Efficient multi-agent collaboration with tool use for online planning in complex table question answering

Wei Zhou, Mohsen Mesgar, Annemarie Friedrich, Heike Adel

Angaben zur Veröffentlichung / Publication details:

Zhou, Wei, Mohsen Mesgar, Annemarie Friedrich, and Heike Adel. 2025. "Efficient multi-agent collaboration with tool use for online planning in complex table question answering." In *Findings of the Association for Computational Linguistics: NAACL 2025 - Annual Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics, 29 April - 4 May 2025, Albuquerque, NM, USA*, edited by Luis Chiruzzo, Alan Ritter, and Lu Wang, 945–68. Stroudsburg, PA: Association for Computational Linguistics (ACL). https://doi.org/10.18653/v1/2025.findings-naacl.54.

Efficient Multi-Agent Collaboration with Tool Use for Online Planning in Complex Table Question Answering

Wei Zhou[1,2] **Mohsen Mesgar**[1] **Annemarie Friedrich**[2] **Heike Adel**[3]
[1]Bosch Center for Artificial Intelligence, Renningen, Germany
[2]University of Augsburg, Germany [3]Hochschule der Medien, Stuttgart, Germany
{wei.zhou|mohsen.mesgar}@de.bosch.com
annemarie.friedrich@uni-a.de adel-vu@hdm-stuttgart.de

Abstract

Complex table question answering (TQA) aims to answer questions that require complex reasoning, such as multi-step or multi-category reasoning, over data represented in tabular form. Previous approaches demonstrate notable performance by leveraging either closed-source large language models (LLMs) or fine-tuned open-weight LLMs. However, fine-tuning LLMs requires high-quality training data, which is costly to obtain. The use of closed-source LLMs poses accessibility challenges and leads to reproducibility issues. In this paper, we propose **M**ulti-**A**gent **C**ollaboration with **T**ool use (MACT), a framework that requires neither fine-tuning nor closed-source models. In MACT, a planning agent and a coding agent that also make use of tools collaborate for TQA. MACT outperforms previous SoTA systems on three out of four benchmarks and performs comparably to the larger and more expensive closed-source model GPT-4 on two benchmarks, even when using only open-weight models without any fine-tuning. Our extensive analyses prove the effectiveness of MACT's multi-agent collaboration in TQA. We release our code publicly.[1]

1 Introduction

The goal of table question answering (TQA) is to answer a question based on data represented in tabular form, optionally also using additional textual context. Recent studies on TQA focus more and more on complex instances, as they are ubiquitous in table data analysis (Zhu et al., 2021; Zhang et al., 2024b; Lu et al., 2023). Solving those complex instances requires performing multiple reasoning steps and/or employing different reasoning strategies (Ghosal et al., 2023). We refer to these aspects as *multi-step* and *multi-category* reasoning, respectively. An example requiring both types of reasoning is shown in the upper left part of Figure 1. To answer the question about the percentage change, a system first needs to use factual knowledge to extract countries in Europe. Then, numerical reasoning is applied to calculate the percentage change and to carry out the comparison.

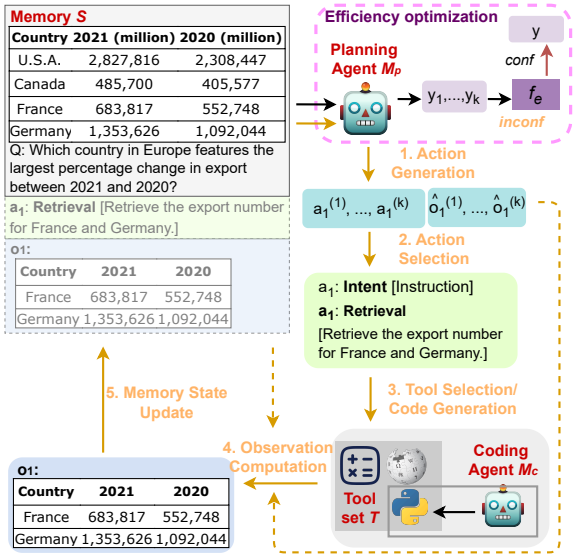

Figure 1: Overview of MACT, an iterative collaboration framework for TQA that consists of five stages for each iteration as well as an efficiency optimization module.

One popular approach for addressing those complex instances in TQA is planning, where step-wise plans are generated and used to guide the reasoning process (Zhang et al., 2024c; Wang et al., 2024; Wu and Feng, 2024; Zhu et al., 2024; Zhao et al., 2024). State-of-the-art works in this direction either fine-tune open-weight large language models (LLMs) (Wu and Feng, 2024; Zhu et al., 2024) or prompt closed-source commercial LLMs (Wang et al., 2024; Zhang et al., 2024c; Zhao et al., 2024). However, **fine-tuning requires high-quality data**, which is usually expensive to obtain (Zhu et al., 2021). Prompting closed-source commercial LLMs can also be **costly and poses challenges to reproducibility**. To the best of our knowledge, existing

[1]https://github.com/boschresearch/MACT

Findings of the Association for Computational Linguistics:
NAACL 2025, pages 945–968

methods leverage a single LLM to perform planning and reasoning, which is sub-optimal in particular if the LLM does not excel at mathematical reasoning or coding (Wu and Feng, 2024). These models struggle with answering questions requiring complex reasoning.

To address these challenges, we propose MACT, a multi-agent collaboration framework with tool use, which neither depends on closed-source LLMs nor requires fine-tuning. In fact, its backbone LLMs can be exchanged flexibly. It incorporates two agents (a planning agent and a coding agent) and a set of tools (a Python interpreter, a calculator and Wikipedia search). The planning agent performs online planning, i.e., it generates a plan iteratively. This breaks down complex problems and helps to address multi-step reasoning. The coding agent and the tool set assist with generating faithful intermediate results. The agents work in a collaborative setting, addressing the challenges of multi-category reasoning as all agents can concentrate on the reasoning types they excel in. An efficiency optimization module which allows the framework to take informed shortcuts.

We conduct experiments on four popular TQA benchmarks that include complex TQA instances. Our framework outperforms previous SoTA systems on three out of four benchmarks. It achieves comparable results to GPT-4 on two benchmarks even when using only open-weight models without any fine-tuning. In comparison to fine-tuned SoTA TQA systems, it demonstrates considerably better generalizability across datasets. Our analysis proves the effectiveness of our proposed collaborative setting of specialized agents. We find that the efficiency optimization module can save up to 33% of iterations without performance degradation.

2 Related Work

We review previous work for three core aspects of MACT: planning, multi-agent collaboration and LLMs with tool use. Table 1 compares MACT with previous TQA systems.

Planning. We categorize previous work into three groups based on planning strategies: *Heuristic coarse-grained planning* consists of two predefined steps of retrieving and aggregating (Ye et al., 2023; Zhou et al., 2024). *Online global planning* generates a plan in the first iteration and revises it in the next one (Zhao et al., 2024). *Online iterative planning* conditions the generation of the next step on the executed results of previous steps (Zhang et al., 2023a; Wang et al., 2024). We opt for online iterative planning for complex TQA, as it provides for more fine-grained steps during problem solving (in contrast to heuristic planning) and emphasizes the dependency among steps (in contrast to online global planning), which is crucial in complex TQA. In contrast to previous work using iterative planning, we introduce an efficiency optimization module to minimize the costs of the framework. To learn how to generate plans, previous work either depends on fine-tuning (Wu and Feng, 2024; Zhu et al., 2024), or strong closed-source models, combined with in-context learning (Wang et al., 2024; Zhang et al., 2023a; Zhao et al., 2024). By contrast, MACT generates effective plans using either closed-source or open-weight models, without the need for fine-tuning.

System	Online planning	No fine-tuning	NL plan	Multi agents	Tools
Raven	✗	-	-	✗	cal, SQL
Binder	✗	-	-	✗	SQL/Python
Lever	✗	-	-	✗	SQL
TableLlaMA	✗	-	-	✗	✗
Dater	✗	✓	✓	✗	✗
TAT-LLM	✗	✗	✓	✗	cal
Chain of table	✓	✓	✗	✗	✗
Reactable	✓	✓	✗	✗	SQL, Python
Protrix	✓	✗	✓	✗	SQL
TAPERA	✓	✓	✓	✗	Python
MACT (ours)	✓	✓	✓	✓	Python, cal, Wiki

Table 1: Comparing MACT with previous works. NL plan stands for using natural language to encode a plan, cal=calculator, and Wiki=Wikipedia.

Multi-agent collaboration. In multi-agent collaboration settings, multiple AI entities collaborate towards a common goal (Talebirad and Nadiri, 2023). We use the term agent to refer to LLMs that interact with executable tools, following Qiao et al. (2024). As far as we know, none of the previous works in TQA utilize multi-agent collaboration. The approaches most closely related to ours explore collaboration among homogenous agents, i.e., all agents use the same backbone but are prompted differently (Liu et al., 2023; Zhao et al., 2024). The effectiveness of this approach relies heavily on strong (closed-source) backbone models. Our work explores multi-agent collaboration for TQA, without any constraint of model type.

Tool use. LLMs have been shown to be ineffective in retrieving information from long tables (Zhou et al., 2024) and carrying out numerical reasoning (Imani et al., 2023). Making use of tools can ensure faithful results of these operations. The most common tools used in TQA are SQL interpreters (Cheng et al., 2022), Python with Pandas dataframes (Gemmell and Dalton, 2023), and calculators (Zhu et al., 2024). In MACT, we use similar tools. Inspired by Shinn et al. (2023), we further add Wikipedia search as an additional tool to assist questions requiring factual knowledge.

3 Method

We propose MACT, a **M**ulti-**A**gent **C**ollaboration framework enriched with a set of **T**ools for TQA. Figure 1 provides an overview of the framework. It consists of four major modules: a memory S, a planning agent M_p, a coding agent M_c, and a tool set T. M_p and M_c are instantiated by (potentially) different LLMs. They collaborate through five core stages: *action generation*, *action selection*, *tool selection/code creation*, *observation computation*, and *memory state update*. The stages are executed iteratively for a maximum of I iterations, where I is a hyper-parameter. We control the overall efficiency of the collaboration via an *efficiency optimization* module. For a new TQA instance, we initialize the memory state $s_0 = (\text{table}, \text{question}, \text{texts})$, i.e., with the input table, the question, and (if given) textual context. All parts of the memory are represented as strings. To represent the table as a string, we use pipes as column separators.

