TART: An Open-Source Tool-Augmented Framework for Explainable Table-based Reasoning

Xinyuan Lu[1] **Liangming Pan**[2*] **Yubo Ma**[3] **Preslav Nakov**[4] **Min-Yen Kan**[1]

[1]National University of Singapore [2]University of Arizona

[3]Nanyang Technology University [4]MBZUAI

luxinyuan@u.nus.edu liangmingpan@arizona.edu yubo001@e.ntu.edu.sg

preslav.nakov@mbzuai.ac.ae kanmy@comp.nus.edu.sg

Abstract

Current Large Language Models (LLMs) exhibit limited ability to understand table structures and to apply precise numerical reasoning, which is crucial for tasks such as *table question answering* and *table-based fact verification*. To address these challenges, we introduce our *Tool-Augmented Reasoning framework for Tables* (TART), which integrates LLMs with specialized tools. TART contains three key components: a *table formatter* to ensure accurate data representation, a *tool maker* to develop specific computational tools, and an *explanation generator* to maintain explainability. We also present the TOOLTAB dataset, a new benchmark designed specifically for training LLMs in table–tool integration. Our experiments indicate that TART achieves substantial improvements over existing methods (*e.g.,* Chain-of-Thought) by improving both the precision of data processing and the clarity of the reasoning process. Notably, TART paired with `CodeLlama` achieves 90.0% of the accuracy of the closed-sourced LLM `GPT-3.5-turbo`, highlighting its robustness in diverse real-world scenarios. Both code and data are openly available at https://github.com/XinyuanLu00/TART.

1 Introduction

Tabular data is prevalent across multiple fields such as scientific research, financial reporting, and healthcare records (Dong et al., 2022). Manual handling of such data can be both routine and error-prone, or may require specialized skills, highlighting the need for automated table reasoning to improve efficiency (Badaro et al., 2023). Typical table-based reasoning tasks include *table question answering* (TQA), which extracts precise information from tables to answer given queries (Chen et al., 2020b; Zhu et al., 2021; Nan et al., 2022) and *table-based fact verification* (TFV), which verifies the correctness of statements by cross-referencing them with facts stored in tables (Wang et al., 2021; Lu et al., 2023c).

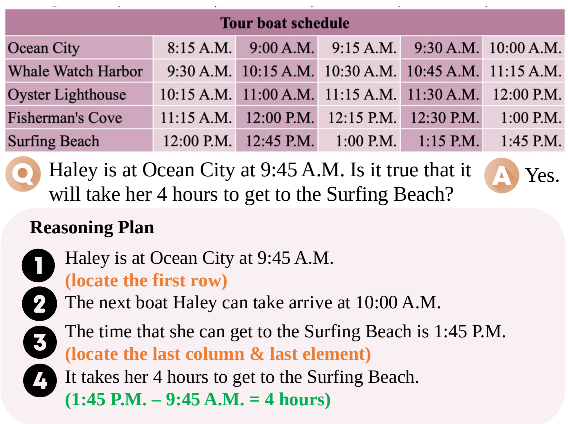


Figure 1: Example of the TableQA task, demonstrating the verification of travel time via boat schedule and the critical skills needed for accurate table reasoning: table structure understanding, precise numerical calculations, and executing sequential reasoning steps.

Modern large language models (LLMs) such as GPT-4 (OpenAI, 2023) have shown remarkable reasoning capabilities across a variety of tasks, spurring interest in their application to table-based tasks (Ye et al., 2023). However, table-based reasoning presents unique challenges for LLMs, which are primarily trained on sequential text data (Zhang et al., 2024a), as illustrated by a real-world example in Figure 1. *(1) Understanding and operating on table structure*: LLMs must adapt to the non-linear format of the tables, which demands unique reasoning skills such as recognizing headers, interpreting the roles of the rows and the columns, and precisely extracting information from relevant table cells. *(2) Precise numerical reasoning:* Tables often contain quantitative information that requires precise calculations and comparisons. LLMs must perform operations such as summation, averaging, or trend analysis accurately, often over multiple cells or tables, which is a shift from their usual text-based reasoning tasks (Herzig et al.,

*Corresponding Author

Findings of the Association for Computational Linguistics: NAACL 2025, pages 4323–4339

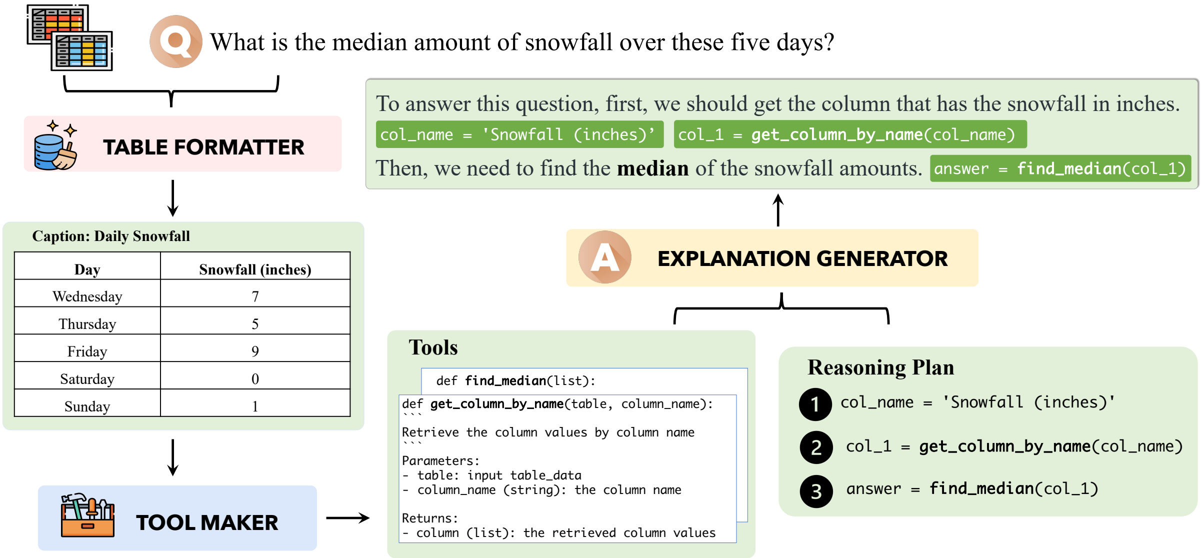


Figure 2: An overall framework of TART, which contains three main modules: (*i*) *table formatter*, which prepares and organizes the raw table data, (*ii*) *tool maker*, which creates specialized tools for precise table manipulation, and (*iii*) *explanation generator*, which produces user-friendly explanations integrating the output of different tools.

2020; Liu et al., 2022). *(3) Reasoning planning:* LLMs often need to plan several reasoning steps ahead. This includes decomposing the original question, determining relevant table parts, and anticipating intermediate calculations or data transformations. All three challenges manifest in Figure 1.

Existing approaches that use LLMs for table-based reasoning can be broadly classified into two categories. One is *chain-of-thought (CoT) reasoning* (Wei et al., 2022), in which the model is prompted to perform step-by-step reasoning over the input table flattened as a textual sequence (Zhang et al., 2024a; Jin and Lu, 2023; Chen, 2023; Ye et al., 2023). Despite its flexibility, CoT often lacks precision in tabular operations and numerical reasoning, such as sorting, counting, and filtering (Wu and Feng, 2024). The second approach, *program-based reasoning (PoT)* (Gao et al., 2023; Chen et al., 2023) prompts the model to generate SQL or Python code to enable precise reasoning (Liu et al., 2024; Zhang et al., 2024b; Wang et al., 2024b; Wu and Feng, 2024). However, this method struggles with reasoning planning and its reasoning is less understandable to humans (Zhang et al., 2024b). Therefore, there is potential value in integrating the advantages of program-based and textual reasoning, to achieve both high precision and explainability in table-based reasoning.

