

Beyond Linearization: Attributed Table Graphs for Table Reasoning

Yuxiang Wang¹, Junhao Gan¹, Shengxiang Gao¹
Shenghao Ye², Zhengyi Yang³, Jianzhong Qi^{1,*}

¹The University of Melbourne ³The University of New South Wales

²The University of Science and Technology of China

{yuxiang.wang8, shengxiang.gao1}@student.unimelb.edu.au

{junhao.gan, jianzhong.qi}@unimelb.edu.au

Abstract

Table reasoning, a task to answer questions by reasoning over data presented in tables, is an important topic due to the prevalence of knowledge stored in tabular formats. Recent solutions use Large Language Models (LLMs) for their semantic understanding and reasoning capabilities. A common paradigm of such solutions linearizes tables to form plain texts that are served as input to LLMs. This paradigm has critical issues. It requires LLMs to infer row-column-cell relations from serialized inputs, makes evidence paths harder to trace, and is subject to the “lost-in-the-middle” issue. To address these issues, we propose Table Graph Reasoner (TABGR), a model that represents tables as an Attributed Table Graph (ATG) without task-specific training. The ATG explicitly preserves row-column-cell structure while enabling graph-based reasoning over traceable evidence paths. We further propose a Question-Guided Personalized PageRank (QG-PPR) mechanism to rerank tabular data and mitigate the lost-in-the-middle issue. Extensive experiments across multiple table reasoning benchmarks show that TABGR consistently outperforms state-of-the-art models by up to 9.7% in accuracy. Our code is available at: <https://github.com/yxw-11/TabGR>

1 Introduction

Table reasoning aims to answer questions by reasoning over data presented in tables. Its importance comes from the prevalence of knowledge stored in tabular formats (Yu et al., 2025).

Recent studies leverage the semantic understanding and reasoning capabilities of Large Language Models (LLMs) for table reasoning. They typically follow two paradigms: (i) *decomposition-based reasoning*, which generates executable code or conducts iterative symbolic operations to decompose

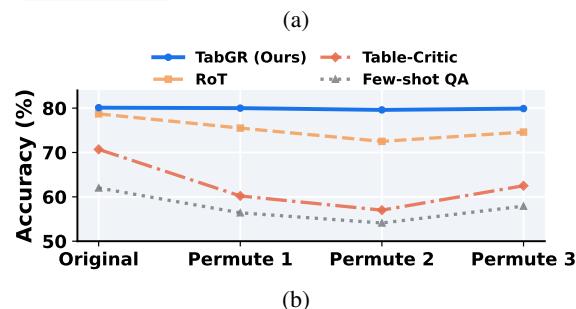
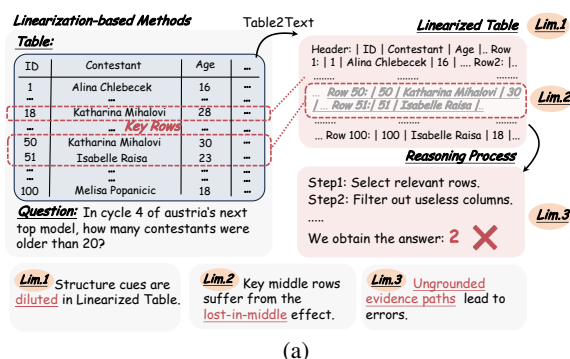


Figure 1: Motivating example. (a) Limitations of linearization-based methods. (b) Performance of different methods on WikiTableQuestions (Pasupat and Liang, 2015) under random table permutations.

tables, and then reasons over the decomposed sub-tables (Cheng et al., 2023; Ye et al., 2023; Wang et al., 2024, 2025b; Yu et al., 2025; Zhang et al., 2025a), and (ii) *full-table reasoning*, which applies LLMs to reason over full tables directly (Zheng et al., 2023; Liu et al., 2024b; Zhang et al., 2025b). Both paradigms linearize tables when feeding them into LLMs (Yin et al., 2020), treating tables as flat sequences (e.g., Markdown).

While table linearization is simple and has been shown to be effective with existing solutions, we observe a few inherent limitations with such a table representation strategy, as illustrated by Figure 1.

(1) Sequential structure dilution: Linearized formats encode table structure through textual markup, but structural cues may dilute and are less direct in token sequences than in row-column-cell

*Corresponding author

reasoning units (Sui et al., 2024). Even with layout cues such as separators and newlines (cf. Figure 1a), or richer formats such as HTML or $\text{\LaTeX}$, LLMs need to interpret these bindings from sequences, causing grounding errors in challenging tables; examples and comparisons are in Appendix K. Existing structure-aware methods often require architectural adaptation or additional training (Yin et al., 2020; Su et al., 2024), incurring substantial costs.

(2) Sensitivity to answer positions: Linearizing tables into a long sequence is vulnerable to the positions of the answers in table reasoning tasks. LLMs are known to be subject to the “lost-in-the-middle” problem (Liu et al., 2024a), i.e., they tend to miss information given in the middle of a long input. As shown in Figure 1b, we run two state-of-the-art (SOTA) methods, Table-Critic (Yu et al., 2025) and RoT (Zhang et al., 2025b), and a simple few-shot LLM prompting method on three random permutations of the table rows and columns of the WikiTableQuestions dataset (Pasupat and Liang, 2015). The table reasoning accuracy fluctuates with each permutation.

(3) Difficult to ground the reasoning process: Linearizing tables often constrains LLMs to plain-text reasoning, where evidence is serialized text rather than specific rows, columns, or cells. As existing methods typically perform reasoning (e.g., Chain-of-Thought (Zheng et al., 2023)) over serialized tables, their reasoning steps are not explicitly grounded in individual row-column-cell relations, making the supporting evidence harder to trace.

To address these limitations, we propose Table Graph Reasoner (TABGR), a novel table reasoning method that introduces an Attributed Table Graph (ATG) for table representation. In the ATG, rows and cells of a table are represented as the graph nodes, while the graph edges encode the column relations. This representation allows LLMs to access row-column-cell relations as graph triples.

Based on the ATG, we propose a Question-Guided Personalized PageRank (QG-PPR) (Jeh and Widom, 2003; Brin and Page, 2012) mechanism to assign question- and structure-aware salience scores to each graph triple by inducing a propagation matrix, and we rerank triples according to their importance. This ensures that crucial information is surfaced to the LLM regardless of its original position in the table.

Besides, graph-based table reasoning allows LLMs to reason over explicit structural relations by identifying salient subgraphs, tracing reasoning

paths, and revealing which row-column-cell connections support the final prediction, making the evidence path easier to inspect.

Our contributions are summarized as follows:

(1) We propose TABGR, a table reasoning model that requires no task-specific training and represents tables as an Attributed Table Graph, explicitly preserving the row-column-cell structure for more effective reasoning.

(2) We introduce a Question-Guided Personalized PageRank mechanism that assigns salience scores to table (i.e., graph) triples, enabling question- and structure-aware triple reranking and hence improving robustness to answer position in tables (a.k.a. the “lost-in-the-middle” issue).

(3) We transform table reasoning into graph-based reasoning, enabling LLMs to produce reasoning paths grounded in row-column-cell triples, making the supporting evidence easier to trace.

(4) Extensive experiments on Table Question Answering (TableQA) and Table Fact Verification (TableFV) benchmarks demonstrate that TABGR not only consistently outperforms SOTA methods in accuracy but also exhibits stronger robustness to permutations of table contents.

2 Related Work

Table reasoning requires models to comprehend both the content and structural information of tables, to support tasks such as TableQA and TableFV. Existing methods differ primarily in how they represent tables and reason over the represented format.

Linearization-based Table Reasoning Most existing methods for TableQA and TableFV follow a “linearize-then-reason” paradigm, converting tables into linearized sequences (e.g., in Markdown format) before feeding them into LLMs. The methods can be further categorized into two groups.

Decomposition-based reasoning methods generate executable code or symbolic operations to decompose tables and questions. For example, Dater (Ye et al., 2023) decomposes tables and questions via LLM-generated programs. Binder (Cheng et al., 2023) iteratively refines executable programs with LLM feedback. TableRAG (Chen et al., 2024) generates programs to retrieve relevant table cells and columns before reasoning. Chain-of-Table (Wang et al., 2024) performs step-wise reasoning by iteratively applying predefined symbolic operations to derive sub-tables. Table-Critic (Yu et al., 2025) improves this pipeline by introducing

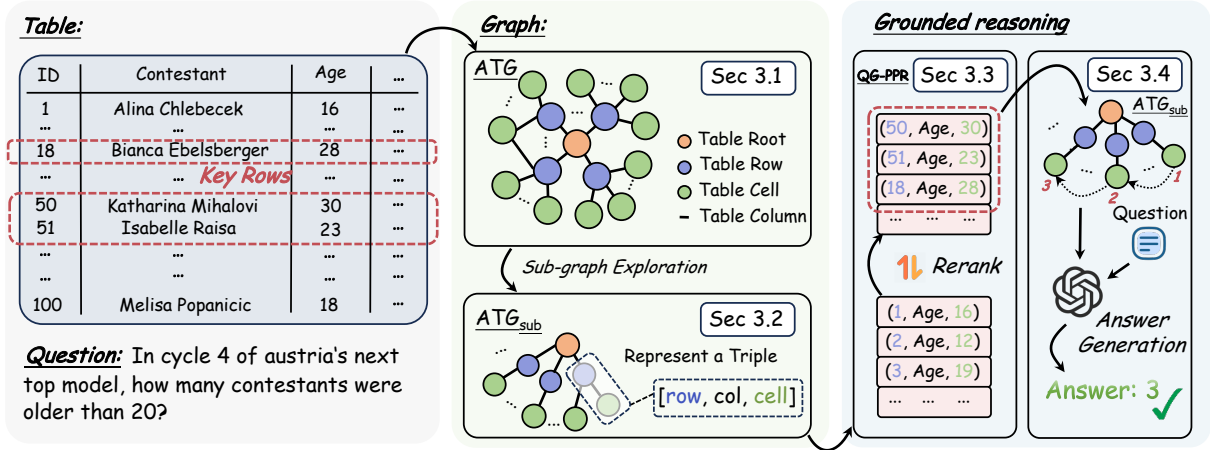


Figure 2: Overview of TABGR, a graph-based framework for table reasoning. The input table is first transformed into an Attributed Table Graph (ATG), which supports Question-Guided Personalized PageRank (QG-PPR) for estimating the salience of table triples and reranking evidence. The reranked evidence then guides an LLM to perform reasoning over the ATG, producing a traceable evidence path grounded in table triples.

intermediate critiques to refine each reasoning step. This line of work can reduce long-context exposure through execution, while TABGR instead explicitly keeps table structure during reasoning.