3.1 Action Generation

To format our plans, we follow ReAct (Yao et al., 2022) that consists of the generation of thoughts, actions and observations. Our framework only requires actions and observations, but following Yao et al. (2022), who demonstrate performance gains from generating thoughts with actions, we adopt their prompting method. Thus, at each iteration $i \leq I$, we prompt M_p to generate a thought z_i, an action a_i and an observation $\hat{o}_i$: $(z_i, a_i, \hat{o}_i) \sim M_p(z_i, a_i, \hat{o}_i | s_{i-1}, \phi_p, \tau_p)$, where s_{i-1} is the previous memory state, and ϕ_p and τ_p are the prompt (provided in A.6) and temperature of the LLM used for M_p, respectively. Note that $\hat{o}_i$ is not the final observation. Instead, it is later used during execution as a particular form of our proposed collaboration between the planning and coding agent (see 3.5). We sample from M_p k times, resulting in k actions $\{a_i^n\}_{n \leq k} = \{a_i^1, a_i^2, ..., a_i^k\}$ and their corresponding estimated observations $\{\hat{o}_i^n\}_{n \leq k} = \{\hat{o}_i^1, \hat{o}_i^2, ..., \hat{o}_i^k\}$ in iteration i. Following Yao et al. (2022), we define an action a_i with two parts: an intent and an instruction, e.g., "*Retrieval* [Retrieve the export number for France and Germany]." The intent encodes the purpose of an action, e.g., *Retrieval* denotes retrieving information from the input table. The instruction (marked with brackets) provides detailed specifications of the intent. Table 2 shows the six types of intents we define for our framework and examples for corresponding instructions.

The intents *Retrieval* and *Calculation* are commonly seen in previous works (Gemmell and Dalton, 2023; Zhu et al., 2024). We use *Retrieval* for any operations extracting information from a table, including direct querying, filtering, and grouping. Instructions that require calculation, counting or comparison are captured by *Calculation*. To fulfill the possible need for external (factual) knowledge that is not present in the table or textual context, we add the intent *Search*, which performs Wikipedia searches to retrieve informative text passages. *Read* covers the need for contextual reasoning in table-text QA. It refers to instructions involving retrieving information from the texts provided as part of TQA instances. The intent *Finish* stops M_p from generating more actions and ends the iterative execution of our framework, providing the final answer in the corresponding instruction. Lastly, we use an intent called *Ask* to retrieve an answer based on the internal knowledge of the planning agent. If M_p fails to generate a valid action at an iteration, it will continue to generate the action at the next iteration until reaching the maximum iteration number I, and return the most common prediction directly from M_p as the final answer (See section 3.7).

3.2 Action Selection

From the set of k actions generated by M_p for iteration i, we use a function f_s to select the most promising action $a_i^* = f_s(s_{i-1}, \{a_i^n\}_{n \leq k})$. We use self-consistency (SC) (Wang et al., 2022) as the selection function, which outputs the most frequent action from the set of sampled actions. In the case of ties, we choose the most frequent action that was sampled first. We provide a comparison with other selection functions in A.4.

Intent	Instruction: Format and Example	Tool Selection	Code Generation	Tool Use/Execution
Retrieval	textual description of what to retrieve, e.g., "sale numbers of 2019"	$t = Python$	M_c	Python interpreter is run on generated code
Calculation	formula or textual description of what to calculate, e.g., "(135-114)/135"	$t = T_{cal}$ if formula, else $t = T_{python}$	M_c if not formula	Calculator is executed or Python interpreter is run on code
Search	entity name, e.g., "Tesla"	$t = T_{srh}$	no	Wikipedia API is called on entity
Read	textual description of required information from input texts, e.g., "when was the target method adopted"	$t = Null$	no	M_p is prompted to extract information from provided textual context
Finish	final answer y	$t = Null$	no	final answer is output and execution stops
Ask	textual description of required information from M_p, e.g., "hours of a day"	$t = Null$	no	M_p is prompted for the information need

Table 2: Overview of intents and instructions of actions and how they are executed within our framework.

3.3 Tool Selection and Use

The tool needed for executing action a_i^* depends on the intent of the action (see columns "Tool Selection" and "Tool Use/Execution" of Table 2). To address the intents *Search*, *Calculation* and *Retrieval*, we introduce a set of tools $T = \{T_{srh}, T_{cal}\, T_{python}\}$. T_{srh} is an API function for Wikipedia search (Shinn et al., 2023) from Langchain.[2] The API takes a target entity specified in the instruction and returns the first paragraph of the corresponding Wikipedia entry. T_{cal} is a calculator, powered by a Python interpreter. It takes a formula generated by M_p and outputs the answer. Note that the instruction of *Calculation* can also be a textual description, such as "Compute the average number of medals for each country in the table." To better address these instructions, we introduce a coding agent M_c with a Python interpreter, denoted as T_{python} (see 3.4). We do not distinguish formulas and text description. This means any instructions with the intent *Calculation* will be firstly passed to T_{cal}. If T_{cal} fails to execute the instruction, T_{python} is applied.

The intent *Retrieval* is also addressed by M_c and T_{python}, i.e., M_c generates Python code based on a given instruction to retrieve target cells in the tables and the Python interpreter returns the executed results. Lastly, For *Read*, *Ask* and *Finish*, no tool is used, denoted as $t = Null$ in Table 2. The answers to the intents *Read* and *Ask* are queried via M_p. For *Read*, M_p reads the answer from a given textual context. For *Ask*, it responds based on its internal knowledge. No execution is performed for *Finish*, as the intent ends MACT with a final answer in its instruction.

[2] https://rubydoc.info/gems/langchainrb/Langchain/Tool/Wikipedia

3.4 Code Generation and Execution

To address textual instructions for *Calculation* actions as well as *Retrieval* actions, we integrate a coding agent M_c, which is an LLM and translates the instructions of a_i^* into Python code snippets $c_i \sim M_c(c_i|a_i^*, s_{i-1}, \phi_c, \tau_c)$. The hyper-parameter τ_c controls the temperature of the coding agent, and ϕ_c is a static, pre-defined prompt (see A.6). We sample k times from M_c to increase the robustness of the system against generated syntax errors, resulting in a set of code snippets $C = \{c_i^n\}_{n \leq k}$. A Python interpreter is run on each c_i, creating a set of executed solutions $\hat{C} = \{\hat{c}_i^n\}_{n \leq k}$.

3.5 Observation Computation

The computation of the final observation o_i depends on the selected tool t_i: if $t_i \in \{T_{cal}, T_{srh}\}$, the corresponding tool returns a deterministic result. If t_i is T_{python}, we select the most frequent element from the combined set $\{\hat{o}_i^n\}_{n \leq k} \cup \hat{C}$ as final observation, where $\{\hat{o}_i^n\}_{n \leq k}$ is the estimated observations sampled from M_p. This strategy features two levels of collaboration: from an ensemble perspective, both M_p and M_c contribute to obtain o_i; from a pipeline perspective, M_c makes use of the outputs (actions) of M_p. If neither T nor M_c are needed to execute the action, the final observation is the most frequent element in $\{\hat{o}_i^n\}_{n \leq k}$.

3.6 Memory State Update and Iteration

After obtaining o_i, we update the memory state with the selected action and the observation of iteration i: $s_i = s_{i-1} + [a_i^*, o_i]$. The framework then continues with the next iteration $i+1$. Note that adding the observation to the memory state also allows M_p to build on top of results from M_c in iteration $i+1$. If $i > I$, the execution stops with a predicted answer directly from M_p. In 98% of the cases, a final answer is given before $i > I$.

3.7 Efficiency Optimization

The iterative collaboration approach of M_p and M_c is highly effective in practice as demonstrated in our experiments. However, questions that do not require multi-step or multi-category reasoning can also be answered directly by M_p. For those instances, we propose an efficiency optimization component that serves as a shortcut for directly outputting an answer in the first iteration.Whether the answer is output directly depends on the confidence of M_p, which we approximate by the degree of self-consistency of its estimated predictions $Y = \{y^1, ..., y^k\}$. Y is obtained by accessing the whole reasoning trace (consisting of j actions and estimated observations) that M_p generates for a given s_0 until the intent *Finish* is output: i.e., $(a_1^k, \hat{o}_1^k, ...a_j^k, \hat{o}_j^k) \sim M_p(a_1^k, \hat{o}_1^k, ...a_j^k, \hat{o}_j^k | s_0, \phi_p, \tau_p)$. The output y^k is the instruction of the action a_j^k with intent *Finish*. To control the trade-off between performance and computation time, we introduce a hyper-parameter $\alpha \in [0..1]$. If the degree of self-consistency, i.e., the number of occurrences of the most frequent prediction in Y is larger than $\alpha * k$ (high degree of SC), M_p outputs this most frequent answer. Otherwise, the collaborative framework as described before is adopted. In short, the smaller α, the more often the system is allowed to use the shortcut, and the larger α, the more confident M_p needs to be in order to take the shortcut.

4 Experiments

We assess the performance of MACT on four TQA benchmarks in comparison to SoTA TQA systems.

Datasets. We choose four TQA datasets that cover different reasoning complexities and domains (See A.1 for more details). **WTQ** (Pasupat and Liang, 2015) is the easiest dataset as it neither requires multi-step nor multi-category reasoning. However, it is a widely used benchmark for TQA in the general domain and enables a fair comparison with recent TQA systems. **TAT** (Zhu et al., 2021) includes hybrid tabular and textual data. Most questions require numerical reasoning. **CRT** (Zhang et al., 2023b) uses Wikipedia tables and involves complex reasoning. **SCITAB** (Lu et al., 2023) contains claims requiring compositional reasoning for verification. We follow the original work to convert it from the setting of fact verification to TQA.

TQA systems for comparison. We categorize the recent TQA systems into two groups indicating if they require LLM fine-tuning or not. We include the following baselines that require fine-tuning: OmniTab (Jiang et al., 2022), TableLlama (Zhang et al., 2024a), ProTrix (Wu and Feng, 2024), TAT-LLM (Zhu et al., 2024) and TableLLM (Zhang et al., 2024b). Except for OmniTab, which is backboned by BART (Lewis et al., 2020), all others build on top of LLaMA 7b (Touvron et al., 2023). Our set of TQA baselines that do not require fine-tuning includes Dater (Ye et al., 2023), Binder (Cheng et al., 2023), Chain-of-Table (Wang et al., 2024), ReAcTable (Zhang et al., 2024c), TabSQLify (Nahid and Rafiei, 2024), Plan-then-Reason (Wu and Feng, 2024), Mix-SC (Liu et al., 2023) and ARC (Zhang et al., 2023b). They all rely on GPT-3.5-turbo.