Inspired by the recent paradigm of tool-augmented language models (Wang et al., 2024a; Schick et al., 2023), we propose ***T****ool-****A****ugmented* ***R****easoning framework for* ***T****ables* (TART), which integrates external tool calling into the chain-of-thought reasoning process, as shown in Figure 2. Initially, TART processes the input table using a specialized module *table formatter* to clean and to format the raw table data, preparing it for the subsequent table operations. Subsequently, the *tool maker* calls specialized tools (Python functions) for tabular manipulation and numerical reasoning (e.g., adding columns, selecting rows, and grouping). Alongside these tools, TART also crafts a *reasoning plan* that outlines the programmatic calling sequence of the tools, specifying the necessary arguments and the expected return values for each call. Finally, following the structured reasoning plan, the *explanation generator* produces a hybrid text-and-program output, integrating calls to external tools into coherent, human-readable chain-of-thought explanations. In doing so, TART efficiently delegates table operations and precise numerical calculations to generated tools while preserving CoT's planning ability and explainability.

To train the modules in TART, we further synthesize the TOOLTAB dataset by distilling knowledge from a teacher LLM. We evaluate TART on nine different table-based reasoning benchmarks. The results highlight the effectiveness of integrating task-specific tools for enhancing complex reasoning capabilities. Notably, TART consistently outperformed the CoT baseline, achieving near-parity with `GPT-3.5-turbo` on benchmarks, showcasing its usefulness in real-world scenarios.

In summary, our contributions are threefold:

• We propose TART, a novel framework that enhances table-based reasoning by integrating tools into the reasoning process, which addresses the limitations of current LLMs in handling table structures and executing precise calculations.

• We develop TOOLTAB, a comprehensive benchmark specifically designed to train LLMs on table-tool integration. It includes diverse real-world tables, uniform format, and careful validation to ensure high-quality training.

• Our experiments confirm that TART not only improves the precision and the explainability of table-based reasoning, but also generalizes effectively to out-of-domain datasets.

2 Related Work

Table-based reasoning tasks involve interpreting and manipulating data from structured tabular sources to answer questions, verify facts, or generate summaries. Early approaches used executable SQL or SPARQL to interact with tabular data (Yin et al., 2016; Yu et al., 2018), or graph neural networks to better encode table structures (Zhong et al., 2020; Yang et al., 2020). However, they typically suffer from poor generalization capabilities due to their reliance on specific table formats and linguistic patterns.

Recent advances in large language models (LLMs) have demonstrated significant potential in this area. Pre-training strategies that align LLMs with sentence-table pairs (Chen et al., 2020a; Herzig et al., 2020; Zhou et al., 2022; Gu et al., 2022; Ye et al., 2023; Glockner et al., 2024) have improved table reasoning capabilities, while frameworks like TAP4LLM (Sui et al., 2024b) optimizes table representations through sampling, augmentation, and serialization. Other works, such as *ReAcTable* (Zhang et al., 2024b) and Chain-of-Table (Wang et al., 2024b), have introduced hybrid or explicit reasoning mechanisms to better integrate tabular data into reasoning chains. These methods primarily rely on textual reasoning strategies, such as chain-of-thought, which often lack the precision necessary for table manipulations and numerical reasoning. Efforts have also been made to evaluate and enhance LLM capabilities for table-related tasks. Studies like Table Meets LLM (Sui et al., 2024a) and Text2Analysis (He et al., 2024) introduce benchmarks for tasks such as cell lookup, row retrieval, and Python-based advanced data analysis, emphasizing challenges in table serialization and query understanding. Meanwhile, TabularNet (Du et al., 2021) proposes novel architectures, combining graph-based and relational representations to improve semantic understanding of tabular data. However, these approaches typically focus on either enhancing representation or evaluation rather than augmenting reasoning precision. In addition to general-purpose frameworks, domain-specific solutions such as EHRAgent (Shi et al., 2024) have been developed for multi-tabular reasoning in specialized domains like electronic health records (EHRs). While EHRAgent integrates domain-specific metadata and debugging strategies, its focus contrasts with TART, which is designed as a domain-agnostic solution for table reasoning across diverse contexts. Similarly, API-Assisted Code Generation (Cao et al., 2023) translates queries into Python programs leveraging fixed APIs, differing from TART 's dynamic tool-augmented reasoning approach.

To address the limitations of prior work, TART extends the use of LLMs with integrated external tools, enabling precise table manipulations and numerical reasoning while maintaining explainability. By combining strategies such as table formatting for better representation, tool-based function execution for precision, and LLM-based reasoning for interpretability, TART advances the state of table-based reasoning tasks.

3 Methodology

Generally, a table-based reasoning model, $f_\theta(\cdot)$, parameterized by θ, takes an input query Q and a table $\mathcal{T}$ to produce a response $Y = f_\theta(Q, \mathcal{T})$. Based on this generic formulation, the nature of Q and Y differs depending on the specific table reasoning task:

in table-based QA, Q is a question and Y is the answer; in table-based fact verification, Q is a claim and Y is its veracity label. A table $\mathcal{T}$ is characterized by a caption P and its contents $T_{i,j} \mid i \leq R_T, j \leq C_T$, where R_T and C_T represent the numbers of rows and columns, respectively. Each cell (i, j) contains data $T_{i,j}$.

To build an accurate and explainable table-reasoning framework, our proposed TART integrates the call to external tools into the chain-of-thought reasoning process. TART consists of three reasoning modules (Figure 2): ① *Table Formatter*; ② *Tool Maker*; ③ *Explanation Generator*.

1. Table Formatter. TART first transforms the original table $\mathcal{T}$ with guidance from the query Q into a formatted table $\mathcal{T}'$. The formatter optimizes data formats, aligns columns, and adjusts data types as needed for the query, producing a well-formatted table that is used in subsequent reasoning.

2. Tool Maker. Given $\mathcal{T}'$, the *tool maker* generates a set of candidate tools S useful for solving Q. It also develops a reasoning plan R that details the high-level reasoning which includes the tool calling order, as well as the necessary arguments and the expected return values for the tool calls.

3. Explanation Generator. Given the reasoning plan R as a programmatic guide for chain-of-thought reasoning, the explanation generator is responsible for producing a user-friendly explanation E that incorporates the use of the tools. The explanation also concludes with the final answer $\mathcal{A}$, derived from the reasoning plan R's execution.

3.1 Table Formatter

We first train a specialized open-sourced large language model as the table formatter $\mathcal{F}$, which transforms the noisy raw input table $\mathcal{T}$, into a more structured and manageable format, $\mathcal{T}'$, to facilitate subsequent reasoning: $\mathcal{T}' = \mathcal{F}(\mathcal{T}, Q)$ where the output table $\mathcal{T}'$ is formatted according to three aspects. 1) *Data Cleaning*: the model formats the cell values, such as removing currency symbols and textual footnotes to facilitate the execution of external functions to perform table operations or numerical reasoning. 2) *Data Standardization* converts different data representations into a uniform format;*e.g.,* transforming the data from "`MM/DD/YYYY`" to a consistent "`YYYY-MM-DD`" format across the entire table. 3) *Error Handling*: the model also fixes obvious errors or missing values in the table, such as automatically inferring header names for columns without the table headers.

We introduce the table formatter to ensure that the data in the input table is uniform and optimized for subsequent reasoning, especially to make it more compatible with function execution. In practice, we transform the formatted table $\mathcal{T}'$ into a Python array, facilitating easier interpretation and processing by subsequent reasoning modules.

3.2 Tool Maker

Recent studies have shown that LLMs have the capability of synthesizing relevant tools by understanding the problem context and creating solutions based on the crafted tools (Schick et al., 2023; Cai et al., 2024; Wang et al., 2024a). Motivated by this, we train another specialized LLM $\mathcal{M}$ as a *tool maker*, which takes as input the reformatted table $\mathcal{T}'$ and the query Q to generate a set of candidate tools S and develops a reasoning plan R that details the high-level reasoning steps: $S, R = \mathcal{M}(\mathcal{T}', Q)$. The tool set $S = \{s_1, \cdots, s_n\}$ consists of n specialized tools, where each tool s_i is a Python function that performs table operations (*e.g.*, `get_column_by_name`), numerical reasoning (*e.g.*, `average`, `argmax`), or higher-level functions (*e.g.*, `linear_regression`). These automated tools are essential to handle reasoning tasks that textual-based LLMs cannot address effectively.