Full-table reasoning methods utilize LLMs to reason over full tables. For example, TaCo (Zheng et al., 2023) performs Chain-of-Thought (CoT) (Wei et al., 2022) reasoning directly over fully linearized tables. RoT (Zhang et al., 2025b) improves CoT prompting by performing row-wise traversal reasoning using fully linearized tables.

Linearization-based table representations are simple and intuitive. However, they encode row-column-cell bindings through serialized markup and create long input contexts, where LLMs may struggle to locate answers hidden in the middle (i.e., the lost-in-the-middle issue (Liu et al., 2024a)).

Graph Modeling for Tabular Tasks Graph structures have been explored to capture dependencies in tabular data. For example, T-RAG (Zou et al., 2025) employs hypergraphs for *table retrieval* in multi-table settings, while ODYSSEY (Agarwal et al., 2025) utilizes hybrid graphs to link tabular cells with external unstructured text for *multi-modal linking*. These methods still rely on *linearized tables* at inference time. GraphOTTER (Li et al., 2025) targets *complex table layout understanding* by modeling irregular tables as undirected graphs and performs predefined reasoning actions to traverse the graphs. Its graph representation primarily captures local connections through cell values and their positional indices, lacking explicit global row-column-cell structure,

while predefined reasoning actions may limit generalizability. Tunes (Nguyen et al., 2025) adopts an entity-centric framework by using LLMs to identify a primary key and infer entity-attribute relations before constructing a graph. Retrieval and reasoning operate at the entity level through hybrid search, where queries are matched to pre-identified entity nodes that are defined by table structure rather than conditioned on the input question, which may limit flexibility across diverse queries.

Different from these methods, TABGR represents tables as an ATG, which maintains row-column-cell connections and enables a QG-PPR mechanism. By deriving salience scores from the ATG rather than plain-text, TABGR can prioritize critical triples regardless of their original positions, thereby mitigating the lost-in-the-middle issues. Meanwhile, TABGR creates reasoning paths aligned with table topology, making the supporting evidence easier to trace.

3 TABGR

We consider an input table $\mathcal{T}$ and an input question Q (in plain-text) that we aim to answer through reasoning over $\mathcal{T}$.

As Figure 2 shows, our Table Graph Reasoner (TABGR) first transforms $\mathcal{T}$ into an Attributed Table Graph (ATG, Section 3.1). The ATG explicitly encodes the row-column-cell structure of $\mathcal{T}$ and serves two purposes: (1) It enables LLMs to perform graph-based table decomposition (Section 3.2), allowing TABGR to iteratively zoom in to the question-relevant elements of $\mathcal{T}$. (2) It in-

duces a sparse propagation matrix for a Question-Guided Personalized PageRank (QG-PPR) mechanism (Section 3.3). By deriving salience scores from the ATG, TABGR reranks graph triples to prioritize the question-critical ones, hence mitigating the “lost-in-the-middle” issue. The reranked triples are fed into an LLM for final answer generation (Section 3.4). This processing pipeline is summarized as Algorithm 1 in Appendix B.

3.1 Attributed Table Graph

We represent a table $\mathcal{T}$ as an attributed table graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where the node set $\mathcal{V}$ contains three types of nodes: (1) cell value nodes $c_j^{(k)}$, each corresponding to the k -th unique value in column col_j ; (2) row anchor nodes r_i , each representing a table row; and (3) a root node representing the entire table and connected to all row anchor nodes, mainly used to maintain graph connectivity. Each cell value node $c_j^{(k)}$ is connected to all row anchor nodes whose rows contain this value in column col_j , and each edge $(r_i, c_j^{(k)}) \in \mathcal{E}$ is annotated with the column header h_j . Importantly, identical values appearing in different columns are treated as distinct nodes and are not merged across columns.

Accordingly, table $\mathcal{T}$ can be viewed as a collection of structured triples defined at the cell level:

$$\langle r_i, h_j, c_{i,j} \rangle, \quad i \in [1, R], j \in [1, C], \quad (1)$$

assuming R rows and C columns in $\mathcal{T}$, and $c_{i,j}$ denotes the cell value in row r_i and column col_j , corresponding to the value node $c_j^{(k)}$ in ATG if the value appears multiple times in the same column.

Constructing $\mathcal{G}$ takes $O(R \cdot C)$ expected time using hashing, where $R \cdot C$ accounts for visiting each cell once in $\mathcal{T}$, and checking value-node uniqueness takes $O(1)$ time during graph construction. This process and any subsequent graph updates are not question-dependent and can be done offline if $\mathcal{T}$ is given beforehand, where the resulting graph occupies $O(R \cdot C)$ space.

3.2 ATG-Based Table Decomposition

TABGR traverses $\mathcal{G}$ to construct a grounded reasoning subgraph that is meant to contain task-relevant structural units (i.e., triples).

The traversal (i.e., reasoning) process starts by identifying *anchor triples*. These include (i) triples whose cell values $c_{i,j}$ *exactly* match a sub-string of the input question $\mathcal{Q}$, and (ii) triples whose edge attribute h_j is considered relevant to $\mathcal{Q}$ by an LLM.

We then iteratively expand the subgraph (i.e. extracted triples), using an LLM *judge* to determine if the current subgraph is sufficient to answer $\mathcal{Q}$. If not, TABGR iteratively selects more columns (multiple columns are allowed for each step), and it adds triples matching the selected columns to the extracted subgraph. Otherwise, the traversal is terminated, and the resulting subgraph, denoted by $\mathcal{G}^*$, is passed to the next step for triple reranking.

3.3 Question-Guided Personalized PageRank

Next, we present our QG-PPR mechanism to rerank triples in $\mathcal{G}^*$, such that the question-critical triples will surface at the top, to catch the attention of LLMs when fed into LLMs for reasoning and answer generation. QG-PPR operates on a triple-level sparse propagation matrix induced from the ATG, where each triple connects only within its row or column, yielding the degree of each triple to be upper bounded by $R + C$. Let n denote the number of triples. Thus, each power iteration of QG-PPR takes $O(n(R + C))$ expected time.

Triple Salience Scores We compute triple-level salience scores using QG-PPR, which is an adapted Personalized PageRank process, with the following iterative update:

$$\mathbf{s}^{(t+1)} = \alpha \mathbf{p}_0 + (1 - \alpha) \hat{A}^\top \mathbf{s}^{(t)}, \quad (2)$$

where $\mathbf{s}^{(t)} \in \mathbb{R}^{n \times 1}$ denotes the salience scores at iteration t , $\mathbf{p}_0$ is a *question-guided personalization vector* (detailed next), $\hat{A}$ is the triple-level sparse propagation matrix (detailed next), and $\alpha \in (0, 1)$ is the teleport probability (Brin and Page, 2012).

We initialize $\mathbf{s}^{(0)}$ with a uniform distribution and apply power iteration (K times) until convergence. In practice, we run 20 iterations, which is sufficient to obtain stable salience scores in our experiments.

Personalization Vector We define a personalization vector $\mathbf{p}_0$, which encodes the question-specific importance of each triple and serves as the restart distribution in Personalized PageRank.

Given a triple $\langle r_i, h_j, c_{i,j} \rangle$, we denote by $p_0(i, j)$ the entry in $\mathbf{p}_0$ associated with this triple. We initialize $\mathbf{p}_0$ using question-aware signals from both exact string matching and an LLM-based semantic selector. We construct the key column name set $\mathcal{H}_q$ and key cell value set $\mathcal{C}_q$ by taking the union of (i) columns/cell values that exactly match a sub-string of $\mathcal{Q}$ and (ii) additional columns/cell values selected by the LLM based on semantic relevance.

All matched candidates are deduplicated before scoring to avoid repeated counting.

Each triple receives an additive initial score:

$$p_0(i, j) = v_{\text{col}} \cdot \mathbb{I}(h_j \in \mathcal{H}_q) + v_{\text{val}} \cdot \mathbb{I}(c_{i,j} \in \mathcal{C}_q), \quad (3)$$

where $\mathbb{I}(\cdot)$ denotes the indicator function. The entries of the personalization vector $\mathbf{p}_0$ are then normalized such that $\sum_{i,j} p_0(i, j) = 1$, yielding a valid probability distribution over the triples. By default, we set $v_{\text{col}} = 1.0$ and $v_{\text{val}} = 2.0$.

To further down-weight overly frequent cell values and emphasize more informative values, we incorporate an inverse document frequency (IDF) (Jones, 2004) term into the value-level contribution v_{val} by $v_{\text{val}} \cdot \text{IDF}(c_{i,j})$. Formally, for each cell value $c_{i,j}$, we define:

$$\text{IDF}(c_{i,j}) = \log \left(1 + \frac{N}{1 + \text{df}(c_{i,j})} \right), \quad (4)$$

where N is the total number of rows in $\mathcal{T}$ and $\text{df}(c_{i,j})$ is the number of rows containing $c_{i,j}$.

Sparse Propagation Matrix To incorporate connectivity information between the triples, we induce a weighted triple-level sparse propagation matrix $\hat{A} \in \mathbb{R}^{n \times n}$, projecting row- and column-level structural constraints onto transitions between triples. The matrix serves as the transition operator for propagating evidence probability from question-relevant triples to potential triples.

Two triples u and v are connected if they belong to the same row or share the same column header in $\mathcal{G}^*$. Their transition weight is defined as:

$$w_{u,v} = \begin{cases} \frac{w_{\text{row}}}{|\mathcal{C}(r_i)|}, & u, v \in \mathcal{C}(r_i), \\ \frac{w_{\text{col}}}{|\mathcal{C}(col_j)|}, & u, v \in \mathcal{C}(col_j), \end{cases} \quad (5)$$

where $\mathcal{C}(r_i)$ and $\mathcal{C}(col_j)$ denote the set of triples from row r_i and column col_j , respectively, and $|\cdot|$ represents set cardinality. Parameters w_{row} and w_{col} modulate the relative strength of information propagation along the row dimension and column dimension, which satisfies $w_{\text{row}} + w_{\text{col}} = 1$. This constraint guarantees that the sum of transition probabilities from any triple equals 1, preserving numerical stability during the subsequent QG-PPR iterations. By tuning these weights, we can flexibly prioritize row content or column content depending on the question type. Further, normalizing by $|\mathcal{C}|$ ensures that the propagation intensity remains invariant to table size and permutation.

