. Here, we provide the implementation of these actions.

- *VisitNode*(*query*:Str). This function/action serves as an intermediary, taking query from the LLM and accessing the corresponding nodes within the graph. Due to the inherent flexibility of the LLM's output, its responses may not always directly align with the node names in the graph.To address this challenge, we use a hybrid approach that combines keyword matching and semantic retrieval by taking the intersection of their results, with semantic retrieval implemented by gte-base. This approach aims to identify the most accurate nodes based on the LLM's output, ensuring a robust connection between the LLM and the graph structure.

- *GetAllNeighbours*(*node*:Tuple). This function retrieves all neighbors of a specified node. Notably, we limit the LLM to specify the exact name of the visited node that it wants to explore. Moreover, our knowledge of the graph's structure allows us to programmatically and accurately determine the neighbors of any given node.

- *GetSharedNeighbours*(*node1*:Tuple, *node2*:Tuple): This function returns the shared neighbors of two specified nodes in the graph. Same with the previous function, we can accurately identify and return the shared neighbors given our knowledge on the graph.

- *AnswerQuestion*(): Answer the user question based on the available information. When the LLM selects this function, it signifies that the available information is sufficient to answer the question. At this point, we guide the LLM to generate the final answer using CoT prompts. The prompt template is depicted in Figure 13.

A.4 Implementation of Prompts

A.4.1 Prompts for Baselines

- *One-shot implicit reasoning*. TableReasoner: Figure 7; TableParser: Figure 8.

- *Iterative implicit reasoning*. We use their official codes from the Github.

- *Chain-of-Table*. We use the official codes from the Github.

A.4.2 Prompts for GraphOTTER

- *Reasoning trace initialization*. Figure 10

- *Graph Reasoning*. Figure 9, Figure 11, Figure 12, Figure 13, and Figure 14.

A.4.3 Prompts for LLM-based Evaluator

We employ an LLM as an evaluator, following the approach of Zhao et al. (2023); Adlakha et al. (2023), to verify the accuracy of predictions made by all methods. The prompt template is illustrated in Figure 15.

B Dataset Details and Statistics

We consider two benchmark datasets of complex table QA, including AIT-QA and Hitab.

Given a complex table in a document, we cannot definitively determine the location of headers beforehand. The flexible nature of complex tables allows headers to appear in various positions, not solely at the top or left edges. This necessitates alternative solutions beyond manual header cell annotation, as it is not a practical approach (Zheng et al., 2023). In this case, our experiments follows this real-world setting and removes the annotated table header information from the both datasets.

As for the AIT-QA, its header cells and data cells are stored separately in two-dimensional lists, recording only the content of the cells without preserving their positional information. An example of a table storage format is shown below. When the number of header cells in each column of the $column\_header$ and row_header lists is consistent, it is possible to reconstruct a table with an unknown header cell structure by following a top-to-bottom, left-to-right order. However, the number of header cells in the last column of the $column\_header$ list (5) is inconsistent with the number of header cells in other columns (2), making it impossible to determine the specific position of each header cell in the last column. Consequently, the reconstruction of a table with an unknown header cell structure is not feasible. Therefore, we removed tables like this where we cannot eliminate the influence of the known headers. Finally, the statistics for the test sets of the two datasets are presented in Table 6.

```
# An example of a table storage format in AIT-QA
{
    "id": "tab-2", # The Table ID.
    "column_header":  # A list of column names in the table.
          Column names can be hierarchical and sublist
         captures the order of hierarchy.
    [
        [...],
        [
            "Increase (decrease) from 2017 (a):",
            "Latin"
        ],
        [
            "Increase (decrease) from 2017 (a):",
            "Total"
        ],
        [
            "",
            "Increase (decrease) from 2017 (a):",
```

