`CoReTab`: Improving Multimodal Table Understanding with Code-driven Reasoning

Van-Quang Nguyen
RIKEN AIP, Japan
`quang.nguyen.jz@riken.jp`

Takayuki Okatani
GSIS, Tohoku University/RIKEN AIP, Japan
`okatani@vision.is.tohoku.ac.jp`

Abstract

Existing datasets for multimodal table understanding, such as MMTab, primarily provide short factual answers without explicit multi-step reasoning supervision. Models trained on these datasets often generate brief responses that offers insufficient accuracy and limited interpretability into how these models arrive at the final answer. We introduce **CoReTab**, a code-driven reasoning framework that produces scalable, interpretable, and automatically verifiable annotations by coupling multi-step reasoning with executable Python code. Using the `CoReTab` framework, we curate a dataset of 115K verified samples averaging 529 tokens per response and fine-tune open-source MLLMs through a three-stage pipeline. We evaluate the resulting model trained on `CoReTab` across 17 MMTab benchmarks spanning table question answering, fact verification, and table structure understanding. Our model achieves significant gains of +6.2%, +5.7%, and +25.6%, respectively, over MMTab-trained baselines, while producing transparent and verifiable reasoning traces. These results establish `CoReTab` as a robust and generalizable supervision framework for improving multi-step reasoning in multimodal table understanding.

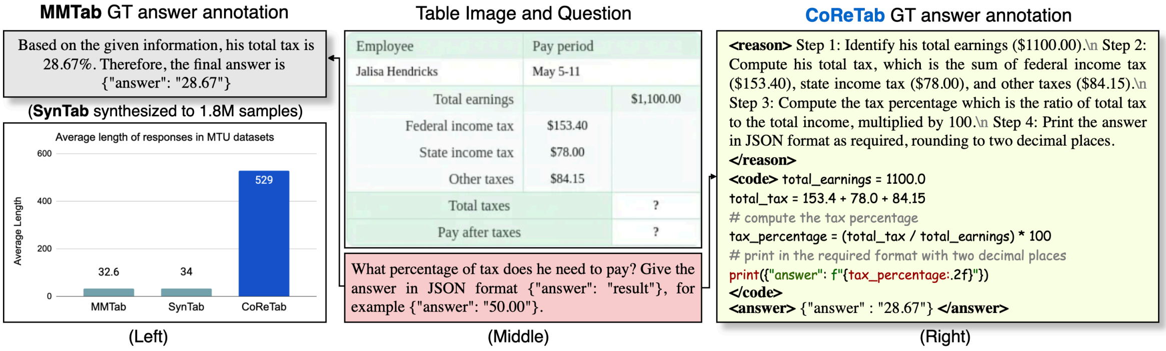
1 Introduction

Tables serve as a primary medium for organizing and communicating (semi-)structured information, with applications spanning finance, science, government, and diverse practical domains (Lautert et al., 2013; Shigarov, 2023). In real-world settings, tables are often embedded as screenshots in documents such as slides, PDFs, or spreadsheets, making the table inherently a visual object. Text-only large language models (LLMs) are ill-suited for such inputs, as extracting text can disrupt spatial layout and introduce inconsistencies. Moreover, directly interacting with table images (e.g., cropping or highlighting) is often more natural and efficient than relying on OCR followed by complex textual representations such as HTML or LaTeX. Prior work shows that even OCR+LLM pipelines substantially underperform multimodal large language models (MLLMs) on table image understanding (Zheng et al., 2024). Consequently, growing attention has focused on multimodal table understanding, which aims to enable machines to comprehend and reason over table images (Li et al., 2022; Luo et al., 2023; Zheng et al., 2024; Zhou et al., 2025). Recent advances in multimodal large language models (Alayrac et al., 2022; OpenAI et al., 2023; Bai et al., 2025) have moved us closer to general-purpose multimodal table understanding, yet their performance remains constrained.

A central bottleneck lies in supervision. Human annotation, as in (Chen et al., 2020; Cheng et al., 2022; Pasupat and Liang, 2015), requires annotators to generate question–answer pairs for each table image, making it accurate but prohibitively costly and slow. Automated synthesis with MLLMs has been explored as an alternative (Zhao et al., 2024; Meng et al., 2024), where the model itself is instructed to generate ⟨question, answer⟩ pairs, given table images. While scalable, this approach struggles with complex table images, resulting in hallucinations and reasoning errors.

Another line of work leverages table codes (textual representations of tables in HTML, LaTeX, or Markdown) that can be rendered into images and paired with generated annotations. Zheng et al. (2024) curated MMTab, a large-scale multimodal dataset of 232K ⟨table image, question, answer⟩ samples by constructing from existing textual table datasets, partially employing a large language model to paraphrase the answers. Zhou et al. (2025) collected more table codes and scaled training resources to 1.8M samples using LLMs to generate question–answer pairs directly from table codes. The annotations in MMTab and SynTab are de-

Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics
Volume 1: Long Papers, pages 6498–6523
March 24-29, 2026 ©2026 Association for Computational Linguistics



MMTab GT answer annotation

Based on the given information, his total tax is 28.67%. Therefore, the final answer is {"answer": "28.67"}

(**SynTab** synthesized to 1.8M samples)

(Left)

Table Image and Question

Employee	Pay period	
Jalisa Hendricks	May 5-11	
Total earnings		$1,100.00
Federal income tax	$153.40	
State income tax	$78.00	
Other taxes	$84.15	
Total taxes		?
Pay after taxes		?

What percentage of tax does he need to pay? Give the answer in JSON format {"answer": "result"}, for example {"answer": "50.00"}.

(Middle)

CoReTab GT answer annotation

<reason> Step 1: Identify his total earnings ($1100.00).\n Step 2: Compute his total tax, which is the sum of federal income tax ($153.40), state income tax ($78.00), and other taxes ($84.15).\n Step 3: Compute the tax percentage which is the ratio of total tax to the total income, multiplied by 100.\n Step 4: Print the answer in JSON format as required, rounding to two decimal places.
</reason>

```
<code> total_earnings = 1100.0
total_tax = 153.4 + 78.0 + 84.15
# compute the tax percentage
tax_percentage = (total_tax / total_earnings) * 100
# print in the required format with two decimal places
print({"answer": f"{tax_percentage:.2f}"})
</code>
```

<answer> {"answer" : "28.67"} **</answer>**

(Right)

Figure 1: (Left-top) Typical response annotation from MMTab (Zheng et al., 2024) which provides limited reasoning supervision. (Left-bottom) Average length of responses of three datasets. (Middle) A pair of ⟨table image, question⟩. (Right) Our **CoReTab** annotation example with step-by-step reasoning and executable Python code.

signed to primarily consist of short answers without multi-step reasoning (Figure 1, left). Models trained on such supervision may excel at question answering but offer little transparency into how these models arrive at the final answer. For real-world applications, where correctness and interpretability are essential, this gap is critical.

It is feasible to use (M)LLMs to automatically generate multi-step reasoning annotations; however, validating the correctness of lengthy reasoning text remains challenging and costly. We address this challenge with **CoReTab**, a **Co**de-driven **Re**asoning framework for multimodal **Tab**le understanding. CoReTab couples natural-language multi-step reasoning with executable code, enabling automatic verification of reasoning correctness. In this framework, an LLM annotator (e.g., Qwen3 (Yang et al., 2025)) is prompted with a table code, a question, and a short ground-truth answer to produce a new annotation that includes both multi-step reasoning and corresponding Python code (Figure 1, right). The Python code is executed and verified against the ground-truth answer, and only consistent samples are retained. Multi-step reasoning text associated with verified code tends to be more accurate and less prone to hallucination. As a result, this closed-loop, code-verifiable framework allows scalable generation of accurate, interpretable reasoning annotations with minimal human effort.

Using this framework, we curate a CoReTab dataset, an augmented version of MMTab containing 115K verified samples across 11 tasks, with average responses being on average an order of magnitude longer than those in MMTab or SynTab. Each annotation includes step-by-step reasoning and executable code, ensuring both interpretability and verifiability. Moreover, the process is cost-efficient: while annotating 2,000 examples for TableBench (Wu et al., 2025) required an estimated $12,000, CoReTab scales to 115K samples at a tiny fraction of that cost using local GPU inference.

To evaluate the effectiveness of CoReTab, we fine-tune Qwen2.5-VL (Bai et al., 2025) using a three-stage training pipeline with LoRA (Hu et al., 2022): (1) table recognition pretraining, (2) instruction tuning on CoReTab, and (3) reinforcement learning optimization. The resulting model consistently outperforms MMTab-trained baselines (e.g., Table-Qwen2.5-VL) across 17 MMTab benchmarks, with average improvements of +6.2% in table question answering, +5.7% in table fact verification, and +25.6% in table structure understanding. Overall, the code-driven reasoning supervision enhances robustness—by teaching models to couple reasoning with executable code—and interpretability, as each prediction provides multi-step reasoning that can be inspected and verified.

We summarize our contributions as follows:

- We introduce CoReTab, the first framework for code-driven reasoning supervision with automatic verification for multimodal table understanding.
- We curate a dataset of 115K verified annotations combining natural-language and executable reasoning, offering richer supervision than existing MMTab and SynTab datasets.
- We demonstrate that the resulting MLLM trained under CoReTab supervision achieves substantial improvements across 17 MMTab benchmarks, delivering more accurate, interpretable, and verifiable answers.

2 Related Work

2.1 Multimodal Table Understanding Datasets

A variety of datasets have been introduced for multimodal table understanding (MTU), such as table recognition (Zhong et al., 2020; Zheng et al., 2021), MMCoQA (Li et al., 2022), and MMTab (Zheng et al., 2024). Among them, MMTab is the most comprehensive, covering 24 tasks across four categories: Table Question Answering (TQA) (Pasupat and Liang, 2015; Cheng et al., 2022; LU et al., 2023; Zhu et al., 2021; Katsis et al., 2021; Nan et al., 2022; Jauhar et al., 2016), Table Fact Verification (TFV) (Chen et al., 2020; Gupta et al., 2020; Akhtar et al., 2022), Table-to-Text generation (T2T) (Parikh et al., 2020; Wiseman et al., 2017), and Table Structure Understanding (TSU). Other works such as UniMMQA (Luo et al., 2023), MMTabQA (Mathur et al., 2024), and MMTBench (Titiya et al., 2025) further integrated visual features into table images for table-based QA and reasoning tasks. More recently, SynTab (Zhou et al., 2025) scaled training resources to 1.2M samples by employing LLMs to synthesize question–answer pairs in short and detailed forms without executable verification, which may limit their consistency and correctness.

2.2 (Multimodal) Large Language Models

Large language models (LLMs) have recently been applied to table understanding via strategies such as prompt engineering (Chen, 2023; Sui et al., 2023), instruction tuning (Zhang et al., 2023b; Li et al., 2023c; Liu et al., 2023c), and tool integration (Lu et al., 2023; Li et al., 2023a; Wang et al., 2024c), giving rise to models like TableLlama (Zhang et al., 2023b) and TableGPT (Li et al., 2023c). However, these approaches primarily process textual tables as inputs rather than table images.

Multimodal large language models (MLLMs) extend LLMs with visual capabilities. MLLMs such as Flamingo (Alayrac et al., 2022), BLIP-2 (Li et al., 2023b), and LLaVA (Liu et al., 2023b) have demonstrated strong performance on general multimodal benchmarks. Document-oriented models like Vary (Wei et al., 2023) and Monkey (Li et al., 2023d) scale to higher resolutions and complex layouts, while Qwen2-VL (Wang et al., 2024a) and Qwen2.5-VL (Bai et al., 2025) are further trained on large-scale in-house datasets, achieving state-of-the-art results in document understanding. Nevertheless, MLLMs' performance remains limited on MMTab benchmarks. Finetuning MLLMs on MMTab and SynTab improves the performance, resulting in Table-LLaVA (Zheng et al., 2024) and SynTab-LLaVA (Zhou et al., 2025).

Recent reasoning-augmented prompting and supervision have improved accuracy and interpretability in text-based domains, ranging from chain-of-thought (Wei et al., 2022) to code interpreter and program-of-thought (Mishra et al., 2022; Chen et al., 2023; Gao et al., 2023). Other research (Lu et al., 2023; Surís et al., 2023) employed LLMs with off-the-shelf modules to perform compositional reasoning and complex visual tasks. For tabular reasoning, Wang et al. (2024c) introduced the Chain-of-Table framework that generates Python code sequences to perform complex operations on textual table data. For reasoning on table images, Cheng et al. (2025) proposed an iterative chain-of-thought generation method to enhance table and document question answering. Building on this line of work, we are the first to introduce the code-driven reasoning framework and a tailored training pipeline to enable MLLMs to excel in MTU tasks across TQA, TFV, and TSU.

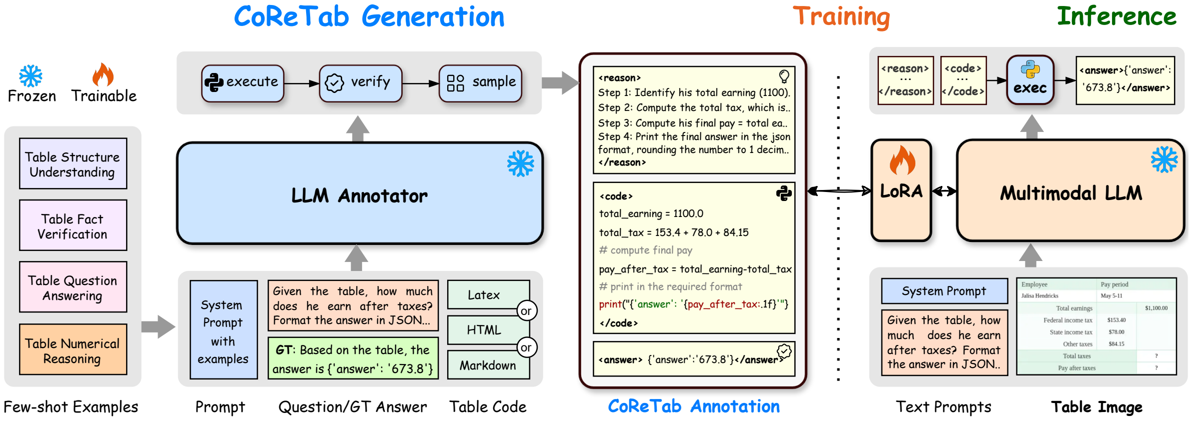
3 `CoReTab` Framework and Dataset

3.1 Data Collection

We construct `CoReTab` from 11 publicly available datasets spanning three major task categories: Table Question Answering (TQA), Table Fact Verification (TFV), and Table Structure Understanding (TSU) (Table 1). These tasks are associated with definitive answers and objective evaluation metrics such as Accuracy, which provide verifiable assessments of generated annotations. Following MMTab (Zheng et al., 2024), we collect table codes (LaTeX, HTML, or Markdown format) as part of the input to the LLM annotator, while the corresponding rendered images are part of ⟨table image, question, answer⟩ `CoReTab` tuples. For comparability and guaranteed correctness of annotations, we reuse pairs of input questions and ground truth answers provided in MMTab, rather than SynTab, which may contain incorrect labels.