Triple Reranking We use the salience scores s derived from the QG-PPR process to rerank evidence both across rows and within each row. This reranking encourages question-critical triples to be placed at the start of the prompt to be fed into an LLM for reasoning and answer generation, regardless of their positions in the original table $\mathcal{T}$.

For each row r_i , we aggregate salience scores of its triples to obtain a row-level importance score:

$$S(r_i) = \sum_j s_{i,j}, \quad (6)$$

where $s_{i,j}$ denotes the salience score of the triple $\langle r_i, h_j, c_{i,j} \rangle$. The rows are then ranked in descending order of $S(r_i)$.

Within each row, the corresponding triples are further ranked according to their individual salience scores $s_{i,j}$ in descending order, yielding an intra-row ordering of cell-level evidence.

For *order-sensitive* questions, where the answer depends on the inherent order of the table (e.g., “who is the first listed player”), the original triple order is preserved. For all other questions, the reranked triples are provided for the downstream reasoning. By prioritizing evidence based on graph-derived salience rather than original position in the table, TABGR effectively mitigates the “lost-in-the-middle” effect and achieves robust performance across different table permutations.

3.4 Answer Generation

The final stage of TABGR utilizes the reranked subgraph to generate both the final answer and explicit reasoning paths. Given question Q and the reranked subgraph $\mathcal{G}_{\text{ranked}}^*$ (in the form of triples), we prompt an LLM to perform Chain-of-Thought (CoT) (Zheng et al., 2023) reasoning over $\mathcal{G}_{\text{ranked}}^*$.

The LLM is instructed to conduct reasoning by going through the graph triples. This process can be formulated as follows:

$$(P, T, Ans) = \text{LLM}(Q, \mathcal{G}_{\text{ranked}}^*), \quad (7)$$

where $P = \{t_1, t_2, \dots, t_k\}$ represents the reasoning path consisting of the sequence of triples $t_i \in \mathcal{G}_{\text{ranked}}^*$ that has been used to derive the answer; T denotes the CoT text, which provides the logical derivation grounded by path P , and Ans is the final generated answer.

By running the CoT process over the reranked graph triples rather than a linearized, plain-text table, TABGR effectively decouples reasoning from

the table’s original sequence. This enables robustness against table permutations and makes reasoning more traceable, since each step in the path P can be mapped back to triples in G_{ranked}^* .

4 Experiments

4.1 Experimental Setup

Datasets We evaluate our model on five table reasoning benchmarks. We first use two widely adopted datasets, following the recent baseline Table-Critic (Yu et al., 2025): (i) **WikiTableQuestions** (denoted as WikiTQ) (Pasupat and Liang, 2015), a TableQA benchmark with 4,344 test samples from 421 tables; and (ii) **TabFact** (Chen et al., 2020), a table fact verification benchmark with 2,024 test samples from 298 tables. To further evaluate the generalizability of our method under more complex and large-scale table settings, we additionally conduct experiments on: (iii) **HiTab** (Cheng et al., 2022), which features hierarchical tables with multi-level headers and nested structures; (iv) **SLQA** (Wang et al., 2025a), a large-table TableQA benchmark; and (v) **RealHiTBench** (Wu et al., 2025a), a realistic hierarchical-table benchmark that enables evaluation on complex layouts with table structures recovered from serialized inputs. Dataset details are provided in Appendix C, and the results on HiTab and RealHiTBench are reported in Appendix G and H, respectively.

Competitors We compare our method TABGR (both w/ and w/o table decomposition, denoted as **TABGR[†]** and **TABGR**, respectively) with two categories of baseline methods:

(1) Decomposition-based reasoning: This category breaks down tables or questions into sub-components before reasoning, including **Dater** (Ye et al., 2023), **Binder** (Cheng et al., 2023), **TableRAG** (Chen et al., 2024), **Chain-of-Table** (Wang et al., 2024), **Table-Critic** (Yu et al., 2025), and **GraphOTTER** (Li et al., 2025), which have been described in Section 2.

(2) Full-table reasoning: This category reasons over the full tables. **End-to-End QA** directly prompts an LLM with linearized full tables and questions. **Few-shot QA** adds examples (table, question, and the corresponding answer) to the LLM input. **Chain-of-Thought (CoT)** (Wei et al., 2022) directly prompts an LLM to generate step-by-step reasoning over the linearized table. **LLM-Reranker + CoT** first prompts the LLM to reorder rows and columns by question relevance, followed

by CoT reasoning, providing a strong yet simple reranking baseline. **Tunes** (Nguyen et al., 2025) and **RoT** (Zhang et al., 2025b) are also tested, which have been described in Section 2.

Implementation Details We run experiments with three LLM families of different sizes, including both open- and closed-source models: (i) GPT-4o-mini (Hurst et al., 2024) and GPT-5-mini (OpenAI, 2026), (ii) LLaMA3.3-70B-Instruct, LLaMA3.1-70B-Instruct, and LLaMA3.1-8B-Instruct (Dubey et al., 2024), and (iii) Qwen2.5-72B-Instruct (Yang et al., 2024). LLaMA3.3-70B-Instruct is used as the default backbone model across all methods tested in the same experiment.

For all baseline methods, we follow their original settings. For our methods, TABGR[†] uses ATG-based decomposition, while TABGR reasons over the full ATG. We set α to 0.15 and 0.35 for the two modes, respectively (detailed hyper-parameter study is included in Appendix D). For all experiments, we use temperature 0.0, i.e., greedy decoding, to minimize randomness for result reproducibility. The detailed prompts used for our methods are included in Appendix A.

Evaluation Metrics For WikiTQ, HiTab, and SLQA, we use the official WikiTQ evaluator and report accuracy based on exact string matching (Pasupat and Liang, 2015). For TabFact, we report binary classification accuracy (true/false).

4.2 Results

Accuracy Results Table 1 reports method performance across different LLM backbones on both datasets. We make the following observations:

(1) TABGR (with full graph reasoning) consistently outperforms all baseline methods across all three LLMs on both datasets. On average, TABGR achieves 78.8% accuracy on WikiTQ and 93.6% on TabFact, i.e., 2.7% and 1.0% improvements over the strongest baseline, RoT, respectively. McNemar’s exact test (Dror et al., 2018) confirms significant gains over RoT on both WikiTQ and TabFact datasets ($p < 10^{-11}$ and $p < 0.01$).

(2) While TABGR[†] (with ATG-based table decomposition) is not as accurate as TABGR due to information loss during decomposition, it outperforms all decomposition-based baselines and is on-par with the strongest full-table reasoning baseline, RoT. On average, TABGR[†] outperforms the best decomposition-based method, Table-Critic, by 2.8% and 1.1% on the two datasets, respectively.

Category	Method	LLaMA3.3-70B		Qwen2.5-72B		GPT-4o-mini		Average	
		WikiTQ	TabFact	WikiTQ	TabFact	WikiTQ	TabFact	WikiTQ	TabFact
Decomp.	Binder (Cheng et al., 2023)	52.2	80.5	57.0	82.2	54.8	83.3	54.7	82.0
	GraphOTTER (Li et al., 2025)	55.1	71.2	55.3	76.9	50.3	67.0	53.6	71.7
	TableRAG (Chen et al., 2024)	-	-	59.7	72.3	55.2	69.6	57.5	71.0
	Dater (Ye et al., 2023)	59.5	87.6	63.0	90.0	65.8	88.3	62.8	88.6
	Chain-of-Table (Wang et al., 2024)	62.1	89.9	68.3	89.7	67.5	88.9	66.0	89.5
	Table-Critic (Yu et al., 2025)	70.1	91.5	77.2	92.6	73.9	91.1	73.7	91.7
	TABGR[†] (ours)	76.9	93.5	<u>76.2</u>	92.8	74.3	91.8	75.8	92.7
	Δ (%)	$\uparrow 9.7$	$\uparrow 2.2$	-	$\uparrow 0.2$	$\uparrow 0.5$	$\uparrow 0.8$	$\uparrow 2.8$	$\uparrow 1.1$
Full-table	End-to-End QA	51.1	81.0	56.6	85.0	52.6	73.5	53.4	79.8
	Few-Shot QA	62.0	82.6	61.7	85.1	57.6	75.1	60.4	80.9
	Chain-of-Thought (Wei et al., 2022)	72.7	91.0	73.6	92.3	72.4	90.5	72.9	91.3
	LLM-Reranker + CoT	75.2	91.3	73.8	89.6	70.3	89.0	73.1	90.0
	Tunes (Nguyen et al., 2025)	76.8	91.2	-	-	<u>75.4</u>	90.4	76.1	90.8
	RoT (Zhang et al., 2025b)	<u>78.7</u>	<u>92.6</u>	<u>77.3</u>	<u>93.6</u>	74.2	<u>92.0</u>	76.7	<u>92.7</u>
	TABGR (ours)	80.1	94.4	79.2	94.0	77.1	92.3	78.8	93.6
	Δ (%)	$\uparrow 1.8$	$\uparrow 1.9$	$\uparrow 2.5$	$\uparrow 0.4$	$\uparrow 2.3$	$\uparrow 0.3$	$\uparrow 2.7$	$\uparrow 1.0$

Table 1: Overall table reasoning performance comparison. TABGR[†] represents our method with ATG-based table decomposition and TABGR represents our method with full-graph reasoning. **Bold** denotes the best performance and underline denotes the second-best. Δ (%) denotes the relative performance gain of TABGR and TABGR[†] compared with the best baseline results.

Method	WikiTQ			TabFact			SLQA
	Small	Medium	Large	Small	Medium	Large	
# Questions	(3,625)	(638)	(81)	(1,942)	(82)	(1,110)	
Table-Critic	72.1	60.8	<u>55.6</u>	91.7	87.8	52.3	
RoT	<u>80.6</u>	<u>71.4</u>	54.3	<u>92.7</u>	<u>90.2</u>	<u>54.1</u>	
TABGR [†]	78.6	67.4	70.4	93.6	92.7	66.7	
TABGR	82.1	72.3	65.4	94.5	93.9	65.5	
Δ (%)	$\uparrow 1.9$	$\uparrow 1.3$	$\uparrow 26.6$	$\uparrow 1.9$	$\uparrow 4.1$	$\uparrow 23.3$	

Table 2: Table reasoning performance comparison with the best baselines across tables in different size ranges: *small* (< 1k tokens), *medium* (1k–4k tokens), and *large* (> 4k tokens). WikiTQ contains table size from small to large. TabFact contains only *small* and *medium* tables, while SLQA focuses on real-world *large* tables.