Unlike previous work that manually defined a small number (< 10) of hand-crafted tools (Lu et al., 2023a; Pan et al., 2023) or retrieved tools from a predefined set (Qin et al., 2024; Ma et al., 2024), we choose to train a specialized *tool maker* model that **learns to *generate* tools dynamically, based on the specifics of the table and the context of the problem.** This approach not only preserves the model's ability to "extract" previously encountered tools from its parametric memory, but also empowers the model to create novel tools as needed for unique problems, as shown in Section 5.2. While generating tools offer greater flexibility, it is crucial to prevent the tool maker from creating overly-specific tools (*e.g.*, `count_people_on_third_floor`), as this would hinder its ability to generalize to new problems. To address this issue, we incorporate *tool abstraction* and *tool deduplication* steps when constructing synthetic data for training the module (Section 3.4).

The model also constructs a high-level reasoning plan $R = [r_1, \cdots, r_N]$, which outlines how tools should be applied. The reasoning plan is formulated as a sequence of N function calls. Each *function call* $r_i = (s_i, A_i, V_i)$ includes the function $s_i \in \mathcal{S}$, the *argument* A_i passed to the function, and the *variable* V_i that stores the result of the function call $s_i(A_i)$.

This reasoning plan acts as a programmatic blueprint, guiding the table-based reasoning process. Both the tool set S and the reasoning plan R are then provided to the explanation generator, producing the final explainable reasoning output.

3.3 Explanation Generator

While program-based reasoning plans are precise, they are often difficult for non-expert users to understand. Moreover, certain types of reasoning, such as commonsense or narrative-based reasoning, are better communicated in natural language. To address this, TART incorporates a specialized module called the *explanation generator* $\mathcal{E}$, which generates chain-of-thought natural language explanations integrated with function calls, following the steps outlined in the reasoning plan R: $O = \mathcal{E}(S, R)$. The final output O of TART provides detailed explanations for the function calls. For example, the function call `get_column_by_name` is explained as, "*First, retrieve the column listing snowfall in inches.*" Additionally, the explanation generator groups related function calls together to create coherent and easy-to-follow explanations, as illustrated in Figure 2.

3.4 Model Training

As no prior work adopts the tool-augmented LLM framework for table reasoning, there does not exist training data to train the modules *Table Formatter*, *Tool Maker* and *Explanation Generator* ($\mathcal{F}$, $\mathcal{M}$, and $\mathcal{E}$) in TART. Previous studies have demonstrated that smaller LLMs can learn from distilling the generated outputs of larger teacher LLMs that have better reasoning capabilities (West et al., 2022; Wang et al., 2023; Kim et al., 2024). Following this, we use a teacher LLM $\mathcal{L}$ to first synthesize tool-integrated solution trajectories for a set of seed table-based reasoning tasks. These high-quality solution trajectories serve as the blueprint from which we automatically extract and rearrange their components to build training sets for $\mathcal{F}$, $\mathcal{M}$, and $\mathcal{E}$.

Training Data Synthesis. For all modules, we use GPT-4 as the teacher LLM $\mathcal{L}$ to generate training data. As shown in Table 5, we select five diverse table reasoning datasets: two from TQA and three from TFV, spanning general knowledge (Wikipedia) as well as domains such as finance, health, and scientific documents. These datasets provide a broad range of reasoning types. We few-shot prompt $\mathcal{L}$ to generate tool-integrated solutions for training instances for each dataset. In each solution, the model is prompted to clean the table, invent tools, and propose a reasoning plan with explanations. We provide our prompt in Appendix F.

After generating the solutions, we evaluate the final answers against the ground truth, retaining only the instances with correct answers. Subsequently, we refine the solutions by removing overly specific tools through *tool abstraction* and *tool deduplication*. Tool abstraction filters out tools that appear only once, keeping those with broader applicability. Tool deduplication consolidates similar tools that perform the same function, but have different names or implementations (e.g., `add` and `sum`). As a result, we obtain 11,701, 9,916, and 9,916 training instances for the table formatter $\mathcal{F}$, tool maker $\mathcal{M}$, and explanation generator $\mathcal{E}$, We refer to this training dataset as TOOLTAB, with detailed statistics provided in Table 6.

Training Configurations. Instruction fine-tuning (Mishra et al., 2022; Chung et al., 2024) has emerged as a critical strategy that directs LLMs to adhere to specified instructions, facilitating their reasoning capability across a wide range of table-based tasks. Therefore, we use open-source LLMs with instruction tuning as the backbone models for the modules of TART, specifically `Llama-2-7b` (Touvron et al., 2023), `Llama-3-8b`, `CodeLlama-7b` (Rozière et al., 2023) and `Deepseek-Coder-7b-Instruct-V1.5` (Guo et al., 2024). We fine-tune all TART modules independently on their respective training datasets from TOOLTAB, using the standard next-token prediction objective.

4 Experiments

Datasets and Baselines. To evaluate TART, we select two categories of benchmarks for table-based reasoning. *(1) Table question answering (TQA)*: `WikiTableQuestion (WTQ)` (Pasupat and Liang, 2015) focuses on simple factoid questions. `HiTab (HIT)` (Cheng et al., 2022), `TabMWP (TMP)` (Lu et al., 2023b) and `FinQA (FQA)` (Chen et al., 2021) datasets focus on numerical reasoning reasoning. `TAT-QA (TAT)` (Zhu et al., 2021) and `HybridQA (HYQ)` (Chen et al., 2020b) require joint reasoning over the table and the text for financial reports and Wikipedia tables, respectively. *(2) Table-based fact verification (TFV)*: We select `TabFact (TAF)` (Chen et al., 2020a), `SCITAB (SCT)` (Lu et al., 2023c), and `PubHealthTab (PHT)` (Akhtar et al., 2022) datasets, which focus on verifying facts based on tables from Wikipedia, scientific articles, and public health articles, respectively.

For baseline comparisons, we select well-known table-based open-source LLMs such as `TableLlama` (Zhang et al., 2024a) , as well as

	Model	Setting	TableFV TabFact	TableFV PubHT	TableFV SCITAB	TableQA TabMWP	TableQA FinQA	Avg. Acc.
I.	**TableLlama**	w/o Fine-tuning	72.3	72.5	67.4	46.8	3.2	52.4
		w/ DirectQA	72.9	70.5	<u>74.2</u>	48.4	3.7	54.0
II.	**Llama2-7b**	w/ DirectQA	64.4	81.2	64.0	55.3	6.4	54.3
		w/ CoT	52.6	55.0	42.7	74.5	4.2	45.8
		w/ TART	69.2	55.0	53.4	88.8	19.2	57.1 (+24.7%)
	Llama3-8b	w/ DirectQA	<u>74.5</u>	**85.9**	**82.0**	68.6	10.6	64.3
		w/ CoT	48.4	62.4	41.0	88.3	8.5	49.7
		w/ TART	69.7	68.5	47.2	92.6	27.1	61.0 (+22.7%)
	CodeLlama-7b	w/ DirectQA	65.4	75.8	64.6	44.7	4.3	51.0
		w/ CoT	45.2	51.7	38.8	70.7	2.7	41.8
		w/ TART	66.5	69.8	44.9	90.1	25.0	59.3 (+41.9%)
	DeepSeek-7b	w/ DirectQA	72.9	76.5	73.0	62.2	9.0	58.7
		w/ CoT	52.1	62.4	45.5	84.6	8.5	50.6
		w/ TART	71.3	69.1	47.8	<u>93.1</u>	30.9	62.4 (+23.3%)
III.	**GPT-3.5-turbo**	w/ TART	78.7	63.6	59.3	88.3	<u>56.4</u>	<u>69.3</u>
	GPT-4	w/ TART	**87.7**	<u>84.1</u>	63.6	**98.3**	**68.5**	**80.4**

Table 1: Performance evaluation across backbone models using the TART framework, highlighting the best (bold) and second-best (underlined) results. The accuracy is calculated on testing sets, with overall average accuracy in the last column (*Avg. Acc.*). The red numbers indicate the average increase percentage over the CoT methods.

text-pretrained models (`Llama2-7b`, `Llama3-8b`) and code-pretrained models (`CodeLlama-7b`, `DeepSeek-Coder-7b`). We choose the 7b and the 8b versions to represent a balance between computational efficiency and the capacity for complex reasoning and generalization. For each model, we fine-tune with two settings: (1) `DirectQA`, where models generate answers directly from questions and tables, and (2) `Chain-of-Thought (CoT)` reasoning, which requires models to formulate a step-by-step reasoning process before concluding with an answer.