3.2 Generation Framework

We employ Qwen3 32B-3A (Yang et al., 2025), a mixture-of-experts model with 3B active parameters out of 32B, as the LLM annotator. This LLM is both sufficiently powerful for our tasks and feasible to deploy on local GPUs, which ensures reproducibility and cost efficiency compared with



```
<reason>
Step 1: Identify his total earning (1100).
Step 2: Compute the total tax, which is..
Step 3: Compute his final pay = total ea..
Step 4: Print the final answer in the json
format, rounding the number to 1 decim..
</reason>
```

```
<code>
total_earning = 1100.0
total_tax = 153.4 + 78.0 + 84.15
# compute final pay
pay_after_tax = total_earning-total_tax
# print in the required format
print("{'answer': '{pay_after_tax:.1f}'"}
</code>
```

```
<answer> {'answer':'673.8'}</answer>
```

Figure 2: Overview of the **CoReTab** framework. It uses an LLM to generate responses as new annotations with multi-step reasoning and Python code, which can be executed and verified against the ground truth with minimal human review. CoReTab annotations provide a rich and interpretable supervision signal for training MLLMs.

proprietary closed-source models. Figure 2 shows the CoReTab framework, in which the LLM annotator takes as input the tuple of ⟨table code, question, its short GT answer⟩and produces a structured CoReTab annotation as a new GT answer. This new annotation provides better supervision signal for training MLLMs.

Input and Prompting. Each generation instance combines a textual table, a question, the ground-truth answer, and a structured system prompt as the input to the LMM annotator, as follows:

Input to the LLM annotator

You are an expert in table analysis. Your task is to process a textual table to answer a question with a code-driven reasoning explanation.

You will be given few-shot examples to guide the expected reasoning and formatting style.

Your output must strictly follow this structure:

1. Reasoning — enclosed in `<reason>` tags. Provide clear, concise, and verifiable reasoning steps that logically lead to the final answer.

2. Python Code — enclosed in `<code>` tags. Write clean, executable Python code that computes the final answer based on the reasoning. The code must print the answer in the correct format.

3. Final Answer — enclosed in `<answer>` tags. This must contain the final answer or the printed output of the code snippet.

Tool Use: {{ `Tool Use Descriptions` }} # optional

Few-shot Examples: {{ `Task examples` }}

New Input: {{ `table, question, GT answer` }}

The system prompt is kept general across tasks, while task-specific few-shot examples are selected from a pool based on the question type and table format (HTML, LaTeX, or Markdown). We further augment the prompt with descriptions of predefined functions (tool-use descriptions) following CodeAct (Wang et al., 2024b), such as `get_cell_content`, when precise table operations are needed for table structure understanding (TSU) tasks; see Appendix for further details.

Generation. Conditioned on this input, the annotator produces a structured annotation consisting of: (1) natural-language reasoning enclosed in **`<reason>`** tags, (2) executable Python code in **`<code>`** tags, and (3) the final answer in **`<answer>`** tags. Together, these components form a CoReTab annotation that exposes intermediate reasoning steps while remaining automatically verifiable.

Post-processing. Despite careful prompt design, LLMs may still produce plausible but incorrect reasoning. To address this, we apply further post-processing steps to filter out the hallucinated outputs. Specifically, we extract the Python code from each generated output, execute it, and compare its result against the ground-truth answer. Only outputs that pass this verification are retained. Additionally, we filter out responses with excessively long reasoning traces and apply down-sampling to maintain dataset balance. Overall, this post-processing procedure mitigates hallucinations and strives to ensure that the resulting annotations are verifiable, all with minimal human-in-the-loop intervention.

3.3 Dataset Analysis

The final CoReTab dataset comprises approximately 115K verified samples across 11 distinct multimodal table understanding tasks, as summarized in Table 1. CoReTab covers tables of varied struc-

Table 1: Breakdown statistics of the `CoReTab` dataset.

Category	Task Name	Dataset	# Samples	Avg. Len
Table Question Answering	Flat TQA	WTQ (2015)	9.5K	335
	Hierarchical TQA	HiTab (2022)	7.5K	257
	Tabular Numerical Reasoning	TABMWP (2023)	30K	287
		TAT-QA (2021)	6K	279
Table Fact Verification	TFV	TabFact (2020)	22K	347
		InfoTabs (2020)	15K	347
Table Structure Understanding	Table Size Detection	TSD (2024)	5K	632
	Table Cell Extraction	TCE (2024)	5K	795
	Table Cell Locating	TCL (2024)	5K	1427
	Merged Cell Detection	MCD (2024)	5K	966
	Row&Col Extraction	RCE (2024)	5K	836
Total			115K	529

tures, formats, and domains, providing a comprehensive and diverse resource for instruction-tuning multimodal LLMs on table understanding tasks. Compared to MMTab and SynTab, `CoReTab` offers substantially richer annotations that explicitly integrate step-by-step reasoning and executable code. On average, `CoReTab` responses contain 529 tokens—nearly 15$\times$ longer than the short ground-truth answers in MMTab (32.6 tokens) and SynTab (34 tokens). This length reflects the multi-step reasoning and detailed derivation that the dataset encourages. See Appendix for more examples of `CoReTab` annotations and other information.

4 CoReTab-Qwen2.5-VL

We adapt the pretrained Qwen2.5-VL (Bai et al., 2025) for multimodal table understanding with *minimal architectural changes* and *low additional computation*. Unlike Table-LLaVA and SynTab-LLaVA, which pretrain models from scratch, we leverage Low-Rank Adaptation (LoRA) (Hu et al., 2022) to inject lightweight trainable parameters into the frozen backbone. LoRA adapters are inserted into the language model layers, enabling efficient finetuning without updating the base model. Each training stage employs *its own set of LoRA adapters*, while adapters from previous stages remain frozen. As shown in Figure 2 (right), CoReTab-Qwen2.5-VL takes a ⟨table image, question⟩ pair as input and generates a structured `CoReTab` response. A system prompt is prepended to the question. During inference, we prioritized the final answer produced by Python code execution; if no output was obtained, the answer enclosed in the `<answer>` tags was used instead.

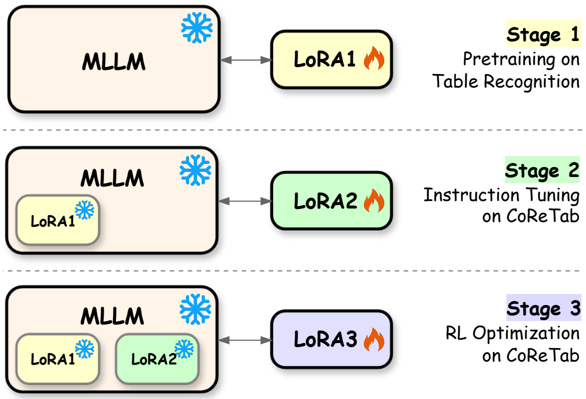


Figure 3: Our three-stage training pipeline: (1) pretraining on table recognition, (2) instruction tuning on `CoReTab`, and (3) optionally RL optimization on `CoReTab`. Separate LoRA adapters are updated in each stage.

4.1 Training Pipeline

As shown in Figure 3, our training pipeline comprises three stages: table recognition pretraining, instruction tuning on `CoReTab`, and RL optimization on `CoReTab` using GRPO. Each stage employs a new set of LoRA adapters, while only Stages 2 and 3 prepend a system prompt to the text input. We denote the final model merged with all three LoRA adapter sets as `CoReTab`-Qwen2.5-VL.

Table Pretraining. As shown in the top of Figure 3, we first adapt Qwen2.5-VL on the MMTab-pre dataset of 150K table recognition samples (Zheng et al., 2024). The model learns to generate textual representations capturing both table structure and content, improving its ability to perceive table layouts and align multimodal features with tabular semantics.

Instruction Tuning. In the second stage, we perform supervised instruction tuning on 115K `CoReTab` samples, covering diverse multi-task scenarios including question answering, reasoning, verification, and structure understanding. LoRA adapters from the pretraining stage are frozen, and a new set of adapters is optimized. We analyze the effect of sharing adapters in the ablation study.

Reinforcement Learning. We further refine the model using GRPO (Shao et al., 2024), training it to maximize a reward function that combines accuracy and format alignment with `CoReTab` preferences following (Guo et al., 2025). LoRA adapters from prior stages remain frozen while a new set is updated. The influence of this stage is also tested in our ablation study.

5 Experiments

5.1 Implementation Details

All experiments use Qwen2.5-VL 7B as the base model. Three separate sets of LoRA adapters are applied sequentially across the three-stage training pipeline, allowing stage-specific adaptation while keeping previous adapters frozen. Each adapter is configured with a rank of 8 and alpha of 32 on all LLM linear layers. We optimize all trainable parameters using AdamW (Loshchilov and Hutter, 2017) with a global batch size of 256 for 2 epochs per each stage. Input table images are resized to ensure the number of image tokens does not exceed 1024. Our training is performed on 8 H200 GPUs. See the Appendix for additional details.

5.2 Experimental Setup

Baselines. We compare `CoReTab` against three categories of methods with results reported in prior work (Zheng et al., 2024; Zhou et al., 2025) if available:

(1) **Open-source MLLMs.** These include LLaVA-1.5 (Liu et al., 2023a), Vary-toy (Wei et al., 2024), Monkey (Li et al., 2023d), mPLUG DocOwl-1.5 (Hu et al., 2024), Internlm Xcomposer-2.5 (Zhang et al., 2023a), Ovis-1.5 (Lu et al., 2024), InternVL2 8B (Chen et al., 2024), Qwen2-VL 7B (Wang et al., 2024a), and TabPedia (Zhao et al., 2024). We also include the zero-shot results of Qwen2.5-VL 7B (Bai et al., 2025).

(2) **Open-source LLMs.** We include LLaMA (Touvron et al., 2023) and its counterpart TableLLaMA (Zhang et al., 2023b) with results reported in (Zheng et al., 2024).

(3) **Closed-source MLLM.** GPT-4V is included as a strong zero-shot baseline, with results reported on 100–200 randomly sampled examples per benchmark in Zheng et al. (2024).

(4) **Trained MLLMs.** We consider SynTab-LLaVA 7B (trained on SynTab & MMTab), Table-LLaVA 7B/13B, and Qwen2.5-VL 7B trained on MMTab (denoted Table-Qwen2.5-VL 7B). We also report results from our `CoReTab`-Qwen2.5-VL 7B trained on `CoReTab`. All models were evaluated in single runs due to computational constraints.

Evaluation Benchmarks and Metrics. We evaluate `CoReTab` on 17 MMTab benchmarks (Zheng et al., 2024) spanning three categories: Table Question Answering (TQA), Table Fact Verification (TFV), and Table Structure Understanding (TSU). Specifically, TQA includes WTQ, HiTab, TAT-QA, and TABMWP; TFV includes TabFact, InfoTabs, and PubHealthTab; and TSU includes TSD, Table Cell Extraction (TCE), Table Cell Locating (TCL), Merged Cell Detection (MCD), and Row&Column Extraction (RCE). For TSU, we additionally evaluate 4 out-of-distribution benchmarks (denoted as TSD^{Φ}, TCE^{Φ}, TCL^{Φ}, and RCE^{Φ}) whose tables do not appear in training.

We adopt task-specific metrics following (Zheng et al., 2024). For tasks with single-value ground truths (TQA, TFV, TCE, TCL), we report Accuracy after normalizing both predictions and references (e.g., lowercasing text, rounding numbers to the same precision). For TSD, we compute Accuracy separately for row and column counts. MCD and RCE are evaluated using cell-level F1 scores, with RCE reported independently for rows and columns.

5.3 Results and Analysis

Table Question Answering Results. Table 2 presents results on five question answering benchmarks: TABMWP, WTQ, HiTab, TAT-QA, and AIT-QA. Among open-source MLLMs, Qwen2.5-VL-7B achieves the highest accuracies, surpassing the closed-source GPT-4V on three benchmarks (WTQ, HiTab, and TAT-QA). However, it still lags behind models trained with task-specific datasets (i.e., SynTab, MMTab, and CoReTab).

Our `CoReTab`-Qwen2.5-VL achieves the highest performance among all compared methods, scoring 97.14% and 82.60% on numerical reasoning tasks (TABMWP and TAT-QA), 60.93% on WTQ, 67.45% on HiTab, and 81.57% on AIT-QA. It consistently improves results compared to Table-Qwen2.5-VL—specifically by over +14.96% on TAT-QA—highlighting the benefit of code execution for precise numerical reasoning. Our model also surpasses SynTab-LLaVA by large margins, despite SynTab-LLaVA being trained on 2.1M samples, indicating that code-driven supervision can offset smaller data scales. Part of this gap may also stem from the stronger pretrained Qwen2.5-VL compared to LLaVA.

