CoRef: A Collaborative Refinement Framework for Table Question Answering

Anonymous ACL submission

Abstract

Table Question Answering (TQA) enables users to query semi-structured tables using natural language. However, current methods suffer from two key challenges: (i) complex layouts which hinder accurate reasoning and (ii) substantial noise that disrupts table-processing code generation. To address these challenges, we propose CoRef, a collaborative refinement framework. First, CoRef employs a Planner and multiple Table Curators, working alongside a Decision Trace Tree to distribute the burdens of decision-making and table curation across specialized agents while also enabling backtracking when needed. Second, CoRef integrates a Code-Refining Memory module, which iteratively refines table-processing code by learning from compiler feedback. Extensive experiments on three popular TQA datasets demonstrate that CoRef outperforms existing methods (74.2% on WikiTQ, 88.6% on TabFact, and 74.7% on HiTab), validating its effectiveness.

1 Introduction

Tables are a common format for organizing structured information and are widely used in various domains, including Wikipedia (Pasupat and Liang, 2015) and government reports (Cheng et al., 2021). Table question answering (TQA) (Zhang et al., 2024b), which focuses on answering user queries based on semi-structured tables, serves as a cornerstone for information extraction (Wang et al., 2021; Ye et al., 2024) and Retrieval-Augmented AI (Zhao et al., 2024a; Pan et al., 2022). Thus, the ability to effectively query and extract meaningful insights from tables is crucial, whether it's for academic research or industry needs.

Large Language Models (LLMs) (Ouyang et al., 2022) have exhibited remarkable capabilities in responding to queries by leveraging their strong natural language understanding (NLU) and reasoning skills. Building on this, recent studies (Chen, 2022; Zhao et al., 2023; Sui et al., 2024a) have utilized In-context Learning (ICL) (Brown et al., 2020) for TQA, offering an alternative to traditional pre-training methods (Herzig et al., 2020; Wang et al., 2021). However, despite these advantages, such methods face limitations due to the context length constraints of LLMs, making it difficult to process excessively large tables. Moreover, LLMs often struggle with reasoning over numerical data within table cells (Schick et al., 2024). Due to such limitations of LLMs, several studies (Cheng et al., 2022; Cao et al., 2023; Ye et al., 2023; Zhang et al., 2023b) have enhanced LLMs by incorporating external tools (Wang et al., 2024b) and environment-observing techniques such as ReAct (Yao et al., 2022). These LLM-based agent frameworks are thus capable of dynamically making table-processing decisions based on the current state of the table, such as querying specific columns using SQL and performing calculations with Python. This paradigm significantly improves the accuracy of TQA by mitigating the inherent limitations of end-to-end (E2E) methods.

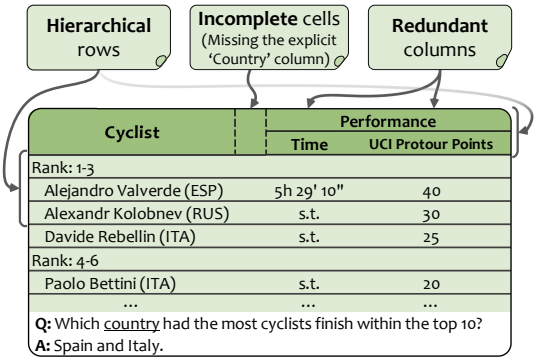

Figure 1: An example of TQA illustrating the characteristics of hierarchical structure, incomplete information and redundant content of a Table.

However, existing agent-based methods still need improvement to address two key challenges in TQA. Firstly, unlike the well-structured, metadata-rich relational database tables found in

Text2SQL (Jin et al., 2022), **semi-structured tables present greater complexity in both layout and content**. As illustrated in Fig. 1, tables frequently contain hierarchical structures, incomplete data, and redundant information. Existing agent-based methods are ill-equipped to address this complexity, as they impose the dual burden of reasoning and table curation on a single agent. In other words, the single agent has to handle intricate decision-making while mastering multiple tools (e.g., SQL Executors) within a limited prompt, thereby violating Simon's principle of bounded rationality (Mintrom, 2015), which posits that a clear division of labor can effectively compensate for the limitations in individual information processing capacity. Secondly, **tables often contain substantial noise**. Existing agent-based methods typically rely on generating code to query these tables. However, such noisy content frequently results in generated code that, despite being logically sound, contains syntax errors that hinder successful execution by external compilers. As illustrated in Fig. 1, elements such as varied punctuation marks and reserved SQL keywords like "Rank" contribute to syntax errors, making it challenging to generate executable table-processing code.

To address these challenges, we propose CoRef, a Collaborative Refinement framework. For the **first** challenge, CoRef introduces a collaborative multi-agent framework, delegating decision-making and table curation to distinct agents. Specifically, CoRef assigns specialized roles: the *Planner* assesses the table's state at each step and decides whether to finalize the answer or delegate table curation tasks to the Table Curators. If delegated, the *Table Curator*s generate and execute the code, then return the refined table to the *Planner* for further reasoning. Additionally, we introduce a Decision Trace Tree that allows backtracking to a previous state upon an incorrect decision, enhancing reasoning and adaptability. For the **second** challenge, given that we adopt the agent-based paradigm that generates code to query tables, we introduce a module named Code-refining Memory (CRM) to ensure that CoRef can generate syntactically correct code in the presence of noisy data. This module utilizes compiler feedback—typically providing information on syntax errors along with explanations—an often overlooked resource in existing TQA methods. By storing the experience gained from previous errors, the module allows the model to refine its capabilities and improve code accuracy over time iteratively. When encountering challenging TQA cases, CoRef can retrieve similar examples from memory, thereby enhancing the reliability of code generation.

Our main contributions include:

- We propose CoRef, a collaborative multi-agent framework for TQA that distributes tasks among specialized agents. Combined with a Decision Trace Tree, CoRef effectively handles complex, noisy tables and generates accurate answers.
- CoRef introduces Code-refining Memory, which stores previous compiler feedback in memory, allowing the model to evolve by learning from past experiences. This helps the model gradually avoid generating code with syntax errors, thereby improving the overall accuracy of CoRef.
- CoRef outperforms SOTA methods in extensive experiments on three public TQA datasets (74.2% on WikiTQ, 88.6% on TabFact, and 74.7% on HiTab), validating its effectiveness.

2 Related Work

2.1 Table Question Answering

Prior to the advent of LLMs, table-related methods (Dong et al., 2022; Jin et al., 2023; Ye et al., 2024) predominantly relied on encoder-only models for downstream tasks. However, these methods required extensive labeled data to achieve optimal performance. With the rise of LLMs and their superior NLU and reasoning, researchers (Zhang et al., 2024b) began integrating them into TQA. LLM-based methods are classified into two categories: E2E methods (Chen, 2022; Wu et al., 2024; Singha et al., 2023; Sui et al., 2024b; Zhao et al., 2023) and those augmented with external tools (Cheng et al., 2022; Ye et al., 2023; Wang et al., 2024c; Nahid and Rafiei, 2024; Zhao et al., 2024b; Zhang et al., 2024a; Abhyankar et al., 2024; Liu et al., 2023; Ji et al., 2024; Mao et al., 2024; Li et al., 2025). E2E methods (Wu et al., 2024; Singha et al., 2023; Sui et al., 2024b; Zhao et al., 2023) involve LLMs directly processing tabular data and queries through their inherent textual reasoning abilities. Nevertheless, E2E methods are constrained by the LLM's inherent weaknesses in numerical computation (Ahn et al., 2024) and the context length constraint, which hampers reasoning over large tables. To overcome E2E limitations, studies (Lu

et al., 2024) use agent-based methods, where LLMs generate intermediate code based on a table, and external executors derive answers by running it. However, these methods rely on single-agent systems and lack effective utilization of compilation tool feedback for iterative self-improvement.