Experimental settings. In MACT, the choice of the planning and coding agent is flexible as no fine-tuning is involved. We experiment with the best open-weight LLMs available at the time of writing: Qwen-2 72B (Yang et al., 2024) (planning agent) and CodeLLaMA-34B (coding agent). They are run on int4 and full precision, respectively. We further use GPT-3.5-turbo as both the planning and coding agents when comparing our method with other TQA systems that use this model. τ_p and τ_c are set to 0.6 for non-repetitive action generation. We set the action and code generation size k to 5, following Liu et al. (2023). The maximum number of iteration I is set to 7, based on empirical results on the development sets. For the efficiency component, we set α to 1 to ensure high confidence of the model. We explore the effects of different values of α in Section 6. Two NVIDIA A100 GPUs are used for running MACT.

5 Results

We first evaluate the performance of MACT in direct comparison to recent TQA systems using closed-source LLMs. Second, we examine how our method performs compared to fine-tuned open-weight LLMs.[3] Following prior work, we use exact match (EM) as the evaluation measure. Lastly, we discuss the efficiency of MACT.

MACT outperforms TQA models on three out of four datasets when using GPT-3.5 as the backbone. The upper part of Table 3 shows MACT using GPT-3.5 as the planning and coding agent in comparison to SoTA TQA systems using GPT-3.5 as the backbone LLM. MACT (GPT-3.5) surpasses the examined TQA models, except for Mix-SC on WTQ. This indicates the effectiveness of our multi-agent strategy compared to single-agent TQA models. We suspect that the performance gap between our approach and Mix-SC comes from data-specific table-cleaning and answer format controls in Mix-SC. In contrast, MACT does not include any dataset-specific pre- or postprocessing steps to keep it generally applicable to any dataset.

MACT outperforms out-of-the-box open-weight LLMs across datasets, demonstrating the effectiveness of specialized agents. The middle part of Table 3 provides the results of MACT using specialized agents (MACT (Qw+CL): Qwen-2 as planning agent, CodeLLaMA as coding agent). As baselines, we compare a setting without a specialized coding agent (MACT (Qw+Qw)) and a setting without a general planning agent (MACT(CL+CL)). As further baselines, we use the two LLMs on their own as well in combination, with Chain-of-thought prompting (Wei et al., 2022). For combination, the single models are prompted five times and combined using SC, as in Liu et al. (2023). This is a direct multi-agent baseline without collaboration and tool use. Both SC(Qw+CL) and MACT (Qw+CL) achieve higher EM scores than individually prompting Qwen and CodeLLaMA, demonstrating the positive effect of using multiple agents for planning and coding. Importantly, MACT (Qw+CL) outperforms SC(Qw+CL) by approximately 6 EM points on average across all datasets, highlighting the superiority of our collaboration technique over simply taking the most frequent predictions from two independent agents. We also find that having an expert coding agent for code generation (MACT (Qw+Qw) vs. MACT (Qw+CL)) improves performance considerably.

[3]We did not run ARC as no code is available. Results for TAT-LLM is only reported on TAT as the model is specially designed for TAT dataset that features both tables and texts as inputs.

	WTQ	TAT	CRT	SCT
closed-source LLM backbones				
GPT-3.5	45.8	39.7	39.3	48.9
Dater	52.8*	22.1	46.8	47.1
Binder	56.7*	0.9	1.24	29.1
Chain-of-Table	59.9*	20.5	33.9	27.6
ReAcTable	52.4*	9.26	29.8	32.1
TabSQLify	64.7*	13.7	42.0	50.9
Plan-then-Reason	65.2*	41.2	44.9	52.5
Mix-SC	**73.6***	54.3	48.6	49.3
ARC	-	-	56.3*	-
MACT	70.4	**64.5**	**57.4**	**55.8**
open-weight LLM backbones				
Qwen (Qw-72b)	60.6	53.6	55.9	55.0
CodeLLaMA (CL-34b)	55.0	29.5	49.7	9.5
SC(Qw-72b+CL-34b)	69.0	56.7	61.4	54.4
MACT (Qw-72b+Qw-72b)	68.6	66.3	59.8	57.3
MACT (CL-34b+CL-34b)	55.2	54.1	43.5	45.0
MACT (Qw-72b+CL-34b)	**72.6**	**66.2**	**64.4**	**59.8**
GPT-4	*72.9*†	*80.8*†	*58.7*†	*63.2*†
Humans (Crowdsourcing)	-	*84.1*†	-	*84.7*†

Table 3: Exact Match results of **models without fine-tuning.** SCT refers to SCITAB. The models are grouped by the LLM they use as their backbone (top: GPT-3.5; middle: open-weight LLMs as indicated in parentheses). Performances marked with * are taken from the original paper. Performances marked with †are taken from Wu and Feng (2024), Zhu et al. (2024), Zhang et al. (2023b) and Lu et al. (2023) for each dataset. We bold the best performances in each group.

MACT with open-weight models delivers comparable performance as closed-source systems. As shown in Table 3, by comparing MACT (Qw+CL) with TQA systems that rely on closed-source LLMs (in the upper part of Table 3), we find that our model outperforms the examined TQA systems for three out of four datasets. Our multi-agent TQA system is more cost-efficient and straightforward to replicate, while delivering superior performance compared to closed-source TQA models. In addition, we show two upper-bounds in the bottom part of Table 3: The performance of human annotators and directly prompting GPT-4 with the table and question. As shown in Table 3, GPT-4 has an advantage on TAT and SCITAB. On WTQ, we observe comparable performances between MACT (QW+CL) and GPT-4. On CRT, our method even

	WTQ	TAT	CRT	SCT
OmniTab (BART 406m)	62.3*	17.1	20.6	29.1
TableLlama (LlaMA-7b)	29.9	17.4	26.9	38.6
Protrix (LlaMA-7b)	48.9*	26.8	40.2	42.4
TAT-LLM (LlaMA-7b)	-	**69.6***	-	-
MACT (LlaMA-7b+CS-7b)	38.1	28.3	40.0	41.1
MACT (Qw-7b+CS-7b)	**58.4**	61.9	**46.4**	**45.9**

Table 4: Exact Match Results of MACT using different LLM agents in **comparison to fine-tuned TQA models**. Performances marked with * refer to the in-domain setting (where fine-tuning took place). SCT refers to SCITAB. CS refers to the deepseek-coder model.

outperforms GPT-4 by 5.7%. CRT is the most complex dataset, requiring multi-step and multi-category reasoning, which direct inference with GPT-4 cannot generally solve. Our step-wise collaborative planning setting is well-suited to such settings. In contrast, there is a large gap between MACT and human performance in SCITAB. SCITAB collects data from scientific papers, in which abbreviations and domain-specific terms are common. These can pose challenges to current systems and models. In TAT, MACT often finds the correct answer but struggles to output it in the correct format (see Sec. 6).

MACT generalizes better across datasets than fine-tuned TQA systems. Table 4 compares our framework with prior fine-tuned TQA models. We present results for MACT with different planning and coding agents: Our standard setting (LlaMA-7b+CS-7b) as well as with a stronger planner (Qw-7b+CS-7b). In general, for fine-tuned TQA models, their performance on the dataset used for fine-tuning is rather high while they suffer from a considerable drop in EM when tested on other datasets. This observation is in line with the findings by Zhang et al. (2024a) and Huang et al. (2024). In contrast, MACT does not use fine-tuned models and can, thus, be applied to any dataset with a good generalization performance. MACT demonstrates comparable results to Protrix when using LlaMA-7b as the planning agent, though it has not been fine-tuned. As expected, using a better planning agent leads to better results. This also shows the robustness of MACT in terms of backbone models.

MACT adapts computational cost to instance complexity. Table 5 compares MACT with other approaches in terms of the total number of LLM calls for each instance. For Binder and Dater, `SC` is performed a fixed number of times regardless of problem complexity. This results in a high number of LLM calls per instance, making them inefficient. In contrast, MACT provides flexibility in generation, as the number of iterations depends on the problem's complexity. For instance, most questions can be solved within three steps for WTQ (see our analysis in A.3). This results in a total of at most 25 LLM calls [4] for each instance. If we incorporate the efficiency optimization module, which potentially saves up to one-third of the iterations (see Section 6), the total number of LLM calls per instance is even lower (approximately 15), making MACT comparable to other approaches in terms of efficiency. The iterative nature of MACT can lead to a higher upper-bound of LLM calls. However, it also allows for solving more complex problems, making the approach more tailored to real-life requirements.

	Number of LLM calls per instance
Binder	50
Dater	100
Chain-of-Table	1-25
ReAcTable	15-125
Mix-SC	10-30
MACT	5-65

Table 5: Number of LLM calls for different approaches. We show lower and upper-bounds if not deterministic.

	WTQ	TAT	CRT	SCITAB
MACT (Qw-72b)	**72.6**	**66.2**	**64.4**	**59.8**
w/o T_{srh}	72.0	66.2	64.6	59.6
w/o $T_{srh} + T_{cal}$	71.3	62.8	63.9	58.2
w/o T+M_c	67.1	61.2	60.4	57.9

Table 6: Ablation study. T_{srh} and T_{cal} refer to the Wikipedia search tool and the calculator tool. T includes the above two tools and a Python interpreter. M_c is the coding agent.

6 Analysis

We conduct various analyses of our framework to back up our claims and contributions. Unless mentioned otherwise, all analyses are performed using Qwen-2 72B as M_p, CodeLlama-34B as M_c, the number of action generation $k = 5$, and selection model $f_s = \texttt{SC}$. To explicitly analyze the effects of multi-agent collaboration with tool use, we do

[4] 2 steps involve action and execution generation, with each five times, plus last step five times of action generation: 2*(5+5)+5.

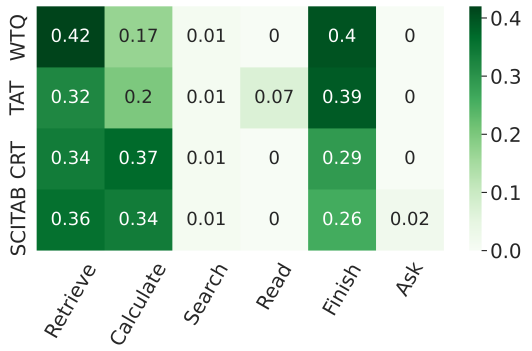

Figure 2: Distribution of action intents by dataset.