Implementation. For TART, we use the answer given by executing the reasoning plan; if the reasoning plan is not executable, we use the answer given by CoT. For each model, we train the model with TOOLTAB while leaving the rest (WTQ, HIT, TAT, and HYQ) as held-out unseen datasets. All experiments were conducted on a GPU server with Intel Xeon Platinum 8480C (224) @ 2.900GHz CPU and 8 NVIDIA H100 (80G) GPUs. The training process for `Llama-2-7b-hf`, `CodeLlama-7b-hf`, and `deepseek-coder-7b-instruct-v1.5` requires a single GPU for approximately 20 hours, using a batch size of 4, learning rate of 5e-5, sequence length of 1500, gradient accumulation steps of 2, and 10 training epochs. Training `Llama-3-8b` required up to two GPUs for around 20 hours with the same settings. To minimize randomness, a temperature of 0.0 was used, while all other hyperparameters for sampling the output from the LLMs remained at their default values. For the closed-source version of TART, we use `GPT-3.5-turbo` and `GPT-4` with two in-context examples.

4.1 Main Results

We first evaluate TART and the baselines on in-domain datasets, where their training sets are used to construct TOOLTAB. The experimental results, as shown in Table 1, reveal a notable performance improvement in our model compared to baseline models. We have four major observations.

1. TART consistently outperforms `CoT` across all four backbone models and datasets. For example, with `CodeLlama-7b` as the backbone model, TART outperforms `DirectQA` and `CoT` by 16.3% and 41.9% on average, respectively. This highlights the effectiveness of integrating task-specific tools in enhancing complex reasoning capabilities.

2. With `CodeLlama-7b` as the backbone model of TART, it achieves the highest accuracy increase of 41.9%., whereas `Llama3-8b` shows the least improvement of 22.7%. This discrepancy is likely because of `CodeLlama-7b`'s specialized pre-training in coding tasks, which enhances the capabilities of creating tools for structured queries and operations.

3. The performance gains of TART also vary for different datasets, with `FinQA` showing the highest increase, while `PubHealthTab` shows the least. This discrepancy suggests that the financial focus

Model	TAF	PHT	SCT	TMP	FQA	Avg. Acc.
Llama2	69.2	55.0	53.4	88.8	19.2	57.1
- TabFT	67.7	39.1	50.4	67.8	17.2	48.4 (-15.2%)
Llama3	69.7	68.5	47.2	92.6	27.1	61.0
- TabFT	69.2	58.9	46.3	65.4	13.8	50.7 (-16.9%)
CodeLlama	66.5	69.8	44.9	90.1	25.0	59.3
- TabFT	66.6	64.9	36.6	69.1	20.2	51.5 (-13.2%)
DeepSeek	71.3	69.1	47.8	93.1	30.9	62.4
- TabFT	58.0	58.8	45.0	71.3	16.7	50.0 (-19.9%)

Table 2: The ablation study of Table Formatter (- *TabFT*) in TART with different backbone models. The red numbers indicate the average accuracy (*Avg. Acc.*) drop percentage without Table Formatter for each model.

of the FinQA dataset, which demands extensive numerical reasoning and structured data manipulation, benefits significantly from the TART approach.

4. Using closed-source models (GPT-3.5-turbo and GPT-4) as the backbone models for TART achieves an average accuracy of 74.9, significantly outperforming the open-source counterparts, which average at 60.0 accuracy. Nonetheless, the highest-performing open-source model, DeepSeek-7b, reaches up to 90.0% of GPT-3.5-turbo's performance and 77.6% of GPT-4, illustrating the competitiveness of open-source models in the creation and use of tools despite the apparent model size gap.

4.2 Ablation Study

Ablation of the Table Formatter. We conduct ablation experiments across all backbone models (Table 2). TART without the Table Formatter led to significant and uniform performance drops of over 10%. This clearly demonstrates the effectiveness of the Table Formatter module in improving reasoning capabilities by ensuring consistent table representations.

Ablation of the Tool Maker. In this ablation, the framework directly generates programs instead of creating modular tools (Figure 3a). While this approach achieves functionality, it impacts both performance and explanability. These tools (e.g., `get_column_by_name`) can be used repeatedly comparing to the tools that are overly specific (e.g.,`calculate_cookie_difference`).

Ablation of the Explanation Generator. The Explanation Generator module does not directly impact the final performance in terms of accuracy, but enhances the explanability of the outputs. For example, in Figure 3(b), TART's final answer provides more human-readble explanations, comparing to the program.

Tab Formt	TAF	PHT	SCT	TMP	FQA
Llama2	71.8/78.5	75.8/66.4	64.0/57.0	93.6/92.0	73.4/37.7
Llama3	76.6/84.7	79.2/67.8	62.4/55.9	94.1/94.4	71.8/40.0
CodeLlama	67.6/78.0	**81.2/66.1**	**64.6/53.9**	94.1/91.5	76.1/35.7
DeepSeek	**70.7/79.7**	72.5/71.3	63.5/51.3	**95.7/93.9**	**74.5/38.6**
Tool Mkr	**TAF**	**PHT**	**SCT**	**TMP**	**FQA**
Llama2	70.2/81.8	65.8/60.2	53.9/61.5	95.7/91.1	61.7/31.0
Llama3	75.5/75.4	71.1/69.8	63.5/52.2	**97.9/92.4**	62.2/38.5
CodeLlama	75.5/85.2	74.5/71.2	**62.9/57.1**	95.7/91.7	68.1/39.8
Deepseek	**76.6/84.7**	**79.2/67.8**	62.4/55.9	94.1/94.4	**71.8/40.0**

Table 3: Results of TART with different backbone modules. The top half uses `deepseek-code-7b` as the Tool Maker, while the bottom half uses `Llama3-8b` as the Table Formatter. The best performance is highlighted in bold and the second-best is underlined. Tab Formt is *Table Formatter* and Tool Mkr is *Tool Maker*.

4.3 Impact of Foundation Models

To explore the optimal module combinations within the TART framework, we explore various pairings of table formatter and toolmaker modules shown in Table 3. We find that using `Llama3-8b` as the table formatter and `DeepSeek-7b` as the tool maker achieves the best average execution rate (76.8) and accuracy (68.6). This aligns with our expectations given that `Llama3-8b` excels in processing long tables while `DeepSeek-7b`, with its pre-training on code, demonstrates superior capability in tool creation (c.f. Table 7 in Appendix B).

5 Discussion

To further explore the usefulness of TART, we focus on the following research questions: 1) What is the performance of TART on out-of-domain (OOD) datasets? (Section 5.1); 2) How does TART create and utilize tools? (Section 5.2); and 3) What is the quality of the explanations generated by TART in actual user cases? (Section 5.3).

5.1 Out-of-Domain Results

We hypothesize that the TART has enhanced generalization capabilities compared to CoT due to its ability to create and use general tools. To validate this, we further evaluate TART across four different OOD datasets: `HiTab (HIT)`, `WikiTableQuestion (WTQ)`, `TAT-QA (TAT)`, and `HybridQA (HYQ)`. The results are shown in Table 4.

The TART method demonstrates variable effectiveness, with notable improvements in certain contexts. For instance, it achieves an average ac-

Question: Hannah baked cookies each day for a bake sale. How many more cookies did Hannah bake on Saturday than on Sunday?