Interestingly, Table-Qwen2.5-VL trained on MMTab also performs well on TABMWP, as MMTab inherits reasoning annotations from TABMWP, emphasizing the value of reasoning-augmented supervision for MLLMs.

Table 2: Evaluation results on 7 MMTab benchmarks for TQA and TFV tasks. For all evaluation metrics, higher values indicate better performance. For Qwen2.5-VL-based models, the images are resized to a maximum of 1024 image tokens. '+Oracle' and '+OCR' denote whether the ground truth or OCR-extracted textual table representations are provided to LLMs, respectively.

			Question Answering					Fact Verification		
Method	**LLM**	**Res.**	**TABMWP**	**WTQ**	**HiTab**	**TAT-QA**	**AIT-QA**	**TabFact**	**InfoTabs**	**PubHTab**
			Acc.	Acc.	Acc.	Acc.	Acc.	Acc.	Acc.	
Open-source MLLM (zero-shot)										
LLaVA v1.5 7B	Vicuna-1.5 7B	336	6.00	51.24	2.03	2.97	-	18.92	8.31	-
Vary-toy 2B	Qwen 1.8B	1024	4.42	7.96	3.42	8.81	9.39	6.33	6.98	-
Monkey 7B	Qwen 7B	896	13.26	19.07	6.41	12.31	-	22.56	22.11	18.89
Docowl1.5 7B	LLaMA2 7B	448	11.41	26.80	11.10	12.44	46.18	27.67	28.74	28.42
IXcomposer2.5 7B	InternLM2 7B	560	26.65	38.09	15.29	19.43	51.08	13.72	8.63	9.17
Ovis1.5 8B	LLaMA3 8B	384	27.37	15.00	6.41	15.67	13.31	8.28	25.04	18.18
InternVL2 8B	InternLM2.5 7B	448	16.51	25.09	7.68	13.99	23.28	28.32	36.63	37.23
Qwen2-VL 7B	Qwen2 7B	-	26.04	37.38	25.19	22.67	67.12	16.29	39.67	32.60
TabPedia 7B	Vicuna-1.5 7B	2560	12.27	20.37	1.22	9.71	17.22	28.84	9.20	21.01
Qwen2.5-VL 7B	Qwen2.5 7B	1024*	54.69	55.23	38.09	56.88	50.59	71.37	70.21	70.15
LLMs (zero-shot)										
LLaMA2+*Oracle*	LLaMA 2 7B	-	17.88	4.26	1.21	3.62	-	4.21	7.55	-
LLaMA2+*OCR*	LLaMA 2 7B	-	16.35	3.91	0.77	5.27	-	4.32	7.17	-
TableLLaMA+*Oracle*	LLaMA 2 7B	-	12.98	31.63	64.71	2.84	-	82.55	2.85	-
TableLLaMA+*OCR*	LLaMA 2 7B	-	17.88	4.26	1.21	3.62	-	44.54	2.18	-
Close-sourced MLLMs (zero-shot)										
GPT-4V Low Res	Unkown	512	60.00	22.50	9.50	19.50	19	45.50	58.50	59.50
GPT-4V Low Res	Unkown	2000	60.50	48.00	27.50	32.50	62.5	45.50	65.60	67.00
Trained MLLMs	**Dataset (#Samples)**									
SynTab-LLaVA 7B	MMTab+SynTab (2.1M)	1536	88.30	39.59	35.66	51.94	28.57	70.78	69.42	68.02
Table-LLaVA 7B	MMTab (232K)	336	57.78	18.43	10.09	12.82	5.48	59.85	65.26	51.03
Table-LLaVA 13B	MMTab (232K)	336	59.77	20.41	10.85	15.67	6.06	65.00	66.91	48.46
Table-Qwen2.5-VL 7B	MMTab (232K)	1024*	91.85	55.51	66.68	65.45	76.61	77.02	69.52	76.96
CoReTab Qwen2.5-VL 7B	CoReTab (115K)	1024*	**97.14**	**60.93**	**67.45**	**82.60**	**81.57**	**84.28**	**76.04**	**80.21**

Table Fact Verification Results. Table 2 reports results on TabFact, InfoTabs, and PubHTab. Among zero-shot models, Qwen2.5-VL already surpasses the closed-source GPT-4V on all three benchmarks, underscoring the competitiveness of open-source MLLMs for factual reasoning tasks.

Finetuning with CoReTab yields absolute gains of +12.91% on TabFact, +5.83% on InfoTabs, and +10.06% on PubHTab. Compared with the MMTab-trained baseline (Table-Qwen2.5-VL), it achieves an average improvement of 5.65% across three tasks. Our responses are also more transparent, derived from explicit reasoning steps and further verified with code execution, rather than only short answers such as "confirmed" or "contradicted" words.

Table Structure Understanding Results. Table 3 reports results on 10 MMTab benchmarks for Table Structure Understanding, including 4 out-of-distribution benchmarks. Among open-source and close-sourced MLLMs in the zero-shot setting, Qwen2.5-VL 7B also attains the best performance such as reaching accuracies of 27.9/59.6 on TSD and 31.12 on TCE. This improves over earlier baselines such as InternVL2-8B and GPT-4v.

Our CoReTab-Qwen2.5-VL 7B substantially outperforms all baselines, achieving 88.70/87.60 on TSD, 85.29 on TCE, 87.41 on TCL, 82.92 on MCD, and 89.98/92.51 on RCE. It substantially outperforms Table-Qwen2.5-VL, achieving an average improvement of +25.56%, highlighting the benefits of reasoning with executable code for accurately detecting table size, identifying merged cells, and localizing cell content. Similar trends hold on out-of-distribution (OOD) tasks (i.e., TSD$^{\Phi}$, TCE$^{\Phi}$, TCL$^{\Phi}$, and RCE$^{\Phi}$).

LLM-only Performance. As shown in Tables 2 and 3, LLM-only methods (including LLaMA and TableLLaMA) substantially underperform multimodal LLMs, highlighting the necessity of MLLMs for effective table image understanding.

Qualitative Results. Figure 4 compares the responses generated by Qwen2.5-VL models trained on either CoReTab or MMTab. The response from our CoReTab-Qwen2.5-VL is accurate and interpretable, providing explicit reasoning traces and executable code for verifying intermediate steps and the final answer. In contrast, Table-Qwen2.5-

Table 3: Evaluation results on 10 Table Structure Understanding benchmarks. For all evaluation metrics, higher values indicate better performance. We denote the OOD benchmarks with Φ. For Qwen2.5-VL-based models, the images are resized to a maximum of 1024 image tokens. '+Oracle' and '+OCR' denote whether the ground truth or OCR-extracted textual table representations are provided to LLMs, respectively.

Method	**LLM**	**Res.**	**TSD**		**TSDΦ**		**TCE**	**TCEΦ**	**TCL**	**TCLΦ**	**MCD**	**RCE**		**RCEΦ**	
			Row Acc.	Col. Acc.	Row Acc.	Col. Acc.	Acc.	Acc.	Acc.	Acc.	F1	Row F1	Col. F1	Row F1	Col. F1
Open-source MLLMs (zero-shot)															
LLaVA v1.5 7B	Vicuna-1.5 7B	336	0.80	2.50	2.40	-	0.22	-	0.62	0.93	1.26	1.66	4.13	-	-
Vary-toy 2B	Qwen 1.8B	1024	1.30	2.20	-	-	1.96	-	0.73	-	0.52	2.01	2.38	-	-
Monkey 7B	Qwen 7B	896	0.80	0.60	-	-	1.46	0.76	1.31	-	0.67	3.89	4.53	4.29	-
Docowl1.5 7B	LLaMA-2 7B	448	0.20	1.50	0.40	0.80	1.00	1.80	0.00	0.00	0.00	2.40	4.60	0.60	1.30
IXcomposer-2.5 7B	InternLM2 7B	560	1.60	8.00	2.40	12.40	3.66	3.80	1.25	3.13	2.30	0.17	0.09	0.00	0.00
Ovis1.5 8B	LLaMA-3 8B	384	5.40	13.00	7.60	16.40	4.69	5.64	2.39	3.33	2.42	13.19	20.30	26.43	37.84
InternVL2 8B	InternLM2.5 7B	448	7.70	34.10	12.40	40.80	11.59	12.36	4.86	9.45	0.79	1.46	9.86	1.11	6.95
Qwen2-VL 7B	Qwen2 7B	-	4.70	15.30	5.20	20.40	9.10	8.35	3.86	6.13	1.03	17.14	23.07	21.52	30.39
TabPedia 7B	Vicuna-1.5 7B	2560	2.80	10.20	4.80	12.40	0.16	0.11	0.00	0.00	0.00	1.53	3.73	1.62	1.21
Qwen2.5-VL 7B	Qwen2.5 7B	1024*	27.90	59.60	44.00	55.20	31.12	42.62	14.33	19.57	0.26	40.71	61.77	38.90	59.96
LLMs (zero-shot)															
LLaMA2+*Oracle*	LLaMA 2 7B	-	1.70	3.60	-	-	0.62	-	0.17	-	-	9.36	18.03	-	-
LLaMA2+*OCR*	LLaMA 2 7B	-	1.30	3.40	-	-	0.35	-	0.15	-	-	8.35	10.45	-	-
TableLLaMA+*Oracle*	LLaMA 2 7B	-	5.30	4.40	-	-	9.35	-	0.82	-	-	4.34	18.03	-	-
TableLLaMA+*OCR*	LLaMA 2 7B	-	3.90	3.70	-	-	3.95	-	0.65	-	-	2.82	2.39	-	-
Close-sourced MLLMs (zero-shot)															
GPT-4V-Low-res	Unknown	512	6.00	24.00	8.00	15.00	3.57	10.29	14.41	17.73	2.12	30.32	56.86	27.69	50.36
GPT-4V-High-res	Unknown	2000	12.50	46.00	19.00	38.00	9.75	14.36	23.38	27.91	3.50	26.44	43.17	48.52	57.14
Trained MLLMs	**Dataset (# samples)**														
SynTab-LLaVA 7B	MMTab+SynTab (2.1M)	1536	56.20	82.40	51.60	62.80	50.29	44.90	60.80	51.33	52.93	54.16	73.33	51.73	55.55
Table-LLaVA 7B	MMTab (232K)	336	33.10	33.20	25.20	16.40	19.45	11.28	29.31	26.10	17.14	31.43	37.93	21.97	18.14
Table-LLaVA 13B	MMTab (232K)	336	34.40	27.60	31.60	14.80	19.53	11.38	29.68	26.17	16.52	31.07	41.49	21.94	18.67
Table-Qwen2.5-VL 7B	MMTab (232K)	1024*	52.90	78.30	52.80	74.80	58.67	64.64	63.11	67.64	39.65	65.13	77.71	72.01	83.97
`CoReTab`-Qwen2.5-VL 7B	`CoReTab` (115K)	1024*	**88.70**	**87.60**	**85.60**	**87.60**	**85.29**	**80.48**	**87.41**	**80.16**	**82.92**	**89.98**	**92.51**	**87.36**	**87.85**

VL produces a shorter, less informative response. Refer to our Appendix for more complex examples.

Prediction Analysis. We also analyzed randomly 100 model's prediction errors per task. For TQA, 72 cases from cell extraction mainly due to small text in large tables or challenging table layouts; 22 cases from incorrect arithmetic logic; 3 cases related to translating reasoning to code. For TFV tasks, 76 cases from cell extraction errors; the remainder from factual reasoning consistency. For TSU tasks, 94 from cell extraction errors; only a small fraction from code-logic issues.

5.4 Ablation study

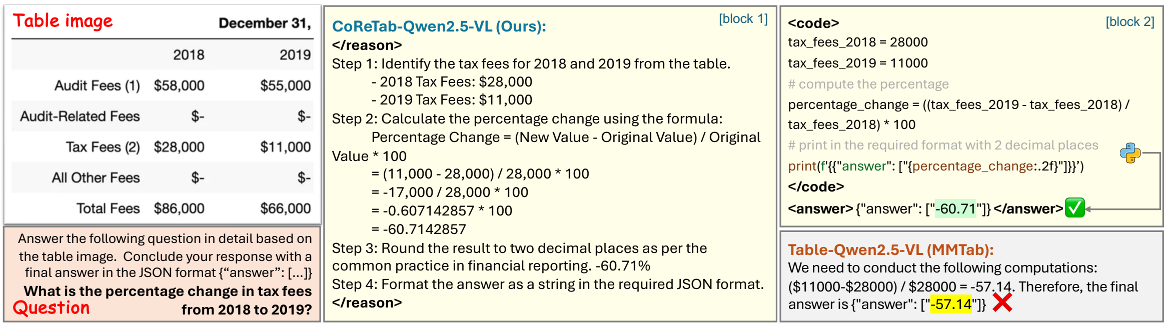
We conduct ablation experiments to assess the contributions of code-driven annotations and configurations to train Qwen2.5-VL with the `CoReTab` dataset. Table 4 reports average results across three benchmark categories—TQA, TFV, and TSU.

Effect of Code-driven Annotations. Removing the **`<code>`** tags results in a large performance drop: -4.19% on TQA, -3.26% on TFV, and a striking -17.96% on TSU. This demonstrates that code-driven annotations are central to the success of `CoReTab`, especially for TSU tasks that require precise localization and detection.

Full-parameter vs. LoRA Fine-tuning. Full-parameter fine-tuning reduces performance relative to LoRA training by 0.76%, 1.37%, and 1.58% on TQA, TFV, and TSU, respectively. This suggests that full-parameter updates may disturb pretrained representations, while LoRA provides a more stable and efficient adaptation strategy.

Shared vs. separate LoRAs. We found that shared LoRA adapters across tasks (block (4)) achieve lower results (-1.2% on average) compared to separate LoRA adapters for each stage, indicating that stage-specific LoRA training better preserves knowledge across the multi-stage pipeline.