Model	Gemini 1.5		Qwen2	
	HRQ	HUQ	HRQ	HUQ
One-shot Implicit Reasoning				
TableReasoner	**71.05**	63.92	73.68	76.98
TableParser	63.16	63.23	65.79	64.60
Iterative Implicit Reasoning				
ReAct	44.74	47.77	60.53	70.10
E5	61.84	58.76	52.63	57.39
MIX-SC	56.58	73.45	**82.89**	84.54
Iterative Explicit Reasoning				
Chain-of-Table	25.00	16.84	48.68	49.48
Ours	69.74	**84.54**	81.58	**90.03**

Table 5: Detailed evaluation on different user questions, categorized by whether their answers are related to the table headers or not. "HRQ" represents "Header-related questions", and "HUQ" represents "Header-unrelated questions". Here, EM is used.

Dataset	Table number	QA Pair	Domain
Hitab	538	1,584	Open domain
AIT-QA	80	367	Airline industry

Table 6: Test set statistics for the two datasets.

```
                "",
                "Domestic",
                "Atlantic"
            ]
        ],
        "row_header":  # A list of row headers in the table. Row
              headers can be hierarchical and sublist captures
            the order of hierarchy.
        [
            [...],
            [
                "Passenger load factor (points)"
            ],
            [
                "11"
            ]
        ],
        "data":  # A list of rows. Each row is a list of row
            entries.
        [
            [...],
            [
                "",
                "",
                "",
                "",
                "",
                "(a) See Part II, Item 6, Selected Financial
                      Data, of this report for the definition of
                      these statistics."
            ]
        ]
}
```

C Case Study of GraphOTTER Reasoning

For better understanding, we involve a case study to reveal the reasoning process of GraphOTTER. The examples of the user question and the complex table from the Hitab dataset are illustrated in Figure 6. The reasoning process of GraphOTTER is provided in Table 7, which details each reasoning step.

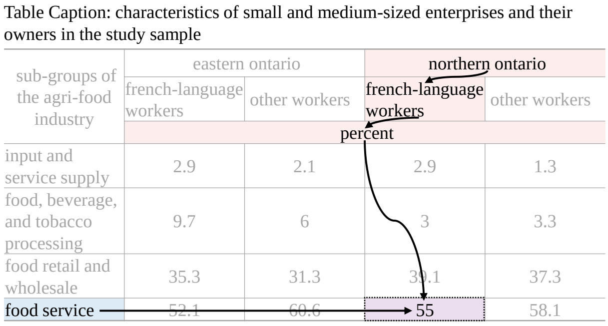


Figure 6: Illustration of the graph reasoning of GraphOTTER. **User Question**: in northern ontario, what proportion of workers who have worked in the restaurant and food services sector was french-language workers?

User Question of Case Study: in northern ontario, what proportion of workers who have worked in the restaurant and food services sector was french-language workers?

D Other Experimental Findings

According to Table 2, GraphOTTER demonstrates balanced performance across both Exact Match (EM) and LLM Evaluation metrics, unlike most baselines where LLM Evaluation scores significantly surpass EM, particularly for direct prompt methods. This balanced performance, achieved through GraphOTTER's graph representation and reasoning framework, signifies the generation of answers that are both factually accurate and adhere to the desired format, a crucial factor in factoid question answering.

Additionally, Table 5 provides a detailed evaluation of the EM metric for different user questions, categorized based on whether their answers are related to the table headers. The results demonstrate that GraphOTTER performs robustly across both question types, underscoring the effectiveness of our self-inferred headers. Also, Tables 3 and 5 show that many methods perform better on questions that do not rely heavily on table headers, possibly because LLMs are typically pretrained on simple tables, which limits their ability to understand flexible table headers in complex tables. Additionally, biases inherent to attention mechanisms(Zhao et al., 2023) may hinder the ability of LLMs to effectively interpret table header data.