2.2 LLM-based Agents

Due to inherent limitations of LLMs like challenges with numerical computation and factual hallucinations (Schick et al., 2024; Wang et al., 2024b), interest in LLM-powered agents has grown (Wang et al., 2024a; Zhang et al., 2023a; Miyake et al., 2024). Numerous studies have proposed methods to improve agent reasoning, such as Tree-of-Thoughts (ToT) (Yao et al., 2024), ReAct (Yao et al., 2022), Self-refine (Madaan et al., 2024), Reflexion (Shinn et al., 2024), and Meta-CoT (Yoran et al., 2023), iteratively refining agents via feedback. In parallel, other works have explored the evolution of agent capabilities (Sumers et al., 2024; Park et al., 2023; Qian et al., 2024, 2023). Additionally, multi-agent frameworks (Hong et al., 2023; Wu et al., 2023; Chen et al., 2023a) have emerged, introducing agents with distinct roles and collaborative strategies, advancing beyond single-agent systems. Techniques such as Debate (Du et al., 2023; Xiong et al., 2023; Liang et al., 2023; Chen et al., 2023b) enhance collaborative reasoning through critical interactions between agents. Despite these advancements, no current research applies multi-agent frameworks or evolving compiler feedback techniques to TQA tasks.

3 Problem Statement

In the TQA task, each instance is represented as a triplet $\langle t, q, a \rangle$, where $t \in \mathbb{T}$ is the semi-structured table, $q \in \mathbb{Q}$ is the question and $a \in \mathbb{A}$ is the golden answer extracted from the associated table t. The table t consists of a header H and a data section D. The header H contains schema information and is designed for visual communication, often featuring a hierarchical organization common in web pages and business documents, as opposed to database sources (see Fig. 1 for an example).

The question-answering process is defined by a mapping function $\phi : \mathbb{Q} \times \mathbb{T} \mapsto \mathbb{A}$. For E2E methods, the function ϕ is encapsulated by a dedicated model $\mathcal{M}$:

$$a' = \mathcal{M}(q, t), \tag{1}$$

where a' denotes the predicted answer. In contrast, agent-based methods involve the LLM $\mathcal{M}$ generating an intermediate program p, then executed by an external tool $\mathcal{T}$ to output the answer a':

$$p = \mathcal{M}(q, t); \quad a' = \mathcal{T}(p, t). \tag{2}$$

Such an agent-based process may iterate several times, refining the program p and improving the accuracy of the final predicted answer.

4 Methodology

We propose CoRef, a Collaborative Refinement framework for TQA (Fig. 2), where multiple agents collaborate using a Decision Trace Tree and Code-refine Memory to iteratively generate code for extracting answers from tables. The following sections elaborate on the agent roles (§ 4.1), the Decision Trace Tree (§ 4.2), and the agent-tool interaction mechanism (§ 4.3). The pseudocode description of CoRef's workflow can be found in Appx B.1, Alg. 1.

4.1 Role Definition

In this framework, specialized agents work together to refine and extract information from tables, managing complex user inquiries by assigning specific tasks to each agent. These agents include:

Planner: The *Planner* agent functions as the core component within the framework. It receives the table and question from the user and, through iterative refinement of the intermediate tables, ultimately derives the final answer. Each decision-making step involves identifying the necessary type of refinement operation—whether to condense or expand the information—and then designating the appropriate *Table Curator* to implement the operation. Once the table information is adequately refined, the *Planner* leverages its text comprehension abilities to deduce the final answer directly.

Table Curator: The *Table Curator* generates code based on the *Planner*'s decisions, executes it via an external compiler, and returns an intermediate table. If execution fails, it iteratively corrects the code using CRM. If errors persist, it analyzes the issue and reports potential causes to the *Planner*, enabling backtracking and reconsidering the decision-making strategy based on the error feedback. Table Curators are further divided into the Python Curator and the SQL Curator, each with distinct prompts suited to their respective programming language. The Python Curator focuses on flattening hierarchical structures and extracting key information

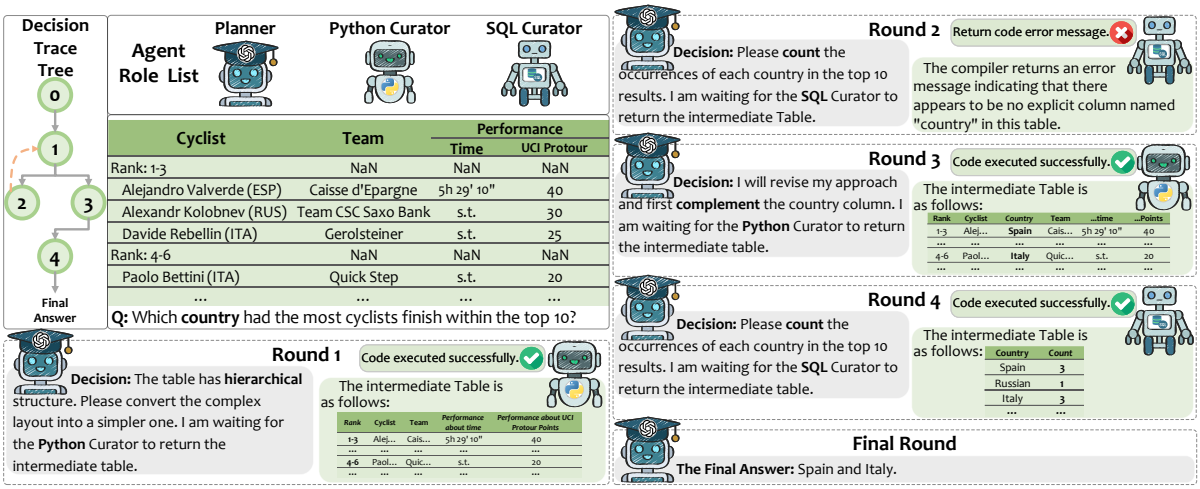

Figure 2: Overview of CoRef. The framework consists of three roles: one *Planner* and two *Table Curator*s, each with distinct responsibilities. Each agent follows a unique prompt defining its role and providing demonstrations. The Planner analyzes the table and assigns tasks, while the *Table Curator*s generate and execute the code using external tools, returning an intermediate table. On the left, the Decision Trace Tree illustrates the *Planner*'s reasoning process, with node numbers corresponding to dialogue rounds. The red dashed arrow represents backtracking from node 2 to node 1 due to an execution error reported from the *SQL Curator*.

with regex, while the SQL Curator excels at retrieving columns and performing aggregations, such as identifying maximum values.

4.2 Decision Trace Tree

LLMs make reasoning errors in complex TQA tasks. The root cause lies in the autoregressive probability paradigm (Zhao et al., 2024c), which inherently lacks the ability to revise previously generated content (Hao et al., 2023) and real-world feedback (Yao et al., 2022). Recent methods that extend inference time (Zhao et al., 2024c), such as ToT (Yao et al., 2024) and SC (Wang et al., 2022), have proven effective in enhancing reasoning. Building on this paradigm, we introduce the Decision Trace Tree, which records the model's reasoning path and enables it to backtrack when receiving error feedback, granting autoregressive LLMs the ability to correct mistakes. As shown in Fig. 2, each node n represents a different table state t_n, where node 0 corresponds to the initial table state t_0. CoRef engages in four dialogue rounds, followed by the final round, where the *Planner* provides the final answer. Consequently, the tree consists of five nodes and an additional node as the final answer. The transition from node n to node $n+1$ in CoRef is formulated as follows:

$$d_n = \mathcal{M}_P(q, t_n), \quad p_n = \mathcal{M}_C(d_n, t_n), \tag{3}$$

$$\mathcal{T}_c(p_n) = \begin{cases} t_{n+1}, & \text{if executed successfully} \\ e_n, & \text{otherwise} \end{cases}. \tag{4}$$

The *Planner* $\mathcal{M}_P$ makes a decision d_n based on the current table state t_n at node n, and the *Curator* $\mathcal{M}_C$ generates the program p_n according to the decision d_n, which is then executed by an external tool executor $\mathcal{T}_c$. If execution succeeds, the updated table state t_{n+1} at node $n+1$ is obtained. If execution fails, an error message e_n is returned. In this case, CoRef backtracks to node n and utilizes the error message e_n from node $n+1$ to attempt a new transition to node $n+2$ by generating a new decision d_{n+1}:

$$d_{n+1} = \mathcal{M}_P(q, t_n, e_n) \tag{5}$$

As illustrated in Fig 2, after backtracking from node 2 to node 1, CoRef then transitions to node 3.