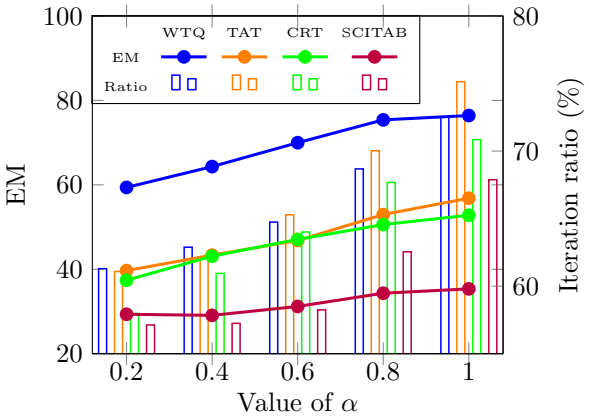

Figure 3: EM (line chart) and iteration ratio (bar chart) against different α. The iteration ratio is calculated by dividing the number of iterations when using efficiency optimization with a specific α by the number of iterations when not using it (without shortcuts).

not use efficiency optimization, which means all instances undergo the iterative collaboration between M_p and M_c with tool use. Further analysis to support our choices of the sampling size k and the maximum number of iterations I are in A.2 and A.3. A case study can be found in A.5.

Effect of Multi-Agent-Collaboration with Tool Use. We explore the effectiveness of specialized agents and tool use in MACT by conducting an ablation study with three scenarios: ablating only T_{srh} (Wikipedia search API), ablating T_{srh} and T_{cal} (calculator), and further ablating the coding agent M_c with a Python interpreter. In cases where M_c or/and tools are ablated, the most frequent estimated observations from M_p are used as the final observations. Our results in Table 6 show that both the tools and the coding agent contribute to the performance of the framework. Nevertheless, they contribute differently to the final performance. For instance, ablating the search tool barely influences the results whereas there are large performance drops when further ablating the coding agent and the Python interpreter. We find that the search tool is barely used whereas the coding agent is called in almost every query. Since Wikipedia is a common pre-training corpus for LLMs, most information might have already been encoded in the LLM. Nevertheless, the search tool can still be helpful given LLMs are known to suffer from hallucinations and the knowledge encoded might not be updated in time. For more specialized domains and sources, the search tool may be crucial. We further observe that the ablation affects WTQ and TAT more than CRT and SCITAB. This might be attributed to dataset features: CRT contains many yes-no questions and SCITAB has been converted from a ternary classification dataset. Thus, chances for guessing the correct final answers are higher than in datasets with a more diverse answer distribution, such as WTQ and TAT. By evaluating our framework on instances from CRT that have answers other than yes/no, we find a performance drop of 8.23 when ablating both tools and the coding agent.

Analysis of Intent Distribution. We report the distribution of the intents of the selected actions for each dataset in Figure 2. We observe that *Retrieve* and *Calculate* are the most frequent intents, along with *Finish*. This indicates that our proposed second agent, the coding agent M_c is used frequently. Different datasets also require different intents. In particular, the framework needs to use the intent *Read* for solving instances in TAT, where textual descriptions are given while this is not the case for the other datasets. We notice that the *Search* intent is used only few times across datasets. This might be because most instances were designed to be solved using the given table and text information. However, when looking into the individual cases where *Search* is used, we still find it useful. For instance, one question from WTQ asks about the number of athletes from American but no information about nationality is given in the table. In this case, *Search* assists with answering the question by adding the nationality for each athlete from Wikipedia. Though the intents *Read*, *Search* and *Ask* are less used compared to others, we still incorporate them to adapt to various use cases that might occur in real-life use cases.

Effect of Efficiency Optimization. To investigate the trade-off between efficiency and accuracy, we plot the model performance against $\alpha \in \{0.2, 0.4, 0.6, 0.8, 1\}$ in Figure 3. We also plot the ratio of the total number of iterations taken to terminate from each tested α to the total number of iterations taken when not using the optimization, i.e., letting the planning agent decide via the *Finish* action when to stop the execution. The best performance is reached for $\alpha = 1$, i.e., when requiring all estimated results to agree with each other to stop the iteration. For SCITAB, for instance, we save approximately 40% of the iterations when setting α to 1 without losing performance compared to not using the optimization component (59.8% vs. 59.7%). On average, adding the efficiency optimization module saves up to 33% of iterations. This shows the effectiveness of the optimization and that users can individually tune the desired trade-off of performance and computation time.

Error Analysis. We randomly sample 50 instances that MACT fails per dataset and conduct an error analysis. About half of the errors come from invalid or wrong code generated by the coding agent M_c. Either M_c fails to make sense of instructions or of complex table structure. The second error type can be attributed to evaluation. We find that about one-third of failures come from strict evaluation metrics (EM). This influences the performance of MACT particularly on the TAT dataset, as it features long text strings as answers. The evaluation challenge has been discussed in many previous works (Wu and Feng, 2024; Li et al., 2024). To estimate the upper-bound of our method, we use GPT-4 as evaluator to determine if a predicted answer is semantically the same as the reference answer. This results in an accuracy of 87.8% on the TAT dataset, compared to 66.2% with EM. The remaining error cases can be largely attributed to the failure of the planning agent in decomposing questions correctly. For instance, one question asks for the score range (min-max) of the top 10 finishers. Apart from retrieving the min and max scores of the top 10 finishers, the planner continues to generate the action: `Calculate[Calculate the range of the scores in the observation 1.]`. This leads to a wrong prediction.

7 Conclusions

We have proposed MACT, a multi-agent collaboration with tool use for table question answering. Unlike previous work, MACT neither requires fine-tuning nor does it depend on closed-source models. In our experiments, our framework demonstrates good generalizability across different benchmark datasets and outperforms a number of state-of-the-art approaches, including closed-source commercial models and fine-tuned models. To boost efficiency, we introduce an efficiency optimization module that saves up to 33% of the iterations in our analysis. In our experiments and analyses, we show that multi-agent collaboration with tools is an effective approach for table question answering.

8 Limitations

MACT is evaluated mainly with single table settings due to the scarcity of datasets featuring multi-table complex reasoning. Though the framework can be extended easily to deal with multiple tables by concatenating them in the inputs, it is still not clear how effective our approach will be in a multi-table setting. Secondly, we only study TQA in the context of English, while there exist many multi-lingual TQA benchmarks and challenges.

Acknowledgements

This work was partially supported by the EU Project SMARTY (GA 101140087).

References

Zhoujun Cheng, Tianbao Xie, Peng Shi, Chengzu Li, Rahul Nadkarni, Yushi Hu, Caiming Xiong, Dragomir Radev, Mari Ostendorf, Luke Zettlemoyer, Noah A. Smith, and Tao Yu. 2023. Binding language models in symbolic languages. *ICLR*, abs/2210.02875.

Zhoujun Cheng, Tianbao Xie, Peng Shi, Chengzu Li, Rahul 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.

Carlos Gemmell and Jeff Dalton. 2023. ToolWriter: Question specific tool synthesis for tabular data. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 16137–16148, Singapore. Association for Computational Linguistics.

Deepanway Ghosal, Preksha Nema, and Aravindan Raghuveer. 2023. ReTAG: Reasoning aware table to analytic text generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 6310–6324, Singapore. Association for Computational Linguistics.

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, Singapore. Association for Computational Linguistics.

Hui Huang, Yingqi Qu, Hongli Zhou, Jing Liu, Muyun Yang, Bing Xu, and Tiejun Zhao. 2024. On the limitations of fine-tuned judge models for llm evaluation. *ArXiv*, abs/2403.02839.

Shima Imani, Liang Du, and H. Shrivastava. 2023. Mathprompter: Mathematical reasoning using large language models. In *Annual Meeting of the Association for Computational Linguistics*.

Zhengbao Jiang, Yi Mao, Pengcheng He, Graham Neubig, and Weizhu Chen. 2022. OmniTab: Pretraining with natural and synthetic data for few-shot table-based question answering. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 932–942, Seattle, United States. Association for Computational Linguistics.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online. Association for Computational Linguistics.

Qianlong Li, Chen Huang, Shuai Li, Yuanxin Xiang, Deng Xiong, and Wenqiang Lei. 2024. Graphotter: Evolving llm-based graph reasoning for complex table question answering.

Tianyang Liu, Fei Wang, and Muhao Chen. 2023. Rethinking tabular data understanding with large language models. *ArXiv*, abs/2312.16702.

Xinyuan Lu, Liangming Pan, Qian Liu, Preslav Nakov, and Min-Yen Kan. 2023. SCITAB: A challenging benchmark for compositional reasoning and claim verification on scientific tables. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7787–7813, Singapore. Association for Computational Linguistics.

Md Mahadi Hasan Nahid and Davood Rafiei. 2024. TabSQLify: Enhancing reasoning capabilities of LLMs through table decomposition. In *2024 Annual Conference of the North American Chapter of the Association for Computational Linguistics*.

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

Shuofei Qiao, Ningyu Zhang, Runnan Fang, Yujie Luo, Wangchunshu Zhou, Yuchen Eleanor Jiang, Chengfei Lv, and Huajun Chen. 2024. Autoact: Automatic agent learning from scratch for qa via self-planning. *ArXiv*, abs/2401.05268.

Noah Shinn, Federico Cassano, Beck Labash, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *Neural Information Processing Systems*.

Yashar Talebirad and Amirhossein Nadiri. 2023. Multi-agent collaboration: Harnessing the power of intelligent llm agents. *ArXiv*, abs/2306.03314.

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

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Huai hsin Chi, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *ArXiv*, abs/2203.11171.

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

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Huai hsin Chi, F. Xia, Quoc Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. *ArXiv*, abs/2201.11903.

Zirui Wu and Yansong Feng. 2024. Protrix: Building models for planning and reasoning over tables with sentence context. *ArXiv*, abs/2403.02177.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Ke-Yang Chen, Kexin Yang, Mei Li, Min Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng,

Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Yang Fan, Yang Yao, Yichang Zhang, Yunyang Wan, Yunfei Chu, Zeyu Cui, Zhenru Zhang, and Zhi-Wei Fan. 2024. Qwen2 technical report. *ArXiv*, abs/2401.05268.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. *ArXiv*, abs/2305.10601.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *ArXiv*, abs/2210.03629.

Yunhu Ye, Binyuan Hui, Min Yang, Binhua Li, Fei Huang, and Yongbin Li. 2023. Large language models are versatile decomposers: Decomposing evidence and questions for table-based reasoning. *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*.

Tianshu Zhang, Xiang Yue, Yifei Li, and Huan Sun. 2024a. TableLlama: Towards open large generalist models for tables. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 6024–6044, Mexico City, Mexico. Association for Computational Linguistics.

Xiaokang Zhang, Jing Zhang, Zeyao Ma, Yang Li, Bohan Zhang, Guanlin Li, Zijun Yao, Kangli Xu, Jinchang Zhou, Daniel Zhang-li, Jifan Yu, Shu Zhao, Juan-Zi Li, and Jie Tang. 2024b. Tablellm: Enabling tabular data manipulation by llms in real office usage scenarios. *ArXiv*, abs/2403.19318.