Pure Program

```
def calculate_cookie_difference(cookie_data):
 saturday_cookies = cookie_data["Saturday"]
 sunday_cookies = cookie_data["Sunday"]
  difference = saturday_cookies - sunday_cookies
return difference
cookie_baking = {"Friday": 163,  "Saturday": 281,  "Sunday": 263}
cookie_difference = calculate_cookie_difference(cookie_baking)
print(cookie_difference)
```

TART's tool-augmented Program

```
table_data = [['Day', 'Number of cookies'], ['Friday', 163], ['Saturday', 281], ['Sunday', 263]]
column_name = 'Number of cookies'
column_ 1 = get_column_by_name(table_data, column_name)
index_1 = get_row_index_by_value(table_data, 'Saturday')
index_2 = get_row_index_by_value(table_data, 'Sunday')
cookies_1 = get_column_cell_value(index_1, column_1)
cookies_2 = get_column_cell_value(index_2, column_1)
answer = subtract(cookies_1, cookies_2)
```

(a) Ablation Study Example of TART's Tool Maker

TART's tool-augmented Program

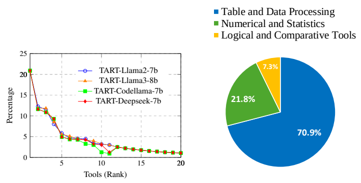
```
table_data = [['Day', 'Number of cookies'], ['Friday', 163], ['Saturday', 281], ['Sunday', 263]]
column_name = 'Number of cookies'
column_ 1 = get_column_by_name(table_data, column_name)
index_1 = get_row_index_by_value(table_data, 'Saturday')
index_2 = get_row_index_by_value(table_data, 'Sunday')
cookies_1 = get_column_cell_value(index_1, column_1)
cookies_2 = get_column_cell_value(index_2, column_1)
answer = subtract(cookies_1, cookies_2)
```

TART's final answer

```
table_data = [['Day', 'Number of cookies'], ['Friday', 163], ['Saturday', 281], ['Sunday', 263]] . First,
we should get the column that has the number of cookies baked each day. <<<column_name = 'Number of cooki
es'>>>;;;<<<column_ 1 = get_column_by_name(table_data, column_name)>>>. Next, we need to find the specifi
c days Hannah baked cookies: Saturday and Sunday. We do this by searching for the row that matches each d
ay's name. <<<column_ 1 = get_column_by_name(table_data, column_name) >>>. Then, we get the number of coo
kies baked on each of these days.<<< index_1 = get_row_index_by_value(table_data, 'Saturday')>>>;;;<<< in
dex_2 = get_row_index_by_value(table_data, 'Sunday')>>>. Finally, we subtract the number of cookies baked
on Sunday from the number baked on Saturday to find out how many more cookies were baked on Saturday. <<<
answer = subtract(cookies_1, cookies_2)>>>.
```

(b) Ablation Study Example of TART's Explanation Generator

Figure 3: Ablation Study of TART's *Tool Maker* and *Explanation Generator*. Panel (a) compares the pure program with TART's tool-augmented program, highlighting the effectiveness of Tool Maker. Panel (b) compares TART's tool-augmented program with TART's final answer, demonstrating the usefulness of Explanation Generator.



Rank	Tool Name
1	get_column_by_name
2	get_column_cell_value
3	get_row_index_by_value
4	extract_price
5	get_row_by_name
6	equal_to
7	get_column_by_index
8	subtract
9	divide
10	add

Figure 4: Analysis of tool usage in the TART framework. Th left and center panels show the distribution and the categories of the top 20 most-used tools across models. The right shows the top-10 tools in TART CodeLlama-7b.

Model	Setting	TQA		Hybrid TQA	
		HIT	WTQ	TAT	HYQ
Llama2-7b	w/ DirectQA	19.1	23.4	15.4	8.5
	w/ CoT	22.1	12.7	20.0	6.7
	w/ TART	19.2	17.0	17.0	6.4
Llama3-8b	w/ DirectQA	**51.1**	**38.8**	20.2	10.1
	w/ CoT	33.8	26.8	<u>29.3</u>	11.0
	w/ TART	<u>34.6</u>	32.5	**29.3**	**12.2**
CodeLlama-7b	w/ DirectQA	17.0	19.1	13.8	6.9
	w/ CoT	16.1	22.6	14.6	9.7
	w/ TART	22.3	30.3	13.8	9.0
Deepseek-7b	w/ DirectQA	27.1	26.2	19.7	11.2
	w/ CoT	20.5	26.2	15.3	8.1
	w/ TART	29.8	<u>33.5</u>	17.0	<u>11.2</u>

Table 4: Out-of-Domain evaluation results for TART framework, highlighting the best (bold) and the second-best (underlined) results.

curacy increase of 29.3% on the WTQ dataset, indicating robust domain-transfer capabilities. The Deepseek-7b backbone model excels, with 30.6% increase in accuracy. We hypothesize that this superiority stems from its pre-training on coding tasks, which equips it with the capability of effectively creating and using tools in novel domains, surpassing pure-text-based pretraining models such as Llama2-7b. The analysis in Section 5.2 supports our hypothesis, suggesting that TART excels in developing generic table reasoning functions that generalize well across various domains.

5.2 Analysis of Tool Creation

We then performed an in-depth analysis of how TART creates and utilizes tools.

5.2.1 Tool Distribution.

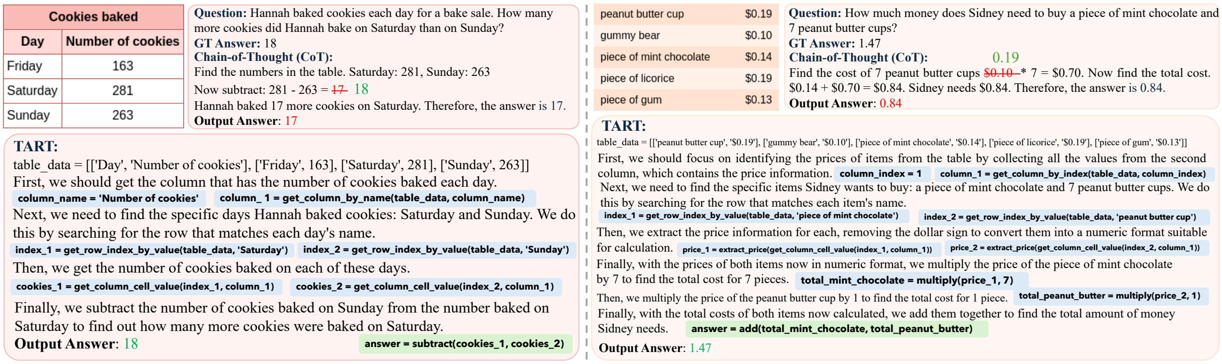
Figure 4 (top-left) illustrates the tool usage distribution across different backbone models in TART, highlighting a long-tail distribution. The most frequently used tools are primarily associated with table processing (e.g., get_column_by_name) and numerical reasoning (e.g., add), aligning with our observations in Section 4.1. Figure 4 (middle) provides a detailed breakdown of tool categories for the top 30 tools, showing that table preprocessing and numerical reasoning tools are the most prevalent. This supports the consistency of tool utilization patterns within TART.

5.2.2 Tool Overlap on OOD Datasets.

Figure 6 illustrates the tool overlap between in-domain datasets and OOD datasets. We find that code pre-training models (CodeLlama-7b and DeepSeek-7b) exhibit a tendency to reuse existing tools when adapting to OOD data. However, text pre-training models demonstrate less overlap, indicating that they tend to solve problems by crafting new tools. The tendency to reuse tools might explain why code pre-training models gain better generalization capabilities in unfamiliar data.

5.2.3 Tool Creation and Usage Analysis.

Figure 7 further reveals that although Llama2-7b frequently reuses tools, it often applies them inap-



(a) Case of numerical calculation error in CoT reasoning. **(b) Case of table location error in CoT reasoning.**

Figure 5: Case Study of TART comparing with CoT reasoning. Panel (a) illustrates a numerical calculation error in CoT where incorrect arithmetic leads to a wrong answer, and panel (b) demonstrates a table location error where CoT fails to retrieve the correct table values. Both errors can be reduced by TART through tool integration.

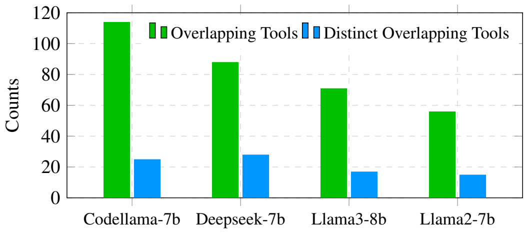


Figure 6: Tool overlap distribution between in-domain and OOD datasets across different backbone models.

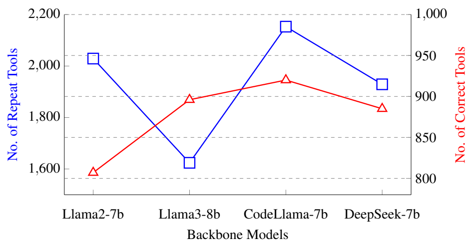


Figure 7: Comparison of the number of repeat tools and correct tools across different backbone models.

propriately. In contrast, `CodeLlama-7b` not only exhibits a high rate of tool reuse but also demonstrates a greater accuracy in their appropriate application. Meanwhile, `Llama3-8b`, despite its lower rate of tool reuse, excels in the correct usage of tools, contributing to its superior performance.