Impact of Reinforcement Learning. Removing reinforcement learning leads to drops of 1.23% and 0.84% on TQA and TFV, but a slight gain (+0.52%) on TSU. We interpret this as reinforcement learning being particularly effective for benchmarks that demand flexible multi-step reasoning (TQA, TFV), while its effect is limited—or even slightly negative—for TSU tasks, which can be solved with



	December 31,	
	2018	2019
Audit Fees (1)	$58,000	$55,000
Audit-Related Fees	$-	$-
Tax Fees (2)	$28,000	$11,000
All Other Fees	$-	$-
Total Fees	$86,000	$66,000

Answer the following question in detail based on the table image. Conclude your response with a final answer in the JSON format {“answer”: [...]}
What is the percentage change in tax fees from 2018 to 2019?

CoReTab-Qwen2.5-VL (Ours):

```
</reason>
Step 1: Identify the tax fees for 2018 and 2019 from the table.
        - 2018 Tax Fees: $28,000
        - 2019 Tax Fees: $11,000
Step 2: Calculate the percentage change using the formula:
        Percentage Change = (New Value - Original Value) / Original
Value * 100
        = (11,000 - 28,000) / 28,000 * 100
        = -17,000 / 28,000 * 100
        = -0.607142857 * 100
        = -60.7142857
Step 3: Round the result to two decimal places as per the
common practice in financial reporting. -60.71%
Step 4: Format the answer as a string in the required JSON format.
</reason>
```

```
<code>
tax_fees_2018 = 28000
tax_fees_2019 = 11000
# compute the percentage
percentage_change = ((tax_fees_2019 - tax_fees_2018) /
tax_fees_2018) * 100
# print in the required format with 2 decimal places
print(f'{{"answer": ["{percentage_change:.2f}"]}}’)
</code>
<answer> {"answer": ["-60.71"]} </answer> ✅
```

Table-Qwen2.5-VL (MMTab):
We need to conduct the following computations: ($11000-$28000) / $28000 = -57.14. Therefore, the final answer is {"answer": ["-57.14"]} ❌

Figure 4: Comparison of responses generated by our CoReTab-Qwen2.5-VL and the Table-Qwen2.5-VL baseline.

Setting	Configuration	TQA	TFV	TSU
(1)	CoReTab-Qwen2.5-VL	77.94	80.18	87.77
(2)	w/o **<code>** tags	73.75	76.92	69.81
	Δ = (2) - (1)	-4.19	-3.26	-17.96
(3)	with full-parameter training	77.18	78.81	87.19
	Δ = (3) - (1)	-0.76	-1.37	-0.58
(4)	with shared LoRA adapters	76.84	79.25	86.14
	Δ = (4) - (1)	-1.10	-0.93	-1.63
(5)	w/o RL optimization	76.71	79.34	88.29
	Δ = (5) - (1)	-1.23	-0.84	+0.52
(6.1)	InternVL3 8B + MMTab	67.32	71.28	61.32
(6.2)	InternVL3 8B + CoReTab	72.58	77.14	74.64
	Δ = (6.1) - (6.2)	5.26	5.86	13.32

Table 4: Ablation experiment results, reporting average performance across three benchmark categories.

simpler reasoning and code.

Impact on other MLLMs. We evaluate our framework on InternVL3-8B (Block 6 in Table 4). The model exhibits the same performance trend as Qwen2.5-VL, showing consistent improvements when trained on CoReTab.

6 Conclusion

We presented **CoReTab**, a code-driven reasoning framework for multimodal table understanding that combines natural-language multi-step reasoning with executable code. By generating scalable, interpretable, and verifiable annotations, CoReTab provides richer supervision to train MLLMs than existing datasets. We have shown that models trained on the CoReTab achieve consistent improvements across 17 MMTab benchmarks, particularly in multi-step reasoning and table structure understanding, while producing transparent, verifiable, and human-aligned responses. Overall, our CoReTab framework can be effectively applied to multimodal table understanding, enhancing both accuracy and interpretability.

7 Acknowledgement

This work was partly supported by JSPS KAKENHI Grant Number 23H00482. This work was written by the authors, with minor phrasing and grammatical polishing provided by ChatGPT. All interpretations and remaining errors are solely the authors’ responsibility.

8 Limitations

While our CoReTab framework provides promising supervision that improves both accuracy and interpretability for MTU, several limitations remain discussed.

First, some reasoning annotations inside `<reason>` tags of verified samples may still contain errors, as they are automatically synthesized by the LLM annotator and verified through direct output comparison without exhaustive human checks. Generating long responses with code-driven reasoning increases inference time. Also, in certain cases, particularly in information retrieval and fact verification tasks, <code> tags may simply print outputs from `<reason>`, reducing the effectiveness of code verification.

Second, our dataset is curated primarily from academic sources in English (e.g., (LU et al., 2023; Chen et al., 2020)) with clean table codes in HTML, $\LaTeX$, and Markdown. In industrial settings, table images may exhibit rotation, occlusion, deformation, or perspective distortions, making accurate table code extraction dependent on robust off-the-shelf table recognition models.

Lastly, CoReTab focuses on tasks with deterministic, verifiable answers (e.g., associated with Accuracy and F1 metrics). Table text generation tasks such as summarization or description remain unexplored; future work could adopt semantic evaluation methods, such as LLM-as-a-judge (Zheng et al., 2023), to extend verification beyond automatic metrics like BLEU.

9 Ethical Considerations

The proposed `CoReTab` dataset is built upon MMTab annotations and other publicly available academic datasets such as WTQ and TabFact, which are distributed under permissive licenses (e.g., MIT, CC-BY-SA-4.0). We do not introduce or collect any new personal information or offensive content beyond what may already exist in the underlying datasets. Consequently, any biases present in the annotations or model outputs reflect those in the source datasets and models.

Our framework is intended solely for research purposes, enabling the development of more robust multimodal table reasoning models, which may also be extended to related domains. For the generated code in `CoReTab`, we follow the safeguards in (Wang et al., 2024b), allowing compilation and execution only for data manipulation and basic numerical or logical operations. Potentially harmful actions (e.g., accessing modules such as `os` or `sys` to delete or modify files) are explicitly prohibited.

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 Association for Computational Linguistics: NAACL 2022*, pages 1–16, Seattle, United States. Association for Computational Linguistics.

Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, and 1 others. 2022. Flamingo: a visual language model for few-shot learning. *Advances in Neural Information Processing Systems*, 35:23716–23736.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, and 1 others. 2025. Qwen2. 5-vl technical report. *arXiv preprint arXiv:2502.13923*.

Wenhu Chen. 2023. Large language models are few(1)-shot table reasoners. In *Findings of the Association for Computational Linguistics: EACL 2023*, pages 1120–1130, Dubrovnik, Croatia. Association for Computational Linguistics.

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*.

Wenhu Chen, Hongmin Wang, Jianshu Chen, Yunkai Zhang, Hong Wang, Shiyang Li, Xiyou Zhou, and William Yang Wang. 2020. Tabfact: A large-scale dataset for table-based fact verification. *Preprint*, arXiv:1909.02164.

Zhe Chen, Weiyun Wang, Yue Cao, Yangzhou Liu, Zhangwei Gao, Erfei Cui, Jinguo Zhu, Shenglong Ye, Hao Tian, Zhaoyang Liu, and 1 others. 2024. Expanding performance boundaries of open-source multimodal models with model, data, and test-time scaling. *arXiv preprint arXiv:2412.05271*.

Kanzhi Cheng, Li YanTao, Fangzhi Xu, Jianbing Zhang, Hao Zhou, and Yang Liu. 2025. Vision-language models can self-improve reasoning via reflection. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 8876–8892.

Zhoujun Cheng, Haoyu Dong, Zhiruo Wang, Ran Jia, Jiaqi Guo, Yan Gao, Shi Han, Jian-Guang Lou, and Dongmei Zhang. 2022. HiTab: A hierarchical table dataset for question answering and natural language generation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1094–1110, Dublin, Ireland. Association for Computational Linguistics.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Pal: Program-aided language models. In *International Conference on Machine Learning*, pages 10764–10799. PMLR.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Vivek Gupta, Maitrey Mehta, Pegah Nokhiz, and Vivek Srikumar. 2020. INFOTABS: Inference on tables as semi-structured data. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2309–2324, Online. Association for Computational Linguistics.

Anwen Hu, Haiyang Xu, Jiabo Ye, Ming Yan, Liang Zhang, Bo Zhang, Ji Zhang, Qin Jin, Fei Huang, and Jingren Zhou. 2024. mplug-docowl 1.5: Unified structure learning for ocr-free document understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 3096–3120.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, and 1 others. 2022. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3.

Sujay Kumar Jauhar, Peter Turney, and Eduard Hovy. 2016. Tabmcq: A dataset of general knowledge tables and multiple-choice questions. *Preprint*, arXiv:1602.03960.

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

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pages 611–626.

Larissa R. Lautert, Marcelo M. Scheidt, and Carina F. Dorneles. 2013. Web table taxonomy and formalization. *SIGMOD Rec.*, 42(3):28–33.

Hongxin Li, Jingran Su, Yuntao Chen, Qing Li, and Zhaoxiang Zhang. 2023a. Sheetcopilot: Bringing software productivity to the next level through large language models. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. 2023b. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. *Preprint*, arXiv:2301.12597.

Peng Li, Yeye He, Dror Yashar, Weiwei Cui, Song Ge, Haidong Zhang, Danielle Rifinski Fainman, Dongmei Zhang, and Surajit Chaudhuri. 2023c. Table-gpt: Table-tuned gpt for diverse table tasks. *Preprint*, arXiv:2310.09263.

Yongqi Li, Wenjie Li, and Liqiang Nie. 2022. Mmcoqa: Conversational question answering over text, tables, and images. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4220–4231.

Zhang Li, Biao Yang, Qiang Liu, Zhiyin Ma, Shuo Zhang, Jingxu Yang, Yabo Sun, Yuliang Liu, and Xiang Bai. 2023d. Monkey: Image resolution and text label are important things for large multi-modal models. *arXiv preprint arXiv:2311.06607*.

Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. 2023a. Improved baselines with visual instruction tuning.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2023b. Visual instruction tuning. In *NeurIPS*.

Qian Liu, Fan Zhou, Zhengbao Jiang, Longxu Dou, and Min Lin. 2023c. From zero to hero: Examining the power of symbolic tasks in instruction tuning. *Preprint*, arXiv:2304.07995.

Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. 2023. Chameleon: Plug-and-play compositional reasoning with large language models. In *The 37th 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. 2023. Dynamic prompt learning via policy gradient for semi-structured mathematical reasoning. In *International Conference on Learning Representations (ICLR)*.

Shiyin Lu, Yang Li, Qing-Guo Chen, Zhao Xu, Weihua Luo, Kaifu Zhang, and Han-Jia Ye. 2024. Ovis: Structural embedding alignment for multimodal large language model. *arXiv preprint arXiv:2405.20797*.

Haohao Luo, Ying Shen, and Yang Deng. 2023. Unifying text, tables, and images for multimodal question answering. In *Findings of the Association for Computational Linguistics: EMNLP 2023*. Association for Computational Linguistics.

Suyash Mathur, Jainit Bafna, Kunal Kartik, Harshita Khandelwal, Manish Shrivastava, Vivek Gupta, Mohit Bansal, and Dan Roth. 2024. Knowledge-aware reasoning over multimodal semi-structured tables. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 14054–14073.

Fanqing Meng, Wenqi Shao, Quanfeng Lu, Peng Gao, Kaipeng Zhang, Yu Qiao, and Ping Luo. 2024. Chartassisstant: A universal chart multimodal language model via chart-to-table pre-training and multitask instruction tuning. *arXiv preprint arXiv:2401.02384*.

Swaroop Mishra, Matthew Finlayson, Pan Lu, Leonard Tang, Sean Welleck, Chitta Baral, Tanmay Rajpurohit, Oyvind Tafjord, Ashish Sabharwal, Peter Clark, and 1 others. 2022. Lila: A unified benchmark for mathematical reasoning. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 5807–5832.

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

OpenAI, :, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mo Bavarian, and 263 others. 2023. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

Ankur Parikh, Xuezhi Wang, Sebastian Gehrmann, Manaal Faruqui, Bhuwan Dhingra, Diyi Yang, and Dipanjan Das. 2020. ToTTo: A controlled table-to-text generation dataset. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1173–1186, Online. Association for Computational Linguistics.

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 (Volume 1: Long Papers)*, pages 1470–1480, Beijing, China. Association for Computational Linguistics.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16. IEEE.

Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 3505–3506.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Alexey Shigarov. 2023. Table understanding: Problem overview. *WIREs Data Mining and Knowledge Discovery*, 13(1):e1482.

Yuan Sui, Mengyu Zhou, Mingjie Zhou, Shi Han, and Dongmei Zhang. 2023. Gpt4table: Can large language models understand structured table data? a benchmark and empirical study. *Preprint*, arXiv:2305.13062.

Dídac Surís, Sachit Menon, and Carl Vondrick. 2023. Vipergpt: Visual inference via python execution for reasoning. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 11888–11898.

Prasham Yatinkumar Titiya, Jainil Trivedi, Chitta Baral, and Vivek Gupta. 2025. Mmtbench: A unified benchmark for complex multimodal table reasoning. *arXiv preprint arXiv:2505.21771*.

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, and 49 others. 2023. Llama 2: Open foundation and fine-tuned chat models. *Preprint*, arXiv:2307.09288.

Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, and 1 others. 2024a. Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution. *arXiv preprint arXiv:2409.12191*.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. 2024b. Executable code actions elicit better llm agents. In *Forty-first International Conference on Machine Learning*.

Zilong Wang, Hao Zhang, Chun-Liang Li, Julian Martin Eisenschlos, Vincent Perot, Zifeng Wang, Lesly Miculicich, Yasuhisa Fujii, Jingbo Shang, Chen-Yu Lee, and 1 others. 2024c. Chain-of-table: Evolving tables in the reasoning chain for table understanding. In *The Twelfth International Conference on Learning Representations*.