4.3 Code-refining Memory

The low accuracy of previous agent-based methods (Zhang et al., 2023b; Wang et al., 2024c) is partly attributed to their tendency to generate code with minor syntactic errors when handling noisy tables. While logically correct, such errors hinder compilation. To mitigate this, CoRef introduces the Code-Refining Memory (CRM) module, activated when a *Table Curator* encounters a coding error.

LLMs inherently exhibit randomness during the generation process, as illustrated below:

$$p_{\text{LLM}}(y|x) = \prod_{t=1}^{T} p(y_t|x, y_{<t}), \tag{6}$$

where x represents the input and y denotes the generated tokens. As a result, LLM-based methods often adopt SC by allowing LLMs to generate diverse reasoning paths (Chen et al., 2023c) in order to improve accuracy. Observations show that when faced with complex samples, LLMs can successfully generate compilable code approximately 1 or 2 times out of 10 attempts. This indicates that the correct code to solve the TQA samples exists within the model's search space. Therefore, we can increase the likelihood of generating the correct solution by prefixing the appropriate context c (Liu et al., 2021):

$$p_{\text{LLM}}(y|c,x) = \prod_{t=1}^{T} p(y_t|c,x,y_{<t}). \quad (7)$$

Compared to Self-Refine (Madaan et al., 2024; Shinn et al., 2024), where LLMs self-correct based on their own feedback, external tools like compilers provide more precise insights, offering specific explanations for code errors. Consequently, CoRef leverages the feedback from the compiler to help the model correct its generated code. Furthermore, some works (Sumers et al., 2024; Park et al., 2023) have introduced the concept of *evolving*, where agents accumulate experience in memory, allowing them to enhance their capabilities and adapt to complex challenges over time. In line with this, the CRM maintains a memory that stores difficult cases encountered during CoRef's operation, facilitating continuous improvement.

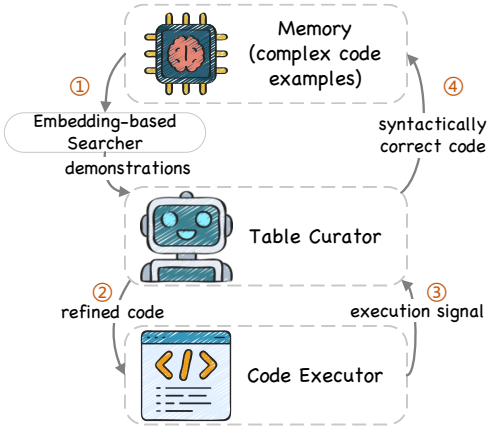

Figure 3: The overview of Code-refining Memory. The *Table Curator* retrieves demos from the memory when handling challenging cases. Successfully compiled code is stored for future reference.

The overview of this module is shown in Fig. 3 while it is also detailed in pseudocode in Appx. B.2. The workflow consists of four steps: 1) When a new instance arrives, the module first retrieves similar examples from the memory using an embedding-based searcher. These examples are challenging cases the model has previously encountered. This step can be formalized as Equation 8:

$$c' = R(x), \quad (8)$$

$$p_{\text{LLM}}(y|c',x) = \prod_{t=1}^{T} p(y_t|c',x,y_{<t}), \quad (9)$$

where R is a function that retrieves examples from the memory that are similar to the current input (line 4 of Alg. 2). Here, c' represents these retrieved examples, including complex cases, corresponding code, and compilation information. 2) The *Table Curator* uses these demonstrations as references to generate code, as described in Equation 9, which is then submitted to the code execution tool $\mathcal{T}_c$. 3) The code execution tool $\mathcal{T}_c$ compiles the code and returns a signal indicating the presence of syntax errors. 4) If the model encounters a compilation error, it iteratively performs self-correction until the code either compiles successfully or exceeds a predefined iteration threshold. Upon success, the syntactically correct code and corresponding sample are stored in memory for future reference. However, if the retry count surpasses the threshold, it suggests an incorrect table manipulation decision—such as attempting to extract country information from a table that lacks a "country" column. In such cases, the *Table Curator* forwards the error message to the *Planner*, aiding in more informed decision-making.

5 Experiments

In this section, we design targeted experiments to address key research questions: **RQ1:** How does the performance of CoRef compare to that of leading single-agent models in terms of the effectiveness and accuracy across various datasets? **RQ2:** How does each proposed sub-module contribute to the overall performance of CoRef? **RQ3:** What errors are associated with methods based on different paradigms? **RQ4:** How does CoRef compare to other baselines regarding time efficiency?

5.1 Datasets & Baselines

We employ three widely studied TQA datasets, with their statistics presented in Table 1. We compare CoRef with the baselines in Table 2, where MixSC (Liu et al., 2023) represents the current SOTA. See Appx. A for details about baselines.

Datasets	Table Info		
	# Test Tables / QA Pairs	Main Domains	Feature
WikiTQ (Pasupat and Liang, 2015)	2108 / 4344	General	Multi-hop Questions
TabFact (Chen et al., 2020)	1695 / 12779	General	Binary Answers
HiTab (Cheng et al., 2021)	538 / 1584	Crime, Health	Hierarchical Layouts

Table 1: Statistics of TQA Datasets.

5.2 Metrics

Due to the inherent randomness of LLMs in generative architectures, achieving an exact match with gold-standard answers can be challenging. For example, given the query "What is the difference in years between constituency 1 and 2?" with the gold answer "4", an LLM-generated response like "4 years" would be marked incorrect under an exact match evaluation. To address this, following prior studies (Cheng et al., 2022; Ye et al., 2023; Liu et al., 2023), we use Semantic Exact Match Accuracy (Cheng et al., 2022) as the evaluation metric across all datasets, comparing predictions with ground truth labels from a semantic perspective.

5.3 Implementation Details

Some methods employ SC and Majority Voting, while others do not. We adhere to their original configurations and distinguish them using '*w/* SC' and '*w/o* SC' in Table 2.

For the underlying LLM of CoRef, we use ChatGPT-3.5-turbo-0613 (GPT-3.5), ChatGPT-4o-mini (4o-mini), and LLaMA-3.1-8B (LLaMA) (Minaee et al., 2024) as the base models. The test set from each dataset is used as the evaluation data, and the prompts are sourced from the training set. We employ AutoGen (Wu et al., 2023) as the multi-agent framework. We refactor the functions that select the next speaker and call external tools in the framework to better align with the current scenario, which is discussed further in Appx. C.2.

5.4 Overall Performance (RQ1)

The comparison results based on GPT-3.5 across three datasets are presented in Table 2. [1]

From Table 2, We can draw the following conclusions: First, CoRef consistently outperformed all baselines, demonstrating clear superiority across all datasets. Notably, for HiTab featuring hierarchical tables, methods relying solely on SQL (e.g., Dater) as an external tool underperformed compared to E2E methods, whereas those integrating Python (e.g., HiTab) showed improved outcomes. CoRef stands out in this context by leveraging the Python *Table Curator* to flatten hierarchical tables, following the step-by-step instructions generated by the Planner. This approach gives a clear advantage over other methods on HiTab. Second, the comparison of different paradigms highlights that the inference with planning (e.g., Chain-of-Table) and external tools (e.g., MixSC) can substantially enhance LLM capabilities——an aspect that CoRef particularly emphasizes. Third, SC is undeniably a straightforward yet highly effective method for improving model performance, as evidenced by the comparison between the w/ SC and w/o SC methods in the table. While CoRef does not explicitly employ SC, it inherently integrates SC-like patterns through SRM and DTT. Finally, the results from H-STAR and MixSC indicate that combining textual and symbolic reasoning significantly boosts LLM accuracy in table reasoning tasks. CoRef can similarly be categorized in this type, enabling different models to specialize in distinct capabilities, thereby achieving a more effective combination of strengths.