To further examine the usefulness of TABGR[†], we break down the tables in the three datasets by size and report method performance over different table size categories in Table 2. We focus on TABGR, TABGR[†], and the two best baselines Table-Critic and RoT (same below).

Overall, larger tables are more challenging, and the different methods achieve lower accuracy over larger tables. TABGR is the most accurate over small or medium tables, while TABGR[†] excels over large tables, demonstrating its effectiveness in identifying the most relevant subgraph for answer generation. The performance gap between our methods and the baseline methods is much larger on the

Method	Shuffled WikiTQ		Shuffled TabFact	
	Row	Row & Col	Row	Row & Col
Table-Critic	$\downarrow 14.1$	$\downarrow 18.7$	$\downarrow 9.7$	$\downarrow 14.0$
RoT	$\downarrow 4.1$	$\downarrow 6.2$	$\downarrow 0.9$	$\downarrow 1.1$
TABGR [†]	$\downarrow 0.9$	$\downarrow 1.3$	$\downarrow 0.5$	$\downarrow 0.7$
TABGR	$\downarrow 0.1$	$\downarrow 0.5$	$\downarrow 0.6$	$\downarrow 0.7$

Table 3: Average relative accuracy changes (%), compared with accuracy on the original datasets.

large tables, emphasizing the “lost-in-the-middle” issue which is addressed by our QG-PPR triple reranking. In contrast, small tables yield the best performance for all methods due to low noise and short context. Notably, TABGR[†] performs even better on large tables than on medium-sized ones for WikiTQ, suggesting that ATG-based decomposition helps large-table reasoning by extracting a question-relevant subgraph from many irrelevant triples. Thus, TABGR is recommended for small and medium tables, while TABGR[†] is preferable for large tables or tight context budgets.

Robustness Analysis To evaluate robustness against answer location in tables, we shuffle table rows and columns with three random seeds and report the average performance drop from the original tables. We identify 1,193 (27.46%) questions in WikiTQ and 642 (31.72%) in TabFact as order-sensitive and retain their original order. The remaining order-insensitive questions are shuffled

Dataset	Category	Method	Input	Output
WikiTQ	Decomp.	Table-Critic	31,192	875
		TABGR [†]	5,063 (0.16×)	285 (0.33×)
	Full-table	RoT	2,013	460
			2,500	205
		TABGR w/o QG-PPR	(1.24×)	(0.45×)
		TABGR	4,739 (2.35×)	223 (0.48×)
TabFact	Decomp.	Table-Critic	30,681	1,008
		TABGR [†]	4,089 (0.13×)	245 (0.24×)
	Full-table	RoT	1,520	249
			1,998	172
		TABGR w/o QG-PPR	(1.31×)	(0.69×)
		TABGR	3,665 (2.41×)	207 (0.83×)

Table 4: Token costs per question. Ratios in parentheses are computed against Table-Critic for decomposition-based methods and against RoT for full-table methods.

for robustness evaluation.

As shown in Table 3, the two baseline models report substantial performance drops when the tables are shuffled, possibly because they are tuned for the original datasets. Our TABGR methods exhibit superior robustness, with only marginal accuracy drops, as our QG-PPR mechanism prioritizes the question-relevant triples regardless of their initial positions in the tables. The slight drops can be explained by random effects of shuffling. Also, when triples receive identical salience scores, their relative order follows the (shuffled) table order.

Efficiency Analysis Table 4 reports the LLM calling costs in terms of the number of input/output tokens per question. TABGR[†] significantly reduces both input/output tokens compared to the best decomposition-based baseline Table-Critic. For example, on WikiTQ, TABGR[†] uses 5,063 input tokens per question, only 0.16× Table-Critic. This reduction is primarily attributed to ATG-based subgraph extraction, which prunes irrelevant table content before answer generation.

For full-table reasoning, TABGR requires more input tokens than RoT does, mainly because QG-PPR constructs question-guided structural evidence through its personalization step. The additional input cost supports stronger structural grounding and robustness, while TABGR still reduces output tokens compared with RoT. For efficiency-sensitive settings, TABGR w/o QG-PPR provides a lightweight alternative that keeps the ATG representation but disables question-

Method	WikiTQ	TabFact
<i>Full-table reasoning</i>		
TABGR	80.1	94.4
w/o QG-PPR	79.5	93.8
w/o Semantic relevance	77.6	92.1
w/o ATG (Linearized CoT)	72.7	91.0
<i>Decomposition reasoning</i>		
TABGR [†]	76.9	93.5
w/o QG-PPR	76.0	93.2
w/o Semantic relevance	74.6	92.4
w/o ATG expansion	73.7	92.0
w/o ATG (Linearized decomp.)	69.0	88.1

Table 5: Ablation study results. Δ (%) indicates the relative performance drop when a module is removed.

guided reranking, using substantially fewer input tokens than full TABGR and fewer output tokens than RoT. As shown in Table 5, this variant incurs only 0.8% and 0.6% relative accuracy drops on WikiTQ and TabFact, respectively, and remains competitive with the strongest full-table baseline.

In Appendix F, we further compare the number of LLM calls of different methods, and report TABGR’s ATG construction time and memory consumption across datasets. The results on SLQA further demonstrate the scalability of TABGR.

Generalization Results We further evaluate TABGR under more challenging table structures and different backbone LLMs. On HiTab, which contains hierarchical tables with multi-level headers and nested structures, TABGR achieves 74.3% accuracy, outperforming GraphOTTER at 72.7%. On RealHiTBench, we use TreeThinker (Wu et al., 2025a) to recover the hierarchical row and column structures from serialized tables. To isolate the effect of table representation and reasoning, both TABGR and RoT use exactly the same recovered structure. Under this controlled setting, TABGR improves the overall accuracy from 53.12% to 54.11%, and the accuracy on structure comprehension from 64.46% to 66.92%. These results show that the gains of TABGR extend to more complex table structures and cannot be attributed solely to upstream structure recovery.

We also examine the effect of the backbone LLM in Appendix I, where TABGR consistently outperforms strong baselines with additional LLMs of different sizes. Beyond the datasets introduced in Section 4, Appendix M evaluates TABGR on TableBench (Wu et al., 2025b), providing further evidence on complex real-world table reasoning.

Ablation Study We study the contribution of each component through controlled ablations. “w/o ATG expansion” uses only anchor triples, “w/o QG-PPR” disables question-guided reranking, “w/o semantic relevance” disables the LLM-based semantic selector for identifying additional question-relevant columns and cell values beyond exact matches, and “w/o ATG” removes ATG representation and graph operations, reducing TABGR to CoT-style reasoning over linearized tables. Table 5 shows that all ablations degrade accuracy, with removing ATG causing the largest performance drop, confirming the importance of explicit row-column-cell grounding. Semantic relevance selection, iterative subgraph expansion, and question-guided reranking provide additional gains. The small drop of TABGR w/o QG-PPR further suggests a cost-sensitive variant under limited token budgets, as discussed in Table 4.

Structural Grounding Analysis. We investigate whether ATG’s gains stem from explicit modeling of table structure or can be achieved with richer serialized textual representations. We compare Markdown, JSON, HTML, and $\text{\LaTeX}$ with ATG on WikiTQ, using the same backbone (LLaMA3.3-70B) and prompting setting.

As shown in Table 6, HTML performs best among the textual formats with 77.3% accuracy, while ATG achieves 80.1%. We further analyze structural errors, including selecting incorrect rows or columns, associating values with incorrect positions, aggregating the wrong set of cells, and using evidence from irrelevant columns. These errors are judged by GPT-5.3 (OpenAI, 2026) using consistent evaluation criteria across all representations. ATG yields a lower error rate of 14.4%, compared with 19.8% for HTML.

The results indicate that richer serialization alone does not fully preserve table structure for reasoning, since the LLM must still recover the relations among rows, columns, and values from the input sequence. ATG instead encodes each cell as a triple $\langle r_i, h_j, c_{i,j} \rangle$, making these relations explicit and providing more reliable evidence grounding. A qualitative example is provided in Appendix K.

Other Results Additional parameter studies, ablations, and case studies are provided in Appendices D–N.

Representation	Accuracy	Struct. Error (%) ↓
Markdown	75.5	21.6
JSON	74.2	23.1
HTML	77.3	19.8
$\text{\LaTeX}$	76.3	20.4
ATG (TABGR)	80.1	14.4

Table 6: Comparison of table representations on WikiTQ. Structural errors include selection, positional, aggregation, and cross-column errors.

5 Conclusion

We proposed TABGR, a graph-based table reasoning method that represents tables as Attributed Table Graphs, which explicitly preserve row-column-cell structure and enable table-permutation robustness via a Question-Guided Personalized PageRank mechanism. Through reasoning over the attributed table graphs, TABGR produces traceable evidence paths grounded in table triples. Extensive experiments show that our method outperforms state-of-the-art methods in accuracy and robustness to row and column ordering, while offering practical efficiency trade-offs across different variants.

Limitations

Our method assumes that table structure, including row-column relations and cell correspondences, is available or recoverable before ATG construction. For serialized or irregular tables, an upstream structure-recovery module may be required, and its errors may propagate to downstream reasoning. Improving robustness to imperfect structure recovery remains an important direction for future work.

Our experiments mainly focus on TableQA and TableFV with objective textual outputs, and do not cover open-ended data analysis. Extending TABGR to multimodal and multilingual table reasoning is left for future work.

Ethics Statement

All datasets and models used in this paper are publicly available, and their usage strictly adheres to the corresponding licenses and terms of use.

Acknowledgments

This work is in part supported by the Australian Research Council (ARC) via Discovery Project DP240101006. Jianzhong Qi is supported by ARC Future Fellowship FT240100170. Junhao Gan is in part supported by ARC DP230102908.