Haoran Wei, Lingyu Kong, Jinyue Chen, Liang Zhao, Zheng Ge, Jinrong Yang, Jianjian Sun, Chunrui Han, and Xiangyu Zhang. 2023. Vary: Scaling up the vision vocabulary for large vision-language models. *Preprint*, arXiv:2312.06109.

Haoran Wei, Lingyu Kong, Jinyue Chen, Liang Zhao, Zheng Ge, En Yu, Jianjian Sun, Chunrui Han, and Xiangyu Zhang. 2024. Small language model meets with reinforced vision vocabulary. *arXiv preprint arXiv:2401.12503*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Sam Wiseman, Stuart Shieber, and Alexander Rush. 2017. Challenges in data-to-document generation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2253–2263, Copenhagen, Denmark. Association for Computational Linguistics.

Xianjie Wu, Jian Yang, Linzheng Chai, Ge Zhang, Jiaheng Liu, Xeron Du, Di Liang, Daixin Shu, Xianfu Cheng, Tianzhen Sun, and 1 others. 2025. Tablebench: A comprehensive and complex benchmark for table question answering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 25497–25506.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Pan Zhang, Xiaoyi Dong, Bin Wang, Yuhang Cao, Chao Xu, Linke Ouyang, Zhiyuan Zhao, Haodong Duan, Songyang Zhang, Shuangrui Ding, Wenwei Zhang, Hang Yan, Xinyue Zhang, Wei Li, Jingwen Li, Kai Chen, Conghui He, Xingcheng Zhang, Yu Qiao, and 2 others. 2023a. Internlm-xcomposer: A vision-language large model for advanced text-image comprehension and composition. *Preprint*, arXiv:2309.15112.

Tianshu Zhang, Xiang Yue, Yifei Li, and Huan Sun. 2023b. Tablellama: Towards open large generalist models for tables. *Preprint*, arXiv:2311.09206.

Weichao Zhao, Hao Feng, Qi Liu, Jingqun Tang, Binghong Wu, Lei Liao, Shu Wei, Yongjie Ye, Hao Liu, Wengang Zhou, and 1 others. 2024. Tabpedia: Towards comprehensive visual table understanding with concept synergy. *Advances in Neural Information Processing Systems*, 37:7185–7212.

Yuze Zhao, Jintao Huang, Jinghan Hu, Xingjun Wang, Yunlin Mao, Daoze Zhang, Zeyinzi Jiang, Zhikai Wu, Baole Ai, Ang Wang, and 1 others. 2025. Swift: a scalable lightweight infrastructure for fine-tuning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 29733–29735.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, and 1 others. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623.

Mingyu Zheng, Xinwei Feng, Qingyi Si, Qiaoqiao She, Zheng Lin, Wenbin Jiang, and Weiping Wang. 2024. Multimodal table understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9102–9124.

Xinyi Zheng, Douglas Burdick, Lucian Popa, Xu Zhong, and Nancy Xin Ru Wang. 2021. Global table extractor (gte): A framework for joint table identification and cell structure recognition using visual context. In *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, pages 697–706.

Xu Zhong, Elaheh ShafieiBavani, and Antonio Jimeno Yepes. 2020. Image-based table recognition: data, model, and evaluation. *Preprint*, arXiv:1911.10683.

Bangbang Zhou, Zuan Gao, Zixiao Wang, Boqiang Zhang, Yuxin Wang, Zhineng Chen, and Hongtao Xie. 2025. Syntab-llava: Enhancing multimodal table understanding with decoupled synthesis. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 24796–24806.

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.

A Details about CoReTab

The `CoReTab` dataset and trained models will be publicly released to facilitate reproducibility and further research on multimodal table understanding. All latest resources will be made available upon publication.

A.1 Additional Generation Details

We use the vLLM engine (Kwon et al., 2023) to accelerate inference. The sampling temperature is set to 0.1 for TSU tasks to ensure code correctness, and 0.3 for other tasks to encourage response diversity. As mentioned in the main paper, we augment the input to the LLM annotator with *Tool Use* text, which describes the predefined functions available for TSU tasks. See Figure 5 for details.

For TSU tasks, we sort the responses by total token length and select the first 5,000 samples to avoid lengthy annotations. For TQA and TFV tasks, we further filter out samples with excessively long text inside `reason` or `code` tags, removing any response where the content within either tag exceeds 1,024 tokens.

A.2 Additional Dataset Statistics

Figure 6 shows the distribution of reasoning lengths and Python code lengths across the 11 annotated tasks in the CoReTab dataset. We observe that for most TQA and TFV tasks, the average Python code is shorter than the corresponding multi-step reasoning explanation, whereas TSU tasks typically produce longer Python code that includes table textual representation for precise detection and localization.

Figure 7 shows the rejection rates of the responses generated by the LLM annotator. To accept or reject a response as a `CoReTab` annotation, we extract its generated Python code and execute it for verification against the ground-truth answers. Here, the rejection rate refers to the proportion of responses whose Python code produces values inconsistent with the corresponding ground-truth answers.

A.3 Dataset Examples

We visualize representative samples from MMTab and `CoReTab` (Sec. 3.3), with examples shown in Fig. 8–18. Each figure displays the input table image, the user query, and the corresponding MMTab and `CoReTab` annotations.

Descriptions of pre-defined functions

Use the following tools to assist in reaching the final answer:

- `get_table_2d`: Get the 2D grid of the HTML table. Args: `html_table: str`. Return: `list`.
- `get_cell_location`: Get the location of a cell in the table. Args: `table_2d: list`, `row_id: int`, `col_id: int`. Return: `str`.
- `get_table_size`: Get the size of the table. Args: `table_2d: list`. Return: `str`.
- `get_cell_value`: Get the value of a cell in the table. Args: `table_2d: list`, `row_id: int`, `col_id: int`. Return: `str`.
- `get_row_values`: Get the values of a row in the table. Args: `table_2d: list`, `row_id: int`. Return: `str`.
- `get_col_values`: Get the values of a column in the table. Args: `table_2d: list`, `col_id: int`. Return: `str`.
- `get_merged_cell_locations`: Get the locations of merged cells in the table. Args: `html_table: str`. Return: `str`.

Figure 5: Descriptions of pre-defined functions used for TSU tasks to the (M)LLM.

Table Question Answering (TQA). Examples from TABMWP, WTQ, HiTab, and TAT-QA are shown in Fig. 8–11. HiTab and AIT-QA contain large and complex tables, yet MMTab annotations mainly provide short answers. For numerical reasoning tasks such as TABMWP and TAT-QA, the solutions typically involve multi-step calculations followed by a final answer. However, MMTab includes only brief answers (short for TAT-QA and a numerical derivation for TABMWP). In contrast, `CoReTab` offers detailed step-by-step reasoning with executable code for all four tasks.

Table Fact Verification (TFV). Examples from TabFact, InfoTabs, and PubHealthTab are shown in Fig. 11–13. These tasks require indexing, querying, and reasoning over the table image to verify a user's statement. If the inferred conclusion aligns with the statement, the output is "correct"; if it contradicts, "incorrect"; and if uncertain, "neutral". MMTab annotations typically provide only the final answer intentionally, whereas `CoReTab` includes explicit reasoning chains and executable code verifying the conclusion.

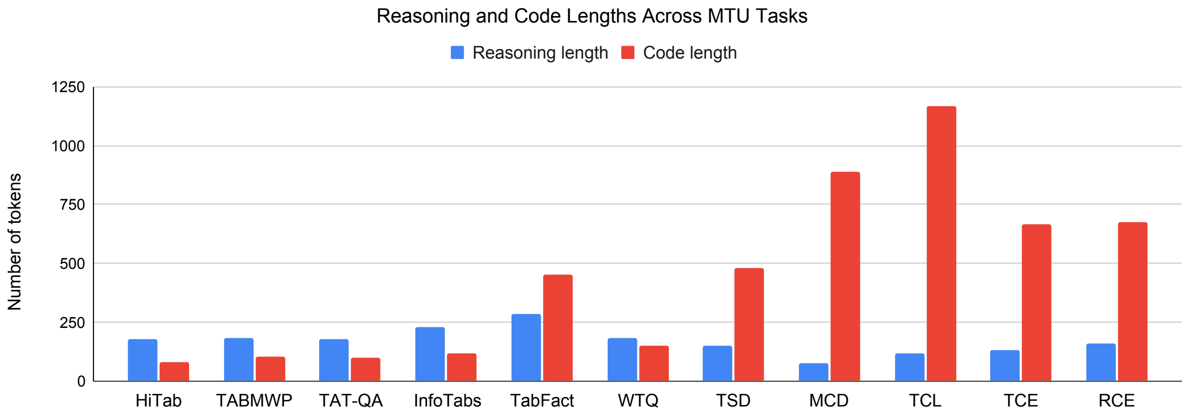


Figure 6: Length of tokens inside **`<reason>`** and **`<code>`** tags over 11 annotated tasks.

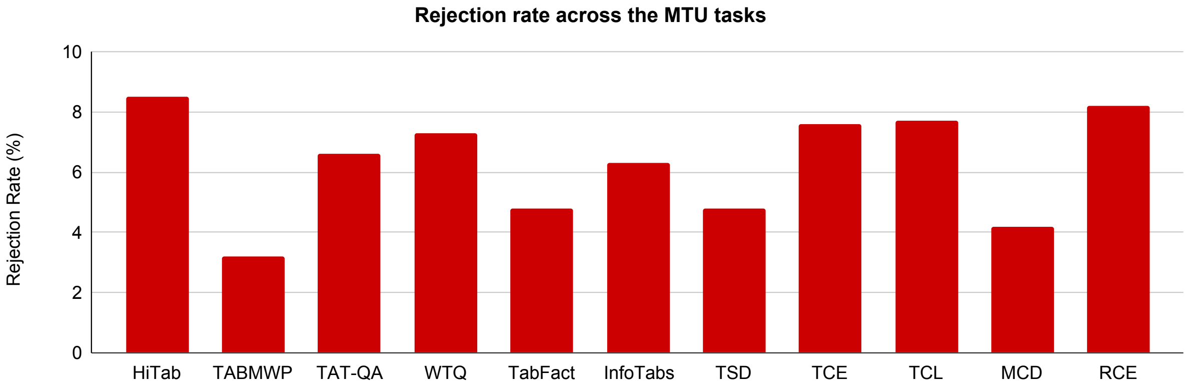


Figure 7: Rejection rates of generated annotations by the LLM over multimodal table understanding tasks.

Table Structure Understanding (TSU). Examples from five tasks are shown in Fig. 14–18: Table Size Detection (TSD) predicts the number of rows and columns; Table Cell Extraction (TCE) retrieves content by specified indices; Table Cell Locating (TCL) identifies cell positions matching a query; Merge Cell Detection (MCD) detects merged regions; and Row/Column Extraction (RCE) outputs complete rows or columns. MMTab annotations are of direct answers, while `CoReTab` provides annotations with reasoning steps and executable code for verification.

B Implementation Details

The objective in stage 1 and 2. In the first two stages, Qwen2.5-VL is trained using the standard cross-entropy loss over both visual and textual tokens. The input to the MLLM is a pair of $\langle$table image I, text query $Q\rangle$, where text query is represented by a sequence of tokens $\mathbf{x} = (x_1, \dots, x_T)$. The model first encodes the image I into a sequence of visual tokens, which are then concatenated with the text tokens. The objective is to maximize the likelihood of the next token conditioned on the preceding multimodal context:

$$\mathcal{L}_{\text{CE}} = -\sum_{t=1}^{T} \log P_\theta(x_t \mid I, x_{<t}), \qquad (1)$$

where P_θ denotes the model's predicted token distribution parameterized by θ. This objective encourages the MLLM to learn joint reasoning across visual and textual modalities while improving its ability to generate coherent and grounded responses.

The objective in stage 3. In stage 3, the model is further refined using RL optimization with GRPO. During training, we sample G rollouts—possible answers S^1 to S^G for each $\langle I, Q\rangle$ pair. The model is trained to maximize the total reward $\mathcal{R}^i$ for each output S^i, defined as

$$\mathcal{R}^i = \mathcal{R}^i_{\text{acc}} + \mathcal{R}^i_{\text{format}}, \qquad (2)$$

where $\mathcal{R}^i_{\text{acc}}$ and $\mathcal{R}^i_{\text{format}}$ are binary rewards indicating correct answer and correct formatting, respectively, following Guo et al. (2025). We then compute the relative advantage $\hat{A}^i$ for each output S^i by normalizing its reward with respect to the other outputs in the rollout, as:

$$\hat{A}^i = \frac{\mathcal{R}^i - \text{mean}(\{\mathcal{R}^1, \ldots, \mathcal{R}^G\})}{\text{std}(\{\mathcal{R}^1, \ldots, \mathcal{R}^G\})}, \quad (3)$$

The overall loss function integrates a clipped surrogate objective with a KL-divergence penalty, formulated as:

$$\mathcal{L}_{\text{GRPO}} = \mathcal{L}_{\text{clip}} - \beta \mathbb{D}_{\text{KL}}[\pi_\theta \| \pi_{\text{ref}}], \quad (4)$$

where $\mathcal{L}_{\text{clip}}$ adopts the proximal policy optimization mechanism. This is to constrain policy updates to avoid excessively large parameter shifts. The KL divergence term $\mathbb{D}_{\text{KL}}$ further regularizes the policy π_θ to stay close to the reference model π_{ref}. Refer to Shao et al. (2024); Guo et al. (2025) for more details.

Implementation Settings. All experiments are conducted on 8 NVIDIA A200 80GB Tensor Core GPUs using DeepSpeed (Rajbhandari et al., 2020; Rasley et al., 2020) with ZeRO optimization (Rajbhandari et al., 2020) and the Swift framework (Zhao et al., 2025) for training. We employ the AdamW optimizer across all three stages, training each for two epochs with a per-GPU batch size of 1. Gradient accumulation is used to achieve a global batch size of 256 for parameter updates. During each training stage, we applied LoRA to the MLLM model with a rank of 8 and a scaling factor (`lora_alpha`) of 32, targeting all linear layers and specific projections (`o_proj`, `up_proj`, `v_proj`, `q_proj`, `gate_proj`, `k_proj`, `down_proj`) in the language model, while keeping the vision encoder frozen. In stages 1 and 2, the learning rate is set to 2×10^{-5}, with a weight decay of 0.1, β_1=0.9, and β_2=0.95.