[1] Results based on 4o-mini are similar to GPT-3.5, as shown in Appx. D.1. However, the situation differs for LLaMA with weaker coding abilities, as discussed in Appx. D.3.

	Method (GPT-3.5 based)	Acc (%)		
		WIKITQ	TABFACT	HITAB
	LLMs w/o external tools			
	E2E w CoT	51.8	70.5	63.5
	E2E w ReAct	66.4	72.8	68.4
	LLMs w/ external tools			
w/o SC	TabSQLify (Nahid and Rafiei, 2024)	64.7	79.5	58.4
w/o SC	Dater (Ye et al., 2023)	65.0	83.5	54.6
w/o SC	ReAcTable (Zhang et al., 2023b)	65.8	83.1	62.4
w/o SC	Chain-Of-Table (Wang et al., 2024c)	67.3	86.6	58.5
w/o SC	H-STAR (Abhyankar et al., 2024)	68.7	83.7	66.2
w/o SC	PoTable (Mao et al., 2024)	62.7	85.9	-
w/o SC	ALTER (Zhang et al., 2024a)	67.4	84.3	52.5
w/o SC	MixSC (Liu et al., 2023)	64.2	-	68.6
w/ SC	Binder (Cheng et al., 2022)	55.1	85.1	54.6
w/ SC	API-Assited (Cao et al., 2023)	42.8	-	70.0
w/ SC	Dater (Ye et al., 2023)	69.0	85.4	57.1
w/ SC	ReAcTable (Zhang et al., 2023b)	68.0	86.1	67.6
w/ SC	ALTER (Zhang et al., 2024a)	70.4	87.2	54.0
w/ SC	GrahOTTER (Li et al., 2025)	-	-	70.8
w/ SC	MixSC (Liu et al., 2023)	73.7	88.5	72.6
	CoRef	**74.2**	**88.6**	**74.7**

Table 2: Main Results. The evaluation results of CoRef and the baselines on three datasets are grouped by their use of external tools and self-consistency.

5.5 Ablation Studies (RQ2)

We perform ablation studies on all datasets to assess the contributions of key modules to CoRef's overall performance improvement: *w/o Collab* (without collaboration, i.e., planning and coding handled by a single agent), *w/o DTT* (without the DTT for backtracking), and *w/o CRM* (without the

Methods	Acc (%)		
	WIKITQ	TABFACT	HITAB
CoRef	**74.2**	**88.6**	**74.7**
w/o Collab	71.1 (↓ 3.1)	86.8 (↓ 1.8)	69.9 (↓ 4.8)
w/o DTT	71.8 (↓ 2.4)	87.0 (↓ 1.6)	71.8 (↓ 2.9)
w/o CRM	71.9 (↓ 2.3)	86.5 (↓ 2.1)	73.1 (↓ 1.6)

Table 3: Ablation results. It presents the contribution of each module in CoRef, including Collaboration, Decision Trace Tree, and Code-Refining Memory. Values in parentheses represent the drop in accuracy.

Code-refining Memory for iterative correction).

The results are shown in Table 3. First, Collaborative agents have a significant impact on HiTab (4.8%) and WikiTQ (3.1%), validating its efficacy in orchestrating strategic decision-making for simplifying intricate table layout or content. Second, facing WikiTQ requires multi-hop reasoning and HiTab demands complex numerical computations, DTT achieves remarkable improvements (2.4% and 2.9% respectively), evidencing its enhanced reasoning capacity for diverse analytical challenges. Finally, CRM effectively mitigates data noise issues prevalent in WikiTQ (2.3%) and TabFact (2.1%), proving critical for generating syntactically robust and semantically precise code interpretations.

5.6 Error Studies (RQ3)

We conduct a comprehensive error analysis of CoRef alongside two representative methods from different paradigms, E2E /w ReAct and MixSC, supplemented by a case study.

5.6.1 Statistical Error Analysis

We sampled 100 cases from HiTab where E2E w/ ReAct failed, while also recording the responses of MixSC and CoRef for these instances. Each erroneous case was manually evaluated to determine the type of error, which we categorized into five types. **Misinterpretation** occurs when the LLM misunderstands the semantics of the question or table, such as failing to recognize hierarchical relationships or misinterpreting the entity referenced in the question. **Coding Errors** involve incorrect code generation, including syntax errors or code that does not accurately reflect decision-making semantics, preventing the retrieval of the correct intermediate table. **Misalignment Issues** arise when the generated output is conceptually correct but does not conform to the required format specified in the instructions. **Reasoning Inconsistency** refers to

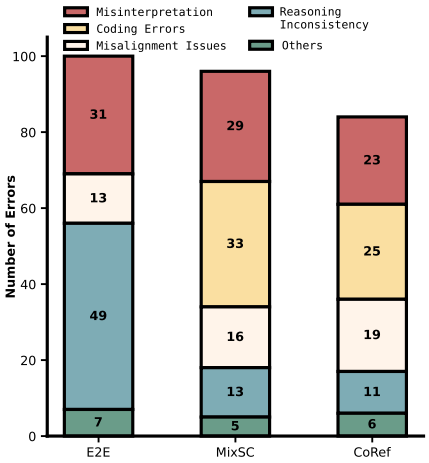

Figure 4: Analysis of error types in 100 samples from HiTab where E2E *w/* ReAct fails.

cases where the LLM's reasoning path is inconsistent, such as deciding to sum all numbers in step two but then providing incorrect values in step three due to its limited numerical calculation capabilities. **Others** include dataset annotation issues, network error or instances where the model exceeds its prescribed iteration limit.

Since the selected samples are those where the E2E method itself makes errors, this indicates that these questions are relatively challenging. As illustrated in Fig. 4, CoRef significantly improves table comprehension by allowing the *Planner* and *Curator*s to collaborate in systematically simplifying the table layout. For the E2E method, the error type with the highest proportion is Reasoning Inconsistency, which is due to LLMs' weakness in numerical computation (Imani et al., 2023; Chen et al., 2024). Although the LLM correctly plans the reasoning path, it fails to provide the final numerically-related answer. From the Fig. 4, both MixSC and CoRef exhibit a substantial reduction in the Reasoning Inconsistency, suggesting that their reliance on external tools effectively mitigates weaknesses in numerical computation. Unfortunately, introducing external tools also increases the possibility of coding errors. CoRef effectively reduces this type of error after introducing CRM. Additionally, CoRef shows a minor increase in Misalignment Issues, which might be due to the context size raising while decreasing the model's ability to adhere to the output format. Appx. C.1 discusses this further.