5.3 Case Study

To gain deeper insights into the advantages of TART over `CoT` reasoning, we conducted a case study, shown in Figure 5. The examples highlight the limitation of `CoT` in numerical reasoning and table preprocessing, such as incorrect calculation in Figure 5(a) and incorrect retrieval inFigure 5(b). Conversely, TART overcomes these challenges effectively via integrating specialized tools like `subtract` and `get_column_by_index`. Despite these strengths, TART still encounters issues related to data type mismatches and incorrect programming plans. A detailed analysis of error types in TART can be found in Appendix E.

6 Conclusion

We introduce an open-source framework to improve table-based reasoning through the *Tool-Augmented Reasoning framework for Tables (*TART*)*. This framework solves the challenges of current LLMs' limited ability to understand table structure and execute precise numerical calculations, and maintains explainability. TART consists of a *table formatter* for accurate data representation, a *tool maker* for creating specialized tools, and an *explanation generator* maintaining interpretable explanations. To train TART, we present the TOOLTAB dataset, a novel benchmark containing a diverse set of real-world tables and their tool-augmented solutions. Experiments across nine benchmarks show that integrating our TART method into different open-sourced LLMs enhances accuracy on table-based reasoning. Furthermore, in-depth analyses reveal that TART effectively learns and uses tools. Future work could extend TART to a multimodal framework by incorporating image-based question-answering and fact-verification to generate richer explanations. Additionally, generating explanations to satisfy the needs of different end users, such as laypersons and experts, could further improve the TART's applicability and impact.

Limitations

Despite the promising results, our proposed framework has certain limitations that warrant further investigation:

Computational Complexity. The TART model may affect efficacy, especially when handling simple questions in quick-response scenarios.

Dataset Coverage. While our efforts have focused on expanding the range of our dataset to include a variety of tableQA datasets, some table-related datasets remain unrepresented in TOOLTAB. As a result, despite TART's capacity to adapt to different OOD datasets and tasks, its performance might still different with the complexities and unique challenges of new table tasks and datasets that it has not yet encountered. Having initiated the development of an expansive, versatile tool-enhanced model for tables, we encourage for continued research in this area to further advance the model's ability to generalize across diverse table configurations.

Ethics Statement

Transparency and Integrity. We ensure that all methodologies, data sources, and technologies used in this study are disclosed transparently. We aim to provide a comprehensive and honest account of our findings, acknowledging both the capabilities and limitations of our proposed solution.

Data Privacy and Security. Our research utilizes datasets that are either publicly available or collected with explicit consent. We adhere to strict data privacy and security protocols to protect the information and ensure it is used solely for the purposes of this research.

Acknowledgements

This research is supported by the Ministry of Education, Singapore, under its MOE AcRF TIER 3 Grant (MOE-MOET32022-0001). The computational work for this article was partially performed on resources of the National Supercomputing Centre, Singapore (`https://www.nscc.sg`).

References

Mubashara Akhtar, Oana Cocarascu, and Elena Simperl. 2022. PubHealthTab: A public health table-based dataset for evidence-based fact checking. In *Findings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 1–16.

Gilbert Badaro, Mohammed Saeed, and Paolo Papotti. 2023. Transformers for tabular data representation: A survey of models and applications. *Transactions of Association Computational Linguistics (TACL)*, 11:227–249.

Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. 2024. Large language models as tool makers. In *Proceedings of The 12th International Conference on Learning Representations (ICLR)*.

Yihan Cao, Shuyi Chen, Ryan Liu, Zhiruo Wang, and Daniel Fried. 2023. Api-assisted code generation for question answering on varied table structures. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP) 2023*, pages 14536–14548.

Wenhu Chen. 2023. Large language models are few(1)-shot table reasoners. In *Findings of the 17th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 1090–1100.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research (TMLR)*, 2023.

Wenhu Chen, Hongmin Wang, Jianshu Chen, Yunkai Zhang, Hong Wang, Shiyang Li, Xiyou Zhou, and William Yang Wang. 2020a. Tabfact: A large-scale dataset for table-based fact verification. In *Proceedings of the 8th International Conference on Learning Representations (ICLR)*.

Wenhu Chen, Hanwen Zha, Zhiyu Chen, Wenhan Xiong, Hong Wang, and William Yang Wang. 2020b. Hybridqa: A dataset of multi-hop question answering over tabular and textual data. In *Findings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1026–1036.

Zhiyu Chen, Wenhu Chen, Charese Smiley, Sameena Shah, Iana Borova, Dylan Langdon, Reema Moussa, Matt Beane, Ting-Hao Huang, Bryan R. Routledge, and William Yang Wang. 2021. Finqa: A dataset of numerical reasoning over financial data. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3697–3711.

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

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac

Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, Dasha Valter, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Y. Zhao, Yanping Huang, Andrew M. Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. 2024. Scaling instruction-finetuned language models. *Journal of Machine Learning Research (JMLR)*, 25:70:1–70:53.

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. In *Proceedings of the 31st International Joint Conference on Artificial Intelligence (IJCAI)*, pages 5426–5435.

Lun Du, Fei Gao, Xu Chen, Ran Jia, Junshan Wang, Jiang Zhang, Shi Han, and Dongmei Zhang. 2021. Tabularnet: A neural network architecture for understanding semantic structures of tabular data. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, pages 322–331.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. PAL: program-aided language models. In *Proceedings of the International Conference on Machine Learning (ICML)*, volume 202, pages 10764–10799.

Max Glockner, Yufang Hou, Preslav Nakov, and Iryna Gurevych. 2024. Grounding fallacies misrepresenting scientific publications in evidence. *CoRR*, abs/2408.12812.

Zihui Gu, Ju Fan, Nan Tang, Preslav Nakov, Xiaoman Zhao, and Xiaoyong Du. 2022. PASTA: table-operations aware fact verification via sentence-table cloze pre-training. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4971–4983.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y. Wu, Y. K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. 2024. Deepseek-coder: When the large language model meets programming - the rise of code intelligence. *CoRR*, abs/2401.14196.

Xinyi He, Mengyu Zhou, Xinrun Xu, Xiaojun Ma, Rui Ding, Lun Du, Yan Gao, Ran Jia, Xu Chen, Shi Han, Zejian Yuan, and Dongmei Zhang. 2024. Text2analysis: A benchmark of table question answering with advanced data analysis and unclear queries. In *Proceedings of the 38th AAAI Conference on Artificial Intelligence (AAAI)*, pages 18206–18215.

Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Martin Eisenschlos. 2020. Tapas: Weakly supervised table parsing via pre-training. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 4320–4333.

Ziqi Jin and Wei Lu. 2023. Tab-cot: Zero-shot tabular chain of thought. In *Findings of the Association for Computational Linguistics (ACL)*, pages 10259–10277.

Joongwon Kim, Bhargavi Paranjape, Tushar Khot, and Hannaneh Hajishirzi. 2024. Husky: A unified, open-source language agent for multi-step reasoning. *CoRR*, abs/2406.06469.

Qian Liu, Bei Chen, Jiaqi Guo, Morteza Ziyadi, Zeqi Lin, Weizhu Chen, and Jian-Guang Lou. 2022. TAPEX: table pre-training via learning a neural SQL executor. In *Proceedings of the 10th International Conference on Learning Representations (ICLR)*.

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

Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. 2023a. Chameleon: Plug-and-play compositional reasoning with large language models. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*.

Pan Lu, Liang Qiu, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, Tanmay Rajpurohit, Peter Clark, and Ashwin Kalyan. 2023b. Dynamic prompt learning via policy gradient for semi-structured mathematical reasoning. In *Proceedings of the 11th International Conference on Learning Representations (ICLR)*.

Xinyuan Lu, Liangming Pan, Qian Liu, Preslav Nakov, and Min-Yen Kan. 2023c. 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 (EMNLP)*, pages 7787–7813.

Yubo Ma, Zhibin Gou, Junheng Hao, Ruochen Xu, Shuohang Wang, Liangming Pan, Yujiu Yang, Yixin Cao, and Aixin Sun. 2024. Sciagent: Tool-augmented language models for scientific reasoning. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 15701–15736.