Yunjia Zhang, Jordan Henkel, Avrilia Floratou, Joyce Cahoon, Shaleen Deep, and Jignesh M. Patel. 2023a. Reactable: Enhancing react for table question answering. *Proceedings of the VLDB Endowment*.

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

Zhehao Zhang, Xitao Li, Yan Gao, and Jian-Guang Lou. 2023b. CRT-QA: A dataset of complex reasoning question answering over tabular data. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 2131–2153, Singapore. Association for Computational Linguistics.

Yilun Zhao, Lyuhao Chen, Arman Cohan, and Chen Zhao. 2024. 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.

Wei Zhou, Mohsen Mesgar, Heike Adel, and Annemarie Friedrich. 2024. Freb-tqa: A fine-grained robustness evaluation benchmark for table question answering. *ArXiv*, abs/2404.18585.

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

Fengbin Zhu, Ziyang Liu, Fuli Feng, Chao Wang, Moxin Li, and Tat seng Chua. 2024. Tat-llm: A specialized language model for discrete reasoning over tabular and textual data. *ArXiv*, abs/2401.13223.

A Appendix

A.1 Datasets

Table A.1 shows their statistics and characteristics. WTQ (Pasupat and Liang, 2015), TAT (Zhu et al., 2021), CRT (Zhang et al., 2023b) and SCITAB (Lu et al., 2023) are publicly available under the licenses of CC-BY-SA-4.05, MIT and MIT, respectively. These licenses all permit us to compose, modify, publish, and distribute additional annotations upon the original dataset.

Dataset	#Test	M-step	M-category	Domain
WTQ	4,344	✗	✗	General
TAT	1,663	✓	✗	Financial
CRT	728	✓	✓	General
SCITAB	1,162	✓	✓	Scientific

Table 7: We use four datasets that vary in reasoning complexity (M-step: multi-step, M-category: multi-category reasoning) and domain. #Test refers to the number of test instances.

A.2 Effect of Sampling Size

Figure 6 shows the effect of the number of generated actions k on the results. Generally, we find that a larger k results in better performance. However, the performance gain is small when increasing k from 5 to 10. We even observe a slight performance drop for SCITAB when increasing k from 5 to 10. Based on these observations, we argue $k = 5$ is a good choice for the number of generated action in MACT.

A.3 Analysis of Iteration Number Distribution

We analyze the distribution of numbers of iterations for each dataset in Figure 7. Most of instances can be solved within seven iterations. Dataset-wise, CRT and SCITAB seem to require more iterations than WTQ and TAT, indicating their difficulties in terms of multi-step reasoning.

A.4 Choice of Selection Model.

In MACT, we use `SC` as the action selection model (see Section **Action Selection**). We now provide results for alternative selection models that have been introduced by prior work. In particular, we compare to LLM-based selection (Yao et al., 2023), log probability (Zhang et al., 2024c), roll-out (Hao et al., 2023) and a combination of all strategies (Hao et al., 2023). In the `LLM` strategy, an LLM evaluator is utilized to select the best action. For consistency with our other results, we use Qwen-72b as the evaluator. We use the same prompts as the original work (Yao et al., 2023). Log probability (`LOG_P`) has been widely used to assist sub-path selection (Zhang et al., 2024c; Hao et al., 2023). However, it can only be used for open-weight LLMs, as it requires access to log probabilities. `ROLL_OUT` estimates future answers by rolling out the current reasoning path and selects the action that leads to the most frequent future answer. For the `COMBINED` method, we use majority voting among all individual selection models. The results in Table 8 show that for WTQ and TAT, `SC` and `COMBINED` lead to the best performance. For CRT and SCITAB, `SC` outperforms `COMBINED`, caused by the comparably poorer performance of `LLM`, `LOG_P` and `ROLL_OUT` on these datasets. `SC` is more efficient than `COMBINED` as the latter requires running all selection models, including the computationally expensive LLM. Overall, this analysis confirms that `SC` as selection model is a good choice.

	WTQ	TAT	CRT	SCITAB
SC	72.6 (2)	66.2 (2)	**64.4** (1)	**59.8** (1)
LLM	70.7 (4)	66.2 (2)	58.4 (5)	56.8 (4)
LOG_P	70.1 (5)	64.9 (5)	61.4 (3)	57.2 (3)
ROLL_OUT	71.9 (3)	65.9 (4)	60.2 (4)	55.5 (5)
COMBINED	**72.7** (1)	**66.6** (1)	62.6 (2)	57.4 (2)

Table 8: Results using different selection models. We put the relative ranking of the models per dataset in parentheses.

A.5 Case Study

We present two reasoning traces selected from CRT and WTQ in Figure 4 and 5, respectively. Figure 4 presents a case involving complex reasoning, where models need to identify top ten finishers, calculate percentage of drivers for each constructor and return the constructor with the highest percentage. In Figure 5, we observe the planning agent has the ability to self-correct in Thought 3.

A.6 MACT Prompts

We provide the prompts used for the planning agent for the examined datasets (WTQ, TAT, CRT and SCITAB) in Figures 8, 9, 10, and 11. Figure 12 and Figure 13 show the prompts used for the coding agent for the action intents *Retrieval* and *Calculation*, respectively.

driver	constructor	laps	time / retired	grid
alberto ascari	ferrari	90	2:50:00	1
juan manuel fangio	maserati	90	+ 1:00	4
nino farina	ferrari	88	+ 2 laps	5
jos	maserati	88	+ 2 laps	2
mike hawthorn	ferrari	87	+ 3 laps	3
felice bonetto	maserati	82	+ 8 laps	16
prince bira	connaught - lea - francis	82	+ 8 laps	19
ken wharton	cooper - bristol	80	+ 10 laps	11
peter whitehead	cooper - alta	79	+ 11 laps	14
louis rosier	ferrari	78	+ 12 laps	24
jimmy stewart	cooper - bristol	79	spun off	15
tony rolt	connaught - lea - francis	70	halfshaft	10
luigi villoresi	ferrari	65	axle	6
onofre	maserati	65	engine	7
alan brown	cooper - bristol	56	overheating	21
peter collins	hwm - alta	56	spun off	23
jack fairman	hwm - alta	54	clutch	27
...	...	...	...	...

Question: **Which constructor had the highest percentage of its drivers finish the race in the top ten?**
Thought 1: I need to find the number of drivers who finished the race in the top ten for each constructor.
Action 1: Retrieval[Retrieve the rows for the top ten finishers.]
Observation 1:

driver	constructor	laps	time / retired	grid
alberto ascari	ferrari	90	2:50:00	1
juan manuel fangio	maserati	90	+ 1:00	4
nino farina	ferrari	88	+ 2 laps	5
jos	maserati	88	+ 2 laps	2
mike hawthorn	ferrari	87	+ 3 laps	3
felice bonetto	maserati	82	+ 8 laps	16
prince bira	connaught - lea - francis	82	+ 8 laps	19
ken wharton	cooper - bristol	80	+ 10 laps	11
peter whitehead	cooper - alta	79	+ 11 laps	14
louis rosier	ferrari	78	+ 12 laps	24

Thought 2: I need to calculate the percentage of drivers for each constructor among the top ten finishers.
Action 2: Calculation[Calculate the percentage of drivers for each constructor among the top ten finishers.]
Observation 2:

Constructor	Percentage
ferrari	40.0
maserati	30.0
connaught lea francis	10
cooper bristol	10
cooper alta	10

Thought 3: I need to identify the constructor with the highest percentage.
Action 3: Retrieval[Retrieve the constructor with the highest percentage.]
Observation 3:

Constructor	Percentage
ferrari	40.0

Thought 4: The constructor with the highest percentage is Ferrari.
Action 4: Finish[**Ferrari**.]

Figure 4: An instance selected from CRT featuring complex reasoning.

Year	Competition	Venue	Position	Notes
2005	World Youth Championships	Marrakech, Morocco	6th	5.05
2006	World Junior Championships	Beijing, China	5th	5.30 m
2008	Olympic Games	Beijing, China	10th	5.45 m
2009	European U23 Championships	Kaunas, Lithuania	8th	5.15 m
...	...	...	...	...

Question: **how many competitions are listed before 2006?**
Thought 1: I need to retrieve the number of competitions that happened before 2006 .
Action 1: Retrieval[retrieve the competitions that happened before 2006.]
Observation 1:

Year	Competition	Venue	Position	Notes
2005	World Youth Championships	Marrakech, Morocco	6th	5.05

Thought 2: I need to count the number of competitions retrieved in observation 1.
Action 2: Calculation[count the number of competitions in the table of observation 1.]
Observation 2: 11

Thought 3:The number of competitions listed before 2006 is 1 (World Youth Championships).
However, the result returned in observation 2 is 11, which is not correct.
There's a single competition before 2006, so the correct answer should be 1.
Action 3: Finish[**1**]

Figure 5: An instance selected from WTQ. We find the planning agent can perform self-correct given previous reasoning traces.

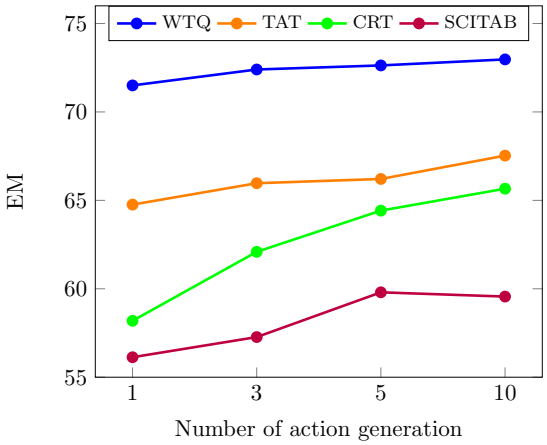

Figure 6: EM against different action generation size.

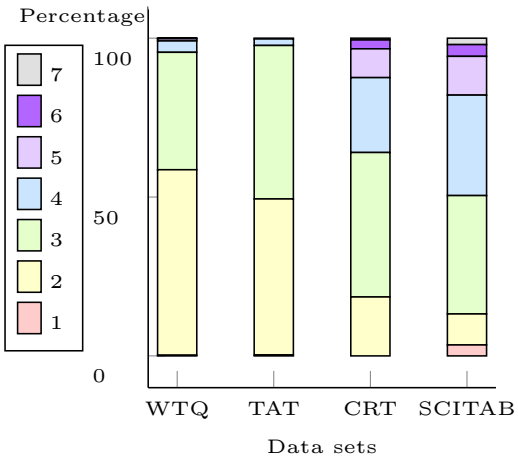

Figure 7: The distribution of number of iterations for each dataset.