5.6.2 Case Study

To highlight the differences between CoRef and existing methods in handling complex table layouts, we conducted a case study comparing E2E *w/* ReAct, MixSC, and CoRef. Using a HiTab sample

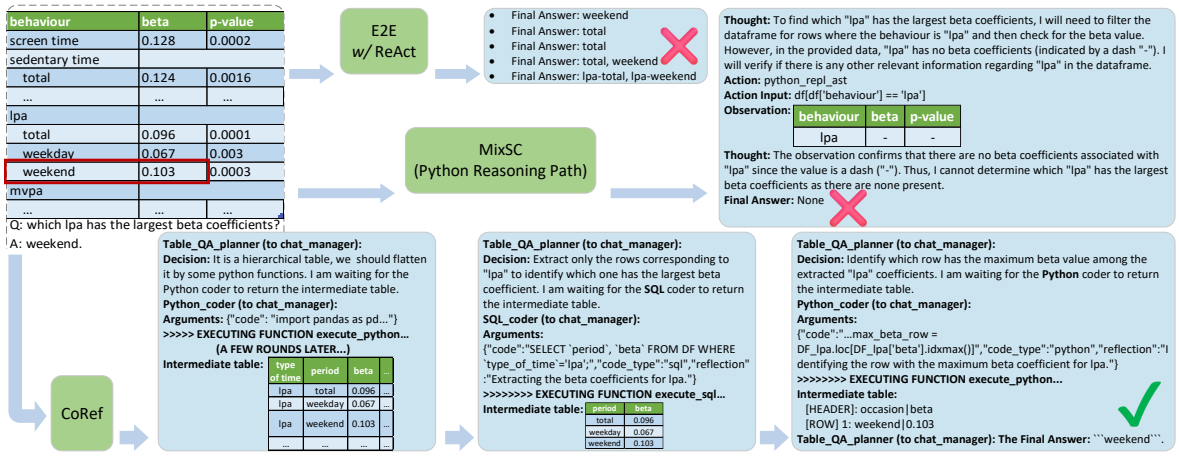

Figure 5: Showcasing the results of E2E, MixSC, and CoRef on the same instance from the HiTab dataset. This instance is challenging due to the hierarchical relationships present in the 'behaviour' column.

with hierarchical data in the first column, we analyze the reasoning processes of these three methods. As shown in Fig. 5, CoRef demonstrates superior reasoning capabilities when handling this complex table. For E2E *w/* ReAct, despite receiving explicit prompts for iterative question formulation under the SC setting, the model provided final answers in all five attempts—none of which were correct. For MixSC, we focus on the outcome of its Python Reasoning Path (as the results from its Textual Reasoning Path closely mirror those of E2E *w/* ReAct and are therefore omitted). It is evident that MixSC failed to accurately grasp the hierarchical structure of the table when generating Python code, leading to incorrect results. In contrast, CoRef successfully refined the table through iterative curation, ultimately delivering the correct answer by relying on a highly streamlined intermediate table.

5.7 Efficiency Analysis (RQ4)

We evaluate efficiency by analyzing the total number of LLM call rounds on the WikiTQ dataset, as this metric is less affected by network instability than total runtime. Table 4 compares CoRef with baselines, detailing the number of rounds per operation and total LLM calls. Notably, for methods using SC (marked with ♠), sampling is fixed at 5 (consistent with the original setting), leading to a five-fold increase in total rounds. Despite its multi-agent framework, CoRef maintains a total LLM call count comparable to the current SOTA approach, MixSC. For locally deployed model runtime comparisons, see Appx. D, where results indicate that CoRef and SOTA methods exhibit nearly identical time efficiency.

Method	Detailed Operations	Total Rounds
E2E *w/* ReAct♠	Query: 1-3	5-15
Binder♠	Neural SQL: 10	50
DATER♠	Decompose Table: 8 Generate Cloze & SQL: 8 Query: 4	100
Chain-of-Table	Dynamic Plan ≤ 5 Generate Args ≤ 19 Query: 1	≤ 25
ReAcTable♠	Plan&Action: ≈ 3	≈ 15
TabSQLify	Table Decompose: 1 Query: 1	2
H-STAR	Row / Column Retrieval: 4-8 Query: 2	6-10
ALTER♠	Table / Query Augmenting: 5 Query: 1	30
MixSC♠	Normalization: 1 Direct Reasoning: 1 Python Reasoning: ≤ 5	≤ 35
CoRef	*Planner*: ≤ 10 *Table Curator* (SQL): ≤ 10 *Table Curator* (Python): ≤ 10	≤ 30

Table 4: Generated sample counts for different methods. SC-based methods (♠) include the total rounds from five model runs.

6 Conclusion

In this paper, we introduce CoRef, a novel multi-agent framework designed to tackle the challenges of TQA. CoRef is built upon three key modules: Collaborative Agents, Decision Trace Tree, and Code-Refining Memory. These components effectively address the limitations of existing LLM-based methods, particularly in handling complex and noisy web tables. Experimental results on three widely studied public TQA datasets demonstrate that CoRef surpasses SOTA methods, underscoring its potential to advance TQA systems.

For future work, we aim to explore more modality-specific tables and test-time scaling in TQA, further enhancing its scalability and applicability.

Limitations

While our approach enhances LLM performance on TQA by incorporating three additional modules, several limitations remain.

Inference Efficiency: Although extending reasoning time is an emerging trend, it inevitably impacts efficiency. Compared to directly prompting the LLM for an answer, our model incurs higher computational costs and longer inference time.

Tree-Based Reasoning Constraints: We employ a tree structure to assist in reasoning, yet its efficiency and performance remain inferior to Monte Carlo Tree Search (MCTS), highlighting a promising direction for future research.

Dependence on LLM Capabilities: Our method relies heavily on the LLM's inherent logical reasoning and code generation abilities. Consequently, its effectiveness may be limited when applied to weaker LLMs, yielding only marginal improvements.

References

Nikhil Abhyankar, Vivek Gupta, Dan Roth, and Chandan K Reddy. 2024. H-star: Llm-driven hybrid sql-text adaptive reasoning on tables. *arXiv preprint arXiv:2407.05952*.

Janice Ahn, Rishu Verma, Renze Lou, Di Liu, Rui Zhang, and Wenpeng Yin. 2024. Large language models for mathematical reasoning: Progresses and challenges. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: Student Research Workshop*, pages 225–237, St. Julian's, Malta. Association for Computational Linguistics.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, T. J. Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeff Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. *ArXiv*, abs/2005.14165.

Yihan Cao, Shuyi Chen, Ryan Liu, Zhiruo Wang, and Daniel Fried. 2023. Api-assisted code generation for question answering on varied table structures. *ArXiv*, abs/2310.14687.

Guangyao Chen, Siwei Dong, Yu Shu, Ge Zhang, Jaward Sesay, Börje F. Karlsson, Jie Fu, and Yemin Shi. 2023a. Autoagents: A framework for automatic agent generation. *ArXiv*, abs/2309.17288.

Justin Chih-Yao Chen, Swarnadeep Saha, and Mohit Bansal. 2023b. Reconcile: Round-table conference improves reasoning via consensus among diverse llms. *arXiv preprint arXiv:2309.13007*.

Justin Chih-Yao Chen, Swarnadeep Saha, and Mohit Bansal. 2023c. Reconcile: Round-table conference improves reasoning via consensus among diverse llms. *arXiv preprint arXiv:2309.13007*.

Wenhu Chen. 2022. Large language models are few(1)-shot table reasoners. *ArXiv*, abs/2210.06710.

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

Zui Chen, Yezeng Chen, Jiaqi Han, Zhijie Huang, Ji Qi, and Yi Zhou. 2024. An empirical study of data ability boundary in llms' math reasoning. *arXiv preprint arXiv:2403.00799*.

Zhoujun Cheng, Haoyu Dong, Zhiruo Wang, Ran Jia, Jiaqi Guo, Yan Gao, Shi Han, Jian-Guang Lou, and Dongmei Zhang. 2021. Hitab: A hierarchical table

dataset for question answering and natural language generation. *ArXiv*, abs/2108.06712.

Zhoujun Cheng, Tianbao Xie, Peng Shi, Chengzu Li, R.K. Nadkarni, Yushi Hu, Caiming Xiong, Dragomir R. Radev, Marilyn Ostendorf, Luke Zettlemoyer, Noah A. Smith, and Tao Yu. 2022. Binding language models in symbolic languages. *ArXiv*, abs/2210.02875.

Haoyu Dong, Zhoujun Cheng, Xinyi He, Mengyu Zhou, Anda Zhou, Fan Zhou, Ao Liu, Shi Han, and Dongmei Zhang. 2022. Table pre-training: A survey on model architectures, pre-training objectives, and downstream tasks. *arXiv preprint arXiv:2201.09745*.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. 2023. Improving factuality and reasoning in language models through multiagent debate. *arXiv preprint arXiv:2305.14325*.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. 2023. Reasoning with language model is planning with world model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 8154–8173.

Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Martin Eisenschlos. 2020. Tapas: Weakly supervised table parsing via pre-training. *ArXiv*, abs/2004.02349.

Sirui Hong, Xiawu Zheng, Jonathan P. Chen, Yuheng Cheng, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zi Hen Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, and Chenglin Wu. 2023. Metagpt: Meta programming for multi-agent collaborative framework. *ArXiv*, abs/2308.00352.

Shima Imani, Liang Du, and Harsh Shrivastava. 2023. Mathprompter: Mathematical reasoning using large language models. *arXiv preprint arXiv:2303.05398*.

Deyi Ji, Lanyun Zhu, Siqi Gao, Peng Xu, Hongtao Lu, Jieping Ye, and Feng Zhao. 2024. Tree-of-table: Unleashing the power of llms for enhanced large-scale table understanding. *arXiv preprint arXiv:2411.08516*.

Nengzheng Jin, Joanna Siebert, Dongfang Li, and Qingcai Chen. 2022. A survey on table question answering: recent advances. In *China Conference on Knowledge Graph and Semantic Computing*, pages 174–186. Springer.

Rihui Jin, Jianan Wang, Wei Tan, Yongrui Chen, Guilin Qi, and Wang Hao. 2023. Tabprompt: Graph-based pre-training and prompting for few-shot table understanding. In *Conference on Empirical Methods in Natural Language Processing*.

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

Tian Liang, Zhiwei He, Wenxiang Jiao, Xing Wang, Yan Wang, Rui Wang, Yujiu Yang, Zhaopeng Tu, and Shuming Shi. 2023. Encouraging divergent thinking in large language models through multi-agent debate. *arXiv preprint arXiv:2305.19118*.

Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. 2021. What makes good in-context examples for gpt-3? *arXiv preprint arXiv:2101.06804*.

Tianyang Liu, Fei Wang, and Muhao Chen. 2023. Rethinking tabular data understanding with large language models. *arXiv preprint arXiv:2312.16702*.

Weizheng Lu, Jiaming Zhang, Jing Zhang, and Yueguo Chen. 2024. Large language model for table processing: A survey. *arXiv preprint arXiv:2402.05121*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. 2024. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36.

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

Shervin Minaee, Tomá Mikolov, Narjes Nikzad, Meysam Asgari Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. 2024. Large language models: A survey. *ArXiv*, abs/2402.06196.

Michael Mintrom. 2015. 12Herbert A. Simon, Administrative Behavior: A Study of Decision-Making Processes in Administrative Organization. In *The Oxford Handbook of Classics in Public Policy and Administration*. Oxford University Press.

Kentaro Miyake, Hiroyoshi Ito, Christos Faloutsos, Hirotomo Matsumoto, and Atsuyuki Morishima. 2024. Netevolve: Social network forecasting using multi-agent reinforcement learning with interpretable features. *Proceedings of the ACM on Web Conference 2024*.

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

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744.

Feifei Pan, Mustafa Canim, Michael Glass, Alfio Gliozzo, and James Hendler. 2022. End-to-end table question answering via retrieval-augmented generation. *arXiv preprint arXiv:2203.16714*.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, UIST '23, New York, NY, USA. Association for Computing Machinery.

Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables. In *Annual Meeting of the Association for Computational Linguistics*.

Cheng Qian, Yufan Dang, Jiahao Li, Wei Liu, Weize Chen, Cheng Yang, Zhiyuan Liu, and Maosong Sun. 2023. Experiential co-learning of software-developing agents. In *Annual Meeting of the Association for Computational Linguistics*.

Cheng Qian, Jiahao Li, Yufan Dang, Wei Liu, Yifei Wang, Zihao Xie, Weize Chen, Cheng Yang, Yingli Zhang, Zhiyuan Liu, and Maosong Sun. 2024. Iterative experience refinement of software-developing agents. *ArXiv*, abs/2405.04219.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2024. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2024. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36.

Ananya Singha, José Cambronero, Sumit Gulwani, Vu Le, and Chris Parnin. 2023. Tabular representation, noisy operators, and impacts on table structure understanding tasks in llms. *Preprint*, arXiv:2310.10358.

Yuan Sui, Mengyu Zhou, Mingjie Zhou, Shi Han, and Dongmei Zhang. 2024a. Table meets llm: Can large language models understand structured table data? a benchmark and empirical study. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*, pages 645–654.

Yuan Sui, Mengyu Zhou, Mingjie Zhou, Shi Han, and Dongmei Zhang. 2024b. Table meets llm: Can large language models understand structured table data? a benchmark and empirical study. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*, pages 645–654.

Theodore R. Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L. Griffiths. 2024. Cognitive architectures for language agents. *Preprint*, arXiv:2309.02427.

Daheng Wang, Prashant Shiralkar, Colin Lockard, Binxuan Huang, Xin Dong, and Meng Jiang. 2021. Tcn: Table convolutional network for web table interpretation. *Proceedings of the Web Conference 2021*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

Zheng Wang, Bingzheng Gan, and Wei Shi. 2024a. Multimodal query suggestion with multi-agent reinforcement learning from human feedback. *Proceedings of the ACM on Web Conference 2024*.

Zhiruo Wang, Zhoujun Cheng, Hao Zhu, Daniel Fried, and Graham Neubig. 2024b. What are tools anyway? a survey from the language model perspective. *arXiv preprint arXiv:2403.15452*.

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. 2024c. Chain-of-table: Evolving tables in the reasoning chain for table understanding. *arXiv preprint arXiv:2401.04398*.

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

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. 2023. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. *arXiv preprint arXiv:2308.08155*.

Xianjie Wu, Jian Yang, Linzheng Chai, Ge Zhang, Jiaheng Liu, Xinrun Du, Di Liang, Daixin Shu, Xianfu Cheng, Tianzhen Sun, et al. 2024. Tablebench: A comprehensive and complex benchmark for table question answering. *arXiv preprint arXiv:2408.09174*.

Kai Xiong, Xiao Ding, Yixin Cao, Ting Liu, and Bing Qin. 2023. Examining inter-consistency of large language models collaboration: An in-depth analysis via debate. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 7572–7590.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2024. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models.

Chao Ye, Guoshan Lu, Haobo Wang, Liyao Li, Sai Wu, Gang Chen, and Junbo Zhao. 2024. Towards cross-table masked pretraining for web data mining. In *Proceedings of the ACM on Web Conference 2024*, pages 4449–4459.

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.

Ori Yoran, Tomer Wolfson, Ben Bogin, Uri Katz, Daniel Deutch, and Jonathan Berant. 2023. Answering questions by meta-reasoning over multiple chains of thought. *arXiv preprint arXiv:2304.13007*.

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

Junjie Zhang, Yupeng Hou, Ruobing Xie, Wenqi Sun, Julian McAuley, Wayne Xin Zhao, Leyu Lin, and Ji rong Wen. 2023a. Agentcf: Collaborative learning with autonomous language agents for recommender systems. *Proceedings of the ACM on Web Conference 2024*.

Xuanliang Zhang, Dingzirui Wang, Longxu Dou, Qingfu Zhu, and Wanxiang Che. 2024b. A survey of table reasoning with large language models. *arXiv preprint arXiv:2402.08259*.

Yunjia Zhang, Jordan Henkel, Avrilia Floratou, Joyce Cahoon, Shaleen Deep, and Jignesh M. Patel. 2023b. Reactable: Enhancing react for table question answering. *ArXiv*, abs/2310.00815.

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*.

Penghao Zhao, Hailin Zhang, Qinhan Yu, Zhengren Wang, Yunteng Geng, Fangcheng Fu, Ling Yang, Wentao Zhang, and Bin Cui. 2024a. Retrieval-augmented generation for ai-generated content: A survey. *arXiv preprint arXiv:2402.19473*.