Swaroop Mishra, Daniel Khashabi, Chitta Baral, and Hannaneh Hajishirzi. 2022. Cross-task generalization via natural language crowdsourcing instructions. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 3470–3487.

Linyong Nan, Chiachun Hsieh, Ziming Mao, Xi Victoria Lin, Neha Verma, Rui Zhang, Wojciech Kryscinski, Hailey Schoelkopf, Riley Kong, Xiangru Tang, Mutethia Mutuma, Ben Rosand, Isabel Trindade, Renusree Bandaru, Jacob Cunningham, Caiming Xiong, and Dragomir R. Radev. 2022. Fetaqa: Free-form table question answering. *Transactions of the Association for Computational Linguistics (TACL)*, 10:35–49.

OpenAI. 2023. GPT-4 technical report. *CoRR*, abs/2303.08774.

Liangming Pan, Xiaobao Wu, Xinyuan Lu, Anh Tuan Luu, William Yang Wang, Min-Yen Kan, and Preslav Nakov. 2023. Fact-checking complex claims with program-guided reasoning. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 6981–7004.

Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing (ACL-IJCNLP)*, pages 1470–1480.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *The 12th International Conference on Learning Representations (ICLR)*.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton-Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2023. Code llama: Open foundation models for code. *CoRR*, abs/2308.12950.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems (NeurIPS)*.

Wenqi Shi, Ran Xu, Yuchen Zhuang, Yue Yu, Jieyu Zhang, Hang Wu, Yuanda Zhu, Joyce C. Ho, Carl Yang, and May Dongmei Wang. 2024. Ehragent: Code empowers large language models for few-shot complex tabular reasoning on electronic health records. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, (EMNLP)*, pages 22315–22339.

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 (WSDM)*, pages 645–654.

Yuan Sui, Jiaru Zou, Mengyu Zhou, Xinyi He, Lun Du, Shi Han, and Dongmei Zhang. 2024b. TAP4LLM: table provider on sampling, augmenting, and packing semi-structured data for large language model reasoning. In *Findings of the Association for Computational Linguistics (EMNLP)*, pages 10306–10323.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton-Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurélien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open foundation and fine-tuned chat models. *CoRR*, abs/2307.09288.

Nancy Xin Ru Wang, Diwakar Mahajan, Marina Danilevsky, and Sara Rosenthal. 2021. Semeval-2021 task 9: Fact verification and evidence finding for tabular data in scientific documents (SEM-TAB-FACTS). In *Proceedings of the 15th International Workshop on Semantic Evaluation, SemEval@ACL/IJCNLP 2021*, pages 317–326.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2023. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 13484–13508.

Zhiruo Wang, Zhoujun Cheng, Hao Zhu, Daniel Fried, and Graham Neubig. 2024a. What are tools anyway? A survey from the language model perspective. *CoRR*, abs/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, and Tomas Pfister. 2024b. Chain-of-table: Evolving tables in the reasoning chain for table understanding. In *Proceedings of The 12th International Conference on Learning Representations (ICLR)*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*.

Peter West, Chandra Bhagavatula, Jack Hessel, Jena D. Hwang, Liwei Jiang, Ronan Le Bras, Ximing Lu, Sean Welleck, and Yejin Choi. 2022. Symbolic knowledge distillation: from general language models to commonsense models. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL)*, pages 4602–4625.

Zirui Wu and Yansong Feng. 2024. Protrix: Building models for planning and reasoning over tables with sentence context. In *Findings of the Association for Computational Linguistics (EMNLP)*, pages 4378–4406.

Xiaoyu Yang, Feng Nie, Yufei Feng, Quan Liu, Zhigang Chen, and Xiaodan Zhu. 2020. Program enhanced fact verification with verbalization and graph attention network. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7810–7825.

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 Conference on Research and Development in Information Retrieval (SIGIR)*, pages 174–184.

Pengcheng Yin, Zhengdong Lu, Hang Li, and Ben Kao. 2016. Neural enquirer: Learning to query tables in natural language. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2308–2314.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3911–3921.

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 (NAACL-HLT)*, pages 6024–6044.

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

Wanjun Zhong, Duyu Tang, Zhangyin Feng, Nan Duan, Ming Zhou, Ming Gong, Linjun Shou, Daxin Jiang, Jiahai Wang, and Jian Yin. 2020. Logicalfactchecker: Leveraging logical operations for fact checking with graph module network. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 6053–6065.

Yuxuan Zhou, Xien Liu, Kaiyin Zhou, and Ji Wu. 2022. Table-based fact verification with self-adaptive mixture of experts. In *Findings of the 60th Association for Computational Linguistics (ACL)*, pages 139–149.

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 (ACL-IJCNLP)*, pages 3277–3287.

A Dataset Composition for TART Training

In Table 5, we show the composition of the seed datasets utilized for training our TART model. These datasets vary in terms of the tasks, the domains, and the types of input and output data. For instance, TabMWP and FinQA focus on TableQA tasks within mathematics and finance domains respectively, requiring a combination of tables, text, and questions as inputs, with short answers as outputs. Meanwhile, PubHealthTab, TabFact, and SCITAB target table fact-checking tasks across health, general Wikipedia, and scientific article domains. These datasets similarly involve tables and claims as inputs but differ in the specifics of the domain-related claims, each producing a short label as an output.

Dataset	Task	Domain	Input	Output
1. TabMWP	TableQA	Maths	Table, Question	Answer (Short)
2. FinQA	TableQA	Finance	Table, Text, Question	Answer (Short)
3. PubHealthTab	Table Fact Checking	Health	Table, Claim	Label (Short)
4. TabFact	Table Fact Checking	Wikipedia	Table, Claim	Label (Short)
5. SCITAB	Table Fact Checking	Scientific Articles	Table, Claim	Label (Short)

Table 5: Statistics of the seed datasets for TART training, highlighting their respective tasks, domains, and the nature of input and output data.

To construct the TOOLTAB, we obtain 11,701, 9,916, and 9,916 training instances for the table formatter $\mathcal{F}$, tool maker $\mathcal{M}$, and explanation generator $\mathcal{E}$, respectively. The detailed statistics is provided in Table 6.

Dataset	Train	Dev	Generated Sample	Executable Sample	Table Formatter	Tool Maker	Explanation Generator
TabMWP	23,059	7,686	6,000	5,835	6,000	5,713	5,713
FinQA	6,251	883	1,984	1,609	1,967	1,148	1,148
TabFact	92,283	12,792	1,866	1,773	1,866	1,701	1,705
PubHealthTab	1,180	152	1,180	1,075	1,180	958	958
SciTab	690	-	690	625	688	396	396
Total	123,463	21,513	5,720	10,917	11,701	9,916	9,916

Table 6: Statistics in dataset TOOLTAB for training TART model.

B Different Backbone Combinations

In the pursuit of identifying optimal module combinations within the TART framework, we explore various pairings of table formatter and tool-maker modules shown in Table 7. The combination of `Llama3-8b` as the table formatter and `DeepSeek-7b` as the tool maker performs the most effective pairing, having the best average execution rate and accuracy (76.8 and 68.6 respectively). This best combination aligns with our expectations given that `Llama3-8b` excels in processing long tables while `DeepSeek-7b`, with its pre-training on code, demonstrates superior capability in tool creation.

C Tool Use on Different Backbone Models

Table 8 show the top 10 tools dominate the table processing (e.g., `get_column_by_name`) and numerical reasoning (e.g., `add`), consistent with our earlier findings in Section 4.1. Further illustrating this, Figure 4 (b) presents a tool categorization for the top 30 functions. Table preprocessing tools constitute the highest percentage at 71.0%, followed by numerical reasoning tools at 21.8%. Together, these categories account for over 90% of tool usage, verifying our assumption that TART is better at table preprocessing and numerical reasoning.

D CoT Baseline Implementation

For a direct and fair comparison with TART, the same number of CoT samples are generated using the same IDs from the TART training dataset. These samples are generated using GPT-4, prompted with two in-context examples (detailed in Appendix F). In total, we obtain 9,916 training instances.