Solve a table question answering task with interleaving Thought, Action, Observation steps. Thought can reason about the current situation, and Action can be four types:
(1) Retrieve[cells], which retrieves certain cell(s) from the table and returns the retrieved cells in string format.
(2) Calculate[formular/instruction], which carries out calculations based on the formular, or the instruction and returns the calculated results.
(3) Search[entity], which searches the exact entity on Wikipedia and returns the first paragraph if it exists.
(4) Finish[answer], which returns the answer and finishes the task.
You may take as many steps as necessary.
Here are some examples:
Table:
Parish	Locality	Parish Priest	Founded	Closed
St Mary	Bacup	Fr Frank Thorpe	1852	----
Our Immaculate Mother & St Anselm	Whitworth	Fr Frank Thorpe	1860	----
St Joseph	Stacksteads	----	1947	2005
St Joseph & St Peter	Newchurch-In-Rossendale	Fr Philip Boast	1915	----
The Immaculate Conception	Haslingden	Fr Canon John Mackie	1854	----
St Veronica (Chapel of Ease)	Helmshore	Served from The Immaculate Conception	1959	----
St James the Less	Rawtenstall	Fr David Lupton, Rural Dean	1828	----
Context:
Question: what's the number of parishes founded in the 1800s?
Thought 1: I need to retrieve the parishes founded 1800s. Since the time 1800s spans from 1800 to 1899, I need to retrieve the parishes whose Found time is between 1800-1899.
Action 1: Retrieve[retrieve the parishes whose Found time is between 1800-1899]
Observation 1:	Parish	Founded
St Mary	1852
Our Immaculate Mother & St Anselm	1860
The Immaculate Conception	1854
St James the Less	1828
Thought 2: I need to count the number of parishes retrieved in observation 1.
Action 2: Calculate[count the number of parishes in the table of observation 1]
Observation 2: 4
Thought 3: The question asks the number of parishes founded in 1800s. The answer is obtained in observation 2, which is 4.
Action 2: Finish[4]

Table:
Election	Number of popular votes	% of popular votes	Total elected seats	+/−
1988	139,982	22.16	61 / 264	
1991	170,757	32.11	83 / 272	22
1994	242,557	35.34	121 / 346	38
1999	271,251	33.45	122 / 390	1
2003	459,640	44.67	194 / 400	72
2007	445,781	39.15	127 / 405	30
2011	464,512	39.34	103 / 412	18
Context:
Question: what is the total number of popular votes cast in 2003?
Thought 1: I need to retrieve the number of popular votes in the election year 2003.
Action 1: Retrieve[retrieve the number of popular votes in the election year 2003]
Observation 1:	Election	Number of popular votes
2003	459,640
Thought 2: The question asks the total number of popular votes in 2003. The answer is retrieved from the observation 1, which is 459,640.
Action 2: Finish[459,640]

Table:
Year	Competition	Venue	Position	Notes
1989	European Indoor Championships	The Hague, Netherlands	10th	
1989	World Indoor Championships	Budapest, Hungary	9th	
1991	World Indoor Championships	Lisbon, Portugal	6th	
1991	World Championships	Tokyo, Japan	5th	5.75 m NR
1992	European Indoor Championships	Genoa, Italy	5th	
1993	World Championships	Stuttgart, Germany	11th	

```
Context:
Question: what peter widen's is the highest finish in all indoor championships?
Thought 1: I need to retrieve peter widen's finish position from the table.
Action 1: Retrieve[retrieve peter widen's position from the table]
Observation 1: | Position |
| 10th |
| 9th |
| 6th |
| 5th |
| 5th |
| 11th |
Thought 2: The question asks the highest finish. Since I already have a table of finished position in observation 1, I need to compare the
finished positions to find out the highest ranking, which is 5th.
Action 2: Finish[5th]
(END OF EXAMPLES)
Now generating the Thought, Action, Observation for the following instance:
Table:
{table}
Context: {context}
Question: {question}
{memory}
```

Figure 8: MACT: Planning agent prompt for WTQ.

Solve a table question answering task with interleaving Thought, Action, Observation steps. Thought can reason about the current situation, and Action can be four types:
(1) Retrieve[cells], which retrieves certain cell(s) from the table and returns the retrieved cells in string format.
(2) Look up[information], which looks up the information in the context (if any) and returns the information in string format.
(3) Calculate[formular/instruction], which carries out calculations based on the formular, or the instruction and returns the calculated results.
(4) Finish[answer], which returns the answer and finishes the task.
You may take as many steps as necessary.
Here are some examples:
Table:
| | 2019 | | 2018
In thousands | $ | % | $ | %
Drinkable Kefir other than ProBugs | $ 71,822 | 77% | $ 78,523 | 76%
Cheese | 11,459 | 12% | 11,486 | 11%
Cream and other | 4,228 | 4% | 5,276 | 5%
ProBugs Kefir | 2,780 | 3% | 2,795 | 3%
Other dairy | 1,756 | 2% | 3,836 | 4%
Frozen Kefir (a) | 1,617 | 2% | 1,434 | 1%
Net Sales | $ 93,662 | 100% | $ 103,350 | 100%
Context: Paragraph 1: Our product categories are: Paragraph 2: Drinkable Kefir, sold in a variety of organic and non-organic sizes, flavors, and types, including low fat, non-fat, whole milk, protein, and BioKefir (a 3.5 oz. kefir with additional probiotic cultures). Paragraph 3: European-style soft cheeses, including farmer cheese in resealable cups. Paragraph 4: Cream and other, which consists primarily of cream, a byproduct of making our kefir. Paragraph 5: ProBugs, a line of kefir products designed for children. Paragraph 6: Other Dairy, which includes Cupped Kefir and Icelandic Skyr, a line of strained kefir and yogurt products in resealable cups. Paragraph 7: Frozen Kefir, available in soft serve and pint-size containers. Paragraph 8: Lifeway has determined that it has one reportable segment based on how our chief operating decision maker manages the business and in a manner consistent with the internal reporting provided to the chief operating decision maker. The chief operating decision maker, who is responsible for allocating resources and assessing our performance, has been identified collectively as the Chief Financial Officer, the Chief Operating Officer, the Chief Executive Officer, and Chairperson of the board of directors. Substantially all of our consolidated revenues relate to the sale of cultured dairy products that we produce using the same processes and materials and are sold to consumers through a common network of distributors and retailers in the United States. Paragraph 9: Net sales of products by category were as follows for the years ended December 31: Paragraph 10: (a) Includes Lifeway Kefir Shop sales Paragraph 11: Significant Customers – Sales are predominately to companies in the retail food industry located within the United States. Two major customers accounted for approximately 22% and 21% of net sales for the years ended December 31, 2019 and 2018, respectively. Two major customers accounted for approximately 17% of accounts receivable as of December 31, 2019 and 2018. Our ten largest customers as a group accounted for approximately 57% and 59% of net sales for the years ended December 31, 2019 and 2018, respectively.
Question: What is the change in the net sales for cheese between 2018 and 2019?
Thought 1: I need to retrive the net sales for cheese for 2018 and 2019.
Action 1: Retrieve[net sales for cheese for 2018 and 2019]
Observation 1: | 2019 | 2018 |
| 11459 | 11486 |
Thought 2: I need to substract the net sales for cheese for 2019 from 2018, which results in the formular: [11459 - 11486]
Action 2: Calculate[11459 - 11486]
Observation 2: -27
Thought 3: The question asks for change in the net sales for cheese between 2018 and 2019. The answer is in the observation 2, which is -27. The unit/scale of the number is thousand, therefore the answer is -27 thousands.
Action 3: Finish[-27 thousands]

Table:
| | Three Months Ended | | % Variation |
| December 31, 2019 | September 29, 2019 | December 31, 2018 | Sequential | Year-Over-Year
| | | (Unaudited, in millions) | |
Automotive and Discrete Group (ADG) | $924 | $894 | $967 | 3.3% | (4.5)%
Analog, MEMS and Sensors Group (AMS) | 1,085 | 968 | 988 | 12.1 | 9.9
Microcontrollers and Digital ICs Group (MDG) | 742 | 688 | 689 | 7.9 | 7.6
Others | 3 | 3 | 4 | — | —
Total consolidated net revenues | $2,754 | $2,553 | $2,648 | 7.9% | 4.0%
Context: Paragraph 1: On a sequential basis, ADG revenues were up 3.3%, driven by an increase in volumes of approximately 8%, partially offset by a decrease in average selling prices of approximately 5%, mostly attributable to product mix. Paragraph 2: AMS revenues increased 12.1% driven by Analog and Imaging products. AMS increase was due to an increase of approximately 5% in average selling prices, entirely due to product mix, and to higher volumes of approximately of 7%. Paragraph 3: MDG revenues increased by 7.9%, mainly driven by Microcontrollers, due to both higher

product mix, and higher volumes of approximately 2%. Paragraph 4: On a year-over-year basis, fourth quarter net revenues increased by 4.0%. ADG revenues decreased 4.5% compared to the year-ago quarter on lower revenues in both Automotive and Power Discrete. The decrease was entirely due to lower average selling prices of approximately 4%, while volumes remained substantially flat. The decrease in average selling prices was a combination of less favorable product mix and lower selling prices. Paragraph 5: AMS fourth quarter revenues grew 9.9% year-over-year, mainly driven by Analog and Imaging. The increase was entirely due to higher average selling prices of approximately 18%, entirely attributable to product mix,
partially offset by lower volumes of approximately 8%. MDG fourth quarter revenues increased by 7.6%, mainly driven by Microcontrollers. The increase was due to higher average selling prices of approximately 9%,
entirely due to improved product mix.
Question: What led to increase in the revenue of ADG on sequential basis?
Thought 1: I need to look up what led to increase in the revenue of ADG on sequential basis in the context.
Action 1: Look up[what led to increase in the revenue of ADG on sequential basis]
Observation 1: On a sequential basis, ADG revenues were up 3.3%, driven by an increase in volumes of approximately 8%, partially offset by a decrease in average selling prices of approximately 5%, mostly attributable to product mix.
Thought 2: The question asks what led to increase in the revenue of ADG on sequential basis, the answer is in the observation 1, which is: an increase in volumes of approximately 8%, partially offset by a decrease in average selling prices of approximately 5%, mostly attributable to product mix. There is no unit or scale needed as the answer is not a number.
Action 2: Finish[an increase in volumes of approximately 8%, partially offset by a decrease in average selling prices of approximately 5%, mostly attributable to product mix.]