References

- Ankush Agarwal, Chaitanya Devaguptapu, and Ganesh S. 2025. Hybrid graphs for table-and-text based question answering using llms. In *NAACL*, pages 858–875.
- Sergey Brin and Lawrence Page. 2012. Reprint of: The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, 56(18):3825–3833.
- Yihan Cao, Shuyi Chen, Ryan Liu, Zhiruo Wang, and Daniel Fried. 2023. Api-assisted code generation for question answering on varied table structures. In *EMNLP*, pages 14536–14548.
- Si-An Chen, Lesly Miculicich, Julian M Eisenschlos, Zifeng Wang, Zilong Wang, Yanfei Chen, Yasuhisa Fujii, Hsuan-Tien Lin, Chen-Yu Lee, and Tomas Pfister. 2024. TableRAG: Million-token table understanding with language models. In *NeurIPS*, pages 74899–74921.
- Wenhu Chen, Hongmin Wang, Jianshu Chen, Yunkai Zhang, Hong Wang, Shiyang Li, Xiyu Zhou, and William Yang Wang. 2020. TabFact: A large-scale dataset for table-based fact verification. In *ICLR*.
- 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 *ACL*, pages 1094–1110.
- Zhoujun Cheng, Tianbao Xie, Peng Shi, Chengzu Li, Rahul Nadkarni, Yushi Hu, Caiming Xiong, Dragomir Radev, Mari Ostendorf, Luke Zettlemoyer, and 1 others. 2023. Binding language models in symbolic languages. In *ICLR*.
- Rotem Dror, Gili Baumer, Segev Shlomov, and Roi Reichart. 2018. The hitchhiker’s guide to testing statistical significance in natural language processing. In *ACL*, pages 1383–1392.
- Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, and 1 others. 2024. The Llama 3 herd of models. *CoRR*, abs/2407.21783.
- Aaron Hurst, Adam Lerer, Adam P. Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, Aleksander Madry, Alex Baker-Whitcomb, Alex Beutel, and 1 others. 2024. GPT-4o system card. *CoRR*, abs/2410.21276.
- Glen Jeh and Jennifer Widom. 2003. Scaling personalized web search. In *WWW*, pages 271–279.
- Ruochun Jin, Xiyue Wang, Dong Wang, Haoqi Zheng, Yunpeng Qi, Silin Yang, and Meng Zhang. 2025. TALON: A multi-agent framework for long-table exploration and question answering. In *EMNLP*, pages 27397–27413.
- Karen Spärck Jones. 2004. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 60(5):493–502.
- 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 *COLING*, pages 5486–5506.
- Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paran-jape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024a. Lost in the Middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173.
- Tianyang Liu, Fei Wang, and Muhao Chen. 2024b. Re-thinking tabular data understanding with large language models. In *NAACL*, pages 450–482.
- Qingyang Mao, Qi Liu, Zhi Li, Mingyue Cheng, Zheng Zhang, and Rui Li. 2026. Potable: Toward systematic thinking via plan-then-execute stage reasoning on tables. *IEEE Trans. Knowl. Data Eng.*, 38(7):4615–4626.
- Md Mahadi Hasan Nahid and Davood Rafiei. 2024. Tabsqlify: Enhancing reasoning capabilities of llms through table decomposition. In *NAACL*, pages 5725–5737.
- Thi-Nhung Nguyen, Hoang Ngo, Dinh Phung, Thuy-Trang Vu, and Dat Quoc Nguyen. 2025. Improving table understanding with LLMs and entity-oriented search. In *COLM*.
- OpenAI. 2026. Openai GPT-5 system card. *CoRR*, abs/2601.03267.
- Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables. In *ACL/IJCNLP*, pages 1470–1480.
- Aofeng Su, Aowen Wang, Chao Ye, Chen Zhou, Ga Zhang, Gang Chen, Guangcheng Zhu, Haobo Wang, Haokai Xu, Hao Chen, Haoze Li, Haoxuan Lan, Jiaming Tian, Jing Yuan, Junbo Zhao, Junlin Zhou, Kaizhe Shou, Liangyu Zha, Lin Long, and 14 others. 2024. TableGPT2: A large multimodal model with tabular data integration. *CoRR*, abs/2411.02059.
- Yuan Sui, Mengyu Zhou, Mingjie Zhou, Shi Han, and Dongmei Zhang. 2024. Table meets LLM: can large language models understand structured table data? A benchmark and empirical study. In *WSDM*, pages 645–654.
- Yuxiang Wang, Junhao Gan, and Jianzhong Qi. 2025a. Towards question answering over large semi-structured tables. *CoRR*, abs/2502.13422.
- Yuxiang Wang, Jianzhong Qi, and Junhao Gan. 2025b. Accurate and regret-aware numerical problem solver for tabular question answering. In *AAAI*, pages 12775–12783.

- Zilong Wang, Hao Zhang, Chun-Liang Li, Julian Martin Eisenschlos, Vincent Perot, Zifeng Wang, Lesly Miculicich, Yasuhisa Fujii, Jingbo Shang, Chen-Yu Lee, and 1 others. 2024. Chain-of-Table: Evolving tables in the reasoning chain for table understanding. In *ICLR*.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-Thought prompting elicits reasoning in large language models. In *NeurIPS*, pages 24824–24837.
- Pengzuo Wu, Yuhang Yang, Guangcheng Zhu, Chao Ye, Hong Gu, Xu Lu, Ruixuan Xiao, Bowen Bao, Yijing He, Liangyu Zha, Wentao Ye, Junbo Zhao, and Haobo Wang. 2025a. Realhitbench: A comprehensive realistic hierarchical table benchmark for evaluating llm-based table analysis. In *ACL*, volume *ACL 2025 of Findings of ACL*, pages 7105–7137.
- Xianjie Wu, Jian Yang, Linzheng Chai, Ge Zhang, Jiaheng Liu, Xeron Du, Di Liang, Daixin Shu, Xianfu Cheng, Tianzhen Sun, Tongliang Li, Zhoujun Li, and Guanglin Niu. 2025b. Tablebench: A comprehensive and complex benchmark for table question answering. In *AAAI*, pages 25497–25506.
- An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, and 22 others. 2024. Qwen2.5 technical report. *CoRR*, abs/2412.15115.
- Yunhu Ye, Binyuan Hui, Min Yang, Binhua Li, Fei Huang, and Yongbin Li. 2023. Large language models are versatile decomposers: Decompose evidence and questions for table-based reasoning. In *SIGIR*, pages 174–184.
- Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. TaBERT: Pretraining for joint understanding of textual and tabular data. *arXiv preprint arXiv:2005.08314*.
- Peiying Yu, Guoxin Chen, and Jingjing Wang. 2025. Table-Critic: A multi-agent framework for collaborative criticism and refinement in table reasoning. In *ACL*, pages 17432–17451.
- Han Zhang, Yuheng Ma, and Hanfang Yang. 2025a. AL-TER: augmentation for large-table-based reasoning. In *NAACL*, pages 179–198.
- Xuanliang Zhang, Dingzirui Wang, Keyan Xu, Qingfu Zhu, and Wanxiang Che. 2025b. RoT: Enhancing table reasoning with iterative row-wise traversals. In *EMNLP*, pages 559–579.
- 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. In *EMNLP*, pages 14786–14802.
- Mingyu Zheng, Hao Yang, Wenbin Jiang, Zheng Lin, Yajuan Lyu, Qiaoqiao She, and Weiping Wang. 2023. Chain-of-Thought reasoning in tabular language models. In *EMNLP*, pages 11006–11019.
- Jiaru Zou, Dongqi Fu, Sirui Chen, Xinrui He, Zihao Li, Yada Zhu, Jiawei Han, and Jingrui He. 2025. RAG over Tables: Hierarchical memory index, multi-stage retrieval, and benchmarking. *arXiv preprint arXiv:2504.01346*.

A Prompts

A.1 Prompts for Graph-Based Retrieval

The prompt used to select the key columns (i.e., edges) for the initial subgraph is as follows.

Initial column selection

You are a table analysis assistant.

Given:

- A table title
- A natural language question about the table
- A list of candidate columns from the table
- One sample row from the table

Task: Select the column names that contain the information necessary to answer the question. Only return the column names (split by commas), nothing else.

Here are some examples:

[Examples]

Title: {title}

Question: {question}

Candidate Columns: {candidate_col}

Sample Row: {sample_row}

Answer:

The prompt used to check if the subgraph fetched is sufficient for question answering is as follows.

Subgraph sufficiency check

You are given a TableQA question and multiple candidate reasoning paths.

Instructions:

- Use ONLY the candidate reasoning paths.
- Treat edges as UNDIRECTED: "(rowK; Col; X)" and "(X; Col; rowK)" both mean rowK have Col = X.
- Decide if the provided paths are sufficient to answer the question.
 - * Sufficient: the given candidate paths can deterministically yield the answer.
 - * Insufficient: necessary link(s) or value(s) are missing or ambiguous.
- If sufficient: Set Finished: True.
- If insufficient: Set Finished: False.
- Output exactly and only the following one section. Do NOT add any other text.

Here are some examples:

[Examples]

Title: {title}

Question: {question}

Candidate Reasoning Paths: {reasoning_paths}

Finished:

The prompt used to select more edges (i.e., column headers) that are used to retrieve corresponding triples to expand the subgraph is as follows.

Additional edge selection

You are given a TableQA question, a set of (possibly incomplete) candidate reasoning paths, and a set of available relations from the same table graph.

Goal:

- Select the MINIMAL SUFFICIENT subset of relations from Available Relations which, when combined with the candidate reasoning paths, is enough to answer the question.

Rules:

- Use ONLY the relation(s) in provided available relations. Do NOT invent new facts.
- Minimality: choose the fewest relations (≥ 1) among available relations that make the answer derivable.
- Tie-breaking: if multiple equally minimal subsets exist, preserve the original order in Available Relations (i.e., pick the earliest lines that work).

Output format (STRICT):

- Output exactly ONE line that starts with: SELECTED_RELATIONS: followed by a Python list literal of strings, e.g. ['relation1', 'relation2'].
- Do NOT output anything else.

Here are some examples:

[Examples]

Title: {title}

Question: {question}

Candidate Reasoning Paths: {reasoning_paths}

Available Relations: {available_relations}

Sample Row: {sample_row}

SELECTED_RELATIONS:

A.2 Prompts for Graph-Based Answer Generation

The prompt used to generate the final answers by both TABGR and TABGR[†] is as follows.

Answer generation

You are given a TableQA question and a list of candidate reasoning paths. Given a question, you should think it step by step and then answer the question. Also provide the final reasoning path enclosed within <paths> ... </paths>.