Yilun Zhao, Lyuhao Chen, Arman Cohan, and Chen Zhao. 2024b. TaPERA: Enhancing faithfulness and interpretability in long-form table QA by content planning and execution-based reasoning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12824–12840, Bangkok, Thailand. Association for Computational Linguistics.

Yu Zhao, Huifeng Yin, Bo Zeng, Hao Wang, Tianqi Shi, Chenyang Lyu, Longyue Wang, Weihua Luo, and Kaifu Zhang. 2024c. Marco-o1: Towards open reasoning models for open-ended solutions. *arXiv preprint arXiv:2411.14405*.

A Baselines

We compare the proposed method with the following baselines:

• **E2E *w/* CoT** (Chen, 2022) equips LLMs with ICL(Brown et al., 2020), utilizing CoT (Wei et al., 2022) reasoning to enable final answer generation.

• **E2E *w/* ReAct** (Yao et al., 2022) enables LLMs to iteratively generate outputs based on the current state of the environment, facilitating dynamic decision-making.

• **Binder** (Cheng et al., 2022) pioneers to employ the agent into WebTQA, which introduces the NeuralSQL to handle irregular Web tables.

• **API-Assited** (Cao et al., 2023) enhances comprehension of hierarchical structures by incorporating a tree-structured table representation.

• **Dater** (Ye et al., 2023) allows the model to progressively decompose complex tables into smaller components during the decision-making process, ultimately deriving the final answer.

• **TabSQLify** (Nahid and Rafiei, 2024) builds on Dater by leveraging SQL queries to manipulate and refine table data.

• **Chain-Of-Table** (Wang et al., 2024c) extends the CoT method to tabular data by transforming input tables and guiding the LLM through intermediate table states to improve reasoning accuracy.

• **ReAcTable** (Zhang et al., 2023b) adopts the ReAct framework (Yao et al., 2022), combining step-by-step reasoning with code execution, generating intermediate tables, and employing a majority voting mechanism.

• **H-STAR** (Abhyankar et al., 2024) introduces a two-step approach that combines table extraction with adaptive reasoning, effectively integrating symbolic (SQL) and semantic (text) methods.

• **PoTable** (Mao et al., 2024) is a novel table-based reasoning method that integrates an LLM-driven operation planner with a Python interpreter, enabling human-like logical stage splits and open-world operations for enhanced accuracy and explainability in structured table analysis.

• **ALTER** (Zhang et al., 2024a) seeks to unlock the latent potential of NL queries (via a query enhancer) and tables (via a table enhancer).

• **MixSC** (Liu et al., 2023) aggregates multiple reasoning pathways (textual and symbolic reasoning) and currently represents the SOTA.

B Algorithms

B.1 Pseudocode of Multiagent Workflow

See Alg. 1.

B.2 Pseudocode of Code-refining Memory

See Alg. 2.

C Discussion

This section discusses some interesting phenomena we encountered, which are worth further explored in the future.

C.1 About the Misalignment Issues.

In Section 5.6, we classify certain types of model errors as "Misalignment Issues". For example, in response to the question in Fig. 1, the model might produce the misaligned output, "Spain and Italy had the most cyclists finish within the top 10," despite the prompt explicitly instructing the use of the simplest language possible. We believe that addressing the model's performance from this perspective may constitute the most efficient approach to improving its overall accuracy, as these errors do not stem from limitations in the model's reasoning capabilities. We further analyzed the reasoning path lengths associated with incorrect answers across the three methods illustrated in Fig. 5. Our findings indicate that samples exhibiting "Misalignment Issues" tend to have longer reasoning paths than the average. This observation suggests that extended contexts may lead the model to lose track of the initial formatting instructions. As shown in Fig. 5, such errors are more prevalent in our method, likely due to the fact that each agent must reference the conversation history when generating responses, which typically results in longer contextual inputs compared to other methods.

C.2 About the Table Curator Selection.

For the *Table Curator* selection strategy, rather than using the 'AUTO' mode employed by most multi-agent frameworks—which relies on an additional LLM-based agent to choose the *Table Curator* based on conversation history—we adopt a rule-based approach. This mode significantly increases the number of LLM calls and exhibits lower accuracy compared to rule-based methods, especially in less complex task scenarios like TQA. In our framework, the *Planner* explicitly designates which *Table Curator* (Python Curator or SQL Curator) should execute the current decision, with string matching incorporated as part of the rule-based method. Additionally, if the number of LLM calls exceeds a predefined round limit r, the answer will be directly returned.

While rigid and limited rules may be inadequate for intricate scenarios, they are largely effective in simpler contexts, such as TQA. In these cases, pre-defined rules can adequately cover the majority of situations, leading to a 5% improvement in accuracy on the WikiTQ dataset compared to the auto mode. Furthermore, the rule-based approach requires the *Planner* to explicitly designate the *Table Curator*, such as the SQL *Table Curator*, which may also prompt the *Planner* to evaluate the feasibility of the current decision.

D Supplementary Experiments

D.1 Supplementary Experiment One

To validate the robustness of our method, Table 5 presents a comparison of different approaches using ChatGPT-4o-mini (4o-mini). The baselines for comparison include the top two methods that achieved the best performance on each dataset based on GPT-3.5, as well as E2E with ReAct. We can draw the following conclusions: First, CoRef still performs the best. Second, As can be seen from the table, methods based on 4o-mini generally outperform those based on GPT-3.5, indicating that the base model's capabilities have a significant impact on the performance of the method. Third, Methods utilizing the model's code generation capabilities still outperform those that rely directly on the model's text understanding abilities.

D.2 Supplementary Experiment Two

Table 6 presents the results, which compares the performance of three representative methods from different paradigms on the open-source LLaMA-3.1-8B model, whose capabilities are not as powerful as the closed-source ChatGPT-4o-mini and ChatGPT-3.5, especially in terms of programming abilities. Since the experiment is conducted locally, we can measure a stable runtime as a reference for efficiency. For evaluation, we select the first 200 samples from each dataset. Both E2E and MixSC are set to self-consistency runs = 5.

The detailed differences in model configurations across the three methods are as follows: E2E w/ ReAct: Since it does not involve code generation, we directly use LLaMA-3.1-8B-Instruct as the base model for testing. MixSC: Utilizing LLaMA-3.1-

Algorithm 1 Multiagent Workflow

Input: Table t, Question q, Round limit r, *Planner* agent $\mathcal{M}_P$, *Table Curator* agent $\mathcal{M}_C$, Code execution tool $\mathcal{T}_c$, Decision Trace Tree D_T
Output: Predicted answer a'

```
curRound ← −1
parentNode ← −1
tracedNode ← 0
tracebackNeeded ← True
while curRound < r do
    curRound ← curRound + 1
    D_T.linkAToB(curRound, parentNode) # The parent node of curRound is numbered parentNode
    if tracebackNeeded is True then # Initiating traceback
        # Retrieve the previous node state from the Decision Trace Tree
        # lastOutput contains the compiler's feedback from the previous round
        t, feedback ← D_T.getNodeState(tracedNode)
        de ← M_P.makeDecision(q, t, feedback)
    else
        de ← M_P.makeDecision(q, t)
    end if
    tracebackNeeded ← False
    if requiresToolCall(de) is True then # Invoke an external tool if needed
        program ← M_C.generateProgram(de, t)
        isCorrect, feedback, t ← T_c.execute(program)
        if isCorrect is False then
            program ← refineCode(de, (t, q), M_C) # Attempt to correct the code using CRM
            isCorrect, feedback, t ← T_c.execute(program)
            if isCorrect is False then # Failed to correct the code, initiating traceback
                tracebackNeeded ← True
                parentNode ← D_T.getParentNode(curRound)
                D_T.store(feedback, parentNode) # Store execMsg for future reference
                tracedNode ← parentNode
                continue
            end if
        end if
        D_T.storeNodeState(curRound, t)
        parentNode ← curRound
    else
        return de # de' is the final answer
    end if
end while
return de
```

Method	WikiTQ	TabFact	HiTab
ChatGPT-4o-mini			
E2E w ReAct	67.1	76.6	69.1
ALTER	70.7	88.9	-
GraphOTTER	-	-	74.5
MixSC	75.8	89.5	73.9
CoRef	**76.3**	**90.0**	**76.8**

Table 5: Experimental results of the best-performing methods from each of the three paradigms based on LLaMA-3.1-8B.