Similar to TART, the CoT baseline was implemented across four different backbone models: `Llama-2-7b-hf`, `Llama3-8b`, `CodeLlama-7b-hf`, and `DeepSeek-Coder-7b-Instruct-V1.5`. Each model was instructed to generate a step-by-step reasoning explanation followed by the final answer as per the instructions: `INSTRUCTION: Given the following table, and question, generate a step-by-step reasoning explanation and the final answer.`

The training process was aligned with that of TART to ensure experimental consistency. `Llama-2-7b-hf`, `CodeLlama-7b-hf`, and `deepseek-coder-7b-instruct-v1.5` each requires a single GPU for approximately 12 hours, using a batch size of 4, a learning rate of 5e-5, a sequence length of 1500, gradient accumulation steps of 2, and 10 training epochs. Training `Llama3-8b` requires up to 2 GPUs for around 10 hours with the same settings.

Module Name		TableFV			TableQA		Avg.
Table Formatter	Tool Maker	TabFact	PubHealthTab	SCITAB	TabMWP	FinQA	Exe./Acc.
Llama2	Llama2	64.9/79.5	65.8/59.2	55.1/60.2	90.4/91.8	65.4/26.0	68.3/63.3
Llama2	Llama3	70.7/75.9	73.2/65.1	64.0/46.5	91.0/93.6	60.6/37.7	71.9/63.8
Llama2	CodeLlama	70.2/76.5	73.8/74.5	**64.6/56.5**	94.7/88.8	71.8/34.1	75.0/66.1
Llama2	Deepseek	71.8/78.5	75.8/66.4	64.0/57.0	93.6/92.0	73.4/37.7	75.7/66.3
Llama3	Llama2	70.2/81.8	65.8/60.2	53.9/61.5	95.7/91.1	61.7/31.0	69.5/65.1
Llama3	Llama3	75.5/75.4	71.1/69.8	63.5/52.2	**97.9/92.4**	62.2/38.5	74.0/65.7
Llama3	CodeLlama	75.5/85.2	74.5/71.2	62.9/57.1	95.7/91.7	68.1/39.8	75.3/69.0
Llama3	Deepseek	**76.6/84.7**	79.2/67.8	62.4/55.9	94.1/94.4	71.8/40.0	**76.8/68.6**
CodeLlama	Llama2	64.9/76.2	69.1/59.2	53.4/58.9	94.1/89.3	66.0/26.6	69.5/62.0
CodeLlama	Llama3	66.5/71.2	75.2/69.6	62.4/57.7	94.1/91.0	60.1/36.3	71.2/65.2
CodeLlama	CodeLlama	64.9/75.4	**77.9/75.0**	68.5/50.8	95.2/92.2	71.3/34.3	75.6/65.5
CodeLlama	DeepSeek	67.6/78.0	81.2/66.1	64.6/53.9	94.1/91.5	76.1/35.7	76.7/65.0
DeepSeek	Llama2	63.3/79.8	67.1/60.0	50.0/56.2	94.7/92.1	63.3/32.8	67.7/64.2
DeepSeek	Llama3	66.5/80.8	65.1/69.1	63.5/54.0	94.1/93.2	59.6/42.9	69.8/68.0
DeepSeek	CodeLlama	67.0/80.2	71.1/70.8	58.4/52.9	96.8/90.1	69.7/36.6	72.6/66.1
DeepSeek	DeepSeek	70.7/79.7	72.5/71.3	63.5/51.3	95.7/93.9	**74.5/38.6**	75.4/67.0

Table 7: The TART framework with different backbone modules,highlighting the best (bold) and the second-best (underlined) results.

Rank	Llama2	Llama3	DeepSeek
1	get_column_by_name	get_column_by_name	get_column_by_name
2	get_column_cell_value	get_column_cell_value	get_column_cell_value
3	get_row_index_by_value	get_row_index_by_value	get_row_index_by_value
4	extract_price	extract_price	extract_price
5	equal_to	equal_to	get_row_by_name
6	get_column_by_index	get_row_by_name	equal_to
7	subtract	get_column_by_index	divide
8	get_row_by_name	divide	get_column_by_index
9	add	subtract	subtract
10	multiply	add	add

Table 8: The top 10 functions across TART `Llama2-7b`, TART `Llama3-8b`, and TART `DeepSeek-7b`.

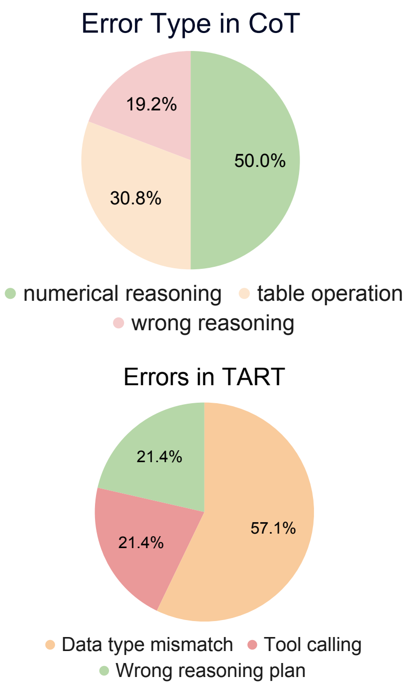


Figure 8: The error types and their distributions of CoT method and our TART framework.

E Error Analysis

To precisely categorize error types in CoT reasoning and TART, we annotate 50 randomly selected error cases for each method. The results (Figure 8) shows that the major error type is incorrect numerical reasoning, followed by errors related to table operations. This analysis verifies the necessity for our proposed TART, which addresses these issues by integrating specialized numerical and table operation tools.

F Prompts

We provide detailed prompts of the TART framework, including the tool discovery process and explanation generation process.

Tool Discovery Prompt:

```
Task Description: Given a table
and a question,
the task is to generate a python
program to
answer the question.
Requirements:
1. First define some functions
to be used in the program.
2. Try to reuse the functions
defined in the previous problems
if possible.
3. When defining a new function,
make sure this function is
general enough to be used in
other problems.
4. Define a function called
solution(table_data) that takes
the table data as input and
returns the answer to the
question.
——
"'
Table: Table Content
Question: Question
Answer: Answer
"'
table_data = table data array

#FUNCTION1 Description
def FUNCTION1():
Function Body

#FUNCTION2 Description
def FUNCTION2():
Function Body
...

def solution(table_data):
Solution Body
return answer

print(solution(table_data))
——
[[FUNCTION_SOLUTION]]
```

Explanation Generation Prompt:

```
Task: Transform Python code
used for a table question
answering task into an easily
understandable explanation in
natural language embedded with
function calls.
Follow these requirements:
1. The explanation should be the
natural language combined with
bracketed segments «< »> for
code.
2. The code segments in the
brackets «< »> should indicate
the line number of the code, with
the format: ###<line number>.
3. Multiple lines of codes are
separated with ';;;' in the
brackets «< »>.
——
"'
Table: Table Content
Question: Question
Answer: Answer
"'
Python Code:
table_data = table data array

def solution(table_data):
Line 1 ###1
Line 2 ###2
...
Line 5 ###5
return answer

print(solution(table_data))

Output Explanation:
First, we should get the column
that ... «<###1 ;;; ###2»>.
...
Finally, we find that «<###5»>.
——
[[OUTPUT_EXPLANATION]]
```

CoT Prompt:

Task Description: Given a table and a question, the task is to generate a step-by-step reasoning explanation and the final answer.

—

```
“‘
Table: Table Content
Question: Question
Answer: Answer
“‘
Python Code:
table_data = table data array


Output Explanation:
To answer this question, first, we ...
Second, to determine..., we compare ...
...

Therefore, the answer is ...
```

—

[[OUTPUT_EXPLANATION]]

TART (GPT-4) Prompt for Table Formatter:

Task Description: Given the following table, context and question, format the table into a python array.

—

```
“‘
Table: Table Content
Question: Question
Answer: Answer
“‘
Python Code:
table_data = table data array
```

—

[[LINEARIZED_TABLE]]

TART (GPT-4) Prompt for Tool Maker:

Task Description: Given the following table, context and question, the table_data, generate the python code to solve it.

—

```
“‘
Table: Table Content
Question: Question
Answer: Answer
table_data = table data array
“‘

Python Code:
#FUNCTION1 Description
def FUNCTION1():
Function Body

#FUNCTION2 Description
def FUNCTION2():
Function Body
...

def solution(table_data):
Solution Body
return answer

print(solution(table_data))
```

—

[[FUNCTION_SOLUTION]]