Rules (strict):

- OUTPUT FORMAT: two sections:
<think> <paths> ... </paths> ... </think>
<answer> ... </answer>

Here are some examples:

[Examples]

YOUR TURN

Title: {title}

Question: {question}

Header: {header}

Table Content: {reasoning_paths}

<think>

<paths>

Algorithm 1: Overall pipeline of TABGR[†]

Input : Table $\mathcal{T}$, question $\mathcal{Q}$
Output : Final answer Ans , Chain-of-Thought text T , and reasoning paths P .

- 1 $\mathcal{G} \leftarrow \text{GraphConstruction}(\mathcal{T})$ ▷ Represent $\mathcal{T}$ as structured triples
- 2 $\hat{A} \leftarrow \text{InducePropagationMatrix}(\mathcal{G})$ ▷ Triple-level transition operator
- 3 $\mathcal{G}^* \leftarrow \text{InitialExtraction}(\mathcal{Q}, \mathcal{G})$ ▷ Exact match and relation selection
- 4 $suf \leftarrow \text{False}, count \leftarrow 0$
- 5 **while** $count < 3$ **do** ▷ LLM-based subgraph sufficiency check
- 6 $suf \leftarrow \text{Judge}(\mathcal{Q}, \mathcal{G}^*)$
- 7 **if** $suf = \text{True}$ **then**
- 8 **break**
- 9 $\mathcal{G}^* \leftarrow \mathcal{G}^* \cup \text{IterativeExpansion}(\mathcal{Q}, \mathcal{G})$ ▷ Select additional edges
- 10 $count \leftarrow count + 1$
- 11 $s \leftarrow \text{QG-PPR}(\mathcal{Q}, \mathcal{G}^*, \hat{A})$ ▷ Saliency estimation via Eq. (5)
- 12 $\mathcal{G}_{ranked}^* \leftarrow \text{Rerank}(\mathcal{G}^*, s)$ ▷ Inter-row and intra-row reranking
- 13 $(P, T, Ans) \leftarrow \text{LLMReasoning}(\mathcal{Q}, \mathcal{G}_{ranked}^*)$ ▷ Generate grounded reasoning path and answer
- 14 **return** P, T and Ans .

B Overall Method Pipeline

Algorithm 1 summarizes the processing pipeline of our TABGR[†] method. TABGR shares a similar pipeline, but without Lines 3 to 10. Its $\mathcal{G}^*$ is just $\mathcal{G}$.

C Dataset Details

Table 7 summarizes the five datasets used in the experiments. All experiments are run with three NVIDIA A100 80 GB GPUs on a cloud GPU server.

Dataset	#Cols. #Rows #Toks. #Toks. #Toks.					#QA Pairs		
	/Tbl.	/Tbl.	/Tbl.	/Cell	/Ans.	Train	Val.	Test
TabFact	6.3	13.5	317.9	3.7	1.0	92,283	12,792	2,024
WikiTQ	6.4	25.4	662.6	4.1	1.7	11,321	2,831	4,344
HiTab	-	-	-	-	-	7,417	1,671	1,584
RealHiTBench	11.9	45.6	-	-	-	-	-	3,752
SLQA	11.0	733.8	9,786.2	1.2	4.4	1,324	239	1,110

Table 7: Dataset statistics. ‘Cols.’: Columns; ‘Toks.’: Tokens; ‘Tbl.’: per table.

D Hyper-parameter Study**D.1 Impact of w_{row} and w_{col}**

We first tune the propagation weights w_{row} and w_{col} under the constraint $w_{row} + w_{col} = 1$. We evaluate a set of representative configurations as shown in Table 8, including row-oriented (larger w_{row}), balanced, and column-oriented (larger w_{col}) configurations. The best-performing configuration

Method	w_{row}	w_{col}	WikiTQ		TabFact	
			Acc.	Δ (%)	Acc.	Δ (%)
TABGR [†]	0.1	0.9	75.8	$\downarrow 1.4$	92.4	$\downarrow 1.2$
	0.3	0.7	76.9	-	92.9	$\downarrow 0.6$
	0.5	0.5	76.4	$\downarrow 0.7$	93.2	$\downarrow 0.3$
	0.7	0.3	76.1	$\downarrow 1.0$	93.5	-
	0.9	0.1	74.2	$\downarrow 3.5$	92.1	$\downarrow 1.5$
TABGR	0.2	0.8	78.8	$\downarrow 1.6$	93.6	$\downarrow 0.8$
	0.4	0.6	79.4	$\downarrow 0.9$	94.0	$\downarrow 0.4$
	0.6	0.4	80.1	-	94.4	-
	0.8	0.2	78.5	$\downarrow 2.0$	93.3	$\downarrow 1.2$

Table 8: Impact of w_{row} and w_{col} ($w_{row} + w_{col} = 1$). Δ indicates the relative performance drop from the best result in each group.

is selected based on validation performance and is used for results reported in the main experiments.

Across both WikiTQ and TabFact, TABGR consistently achieves its best performance with $w_{row} = 0.6$ and $w_{col} = 0.4$, indicating a stable preference for moderately stronger row-wise propagation. This suggests that TABGR often benefits from aggregating multiple attributes within the same row, while still preserving sufficient column-level interactions for cross-row comparison. In contrast, TABGR[†] exhibits more dataset-dependent sensitivity to the propagation weights. On WikiTQ, which typically requires multi-step reasoning involving cross-row filtering and comparison, a stronger column-wise propagation is preferred. On TabFact, which focuses on binary verification and often requires checking the consistency

Method	α	WikiTQ		TabFact	
		Acc.	Δ (%)	Acc.	Δ (%)
TABGR [†]	0.05	75.6	$\downarrow 1.7$	92.8	$\downarrow 0.7$
	0.15	76.9	-	93.5	-
	0.25	76.1	$\downarrow 1.0$	93.1	$\downarrow 0.4$
TABGR	0.25	79.2	$\downarrow 1.1$	93.9	$\downarrow 0.5$
	0.35	80.1	-	94.4	-
	0.45	79.5	$\downarrow 0.7$	94.0	$\downarrow 0.4$

Table 9: Impact of the teleport probability α . Δ (%) indicates the relative performance drop from the best result in each group.

of multiple attributes within the same row, stronger row-wise propagation proves more effective.

From a reasoning perspective, these trends reflect how the two methods process information. TABGR performs global reasoning over the entire graph structure, allowing salience signals to propagate broadly and resulting in more stable performance across hyper-parameter settings. TABGR[†], by contrast, relies more on iterative reasoning over progressively constructed subgraphs, making its performance more sensitive to how relevance is propagated along row or column dimensions.

D.2 Impact of α

Next, we investigate the impact of the teleport probability α , with results summarized in Table 9.

The optimal values of α again vary for the two methods. TABGR[†] achieves peak performance at $\alpha = 0.15$, whereas TABGR performs the best at $\alpha = 0.35$. This suggests that decomposition-based reasoning, which operates on subgraphs, benefits from a lower restart probability to allow for deeper structural propagation over the graphs. Conversely, reasoning over full tables requires a larger α to maintain focus on question-relevant anchors, since it receives larger graphs.

D.3 Impact of Number of Iterations in QG-PPR

Table 10 reports the impact of the power iteration count, K , used by QG-PPR. For both TABGR[†] and TABGR, their accuracy improves slightly from $K = 10$ to $K = 20$ and remains unchanged at $K = 30$. Theoretically, the error of the power method for PPR decreases at a geometric rate of $(1 - \alpha)^K$. With our default teleport probability $\alpha = 0.15$ for TABGR[†] and $\alpha = 0.35$ for TABGR, the residual error terms $(0.85)^{20} \approx 0.038$ and $(0.65)^{20} \approx 0.00018$ are sufficiently small to sta-

bilize the relative ranking of the table triples.

Method	K	WikiTQ		TabFact	
		Acc.	Δ (%)	Acc.	Δ (%)
TABGR [†]	10	76.4	$\downarrow 0.7$	93.1	$\downarrow 0.4$
	20	76.9	-	93.5	-
	30	76.9	0.0	93.5	0.0
TABGR	10	79.8	$\downarrow 0.4$	94.2	$\downarrow 0.2$
	20	80.1	-	94.4	-
	30	80.1	0.0	94.4	0.0

Table 10: Impact of the number of iterations, K .

D.4 Impact of the Initial Scoring Weights

We also evaluate the impact of the initial scoring weights v_{col} and v_{cell} , which affect the question-aware relevance distribution in the personalization vector $\mathbf{p}_0$.

As Table 11 shows, setting $v_{\text{col}} = 1.0$ and $v_{\text{cell}} = 2.0$ yields the best accuracy. This observation aligns with our intuition that column headers guide the relevant table fields, while specific cell values play a more direct role in grounding evidence for reasoning. Incorporating the IDF term further adjusts v_{cell} by down-weighting frequent values, helping the methods emphasize more informative evidence. Overall, the relatively small performance variations across different configurations indicate that our graph-based propagation is robust to the choice of initial scoring weights.

v_{col}	v_{cell}	WikiTQ (Acc.)		TabFact (Acc.)	
		TABGR [†]	TABGR	TABGR [†]	TABGR
2.0	1.0	76.6	79.9	93.4	94.2
1.0	1.0	76.5	79.7	93.3	94.2
1.0	2.0	76.9	80.1	93.5	94.4

Table 11: Impact of the initial scores v_{col} and v_{cell} .

E Additional Ablation Study

We conduct additional ablation studies to further evaluate the effectiveness of QG-PPR.

In Table 12, we remove QG-PPR (“w/o QG-PPR”) and rerun our methods on the shuffled datasets as described earlier. Both TABGR and TABGR[†] become more vulnerable to the shuffles, recording larger drops in accuracy, e.g., 5.9% vs. 0.5% for TABGR w/o QG-PPR and TABGR on WikiTQ with randomly shuffled rows and columns.

Table 13 reports accuracy results when QG-PPR is replaced by naive PageRank (“w/ PageRank”),

Method	Shuffled WikiTQ		Shuffled TabFact	
	Row	Row & Col	Row	Row & Col
TABGR [†]	↓ 0.9	↓ 1.3	↓ 0.5	↓ 0.7
TABGR [†] w/o QG-PPR	↓ 6.0	↓ 7.1	↓ 1.0	↓ 1.2
TABGR	↓ 0.1	↓ 0.5	↓ 0.6	↓ 0.7
TABGR w/o QG-PPR	↓ 5.8	↓ 5.9	↓ 1.2	↓ 1.5

Table 12: Impact of QG-PPR on permutation robustness.