Step ID	**Reasoning Step or Action**	**Current Reasoning Trace** (in the form of subgraph)
0	*Graph Representation Initialization* via finding cells that relevant to the user question.	[(1, 3, 'french-language workers'), (6, 0, 'food service')] has the following shared neighbors: [(1, 0, 'sub-groups of the agri-food industry'), (6, 3, '55.0')]. [(6, 0, 'food service'), (0, 3, 'northern ontario')] has the following shared neighbors: [(0, 0, 'sub-groups of the agri-food industry'), (6, 3, '55.0')]. [(0, 3, 'northern ontario'),SAME COLUMN,(1, 3, 'french-language workers')]
1	**Thought**: The question asks for the percentage of French-language workers in the "food services" sector in Northern Ontario. The graph includes both "Northern Ontario" and "Food Services," but the percentages are not provided, so we need to find them in the same column. **Action**: GetAllNeighbours((1, 3, "french-language workers")) *Action Output*: Same row: [(1, 0, 'sub-groups of the agri-food industry'), (1, 1, 'french-language workers'), (1, 2, 'other workers'), (1, 3, 'french-language workers'), (1, 4, 'other workers')]. Same column: [(0, 3, 'northern ontario'), (1, 3, 'french-language workers'), (2, 3, 'percent'), (3, 3, '2.9'), (4, 3, '3.0'), (5, 3, '39.1'), (6, 3, '55.0')]. **Update**: No update	Same as previous one
2	**Thought**: We found the nodes for "Francophone Workers" and "Percentage" in the "Northern Ontario" column. Shared neighbors of 'Percentage' and 'Food Services' (6, 3, '55.0') suggest that 55.0 is the percentage of French-speaking workers in the 'Food Services' subgroup in Northern Ontario. **Action**: VisitNode('Percentage') *Action Output*: None **Update**: Add node 'percent'	[(1, 3, 'french-language workers'), (6, 0, 'food service')] has the following shared neighbors: [(1, 0, 'sub-groups of the agri-food industry'), (6, 3, '55.0')]. [(6, 0, 'food service'), (0, 3, 'northern ontario')] has the following shared neighbors: [(0, 0, 'sub-groups of the agri-food industry'), (6, 3, '55.0')]. [(6, 0, 'food service'), (2, 3, 'percent')] has the following shared neighbors: [(2, 0, 'sub-groups of the agri-food industry'), (6, 3, '55.0')]. [(0, 3, 'northern ontario'),SAME COLUMN,(1, 3, 'french-language workers'),SAME COLUMN,(2, 3, 'percent')]
3	**Thought**: We found the nodes for "Francophone Workers" and "Percentage" in the "Northern Ontario" column. Shared neighbors of 'Percentage' and 'Food Services' (6, 3, '55.0') suggest that 55.0 is the percentage of French-speaking workers in the 'Food Services' subgroup in Northern Ontario. **Action**: GetSharedNeighbours((2, 3, 'percent'), (6, 0, 'food service')) *Action Output*: [(2, 0, 'sub-groups of the agri-food industry'), (6, 3, '55.0')] **Update**: No update	Same as previous one
4	**Thought**: We found the nodes for "Francophone Workers" and "Percentage" in the "Northern Ontario" column. Shared neighbors of 'Percentage' and 'Food Services' (6, 3, '55.0') indicate that 55.0 is the percentage of French-speaking workers in the 'Food Services' subgroup in Northern Ontario. **Action**: AnswerQuestion()	Same as previous one
5	**Answer Generation**: 55.0	

Table 7: Reasoning process of GraphOTTER. *User Question*: in northern ontario, what proportion of workers who have worked in the restaurant and food services sector was french-language workers? *Answer*: 55.

```
Read the table below regarding "{table_caption}" to answer the following questions.
The output format should be "The answer is xxx, yyy, zzz". If the answer contains multiple
segments of text, please separate them with commas(",").

{table}

read the table first, and then answer the given question.

Question: {question}
```