Table:
 | March 31, |
 | 2019 | 2018
Raw materials | $74.5 | $26.0
Work in process | 413.0 | 311.8
Finished goods | 224.2 | 138.4
Total inventories | $711.7 | $476.2
Context: Paragraph 1: Inventories Paragraph 2: The components of inventories consist of the following (in millions): Paragraph 3: Inventories are valued at the lower of cost and net realizable value using the first-in, first-out method. Inventory impairment charges establish a new cost basis for inventory and charges are not subsequently reversed to income even if circumstances later suggest that increased carrying amounts are recoverable.
Question: What was the percentage change in total inventories between 2018 and 2019?
Thought 1: I need to retrieve the total inventories of 2018 and 2019 respectively.
Action 1: Retrieve[total inventories of 2018 and 2019]
Observation 1: | 2019 | 2018 |
| $711.7 | $476.2 |
Thought 2: I need to calculate the percentage using the formular: [((711.7-476.2)/476.2)*100]
Action 2: Calculate[((711.7-476.2)/476.2)*100]
Observation 2: 49.45.
Thought 3: The question asks for percentage change of total inventories between 2018 and 2019. The answer is in the observation 2, which is 49.45. The unit/scale of the number is percentage, therefore the answer is 49.45%
Action 3: Finish[49.45%]
(END OF EXAMPLES)
Now generating the Thought, Action, Observation for the following instance:
Table:
{table}
Context: {context}
Question: {question}
{memory}

Figure 9: MACT: Planning agent prompt for TAT.

Solve a table question answering task with interleaving Thought, Action, Observation steps. Thought can reason about the current situation, and Action can be four types:
(1) Retrieve[cells], which retrieves certain cell(s) from the table and returns the retrieved cells in string format.
(2) Calculate[formular/instruction], which carries out calculations based on the formular, or the instruction and returns the calculated results.
(3) Search[entity], which searches the exact entity on Wikipedia and returns the first paragraph if it exists.
(4) Finish[answer], which returns the answer and finishes the task.
You may take as many steps as necessary.
Here are some examples:
Table:
rank	player	county	tally	total	matches	average
1	pádraig horan	offaly	5 - 17	32	4	8
2	billy fitzpatrick	kilkenny	2 - 24	30	4	7.5
3	tony o ’sullivan	cork	0 - 28	28	4	7
4	p j molloy	galway	3 - 11	20	2	10
5	christy heffernan	kilkenny	3 - 9	18	4	4.5
5	pat horgan	cork	0 - 18	18	4	4.5
Table tittle: 1982 all - ireland senior hurling championship
Question: How many players in the 1982 all-Ireland senior hurling championship had a higher average score per game than the overall average score per game of the competition?
Thought 1: I need to first get the overall average score per game of the competition.
Action 1: Calculate[Calculate the overall average score of the competition.]
Observation 1: 6.916666666666667
Thought 2: I need to retrieve the part of the table that only includes players with a higher average score than 6.916666666666667.
Action 2: Retrieve[Retrieve rows where the average score is larger than 6.92.]
Observation 2:	rank	player	county	tally	total	matches	average
1	pádraig horan	offaly	5 - 17	32	4	8
2	billy fitzpatrick	kilkenny	2 - 24	30	4	7.5
3	tony o ’sullivan	cork	0 - 28	28	4	7
4	p j molloy	galway	3 - 11	20	2	10
Thought 3: I need to count the number of rows in the observatoin 2.
Action 3: Calculate[Count the number of rows in the observation 2.]
Observation 3: 4.
Thought 4: The question asks the number of players having a larger average scores than the overall average score. From the observation 3, we know that the answer is 4.
Action 4: Finish[4.]

Table:
season	competition	round	opponent	home	away
2013 - 14	uefa europa league	3q	hapoel ramat gan	0 - 0	1 - 0
2013 - 14	uefa europa league	play - off	pasching	2 - 0	2 - 1
2013 - 14	uefa europa league	group h	sevilla	1 - 2	-
2013 - 14	uefa europa league	group h	slovan liberec	-	1 - 2
2013 - 14	uefa europa league	group h	freiburg	-	1 - 1
Table title: g.d. estoril praia.
Question: Was there a correlation between GD Estoril Praia’s performance in home games and away games during the 2013-14 UEFA Europa League competition? Answer with only ’Yes’ or ’No’ that is most accurate and nothing else.
Thought 1: I need to first find the column of "home" and "away".
Action 1: Retrieve[Retrieve the the column of "home" and "away".]
Observation 1:	home	away
0 - 0	1 - 0
2 - 0	2 - 1
1 - 2	-
-	1 - 2
-	1 - 1
Thought 2: I need to compare the outcome of "home" and "away" games in observation 1. I do not find there is no correlation between "home" and "away" games. As a result, the answer is No.
Action 2: Finish[No.]

```
Table:
| Country | Export 2021(million) | Export 2020(million) |
| U.S.A | 2,827,816 | 2,308,471 |
| Canada | 485,700 | 405,577 |
| France | 683,817 | 552,748 |
| Germany | 1,353,626 | 1,092,044 |
Table title: Export of different countries.
Question: Which country in Europe features the largest percentage change in export number between 2021 and 2020?
Thought 1: I need to identify the rows where the countries are in Europe. Since France and Germany are in Europe, I need to extract rows where the countries is either France or Germany.
Action 1: Retrieve[Retrieve tne rows where the country are either France or Germany.]
Observation 1: | Country | Export_2021 | Export_2020 |
| France| 683,817  | 552,748 |
| Germany | 1,353,626  | 1,092,044 |
Thought 2: I need to calculate the percentage change of export between 2021 and 2022 in observation 1.
Action 2:  Calculate[Calculate tne percentage change between 2021 and 2022 in the observation 1.]
Observation 2: | Country | Percentage_change |
| France| 23.71 |
| Germany | 23.95 |
Thought 3: The question asks the country in Europe that features the largest percentage change. From observation 2, we can see that Germany has the largest percentage change 23.95%. Therefore the answer is Germany.
Action 3: Finish[Germany.]
(END OF EXAMPLES)
Now generating the Thought, Action, Observation for the following instance:
Table:
{table}
Context: {context}
Question: {question}
{memory}
```

Figure 10: MACT: Planning agent prompt for CRT.

Solve a table question answering task with interleaving Thought, Action, Observation steps. Thought can reason about the current situation, and Action can be four types:
(1) Retrieve[cells], which retrieves certain cell(s) from the table and returns the retrieved cells in string format.
(2) Calculate[formular/instruction], which carries out calculations based on the formular, or the instruction and returns the calculated results.
(3) Search[entity], which searches the exact entity on Wikipedia and returns the first paragraph if it exists.
(4) Finish[answer], which returns the answer and finishes the task.
You may take as many steps as necessary.
Here are some examples:
Table:
Model	Training data	Overall	Easy	Hard
BERT-large-FT	B-COPA	74.5 (±0.7)	74.7 (±0.4)	74.4 (±0.9)
BERT-large-FT	B-COPA (50%)	74.3 (±2.2)	76.8 (±1.9)	72.8 (±3.1)
BERT-large-FT	COPA	76.5 (±2.7)	83.9 (±4.4)	71.9 (±2.5)
RoBERTa-large-FT	B-COPA	89.0 (±0.3)	88.9 (±2.1)	89.0 (±0.8)
RoBERTa-large-FT	B-COPA (50%)	86.1 (±2.2)	87.4 (±1.1)	85.4 (±2.9)
RoBERTa-large-FT	COPA	87.7 (±0.9)	91.6 (±1.1)	85.3 (±2.0)
Context: Table caption: Table 5: Results of fine-tuned models on Balanced COPA. Easy: instances with superficial cues, Hard: instances without superficial cues.
Claim: RoBERTa-large outperforms BERT-large when fine-tuned on full and balanced COPA.
Question: Is the above claim true or false ?
Thought 1: I need to retrieve the performance of RoBERTa-large and BERT-large on full and balanced COPA. The B-COPA is the acronym of the balanced COPA. Therefore, I need to retrieve Retrieve the rows where model is BERT-large-FT or RoBERTa-large-FT and training data is COPA or B-COPA.
Action 1: Retrieve[Retrieve the rows where model is BERT-large-FT or RoBERTa-large-FT and training data is COPA or B-COPA.]
Observation 1:	Model	Training data	Overall	Easy	Hard
BERT-large-FT	B-COPA	74.5 (±0.7)	74.7 (±0.4)	74.4 (±0.9)
BERT-large-FT	COPA	76.5 (±2.7)	83.9 (±4.4)	71.9 (±2.5)
RoBERTa-large-FT	COPA	87.7 (±0.9)	91.6 (±1.1)	85.3 (±2.0)
RoBERTa-large-FT	B-COPA	89.0 (±0.3)	88.9 (±2.1)	89.0 (±0.8)
Thought 2: In observation 1, RoBERTa-large has an overall performance of 87.7 on COPA. BERT-large has an overall performance of 76.5 on COPA. Therefore, RoBERTa-large outperforms BERT-large on COPA. For balanced COPA, RoBERTa-large has an overall performance of 89.0, BERT-large has an overall performance of 74.5. RoBERTa-large outperforms BERT-large on balanced COPA. Since RoBERTa-large outperforms BERT-large on both full and balanced COPA, the claim is true.
Action 2: Finish[True]