8B-Instruct for textual reasoning and LLaMA-3.1-8B-Code for symbolic reasoning. The final answer is determined by aggregating the results from both reasoning paths. CoRef: The *Planner* agent is based on LLaMA-3.1-8B-Instruct, while all *Curators* use LLaMA-3.1-8B-Code.

Method	WikiTQ	TabFact	HiTab	Avg. Time/Sample
LLaMA-3.1-8B				
E2E w ReAct	**34.7**	53.2	48.0	2.5 s
MixSC	31.6	54.7	44.6	10.0 s
CoRef	33.1	**55.2**	47.7	9.2 s

Table 6: Experimental results of the best-performing methods from each of the three paradigms based on LLaMA-3.1-8B.

Observation: In time efficiency, CoRef exhibits a minor advantage compared to the current SOTA MixSC. Regarding accuracy, on the more challenging WikiTQ and HiTab datasets, both existing SOTA methods and CoRef underperform compared to the end-to-end approach. In contrast, TabFact, with its more straightforward table structure, enables the 8B model to achieve higher code accuracy. **Analysis**: 8B models struggle to generate compilable code when dealing with com-

Method	WikiTQ	TabFact	HiTab
MixSC w/ SC	72.7 ± 0.35	88.6 ± 0.21	72.6 ± 0.57
CoRef	**73.4** ± 0.64	**89.1** ± 0.36	**73.4** ± 0.61

Table 7: The mean and standard deviation of five test runs on 200 samples from the HiTab dataset based on GPT-3.5.

plex tables or questions. This explains why high-performing table question-answering methods predominantly rely on GPT-3.5 or even better ones, as symbolic paradigm approaches heavily depend on the model's strong coding capabilities.

D.3 Supplementary Experiment Three

Supplementary experiment two compares MixSC (SOTA) and CoRef by selecting the first 200 samples from each dataset and repeating the tests five times using GPT-3.5. Despite some variations, CoRef consistently outperformed MixSC, shown in Table 7.

E More Case Studies

More case studies of CoRef are shown in Fig. 6, Fig. 7 and Fig. 8 .

Figure 6: A demo from WikiTQ. In this example, *Planner* (referred to as Table_QA_Planner in the figure) can derive the answer from the returned intermediate table with the assistance of the SQL *Table Curator* (referred to as SQL_coder in the figure) in just one iteration.

Algorithm 2 Code-Refining Memory (CRM)

Input: Problematic intermediate program p, Table t, Question q, Round limit r, *Table Curator* agent $\mathcal{M}_C$, Memory module $\mathcal{M}_e$, Code execution tool $\mathcal{T}_c$
Output: Refined program p_{new}, the feedback from executor `feedback`

 # Numbers in comments correspond to those in Fig. 3
`curRound` $\leftarrow 0$
while `curRound` $< r$ **do**
  $c' \leftarrow \mathcal{M}_e$.`search`$(p, (t, q))$ *# ① Retrieve relevant cases from memory*
  $p_{new} \leftarrow \mathcal{M}_C$.`makeDecision`$(c', (t, q))$ *# ② Generate a refined program*
  `isCorrect`, `feedback` $\leftarrow \mathcal{T}_c$.`execute`(p_{new}) *# ③ Execute the refined program*
  if `isCorrect` **is True then**
    $\mathcal{M}_e$.`store`$(p_{new}, (t, q),$ `feedback`$)$ *# ④ Store successful refinement in memory*
    break
  end if
end while
 # The feedback includes execution details of the corrected code, accounting for cases where execution still fails after maximum refinement attempts.
return p_{new}, feedback

Figure 7: A demo from TabFact. In this example, *Planner* (referred to as Table_QA_Planner in the figure) assigned both the SQL *Table Curator* (referred to as SQL_coder in the figure) and Python *Table Curator* (referred to as Python_coder in the figure) to generate code, and obtained the final answer after the two round of code generation.

Deliver_Proxy (to chat_manager):

Table 'DF':

[Caption]: agri-food industry sub-groups for workers aged 15 years and over, two agricultural regions of ontario, 2011

sub-groups of the agri-food industry | eastern ontario | eastern ontario northern ontario | northern ontario

sub-groups of the agri-food industry | french-language workers other workers | french-language workers other workers

sub-groups of the agri-food industry | percent | percent | percent | percent

input and service supply|2.9|2.1|2.9|1.3

food, beverage, and tobacco processing| 9.7|6.0|3.0|3.3

food retail and wholesale|35.3| 31.3| 39.1|37.3

food service|52.1|60.6|55.0| 58.1

Query: in eastern ontario, what percent of french-language workers have worked in the restaurant and food services sector?
Next speaker: Table_QA_Planner

Table_QA_Planner (to chat_manager):

Decision: There is Hierachical Information within the table. Please help me flatten it. I am waiting for the Python coder to return the intermediate table.
Python_coder (to chat_manager):
Header_Row_Flattened_1: {'percent of french-lang workers in eastern ontario'}{'percent of other workers in eastern ontario'}

Header_Row_Flattened_2: {'percent of french-lang workers in northern ontario'}{'percent of other workers in northern ontario'}
Deliver_proxy (to chat_manager):
Table 'DF':

[Caption]: agri-food industry sub-groups for workers aged 15 years and over, two agricultural regions of ontario, 2011

[HEADER]:sub_groups_of_the_agri_food_industry|percent_of_french_lang_workers_in_eastern_ontario|percent_of_other_workers_in_eastern_ontario|percent_of_french_lang_workers_in_northern_ontario|percent_of_other_workers_in_northern_ontario

---

[ROW] 1: input and service supply|2.9|2.1|2.9|1.3

[ROW] 2: food, beverage, and tobacco processing|9.7|6.0|3.0|3.3

[ROW] 3: food retail and wholesale|35.3|31.3|39.1|37.3

[ROW] 4: food service|52.1|60.6|55.0|58.1

Query: in eastern ontario, what percent of french-language workers have worked in the restaurant and food services sector?

Next speaker: Table_QA_Planner
Table_QA_Planner (to chat_manager):
Decision: Please extract only the row for the "food service" sub-group and keep the relevant columns for the percent of French-language workers in Eastern Ontario. I am waiting for the SQL coder to return the intermediate table.

Next speaker: SQL_coder

SQL_coder (to chat_manager):

***** Suggested tool call (call_hInK88NqchNlL5flmGir9JEQ): execute_sql *****

Arguments:

{"code":"SELECT `percent_of_french_lang_workers_in_eastern_ontario` FROM DF WHERE `sub_groups_of_the_agri_food_industry`='food service';","code_type":"sql","reflection":"Extracting the percent of French-language workers in Eastern Ontario for the food service sub-group."}

Next speaker: code_executor

>>>>>>>> EXECUTING FUNCTION execute_sql...

***** Response from calling tool (call_hInK88NqchNlL5flmGir9JEQ) *****

{"Intermediate table 'DF':\n": "[HEADER]: percent_of_french_lang_workers_in_eastern_ontario\n---\n[ROW] 1: 52.1"}

Next speaker: Table_QA_Planner

Table_QA_Planner (to chat_manager):

The Final Answer: ```52.1%```.

Next speaker: deliver_proxy

deliver_proxy (to chat_manager):

The Final Answer: ```52.1%```.

---

gold label: 52.1

Figure 8: A demo from HiTab. In this example, *Planner* (referred to as Table_QA_Planner in the figure) first instructs the Python *Table Curator* (referred to as Python_coder in the figure) to flatten the hierarchical structure of the table and then relies on SQL *Table Curator* (referred to as SQL_coder in the figure) to obtain the final answer.