During stage 3 with GRPO, we set the number of groups to G=4 (i.e., four rollouts per $\langle I, Q \rangle$ pair) and use a sampling temperature of 0.5 to encourage diverse reasoning paths. The maximum sequence length is set to L=4096 to allow the model to generate complete responses containing both natural language reasoning and executable Python code. The KL-divergence coefficient β in Eq. 4 is fixed at 0.05, and the learning rate for the policy model is 2×10^{-6}.

During inference, we employ the vLLM engine (Kwon et al., 2023) for efficient decoding. To ensure deterministic and reproducible outputs, we set the sampling temperature to `0`, corresponding to greedy decoding. This configuration eliminates randomness in token selection, allowing consistent evaluation across all samples.

B.1 System Prompt to `CoReTab`-Qwen2.5-VL

The system prompt is prepended to the `CoReTab`-Qwen2.5-VL model during both training and inference, differing slightly from the one used for the LLM annotator. For TSU tasks, the descriptions of pre-defined functions are optionally appended to the system prompt before being passed to Qwen2.5-VL. The input to the MLLM is structured as follows:

Structured input to `CoReTab`-Qwen2.5-VL

You are an expert in table analysis. Your task is to process a table image to answer a question with a code-driven reasoning explanation.

Your output must strictly follow this structure:

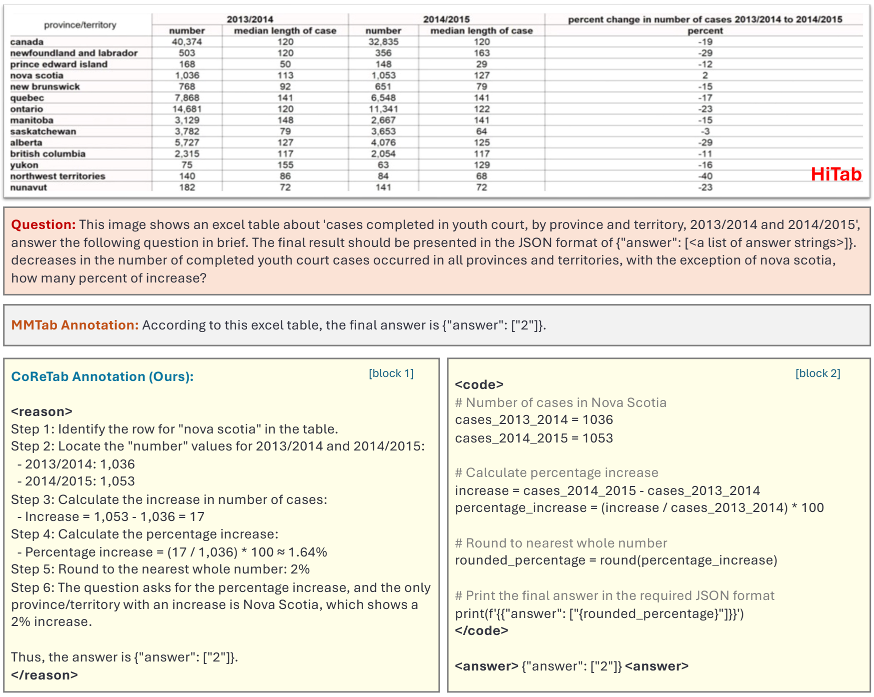
1. Reasoning — enclosed in <reason> tags. Provide clear, concise, and verifiable reasoning steps that logically lead to the final answer.
2. Python Code — enclosed in <code> tags. Write clean, executable Python code that computes the final answer based on the reasoning. The code must print the answer in the correct format.
3. Final Answer — enclosed in <answer> tags. This must contain the final answer or the printed output of the code snippet.

Tool Use: {{ `Tool Use Descriptions` }} # optional

Input: {{ `table image, question` }}

C Additional Qualitative Results

We provide additional results from `CoReTab`-Qwen2.5-VL on the test samples in Fig. 19–22. The results show that the model generates correct answers along with detailed step-by-step reasoning and executable Python code that can be verified.



province/territory	2013/2014		2014/2015		percent change in number of cases 2013/2014 to 2014/2015
	number	median length of case	number	median length of case	percent
canada	40,374	120	32,835	120	-19
newfoundland and labrador	503	120	356	163	-29
prince edward island	168	50	148	29	-12
nova scotia	1,036	113	1,053	127	2
new brunswick	768	92	651	79	-15
quebec	7,868	141	6,548	141	-17
ontario	14,681	120	11,341	122	-23
manitoba	3,129	148	2,667	141	-15
saskatchewan	3,782	79	3,653	64	-3
alberta	5,727	127	4,076	125	-29
british columbia	2,315	117	2,054	117	-11
yukon	75	155	63	129	-16
northwest territories	140	86	84	68	-40
nunavut	182	72	141	72	-23

HiTab

Question: This image shows an excel table about 'cases completed in youth court, by province and territory, 2013/2014 and 2014/2015', answer the following question in brief. The final result should be presented in the JSON format of {"answer": [<a list of answer strings>]}. decreases in the number of completed youth court cases occurred in all provinces and territories, with the exception of nova scotia, how many percent of increase?

MMTab Annotation: According to this excel table, the final answer is {"answer": ["2"]}.

CoReTab Annotation (Ours): [block 1]

<reason>
Step 1: Identify the row for "nova scotia" in the table.
Step 2: Locate the "number" values for 2013/2014 and 2014/2015:
- 2013/2014: 1,036
- 2014/2015: 1,053

Step 3: Calculate the increase in number of cases:
- Increase = 1,053 - 1,036 = 17

Step 4: Calculate the percentage increase:
- Percentage increase = (17 / 1,036) * 100 ≈ 1.64%

Step 5: Round to the nearest whole number: 2%
Step 6: The question asks for the percentage increase, and the only province/territory with an increase is Nova Scotia, which shows a 2% increase.

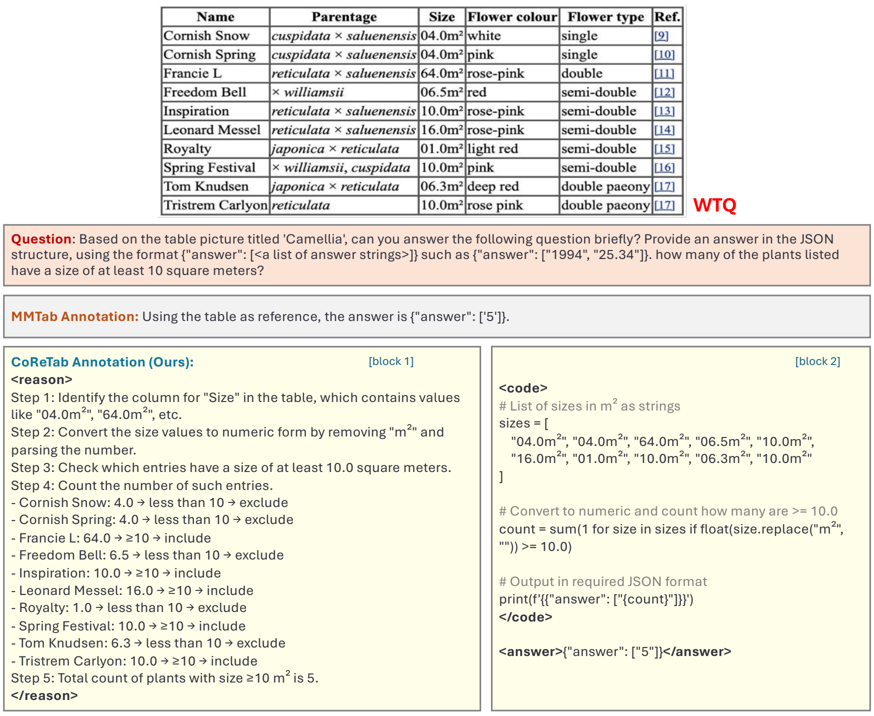
Thus, the answer is {"answer": ["2"]}.
</reason>

[block 2]

```
<code>
# Number of cases in Nova Scotia
cases_2013_2014 = 1036
cases_2014_2015 = 1053

# Calculate percentage increase
increase = cases_2014_2015 - cases_2013_2014
percentage_increase = (increase / cases_2013_2014) * 100

# Round to nearest whole number
rounded_percentage = round(percentage_increase)

# Print the final answer in the required JSON format
print(f'{{"answer": ["{rounded_percentage}"]}}')
</code>
```

<answer> {"answer": ["2"]} **<answer>**

Figure 8: MMTab and CoReTab annotations for a sample from HiTab.



Name	Parentage	Size	Flower colour	Flower type	Ref.
Cornish Snow	*cuspidata* × *saluenensis*	04.0m^2	white	single	[9]
Cornish Spring	*cuspidata* × *saluenensis*	04.0m^2	pink	single	[10]
Francie L	*reticulata* × *saluenensis*	64.0m^2	rose-pink	double	[11]
Freedom Bell	× *williamsii*	06.5m^2	red	semi-double	[12]
Inspiration	*reticulata* × *saluenensis*	10.0m^2	rose-pink	semi-double	[13]
Leonard Messel	*reticulata* × *saluenensis*	16.0m^2	rose-pink	semi-double	[14]
Royalty	*japonica* × *reticulata*	01.0m^2	light red	semi-double	[15]
Spring Festival	× *williamsii, cuspidata*	10.0m^2	pink	semi-double	[16]
Tom Knudsen	*japonica* × *reticulata*	06.3m^2	deep red	double paeony	[17]
Tristrem Carlyon	*reticulata*	10.0m^2	rose pink	double paeony	[17]

WTQ

Question: Based on the table picture titled 'Camellia', can you answer the following question briefly? Provide an answer in the JSON structure, using the format {"answer": [<a list of answer strings>]} such as {"answer": ["1994", "25.34"]}. how many of the plants listed have a size of at least 10 square meters?

MMTab Annotation: Using the table as reference, the answer is {"answer": ['5']}.

CoReTab Annotation (Ours): [block 1]
<reason>
Step 1: Identify the column for "Size" in the table, which contains values like "04.0m^2", "64.0m^2", etc.
Step 2: Convert the size values to numeric form by removing "m^2" and parsing the number.
Step 3: Check which entries have a size of at least 10.0 square meters.
Step 4: Count the number of such entries.
- Cornish Snow: 4.0 → less than 10 → exclude
- Cornish Spring: 4.0 → less than 10 → exclude
- Francie L: 64.0 → ≥10 → include
- Freedom Bell: 6.5 → less than 10 → exclude
- Inspiration: 10.0 → ≥10 → include
- Leonard Messel: 16.0 → ≥10 → include
- Royalty: 1.0 → less than 10 → exclude
- Spring Festival: 10.0 → ≥10 → include
- Tom Knudsen: 6.3 → less than 10 → exclude
- Tristrem Carlyon: 10.0 → ≥10 → include

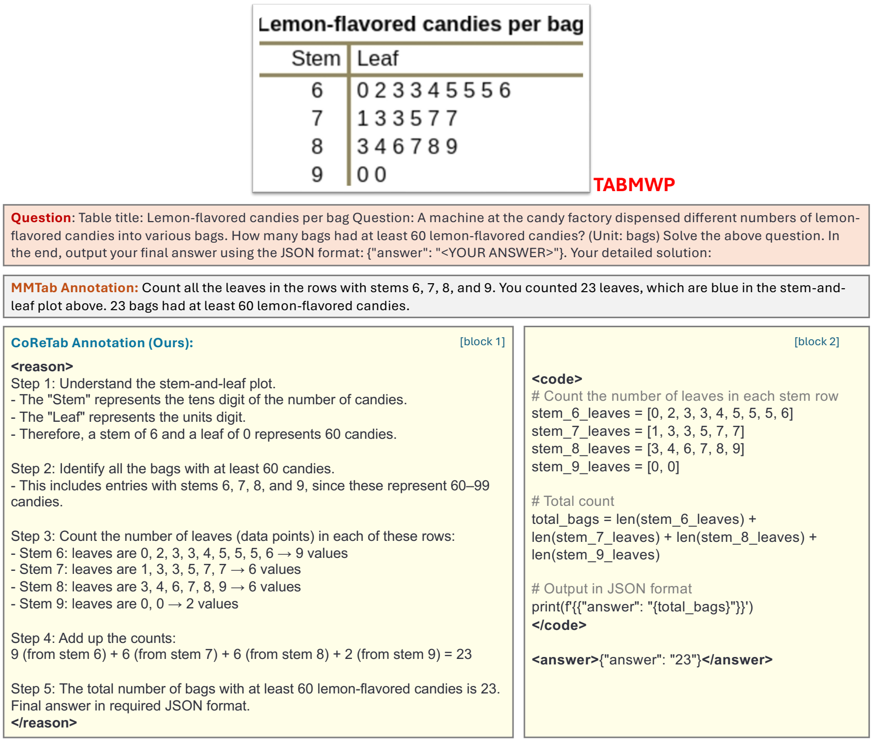
Step 5: Total count of plants with size ≥10 m^2 is 5.
</reason>

[block 2]

```
<code>
# List of sizes in m² as strings
sizes = [
    "04.0m²", "04.0m²", "64.0m²", "06.5m²", "10.0m²",
    "16.0m²", "01.0m²", "10.0m²", "06.3m²", "10.0m²"
]

# Convert to numeric and count how many are >= 10.0
count = sum(1 for size in sizes if float(size.replace("m²",
"")) >= 10.0)

# Output in required JSON format
print(f'{{"answer": ["{count}"]}}')
</code>
```

<answer>{"answer": ["5"]}**</answer>**

Figure 9: MMTab and CoReTab annotations for a sample from WTQ.