Figure 7: Prompt for TableReasoner

```
Suppose you are an expert in statistical analysis.
You will be given a table described in a special format.
Your task is to answer the questions based on the content of the table.

The table is described as follows:
1. The title means the title of the table.
2. We represent cell tuples as (C1, C2, C3), where C1 denotes the row, C2 denotes the
column, and C3 denotes the content.

For examples:
The tuple (7, 0, 416) represents a cell at row 7, column 0, with a value of 416. Make sure
you read and understand these instructions carefully.

Let's think step by step as follows and give full play to your expertise as a statistical
analyst:
1. Clearly understand the question and the information needed to answer the question to
determine the necessary information to extract.
2. Have a comprehensive understanding of the data in the table, including the meaning, data
 types, and formats of each column and row tuples (Note: There are usually summative tuples
 in the table, such as all, combine, total, sum, average, mean, etc. These tuples help you
skip a lot of operations).
3. Perform statistical, calculation, sorting, grouping, or other operations on the tuples
you selected before to extract useful information based on the question's requirements.

You MUST answer each question in the format below line by line (Note: Keep your answer
concise):
1. Cell: The cell tuples most relevant to the answer.
2. Operation: the operation you performed on the tuples you selected.
3. Answer: your answer (A number, noun, phrase, or set of data).
And if the answer is not contained within the context, say "I don't know".

Title: {TABLE_TITLE_HERE}

Cells: {TABLE_NON_HEADER_HERE}

Q: {QUSTION_HERE}
A:
```

Figure 8: Prompt for TableParser

```
Suppose you are an expert in statistical analysis.
You will be given a table described in a special format.
Your task is to answer the Question based on the content of the table.

Graph Definition: We consider each cell in the table as a node in the graph, represented by
 the tuple (Row Index, Column Index, Cell Content). For example, (1, 0, "test1") represents
 the node in the 1st row and 0th column, with content "test1". The tuples (1, 0, "test1"),
(1, 4, "test2"), and (3, 0, "test3") represent three nodes, where "test2" and "test1" have
a SAME ROW relationship, "test3" and "test1" have a SAME COLUMN relationship, and "test1"
is the shared neighbor of "test2" and "test3".
```

Figure 9: System instruction for GraphOTTER's graph reasoning

```
====================================  System Instruction ===================================
Suppose you are an expert in statistical analysis.
You will be given a Table described in a special format.
Your task is to identify the cells in the Table that is most relevant to the Question.

Each cell in the Table is represented by a tuple (Row Index, Column Index, Cell Content).
For example, the tuple (7, 0, "416") represents a cell at row 7, column 0, with a value of
"416".
Make sure you read and understand these instructions carefully.

=========================================  Prompt =========================================
Let's think step by step as follows and give full play to your expertise as a statistical
analyst:
1. **Understand the Question**: Clearly understand the Question and the information needed
to answer the Question to determine the necessary information to extract.
2. **Analyze the Data Structure**: Have a comprehensive understanding of the data in the
Table, including the meaning, data types, and formats of each cell tuples.
3. **Select Relevant Data**: Based on the Question, identify the most relevant cell tuples.
 **Note:** Pay special attention to the header cell tuples in the Table, as they are often
more relevant to the Question's semantics and can help in identifying the related evidence
cell tuples.

{examples}

{table}

**Question:** {question}

Output format instructions:
1. Outputs cell tuples in descending order of relevance.
2. Using this JSON schema: Tuple = {"tuple": tuple, "explanation": str}.  Return a `list[
Tuple]`.
```