Method	WikiTQ		TabFact	
	Acc.	Δ	Acc.	Δ
Few-Shot QA	62.0	-	82.6	-
w/ Row-based Reranking	58.9	↓ 5.0	74.3	↓ 10.0
TABGR	80.1	-	94.4	-
w/ PageRank	78.9	↓ 1.5	92.5	↓ 2.0

Table 13: Impact of QG-PPR on improving accuracy.

which uses uniform initialization without a personalization vector, i.e., to propagate importance without explicitly modeling question relevance.

We further compare against a Few-Shot QA baseline with a row-based reranking strategy. Few-Shot QA relies on linearized table representations, and the row-based reranking ranks table rows according to LLM-predicted importance scores, with higher-ranked rows placed earlier in the LLM input. This setting allows us to examine whether simple LLM-based reranking is sufficient to improve reasoning performance under a table linearization strategy.

The results show that both alternative reranking strategies result in accuracy drops. This indicates that reranking alone is insufficient to address the limitations of linearized table reasoning, and highlights the importance of explicitly modeling table structure, as enabled by QG-PPR’s graph-based representation and question-guided propagation.

F Efficiency and Scalability Analysis

ATG construction Table 14 reports the ATG construction time. We see that it takes less than 5 seconds to construct the ATGs required for the 421 tables of the WikiTQ dataset, i.e., 0.010 s per table and it is even faster for TabFact, as the average table size is smaller in TabFact. Even for large-table QA benchmark, SLQA (with 27 tables), it takes only 0.019 s to construct the ATG per table. Also, the maximum time (Max Time) taken for any table in the datasets is below 0.5 seconds, showing the high practicality of our method.

In terms of memory usage, constructing an ATG

Dataset	# Questions	Total Time (s)	Avg. Time/T (s)	Max Time (s)
WikiTQ	4,344	4.28	0.010	0.36
TabFact	2,024	0.73	0.002	0.15
SLQA	1,100	0.55	0.019	0.40

Table 14: ATG construction times.

requires approximately 3 MB per table for WikiTQ, 1.4 MB for TabFact, and 45 MB for SLQA (average of 9,786 tokens per table). This efficiency comes from representing the graph using a CSR sparse matrix, which is well suited for the highly sparse structure of ATGs. Importantly, ATG construction and updates can be performed fully offline as it is query independent, hence minimizing the risk of out-of-memory (OOM) issues during inference.

QG-PPR We further study the running time of QG-PPR matrix computation. On average, each question takes 0.17 s on WikiTQ and 0.08 s on TabFact for this computation. Even on the large-table benchmark SLQA, the running time is only 2.5 s per question. These results demonstrate the efficiency of QG-PPR.

To compare the efficiency of QG-PPR with a simpler LLM-Reranker pipeline, we report the average number of input and output tokens per question in Table 15. On the WikiTQ dataset, although TABGR uses more input tokens, its number of output tokens is much smaller. Since LLM inference latency is largely dominated by the length of generated outputs, the effective inference time of TABGR is only 22% of that of the simpler reranking method. Similarly, on TabFact, the inference time of TABGR is only 31% of that of the LLM-Reranker.

These results show that QG-PPR achieves higher accuracy (shown in Table 1) while substantially reducing output generation time by avoiding prompting the LLM to fully rerank all table cells. As a result, TABGR is not only more accurate but also more efficient in inference than the simpler LLM-Reranker.

Number of LLM Calls As shown in Table 16, existing decomposition-based methods typically require many iterative LLM calls per question (up to 100 calls on WikiTQ). TABGR[†] significantly reduces the number of required LLM calls through ATG-based decomposition. The full-table variant TABGR requires only three LLM calls per question on average, achieving highly efficient inference.

Dataset	Method	Input Tokens	Output Tokens
WikiTQ	LLM-Reranker + CoT	2,520	1,032
	TABGR	4,753	223
TabFact	LLM-Reranker + CoT	1,638	677
	TABGR	3,665	207

Table 15: Token consumption compared with an LLM-based reranker.

Method	# LLM Calls
Dater (Ye et al., 2023)	100
Binder (Cheng et al., 2023)	50
Table-Critic (Yu et al., 2025)	≤ 32
Chain-of-Table (Wang et al., 2024)	≤ 25
TABGR [†]	≤ 9
TABGR	3

Table 16: Comparison of the upper bound of LLM calls per question across methods on the WikiTQ dataset.

G Experiments on HiTab

We further evaluate our method on HiTab to evaluate its generalizability. HiTab is characterized by its intricate hierarchical headers and nested structures, which pose a significant challenge.

To handle the hierarchical structures in HiTab, TABGR introduces two structural adaptations during triple construction to preserve structural and semantic context. First, we flatten multi-level column headers by recursively merging parent and child headers into a single semantic attribute path (e.g., Region-Worker Type-Metric), which is embedded into every triple in the column name h_j . Second, we adapt $c_{i,j}$ by propagating row-level context by forward-filling merged cells in the leftmost columns, ensuring that each triple captures the complete row hierarchy (e.g., Year-Industry).

We compare TABGR with recent methods, following their original implementations. In particular, TableParser (Zhao et al., 2023), GraphOTTER (Li et al., 2025), and TABGR are all evaluated using Qwen2-72B-Instruct as the backbone LLM, to ensure consistency with their reported settings.

As shown in Table 17, TABGR outperforms the strongest baseline GraphOTTER, which is also a graph-based reasoning method designed for complex table understanding. This result further demonstrates the flexibility of TABGR in handling more complex table structures.

H Experiments on RealHiTBench

Following our TableQA/TableFV setting, we evaluate 2,384 questions from the three objective-

Method	Accuracy
TableParser (Zhao et al., 2023)	44.6
GPT-3.5 (Zhao et al., 2023)	50.0
code-davinci-002 (Cao et al., 2023)	69.3
GraphOTTER (Li et al., 2025)	72.7
TABGR (ours)	74.3

Table 17: Accuracy results on HiTab.

Method	Fact Checking	Numerical Reasoning	Structure Compr.	Overall Accuracy
LLaMA3.3-70B	53.08	36.58	55.81	48.20
RoT w/ Markdown	49.71	41.89	61.87	49.20
RoT w/ TreeThinker	53.99	45.91	64.46	53.12
TABGR w/ TreeThinker	54.81	46.43	66.92	54.11

Table 18: Performance comparison on RealHiTBench using LLaMA3.3-70B-Instruct.

answer categories: *Fact Checking*, *Numerical Reasoning*, and *Structure Comprehending*. We exclude *Data Analysis* and *Chart Generation* due to their open-ended or executable outputs.

We use TreeThinker (Wu et al., 2025a) to recover hierarchical row- and column-header structures from serialized $\text{\LaTeX}$ tables. RoT w/ TreeThinker and TABGR w/ TreeThinker use exactly the same recovered hierarchy, with RoT serializing it as textual input and TABGR converting it into complete row/column paths and graph triples. RoT w/ Markdown directly reasons over the flat Markdown representation. All methods use LLaMA3.3-70B-Instruct as the backbone.

As shown in Table 18, TABGR achieves the best overall Accuracy of 54.11%. Compared with RoT using the same TreeThinker-recovered structure, TABGR improves overall accuracy from 53.12% to 54.11%, and Structure Comprehending from 64.46% to 66.92%. This controlled comparison shows that the improvement is not solely due to upstream structure recovery, but also benefits from TABGR’s graph-based structural representation and reasoning.

I Experiments Across Different LLMs

LLaMA3.1-8B To study the impact of the size of the backbone LLM, we use a smaller LLM, LLaMA3.1-8B, in addition to the LLMs used in Table 1. As shown in Table 19, the accuracy drops for all methods compared with those using LLaMA3.1-70B, which is expected. Our methods still outperform the best baselines in the respective categories, showing their robustness against LLM size. TABGR[†] is now slightly more ac-

Method	WikiTQ	TabFact
Table-Critic (Yu et al., 2025)	45.5	68.9
RoT (Zhang et al., 2025b)	<u>63.7</u>	<u>74.8</u>
TABGR [†]	56.5	75.5
TABGR	64.2	75.2

Table 19: Accuracy comparison using LLaMA3.1-8B.

Method	WikiTQ	TabFact
TabSQLify (Nahid and Rafiei, 2024)	57.0	70.7
Chain-of-Table (Wang et al., 2024)	62.2	85.6
PoTable (Mao et al., 2026)	65.6	87.1
Tunes (Nguyen et al., 2025)	<u>75.4</u>	<u>90.4</u>
TABGR [†]	74.2	90.9
TABGR	76.5	91.6

Table 20: Accuracy comparison using LLaMA-3.1-70B.

curate than TABGR on TabFact, because questions in TabFact mainly rely on localized evidence, making decomposition-based reasoning particularly beneficial for the smaller LLM with limited information-filtering capability.

LLaMA3.1-70B A few baselines, TabSQLify (Nahid and Rafiei, 2024), PoTable (Mao et al., 2026), and Tunes (Nguyen et al., 2025), have used LLaMA3.1-70B in their original papers. For consistency, here, we also use LLaMA3.1-70B as the backbone LLM for TABGR. The results in Table 20 show that TABGR outperforms all baselines with this backbone LLM.

GPT-5-mini We further evaluate with GPT-5-mini, a more recent model in the GPT series. The results in Table 21 show that TABGR continues to outperform the strongest baselines when all are powered by GPT-5-mini.

J Case Study

We conduct a case study comparing the outputs of our TABGR with those of two strong baselines: Table-Critic and RoT.

As shown in Figure 3, TABGR leverages the ATG structure to precisely identify empty cells, whereas RoT and Table-Critic suffer from structural misinterpretation and hallucination, leading to the incorrect inclusion of Row 1.

J.1 Grounded and Traceable Reasoning

We provide qualitative examples to illustrate how ATG triples make the reasoning process more traceable. The goal is not to claim that textual CoT

Method	WikiTQ	TabFact
Chain-of-Thought (Zheng et al., 2023)	76.4	<u>94.5</u>
RoT (Zhang et al., 2025b)	<u>81.7</u>	93.6
TABGR [†]	83.8	95.2
TABGR	85.4	95.8

Table 21: Accuracy comparison using GPT-5-mini.

Player	Goals	Country/Venue	Notes
Marcos Pizzelli	1	Armenia	International goal record
Marcos Pizzelli	1	Cyprus	International goal record
Marcos Pizzelli	1	Cyprus	International goal record
...	...	...	...

Table 22: Table excerpt for the assumption-drift case study.

are not detailed, but to show that TABGR aligns reasoning steps with concrete table evidence.