Lemon-flavored candies per bag

Stem	Leaf
6	0 2 3 3 4 5 5 5 6
7	1 3 3 5 7 7
8	3 4 6 7 8 9
9	0 0

TABMWP

Question: Table title: Lemon-flavored candies per bag Question: A machine at the candy factory dispensed different numbers of lemon-flavored candies into various bags. How many bags had at least 60 lemon-flavored candies? (Unit: bags) Solve the above question. In the end, output your final answer using the JSON format: {"answer": "<YOUR ANSWER>"}. Your detailed solution:

MMTab Annotation: Count all the leaves in the rows with stems 6, 7, 8, and 9. You counted 23 leaves, which are blue in the stem-and-leaf plot above. 23 bags had at least 60 lemon-flavored candies.

CoReTab Annotation (Ours): [block 1]

<reason>
Step 1: Understand the stem-and-leaf plot.
- The "Stem" represents the tens digit of the number of candies.
- The "Leaf" represents the units digit.
- Therefore, a stem of 6 and a leaf of 0 represents 60 candies.

Step 2: Identify all the bags with at least 60 candies.
- This includes entries with stems 6, 7, 8, and 9, since these represent 60–99 candies.

Step 3: Count the number of leaves (data points) in each of these rows:
- Stem 6: leaves are 0, 2, 3, 3, 4, 5, 5, 5, 6 → 9 values
- Stem 7: leaves are 1, 3, 3, 5, 7, 7 → 6 values
- Stem 8: leaves are 3, 4, 6, 7, 8, 9 → 6 values
- Stem 9: leaves are 0, 0 → 2 values

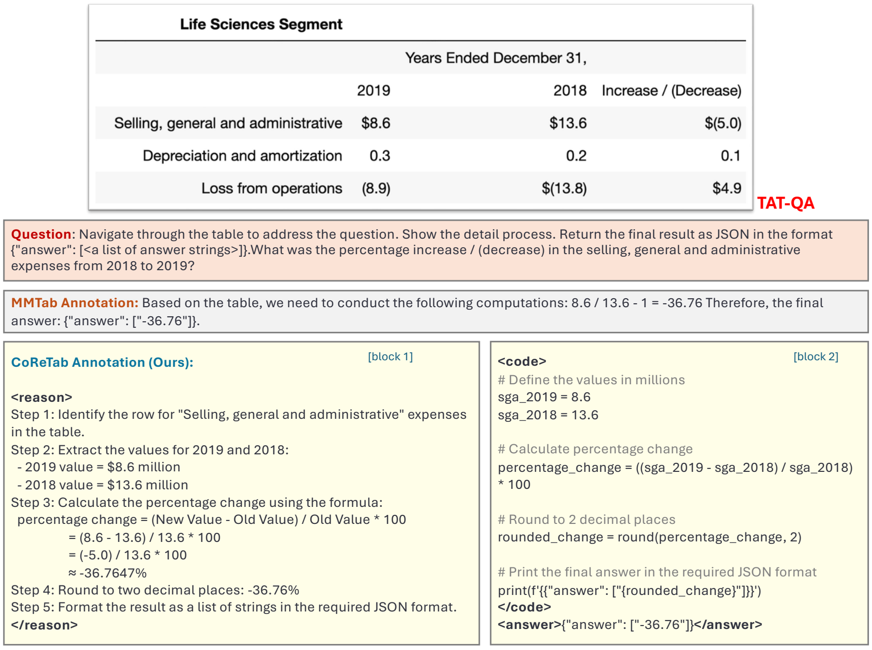
Step 4: Add up the counts:
9 (from stem 6) + 6 (from stem 7) + 6 (from stem 8) + 2 (from stem 9) = 23

Step 5: The total number of bags with at least 60 lemon-flavored candies is 23.
Final answer in required JSON format.
</reason>

[block 2]

```
<code>
# Count the number of leaves in each stem row
stem_6_leaves = [0, 2, 3, 3, 4, 5, 5, 5, 6]
stem_7_leaves = [1, 3, 3, 5, 7, 7]
stem_8_leaves = [3, 4, 6, 7, 8, 9]
stem_9_leaves = [0, 0]

# Total count
total_bags = len(stem_6_leaves) +
len(stem_7_leaves) + len(stem_8_leaves) +
len(stem_9_leaves)

# Output in JSON format
print(f'{{"answer": "{total_bags}"}}')
</code>

<answer>{"answer": "23"}</answer>
```

Figure 10: MMTab and `CoReTab` annotations for a sample from TABMWP.



Life Sciences Segment			
	Years Ended December 31,		
	2019	2018	Increase / (Decrease)
Selling, general and administrative	$8.6	$13.6	$(5.0)
Depreciation and amortization	0.3	0.2	0.1
Loss from operations	(8.9)	$(13.8)	$4.9

TAT-QA

Question: Navigate through the table to address the question. Show the detail process. Return the final result as JSON in the format {"answer": [<a list of answer strings>]}.What was the percentage increase / (decrease) in the selling, general and administrative expenses from 2018 to 2019?

MMTab Annotation: Based on the table, we need to conduct the following computations: 8.6 / 13.6 - 1 = -36.76 Therefore, the final answer: {"answer": ["-36.76"]}.

CoReTab Annotation (Ours): [block 1]

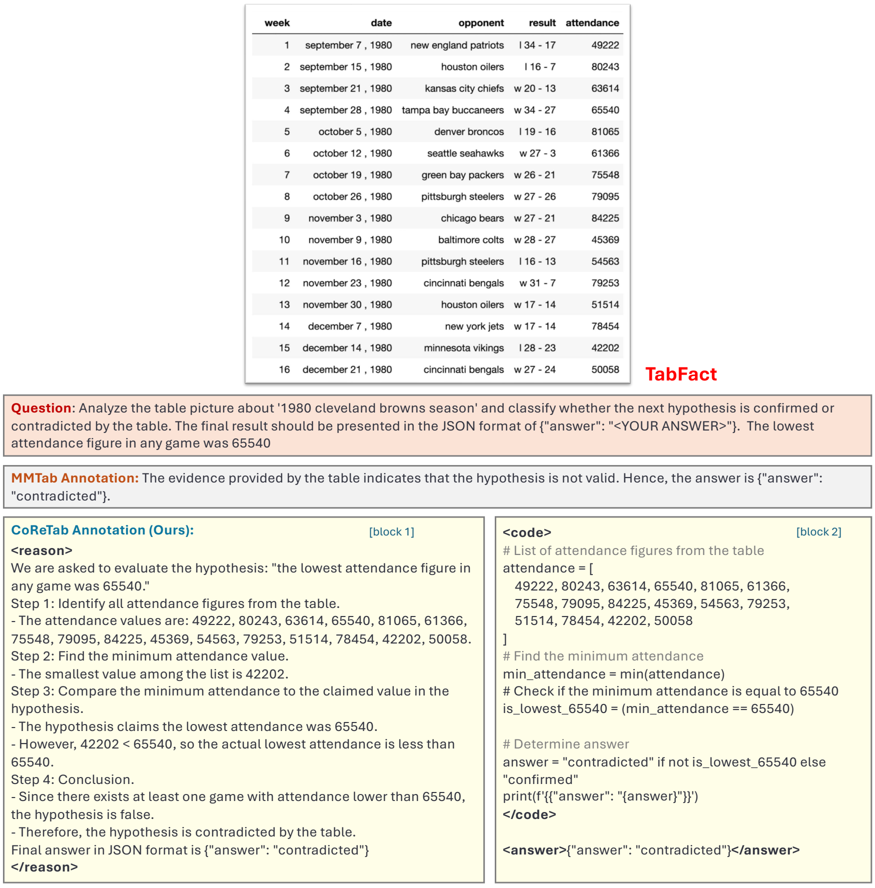
<reason>
Step 1: Identify the row for "Selling, general and administrative" expenses in the table.
Step 2: Extract the values for 2019 and 2018:
- 2019 value = $8.6 million
- 2018 value = $13.6 million

Step 3: Calculate the percentage change using the formula:
percentage change = (New Value - Old Value) / Old Value * 100
= (8.6 - 13.6) / 13.6 * 100
= (-5.0) / 13.6 * 100
≈ -36.7647%
Step 4: Round to two decimal places: -36.76%
Step 5: Format the result as a list of strings in the required JSON format.
</reason>

[block 2]

```
<code>
# Define the values in millions
sga_2019 = 8.6
sga_2018 = 13.6

# Calculate percentage change
percentage_change = ((sga_2019 - sga_2018) / sga_2018)
* 100

# Round to 2 decimal places
rounded_change = round(percentage_change, 2)

# Print the final answer in the required JSON format
print(f'{{"answer": ["{rounded_change}"]}}')
</code>
<answer>{"answer": ["-36.76"]}</answer>
```

Figure 11: MMTab and `CoReTab` annotations for a sample from TAT-QA.



week	date	opponent	result	attendance
1	september 7 , 1980	new england patriots	l 34 - 17	49222
2	september 15 , 1980	houston oilers	l 16 - 7	80243
3	september 21 , 1980	kansas city chiefs	w 20 - 13	63614
4	september 28 , 1980	tampa bay buccaneers	w 34 - 27	65540
5	october 5 , 1980	denver broncos	l 19 - 16	81065
6	october 12 , 1980	seattle seahawks	w 27 - 3	61366
7	october 19 , 1980	green bay packers	w 26 - 21	75548
8	october 26 , 1980	pittsburgh steelers	w 27 - 26	79095
9	november 3 , 1980	chicago bears	w 27 - 21	84225
10	november 9 , 1980	baltimore colts	w 28 - 27	45369
11	november 16 , 1980	pittsburgh steelers	l 16 - 13	54563
12	november 23 , 1980	cincinnati bengals	w 31 - 7	79253
13	november 30 , 1980	houston oilers	w 17 - 14	51514
14	december 7 , 1980	new york jets	w 17 - 14	78454
15	december 14 , 1980	minnesota vikings	l 28 - 23	42202
16	december 21 , 1980	cincinnati bengals	w 27 - 24	50058

TabFact

Question: Analyze the table picture about '1980 cleveland browns season' and classify whether the next hypothesis is confirmed or contradicted by the table. The final result should be presented in the JSON format of {"answer": "<YOUR ANSWER>"}. The lowest attendance figure in any game was 65540

MMTab Annotation: The evidence provided by the table indicates that the hypothesis is not valid. Hence, the answer is {"answer": "contradicted"}.

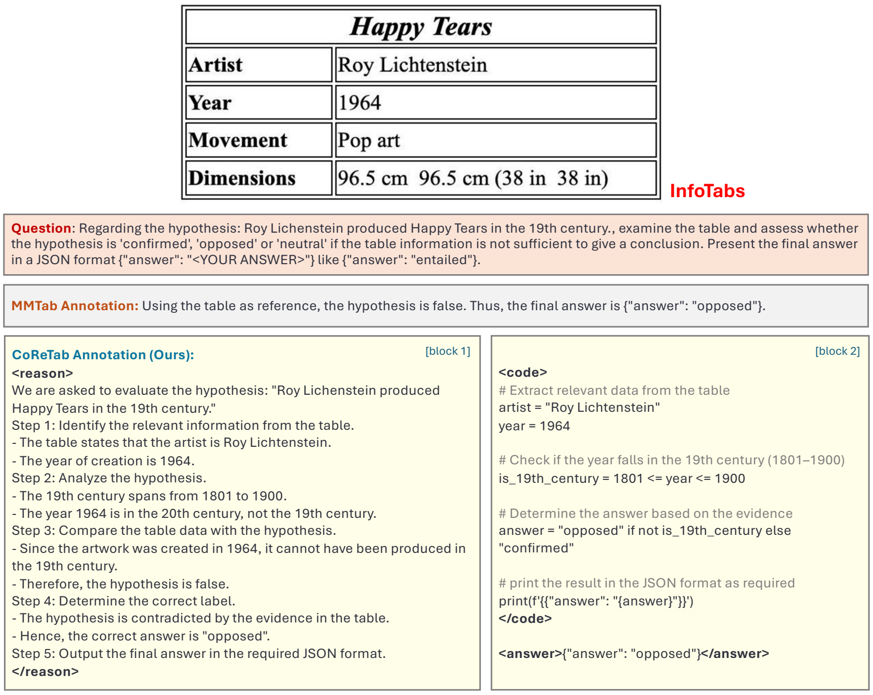
CoReTab Annotation (Ours): [block 1]

<reason>
We are asked to evaluate the hypothesis: "the lowest attendance figure in any game was 65540."
Step 1: Identify all attendance figures from the table.
- The attendance values are: 49222, 80243, 63614, 65540, 81065, 61366, 75548, 79095, 84225, 45369, 54563, 79253, 51514, 78454, 42202, 50058.
Step 2: Find the minimum attendance value.
- The smallest value among the list is 42202.
Step 3: Compare the minimum attendance to the claimed value in the hypothesis.
- The hypothesis claims the lowest attendance was 65540.
- However, 42202 < 65540, so the actual lowest attendance is less than 65540.
Step 4: Conclusion.
- Since there exists at least one game with attendance lower than 65540, the hypothesis is false.
- Therefore, the hypothesis is contradicted by the table.
Final answer in JSON format is {"answer": "contradicted"}
</reason>

[block 2]

```
<code>
# List of attendance figures from the table
attendance = [
   49222, 80243, 63614, 65540, 81065, 61366,
   75548, 79095, 84225, 45369, 54563, 79253,
   51514, 78454, 42202, 50058
]
# Find the minimum attendance
min_attendance = min(attendance)
# Check if the minimum attendance is equal to 65540
is_lowest_65540 = (min_attendance == 65540)

# Determine answer
answer = "contradicted" if not is_lowest_65540 else
"confirmed"
print(f'{{"answer": "{answer}"}}')
</code>

<answer>{"answer": "contradicted"}</answer>
```

Figure 12: MMTab and CoReTab annotations for a sample from TabFact.