Table:
Model	Training data	Overall	Easy	Hard
BERT-large-FT	B-COPA	74.5 (±0.7)	74.7 (±0.4)	74.4 (±0.9)
BERT-large-FT	B-COPA (50%)	74.3 (±2.2)	76.8 (±1.9)	72.8 (±3.1)
BERT-large-FT	COPA	76.5 (±2.7)	83.9 (±4.4)	71.9 (±2.5)
RoBERTa-large-FT	B-COPA	89.0 (±0.3)	88.9 (±2.1)	89.0 (±0.8)
RoBERTa-large-FT	B-COPA (50%)	86.1 (±2.2)	87.4 (±1.1)	85.4 (±2.9)
RoBERTa-large-FT	COPA	87.7 (±0.9)	91.6 (±1.1)	85.3 (±2.0)
Context: Table caption: Table 5: Results of fine-tuned models on Balanced COPA. Easy: instances with superficial cues, Hard: instances without superficial cues.
Claim: The difference between RoBERTa-large-FT and BERT-large-FT is 3.8 points on B-COPA, which is significantly smaller than the difference in COPA.
Question: Is the above claim true or false?
Thought 1: I need to retrieve the performance of RoBERTa-large-FT and BERT-large-FT on B-COPA and COPA.
Action 1: Retrieve[Retrieve the overall performance of BERT-large-FT and RoBERTa-large-FT on COPA and B-COPA training set.]
Observation 1:	Model	Training data	Overall
BERT-large-FT	B-COPA	74.5 (±0.7)
BERT-large-FT	COPA	76.5 (±2.7)
RoBERTa-large-FT	COPA	87.7 (±0.9)
RoBERTa-large-FT	B-COPA	89.0 (±0.3)
Thought 2: I need to compare the performance difference between RoBERTa-large-FT and BERT-large-FT on B-COPA.
Action 2: Calculate[89.0-74.5]
Observation 2: 14.5

```
Thought 3: I need to compare the performance difference between RoBERTa-large-FT and BERT-large-FT on COPA.
Action 3: Calculate[87.7-76.5]
Observation 3: 11.2
Thought 4: The difference between RoBERTa-large-FT and BERT-large-FT on B-COPA is 14.5 (in the observation 2). This is significantly larger than
the difference on COPA, which is 11.2 (in the observation 3). Therefore , the Claim is false.
Action 4: Finish[False]

Table:
| Model | Joint Acc. |
| COMER | 88.64% |
| - Hierachical-Attn | 86.69% |
| - MLP | 83.24% |
Context: Table caption: Table 4: The ablation study on the WoZ 2.0 dataset with the joint goal accuracy on the test set. For ''- Hierachical-Attn
'', we remove the residual connections between the attention modules in the CMR decoders and all the attention memory access are based on the
output from the LSTM. For ``- MLP '', we further replace the MLP with a single linear layer with the non-linear activation.
Claim: [CONTINUE] The effectiveness of our hierarchical attention design is proved by an accuracy drop of 1.95% after removing residual
connections and the hierarchical stack of our attention modules.
Question: Is the above claim true or false?
Thought 1: I need to retrieve the perfomance of COMER with the hierarchical attention design.
Action 1: Retrieve[Retrieve the joint acc. of the model COMER.]
Observation 1: 88.64%.
Thought 2: I need to retrieve the perfomance of COMER without the hierarchical attention design. From the context, I find that the model under the
name ''- Hierachical - Attn '' corresponds to the model without the residual connections between the attention modules. So I need to retrieve the
joint acc. of the model ''- Hierachical-Attn''.
Action 2: Retrieve[Retrieve the joint acc. of the model - Hierachical-Attn.]
Observation 2: 86.69%.
Thought 3: I need to calculate the difference in model performances of COMER and - Hierachical-Attn.
Action 3: Calculate[88.64-86.69]
Observation 3: 1.95.
Thought 4: Since the claim mentioned after removing residual connections and the hierarchical stack of our attention modules, there is an accuracy
drop of 1.95%. This is true from the observation 3. Therefore the claim is True.
Action 4: Finish[True]
(END OF EXAMPLES)
Now generating the Thought, Action, Observation for the following instance:
Table:
{table}
Context: {context}
Question: {question}
{memory}
```

Figure 11: MACT: Planning agent prompt for SCITAB.

````
You are given an instruction and a table in pandas dataframe format. Write python code in one code block to retrieve the most relevant rows or/and columns according to the instruction. Return the result in pandas dataframe format and rename it after 'new_table'. Do not use print in the code.
Below are two examples:
Instruction: extract the score of the game between the teams on 6 February 1922.
Table dateframe code: import pandas as pd
data={"Tie no": ["1", "2", "3", "Replay", "4", "5", "6", "7", "8", "9", "10", "11", "Replay", "12", "Replay", "13", "Replay", "Replay", "14", "Replay", "15", "16"], "Home team": ["Liverpool", "Preston North End", "Southampton", "Cardiff City", "Leicester City", "Nottingham Forest", "Aston Villa", "Bolton Wanderers", "Swindon Town", "Tottenham Hotspur", "Barnsley", "Northampton Town", "Stoke", "Brighton & Hove Albion", "Huddersfield Town", "Bradford City", "Notts County", "Notts County", "Crystal Palace", "Millwall", "Southend United", "Bradford Park Avenue"], "Score": ["0\u20131", "3\u20131", "1\u20131", "2\u20130", "2\u20130", "3\u20130", "1\u20130", "1\u20133", "0\u20131", "1\u20130", "3\u20131", "2\u20132", "3\u20130", "0\u20130", "2\u20130", "1\u20131", "0\u20130", "1\u20130", "0\u20130", "2\u20130", "0\u20131", "2\u20133"], "Away team": ["West Bromwich Albion", "Newcastle United", "Cardiff City", "Southampton", "Fulham", "Hull City", "Luton Town", "Manchester City", "Blackburn Rovers", "Watford", "Oldham Athletic", "Stoke", "Northampton Town", "Huddersfield Town", "Brighton & Hove Albion", "Notts County", "Bradford City", "Bradford City", "Millwall", "Crystal Palace", "Swansea Town", "Arsenal"], "Date": ["28 January 1922", "28 January 1922", "28 January 1922", "1 February 1922", "28 January 1922", "28 January 1922", "28 January 1922", "28 January 1922", "28 January 1922", "28 January 1922", "28 January 1922", "28 January 1922", "1 February 1922", "28 January 1922", "1 February 1922", "28 January 1922", "1 February 1922", "6 February 1922", "28 January 1922", "1 February 1922", "28 January 1922", "28 January 1922"]}
df=pd.DataFrame(data)
Code: ```Python
# Filter based on the date
filtered_df = df[df['Date'] == '6 February 1922']
# Rename the dataframe
new_table = filtered_df
```

Instruction: retrieve the number of passengers for Los Angeles and Saskatoon from the table in 2013.
Table dataframe code: import pandas as pd
data={"Rank": ["1", "2", "3", "4", "5", "6", "7", "8", "9"], "City": ["United States, Los Angeles", "United States, Houston", "Canada, Calgary", "Canada, Saskatoon", "Canada, Vancouver", "United States, Phoenix", "Canada, Toronto", "Canada, Edmonton", "United States, Oakland"], "Passengers": ["14,749", "5,465", "3,761", "2,282", "2,103", "1,829", "1,202", "110", "107"], "Ranking": ["", "", "", "4", "", "1", "1", "", ""], "Airline": ["Alaska Airlines", "United Express", "Air Transat, WestJet", "", "Air Transat", "US Airways", "Air Transat, CanJet", "", ""]}
df=pd.DataFrame(data)
Code: ```Python
# Filter the rows for Los Angeles and Saskatoon in 2013
filter_la = (df['City'] == 'United States, Los Angeles') & (df['Rank'] == '1')
filter_sask = (df['City'] == 'Canada, Saskatoon') & (df['Rank'] == '4')
# Apply the filter and store the result in 'new_table'
new_table = df.loc[filter_la | filter_sask, ['City', 'Passengers']]
# Rename the columns as required
new_table.columns = ['City', 'Passengers_2013']
```
Now please write code for the following instruction.
Instruction:{instruction}
Table dataframe code:{table_df}
Code:
````

Figure 12: MACT: Coding agent prompt for retrieval.

```
According to the instruction, write python code in one code block to perform calculations based on the given pandas dataframe. Return the final result after the variable name final_result. The final result can be of either pandas dataframe or string type. Do not use other data type. Do not use print statement in the code block.
Below are two examples:
Instruction: count how many buildings have a height under 200 ft.
Dataframe code: import pandas as pd
data={"Rank": ["1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12", "13", "14", "15", "16", "17", "18", "19", "20", "21=", "21=", "23", "24", "25", "26", "27", "28", "29", "30"], "Name": ["Rhodes State Office Tower", "LeVeque Tower", "William Green Building", "Huntington Center", "Vern Riffe State Office Tower", "One Nationwide Plaza", "Franklin County Courthouse", "AEP Building", "Borden Building", "Three Nationwide Plaza", "One Columbus Center", "Columbus Center", "Capitol Square", "Continental Center", "PNC Bank Building", "Miranova Condominiums", "Fifth Third Center", "Motorists Mutual Building", "Midland Building", "The Condominiums at North Bank Park", "Lincoln Tower Dormitory", "Morrill Tower Dormitory", "Hyatt Regency Columbus", "Key Bank Building", "Adam's Mark Hotel", "Town Center", "8 East Broad Street", "Huntington Building", "Ohio Judicial Center", "16 East Broad Street"], "Height\\nft / m": ["629 / 192", "555 / 169", "530 / 162", "512 / 156", "503 / 153", "485 / 148", "464 / 141", "456 / 139", "438 / 134", "408 / 124", "366 / 112", "357 / 109", "350 / 107", "348 / 106", "317 / 97", "314 / 96", "302 / 92", "286 / 87", "280 / 85", "267 / 81", "260 / 79", "260 / 79", "256 / 78", "253 / 77", "243 / 74", "226 / 69", "212 / 64.6", "202 / 59.4", "200 / 57.9", "180 / 64.4"], "Floors": ["41", "47", "33", "37", "32", "40", "27", "31", "34", "27", "26", "25", "26", "26", "25", "26", "25", "21", "21", "20", "26", "26", "20", "20", "16", "17", "17", "13", "14", "13"], "Year": ["1973", "1927", "1990", "1984", "1988", "1976", "1991", "1983", "1974", "1989", "1987", "1964", "1984", "1973", "1977", "2001", "1998", "1973", "1970", "2007", "1967", "1967", "1980", "1963", "1961", "1974", "1906", "1926", "1933", "1900"], "Notes": ["Has been the tallest building in Columbus and the tallest mid-block skyscraper in Ohio since 1973. Tallest building constructed in Columbus in the 1970s.", "Tallest building constructed in Columbus in the 1920s.", "Tallest building constructed in Columbus in the 1990s.", "Tallest building constructed in Columbus in the 1980s.", "", "", "", "", "", "", "", "Tallest building constructed in Columbus in the 1960s. Was built as the Bank One Tower.", "", "", "", "Tallest residential building in the state of Ohio. Tallest building built in the 2000s.", "", "", "", "", "", "", "", "", "", "", "", "", "", "", ""]}
df=pd.DataFrame(data)
Information:
Code: ```Python
# Conversion of height from string to numeric
df['Height'] = df['Height\\nft / m'].str.split(' / ').str[0].astype(int)
# Filter buildings with height under 200 ft
buildings_under_200ft = df[df['Height'] < 200]
# Counting the number of buildings
final_result = len(buildings_under_200ft)
```

Instruction: calculate the average of gold medals for the top 5 nations.
Dataframe code: import pandas as pd
data={"Rank": ["1", "2", "3", "4", "5"], "Nation": ["United States", "Jamaica", "Netherlands", "Bahamas", "Ukraine"], "Gold": ["5", "4", "2", "1", "1"], "Silver": ["6", "1", "0", "1", "0"], "Bronze": ["5", "1", "0", "0", "1"], "Total": ["16", "6", "2", "2", "2"]}
df=pd.DataFrame(data)
Code: ```Python
top_5_medals = df.["Gold"].astype(int).sum()
final_result = top_5_medals / 5
```
Now generate python code according to the following instruction.
Instruction: {instruction}
Dataframe code: {table_df}
Code:
```

Figure 13: MACT: Coding agent prompt for calculation.