Figure 10: Prompt and system instruction for GraphOTTER's reasoning trace initialization. Subsequently, we identify the intersection of the results generated by this prompt and the results returned by the gte-base retriever.

```
You can interact with the graph through two steps: "Thought" and "Action", to complete the
question answering task step by step:
1. In the "Thought" step, thoroughly examine the question and the existing data. Determine
if the current data is sufficient to answer the question:
    a. If the existing information is sufficient, proceed the "Action" step and call the "
    AnswerQuestion" function to give the answer.
    b. If more information is needed, call the functions in the 'Action' step to obtain
    useful information.
2. In the "Action" step, you can call the following functions to get more node information
from the graph :
    a. VisitNode(query): Retrieve the node from the graph that is semantically closest to
    the keyword 'query'(given as a str). Note: 'query' cannot be the known Cell Content, to
     avoid meaningless calls;
    b. GetAllNeighbours(node): Get all neighboring nodes in the same row and column of the
    specified node (given as a tuple) from the graph.
    c. GetSharedNeighbours(node1, node2): Get all shared neighbors between two specified
    nodes (also represented as tuples) in the graph.
    d. AnswerQuestion(): Answer the question based on the available information.

Let's think step by step as follows and give full play to your expertise as a statistical
analyst:
1. **Understand the Question**: Clearly understand the Question, clarify the relationships
between the existing data, and organize the information needed to answer the Question.
2. **Analyze the Data Structure**: Have a comprehensive understanding of the data in the
graph, including the meaning, data types, and formats of each nodes. **Note:** 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 nodes help you skip a lot of operations.
3. **Select Relevant Data**: Based on the Question, identify the most relevant nodes. (**
Note:** Some cells may have identical Cell Content, so avoid greedy searches when necessary
; Focus on the most relevant and directly related data to the Question at hand.)
4. **Avoid Redundant Calculations**: Before performing any calculations or operations,
first check if the needed information is already available in the graph. If so, directly
use this information.

Table: {Table}

Question: {Question}

{Reasoning Trace}

{Interaction History}

Please integrate all the current information to output your Thought step {step}. The output
 format is: "Thought step {step}: {your thought}"
```

Figure 11: Thought Step Prompt of GraphOTTER. Note that the Update Step does not rely on prompt.

```
You can interact with the graph through two steps: "Thought" and "Action", to complete the
question answering task step by step:
1. In the "Thought" step, thoroughly examine the question and the existing data. Determine
if the current data is sufficient to answer the question:
    a. If the existing information is sufficient, proceed the "Action" step and call the "
    AnswerQuestion" function to give the answer.
    b. If more information is needed, call the functions in the 'Action' step to obtain
    useful information.
2. In the "Action" step, you can call the following functions to get more node information
from the graph :
    a. VisitNode(query): Retrieve the node from the graph that is semantically closest to
    the keyword 'query'(given as a str). Note: 'query' cannot be the known Cell Content, to
     avoid meaningless calls;
    b. GetAllNeighbours(node): Get all neighboring nodes in the same row and column of the
    specified node (given as a tuple) from the graph.
    c. GetSharedNeighbours(node1, node2): Get all shared neighbors between two specified
    nodes (also represented as tuples) in the graph.
    d. AnswerQuestion(): Answer the question based on the available information.

Table: {Table}

Question: {Question}

{Reasoning Trace}

{Interaction History}

Based on the results of your previous Thought step {step}, output your Action Step and
Explanation.
Using this JSON schema: ActionStep = {"Function" : {"function_name": str, "parameters":
list[tuple] | list[str] }, "Explanation": str}. Return a `list[ActionStep]`.
```