Case: Assumption drift. Table 22 shows an excerpt of the original table. For the question “internationally, Marcos Pizzelli has scored multiple goals in Armenia and what other country?”, the gold answer is “Cyprus”.

A CoT baseline predicts “Andorra” after shifting the question from player-level scoring records to a team-level proxy. This introduces an unsupported assumption that is ungrounded in the target table evidence.

TABGR instead grounds the reasoning path in triples such as $\langle r_i, \text{player}, \text{Marcos Pizzelli} \rangle$ and $\langle r_i, \text{country/venue}, \text{Cyprus} \rangle$. It counts countries directly from the selected rows and excludes Armenia as specified by the question. The selected triples show that “Cyprus” appears multiple times for Marcos Pizzelli, making it the remaining country with multiple goals. This example shows that a traceable triple-level path can help localize whether an answer is supported by the table or introduced by an unsupported reasoning shortcut.

K Analyses on Structural Grounding

The results in Table 6 show that ATG achieves higher accuracy and fewer structural errors than serialized textual table representations. Here, we provide a qualitative comparison to illustrate how explicit row-column-cell modeling helps avoid grounding errors that can still occur with richer textual representations.

Case Study

Question: During what time period was there no shirt sponsors?

Gold Answer: 1982–1985

Table Content:

Year	Kit Manufacturer	Shirt Sponsor	Back of Shirt Sponsor	Short Sponsor
1977–1978	[Empty]	National Express	[Empty]	[Empty]
1982–1985	Umbro	[Empty]	[Empty]	[Empty]
1985–1986	Umbro	Whitbread	[Empty]	[Empty]
1986–1988	Henson	Duraflex	[Empty]	[Empty]
1988–1989	[Empty]	Gulf Oil	[Empty]	[Empty]
...	...	...	...	...
2011–2013	Errea	Mira Showers	Barr Stadia	Gloucestershire Echo

Model	Reasoning Process & Prediction
TABGR (Ours)	Thought: To find the time period when there was no shirt sponsor, we need to examine the "Shirt Sponsor" column in the table. The table lists various years and their corresponding shirt sponsors. We are looking for rows where the "Shirt Sponsor" is empty, indicating no sponsor. Upon reviewing the table, we find one row where the "Shirt Sponsor" is empty: (row2; Shirt Sponsor;) with the year range (row2; Year; 1982–1985). Therefore, the time period when there was no shirt sponsor is 1982–1985. Reasoning Path: (row2; Shirt Sponsor;) → (row2; Year; 1982–1985) Prediction: 1982–1985 ✓ Correct
Table-Critic	Thought: To determine the time period with no shirt sponsor, we need to examine the "shirt sponsor" column in the original table. The rows with no shirt sponsor are: Row 1: 1977–1978 (no shirt sponsor); Row 2: 1982–1985 (no shirt sponsor). However, we also notice that there are other rows with no shirt sponsor: Row 6 is not one of them since it has gulf oil as the shirt sponsor, but row 1 and row 2 are. Given the information, the time period with no shirt sponsor is indeed 1977–1978 and 1982–1985. The Prediction Answer only mentions one of these time periods (1982–1985), but it does not account for the other time period (1977–1978). However, since 1982–1985 is indeed one of the correct time periods, the Prediction Answer is partially correct but incomplete. Analysis: Lost Structural Information and Model Confusion. The critic misinterprets the table structure (identifying Row 1 as empty) and incorrectly penalizes the correct partial prediction, leading to an over-corrected redundant answer. Prediction: 1977–1978, 1982–1985 ⚡ Incorrect
RoT	Thought: To answer the question, I need to figure out during which time period there was no shirt sponsor based on the given table. Let me look at the table again. The table has several columns: Year, Kit Manufacturer, Shirt Sponsor, Back of Shirt Sponsor, and Short Sponsor. My task is to find the time period when the "Shirt Sponsor" column is blank. Let me go through each row one by one. 1. 1977–1978: The "Shirt Sponsor" column is blank. That's one time period. 2. 1982–1985: The "Shirt Sponsor" column is also blank. That's another time period. 3. 1985–1986: The "Shirt Sponsor" column has a value, "Whitbread"... [omitted rows 4–17 for brevity, but similar logic]... So, the time periods when there was no shirt sponsor are 1977–1978 and 1982–1985. Analysis: Structural Grounding Error. The model incorrectly interprets Row 1 as blank, failing to ground "National Express" to its correct column due to the limitations of table linearization. Prediction: 1977–1978, 1982–1985 ⚡ Incorrect

Figure 3: Case study comparing TABGR with strong baselines: Table-Critic and RoT.

isn	name	arrival date	departure date	notes
82	Rasul Kudayev	—	2004-02-27	Repatriated to Russia on January 3, 2004...
...	...	...	...	...

Table 23: Table excerpt for the column-grounding case study.

K.1 Case Study of Row-Column-Cell Grounding

Table 23 shows an excerpt of a table for the question “when was the first departure?”, whose gold answer is “2004-02-27”.

The L^AT_EX-based method incorrectly predicts “January 3, 2004”, using a date from the *notes* column instead of “2004-02-27” from the *departure date* column. Although L^AT_EX provides richer layout information than simpler textual formats, the model must still infer the association between each value and its corresponding column from the serialized input. In con-

trast, TABGR explicitly represents the relevant evidence as $\langle r_1, \text{departure date}, 2004-02-27 \rangle$. The date “January 3, 2004” is separately associated with the *notes* column, preventing it from being confused with the target departure date. TABGR therefore returns the correct answer, “2004-02-27”.

This example complements the quantitative results in Table 6. Richer textual representations can preserve more layout information, but they still require the LLM to recover row-column-cell relations from the input sequence. ATG encodes these relations explicitly, providing more reliable grounding and making the supporting evidence directly traceable.

L Comparison with Program-based Methods

Program-based methods can mitigate long-context issues by avoiding full-table processing through

Method	GPT-4o-mini	Qwen2.5-72B
Python QA	51.2	41.9
Binder	61.4	58.5
DATER	59.8	60.2
TableRAG	55.2	59.6
TabSQLify	56.5	—
TALON	71.0	69.6
TABGR [†]	74.3	76.2
TABGR	77.1	79.2

Table 24: Comparison with representative program-based methods.

Method	ROUGE-L
Chain-of-Thought	38.2
RoT	44.8
TABGR	45.5

Table 25: Additional evaluation on TableBench.

Python or SQL execution. However, this paradigm may introduce program-generation and execution errors, including wrong column/filter selection and information loss from early filtering or aggregation.

We compare TABGR with representative program-based methods on table understanding tasks. Python QA directly generates Python code to answer questions. Binder iteratively refines and executes generated programs. DATER (Ye et al., 2023) decomposes tables and questions into sub-tables and uses SQL to answer sub-questions. TableRAG (Chen et al., 2024) retrieves relevant cells through generated Python programs. TabSQLify (Nahid and Rafiei, 2024) generates SQL queries based on table schemas. TALON (Jin et al., 2025) uses a multi-agent framework with code execution.

As shown in Table 24, TABGR outperforms the best baseline by 6.1 points on GPT-4o-mini and 9.6 points on Qwen2.5-72B. Although program-based methods can avoid processing full table sequences, they may break the original table into partial views or introduce program-generation and execution errors. In contrast, TABGR and TABGR[†] preserve table structure and sufficient context through ATG, leading to stronger performance. Program execution is therefore complementary to our work and could be combined with TABGR in future systems.

M Additional Evaluation on TableBench

To further evaluate TABGR in more challenging table reasoning settings, we conduct additional experiments on TableBench (Wu et al., 2025b).

Method	WikiTQ	TabFact
Claude Opus 4.7	78.6	91.2
TABGR w/ Claude Opus 4.7	80.9	93.7

Table 26: Evaluation with Claude Opus 4.7 on WikiTQ and TabFact.

Model	mpg
Fiat 500 1.2 POP	46
Mini Cooper COUPE 6M 3DR 1.6L Diesel	52

Table 27: Excerpt for the entity-comparison failure case.

TableBench contains complex real-world table reasoning examples and complements the main benchmarks used in Section 4.

As shown in Table 25, TABGR achieves the best ROUGE-L among the compared methods, with a modest gain over RoT. This suggests that ATG-based reasoning can generalize to more complex table reasoning settings. The smaller margin suggests that TableBench remains challenging and may benefit from stronger hybrid reasoning strategies.

N Frontier Model Evaluation and Failure Cases

To examine whether standard table reasoning benchmarks remain challenging for frontier LLMs, we evaluate Claude Opus 4.7 (via Anthropic API) on the full test sets of WikiTQ and TabFact. As shown in Table 26, TABGR further improves Claude Opus 4.7 on both datasets, demonstrating that our structured reasoning framework remains beneficial even with a strong frontier LLM. This suggests that these benchmarks remain challenging, particularly for precise filtering, comparison and structural reasoning.

Failure Case 1: Entity comparison error. Table 27 shows a WikiTQ example that requires comparing numerical values across entities. The question asks which vehicle gets better mpg. The gold answer is “Mini Cooper”, since $52 > 46$, but the frontier model predicts “Fiat 500”. This indicates that even frontier models can make incorrect entity comparisons when reasoning over tabular values.

Failure Case 2: Counting and filtering error. Table 28 shows an example where the question asks how many singles are only a single. The gold answer is 18, while the frontier model predicts 22. The error comes from incorrect filtering on the

Single	Album
...	A World Called You
...	A World Called You
...	single only
...	A World Called You

Table 28: Excerpt for the counting/filtering failure case.

Year	Name	Year	Name
...	...	1962–63	Mrs S. Von Wielligh
...	...	1963–64	Mr F.J. Van Heerden

Table 29: Excerpt for the structural-lookup failure case.

Album column, where entries associated with an album are over-counted as “single only” cases.

Failure Case 3: Structural lookup error. Table 29 shows an example with repeated column groups. The question asks who is listed next after “Mrs S. Von Wielligh”. The gold answer is “Mr F.J. Van Heerden”, but the frontier model fails to follow the local table order under the repeated column structure. This reflects a structural lookup error, where the model does not correctly track adjacency within the intended column group.

Analysis. These failure cases show that frontier LLMs can still struggle with table reasoning operations that require faithful grounding to table structure, including numerical comparison, column-based filtering, and local structural lookup. These errors are consistent with the motivation of TABGR, which represents table evidence as structured triples and ranks question-relevant evidence before answer generation.