Figure 12: Action Step Prompt of GraphOTTER. Note that the Update Step does not rely on prompt.

```
Let's think step by step as follows and give full play to your expertise as a statistical
analyst:
1. **Understand the Question**: Clearly understand the Question, clarify the relationships
between the existing data, and organize the information needed to answer the Question.
2. **Analyze the Data Structure**: Have a comprehensive understanding of the data in the
graph, including the meaning, data types, and formats of each nodes. **Note:** 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 nodes help you skip a lot of operations.
3. **Select Relevant Data**: Based on the Question, identify the most relevant nodes.
4. **Avoid Redundant Calculations**: Before performing any calculations or operations,
first check if the needed information is already available in the graph. If so, directly
use this information.
5. **Synthesize the Answer**: Use the selected data to construct a clear and concise answer
. Ensure that the final answer directly addresses the question, using the most relevant and
 accurate data from the graph.

You MUST answer each question step by step as follows(Note: Keep your answer concise):
1. Cell: The nodes most relevant to the answer.
2. Operation: the operation you performed on the nodes you selected.
3. Explanation: your explanation.
4. Answer: your final answer.
And if you need to extract relevant Cell Content from the graph as answer, do not add any
units, symbols, or other explanatory text. Ensure that the extracted Content matches the
original Cell Content in the graph exactly.

Table: {table}

Question: {Question}

{Reasoning Trace}

{Interaction History}

Please integrate all the current information to output your answer.
Using this JSON schema: Answer = {"cells" : list[str], "operation": str, "explanation": str
, "answer": list[str] }. Return a `Answer`.
```

Figure 13: Answer Generation Prompt of GraphOTTER

```
==================================== Reasoning Trace ======================================
The reasoning trace below displays the neighbor relationships of the nodes/steps from your
interaction history. This trace will dynamically update during your interaction process.
**Reasoning Trace:**
[(4, 4, '37.7'), (0, 2, 'as a percent of total female population in each cma/municipality')
] has the following shared neighbors: [(0, 4, 'largest visible minority group as a percent
of total female population in each cma/municipality'), (4, 2, '66.5')].
[(4, 0, 'brampton'), (0, 4, 'largest visible minority group as a percent of total female
population in each cma/municipality')] has the following shared neighbors: [(0, 0, '
municipalities'), (4, 4, '37.7')].
[(4, 0, 'brampton'), (0, 2, 'as a percent of total female population in each cma/
municipality')] has the following shared neighbors: [(0, 0, 'municipalities'), (4, 2,
'66.5')].
[(4, 0, 'brampton'),SAME ROW,(4, 4, '37.7')]
[(0, 4, 'largest visible minority group as a percent of total female population in each cma
/municipality'),SAME COLUMN,(4, 4, '37.7')]
[(0, 2, 'as a percent of total female population in each cma/municipality'),SAME ROW,(0, 4,
 'largest visible minority group as a percent of total female population in each cma/
municipality')]

==================================  Interaction History ====================================
The steps below represent your interaction history on the graph. Please refer to them when
selecting your next function.
Step 1:["GetSharedNeighbours((4, 0, 'brampton'), (0, 2, 'as a percent of total female
population in each cma/municipality'))"]

Thought step 1: The question asks for the percentage of the total female population in
Brampton, where South Asian women and girls were the largest visible minority group. The
nodes (4, 4, '37.7') and (4, 0, 'brampton') are relevant to the answer, as well as the
shared neighbors (0, 4, 'largest visible minority group as a percent of total female
population in each cma/municipality') and (0, 2, 'as a percent of total female population
in each cma/municipality'). The percentage of the total female population in Brampton is
the information needed to answer the question.
Action Step 1:
    1. Function: GetSharedNeighbours((4, 0, 'brampton'), (0, 2, 'as a percent of total
    female population in each cma/municipality')), Explanation: To find the percentage of
    the total female population in Brampton, we need to get the shared neighbors of the
    nodes (4, 0, 'brampton') and (0, 2, 'as a percent of total female population in each
    cma/municipality').
Observation Step 1:
    1. The result of GetSharedNeighbours((4, 0, 'brampton'), (0, 2, 'as a percent of total
    female population in each cma/municipality')) is: [(4, 0, 'brampton'), (0, 2, 'as a
    percent of total female population in each cma/municipality')] has the following shared
     neighbors: [(0, 0, 'municipalities'), (4, 2, '66.5')].
```

Figure 14: A example of Reasoning Trace and Interaction History of GraphOTTER

```
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 15: LLM Evaluation Prompt