Happy Tears	
Artist	Roy Lichtenstein
Year	1964
Movement	Pop art
Dimensions	96.5 cm 96.5 cm (38 in 38 in)

InfoTabs

Question: Regarding the hypothesis: Roy Lichenstein produced Happy Tears in the 19th century., examine the table and assess whether the hypothesis is 'confirmed', 'opposed' or 'neutral' if the table information is not sufficient to give a conclusion. Present the final answer in a JSON format {"answer": "<YOUR ANSWER>"} like {"answer": "entailed"}.

MMTab Annotation: Using the table as reference, the hypothesis is false. Thus, the final answer is {"answer": "opposed"}.

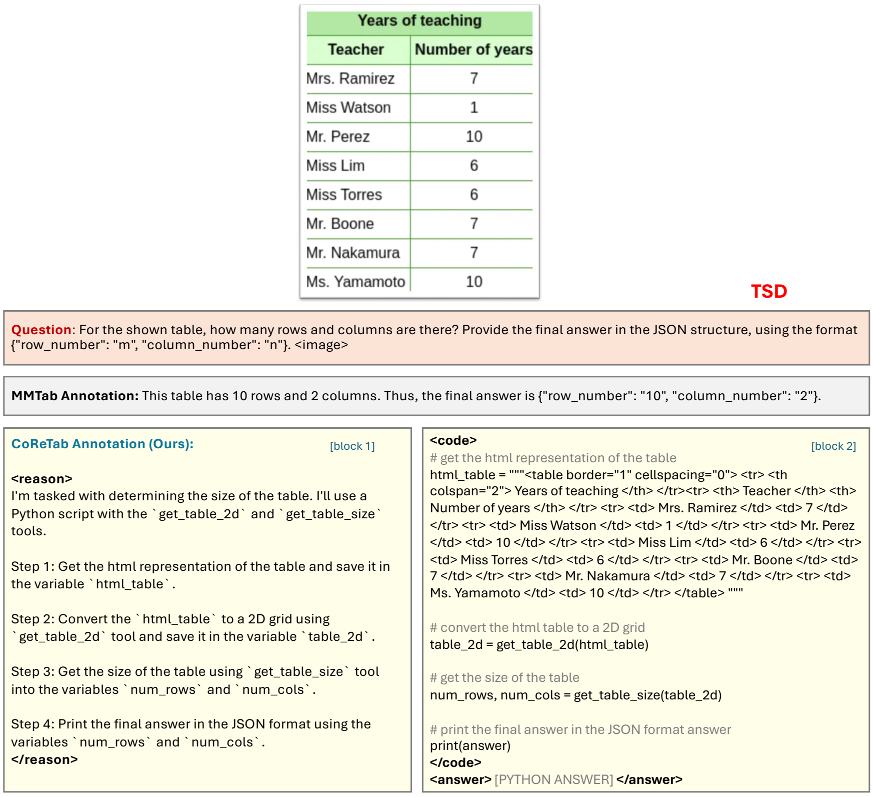
CoReTab Annotation (Ours): [block 1]

<reason>
We are asked to evaluate the hypothesis: "Roy Lichenstein produced Happy Tears in the 19th century."
Step 1: Identify the relevant information from the table.
- The table states that the artist is Roy Lichenstein.
- The year of creation is 1964.
Step 2: Analyze the hypothesis.
- The 19th century spans from 1801 to 1900.
- The year 1964 is in the 20th century, not the 19th century.
Step 3: Compare the table data with the hypothesis.
- Since the artwork was created in 1964, it cannot have been produced in the 19th century.
- Therefore, the hypothesis is false.
Step 4: Determine the correct label.
- The hypothesis is contradicted by the evidence in the table.
- Hence, the correct answer is "opposed".
Step 5: Output the final answer in the required JSON format.
</reason>

[block 2]

```
<code>
# Extract relevant data from the table
artist = "Roy Lichtenstein"
year = 1964

# Check if the year falls in the 19th century (1801–1900)
is_19th_century = 1801 <= year <= 1900

# Determine the answer based on the evidence
answer = "opposed" if not is_19th_century else
"confirmed"

# print the result in the JSON format as required
print(f'{{"answer": "{answer}"}}')
</code>

<answer>{"answer": "opposed"}</answer>
```

Figure 13: MMTab and CoReTab annotations for a sample from InfoTabs.



Years of teaching	
Teacher	**Number of years**
Mrs. Ramirez	7
Miss Watson	1
Mr. Perez	10
Miss Lim	6
Miss Torres	6
Mr. Boone	7
Mr. Nakamura	7
Ms. Yamamoto	10

TSD

Question: For the shown table, how many rows and columns are there? Provide the final answer in the JSON structure, using the format {"row_number": "m", "column_number": "n"}. <image>

MMTab Annotation: This table has 10 rows and 2 columns. Thus, the final answer is {"row_number": "10", "column_number": "2"}.

CoReTab Annotation (Ours): [block 1]

<reason>
I'm tasked with determining the size of the table. I'll use a Python script with the `get_table_2d` and `get_table_size` tools.

Step 1: Get the html representation of the table and save it in the variable `html_table`.

Step 2: Convert the `html_table` to a 2D grid using `get_table_2d` tool and save it in the variable `table_2d`.

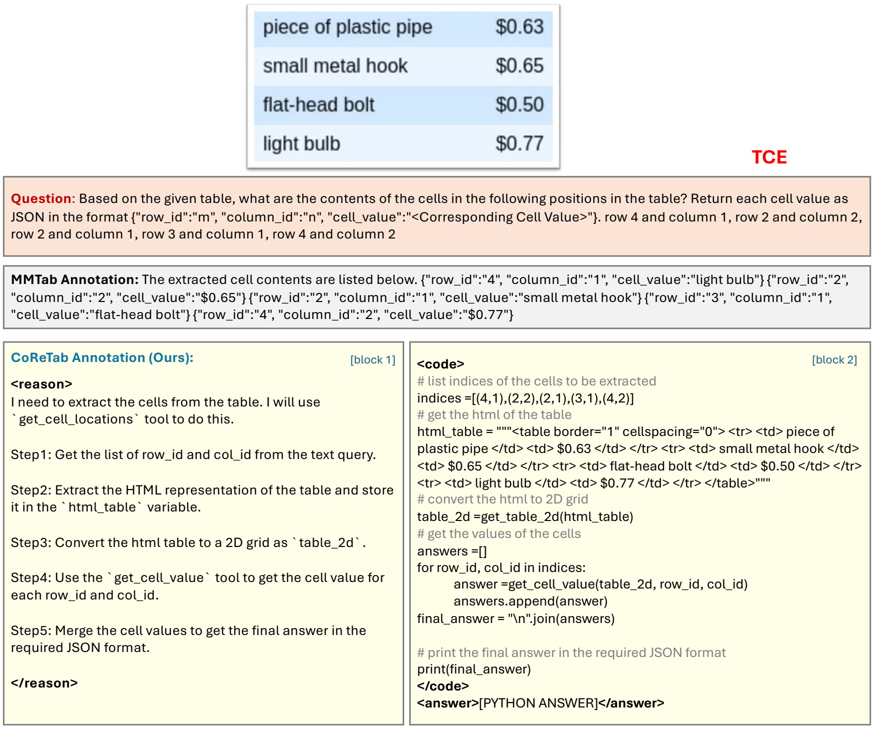
Step 3: Get the size of the table using `get_table_size` tool into the variables `num_rows` and `num_cols`.

Step 4: Print the final answer in the JSON format using the variables `num_rows` and `num_cols`.
</reason>

[block 2]

```
<code>
# get the html representation of the table
html_table = """<table border="1" cellspacing="0"> <tr> <th colspan="2"> Years of teaching </th> </tr><tr> <th> Teacher </th> <th> Number of years </th> </tr> <tr> <td> Mrs. Ramirez </td> <td> 7 </td> </tr> <tr> <td> Miss Watson </td> <td> 1 </td> </tr> <tr> <td> Mr. Perez </td> <td> 10 </td> </tr> <tr> <td> Miss Lim </td> <td> 6 </td> </tr> <tr> <td> Miss Torres </td> <td> 6 </td> </tr> <tr> <td> Mr. Boone </td> <td> 7 </td> </tr> <tr> <td> Mr. Nakamura </td> <td> 7 </td> </tr> <tr> <td> Ms. Yamamoto </td> <td> 10 </td> </tr> </table> """

# convert the html table to a 2D grid
table_2d = get_table_2d(html_table)

# get the size of the table
num_rows, num_cols = get_table_size(table_2d)

# print the final answer in the JSON format answer
print(answer)
</code>
<answer> [PYTHON ANSWER] </answer>
```

Figure 14: MMTab and CoReTab annotations for a sample from TSD.



piece of plastic pipe	$0.63
small metal hook	$0.65
flat-head bolt	$0.50
light bulb	$0.77

TCE

Question: Based on the given table, what are the contents of the cells in the following positions in the table? Return each cell value as JSON in the format {"row_id":"m", "column_id":"n", "cell_value":"<Corresponding Cell Value>"}. row 4 and column 1, row 2 and column 2, row 2 and column 1, row 3 and column 1, row 4 and column 2

MMTab Annotation: The extracted cell contents are listed below. {"row_id":"4", "column_id":"1", "cell_value":"light bulb"}{"row_id":"2", "column_id":"2", "cell_value":"$0.65"}{"row_id":"2", "column_id":"1", "cell_value":"small metal hook"}{"row_id":"3", "column_id":"1", "cell_value":"flat-head bolt"}{"row_id":"4", "column_id":"2", "cell_value":"$0.77"}

CoReTab Annotation (Ours): [block 1]

<reason>
I need to extract the cells from the table. I will use `get_cell_locations` tool to do this.

Step1: Get the list of row_id and col_id from the text query.

Step2: Extract the HTML representation of the table and store it in the `html_table` variable.

Step3: Convert the html table to a 2D grid as `table_2d`.

Step4: Use the `get_cell_value` tool to get the cell value for each row_id and col_id.

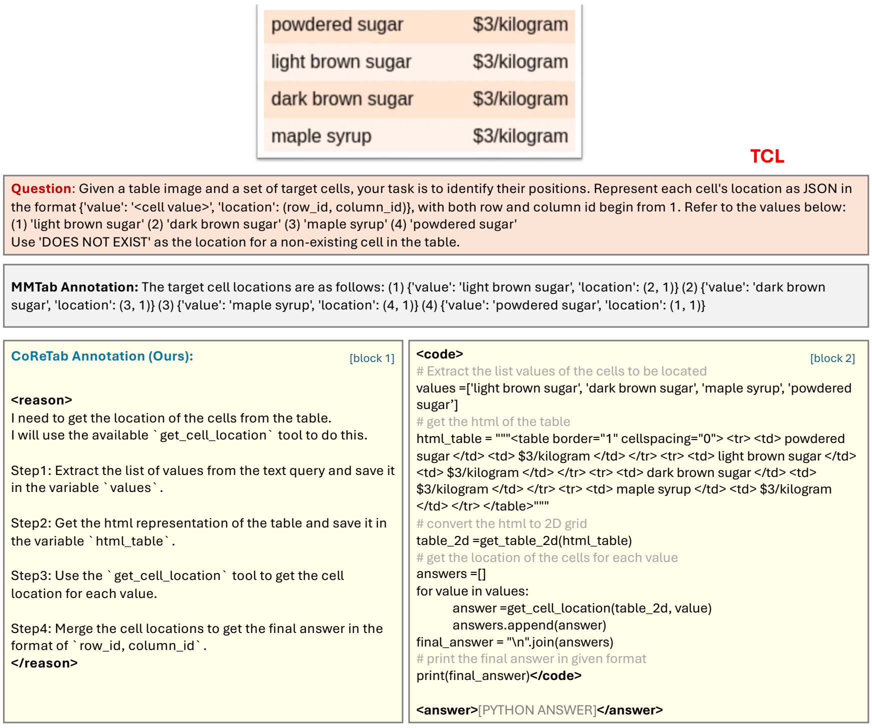
Step5: Merge the cell values to get the final answer in the required JSON format.

</reason>

[block 2]

```
<code>
# list indices of the cells to be extracted
indices =[(4,1),(2,2),(2,1),(3,1),(4,2)]
# get the html of the table
html_table = """<table border="1" cellspacing="0"> <tr> <td> piece of plastic pipe </td> <td> $0.63 </td> </tr> <tr> <td> small metal hook </td> <td> $0.65 </td> </tr> <tr> <td> flat-head bolt </td> <td> $0.50 </td> </tr> <tr> <td> light bulb </td> <td> $0.77 </td> </tr> </table>"""
# convert the html to 2D grid
table_2d =get_table_2d(html_table)
# get the values of the cells
answers =[]
for row_id, col_id in indices:
        answer =get_cell_value(table_2d, row_id, col_id)
        answers.append(answer)
final_answer = "\n".join(answers)

# print the final answer in the required JSON format
print(final_answer)
</code>
<answer>[PYTHON ANSWER]</answer>
```

Figure 15: MMTab and CoReTab annotations for a sample from TCE.



powdered sugar	$3/kilogram
light brown sugar	$3/kilogram
dark brown sugar	$3/kilogram
maple syrup	$3/kilogram

TCL

Question: Given a table image and a set of target cells, your task is to identify their positions. Represent each cell's location as JSON in the format {'value': '<cell value>', 'location': (row_id, column_id)}, with both row and column id begin from 1. Refer to the values below: (1) 'light brown sugar' (2) 'dark brown sugar' (3) 'maple syrup' (4) 'powdered sugar'
Use 'DOES NOT EXIST' as the location for a non-existing cell in the table.

MMTab Annotation: The target cell locations are as follows: (1) {'value': 'light brown sugar', 'location': (2, 1)} (2) {'value': 'dark brown sugar', 'location': (3, 1)} (3) {'value': 'maple syrup', 'location': (4, 1)} (4) {'value': 'powdered sugar', 'location': (1, 1)}

CoReTab Annotation (Ours): [block 1]

<reason>
I need to get the location of the cells from the table.
I will use the available `get_cell_location` tool to do this.

Step1: Extract the list of values from the text query and save it in the variable `values`.

Step2: Get the html representation of the table and save it in the variable `html_table`.

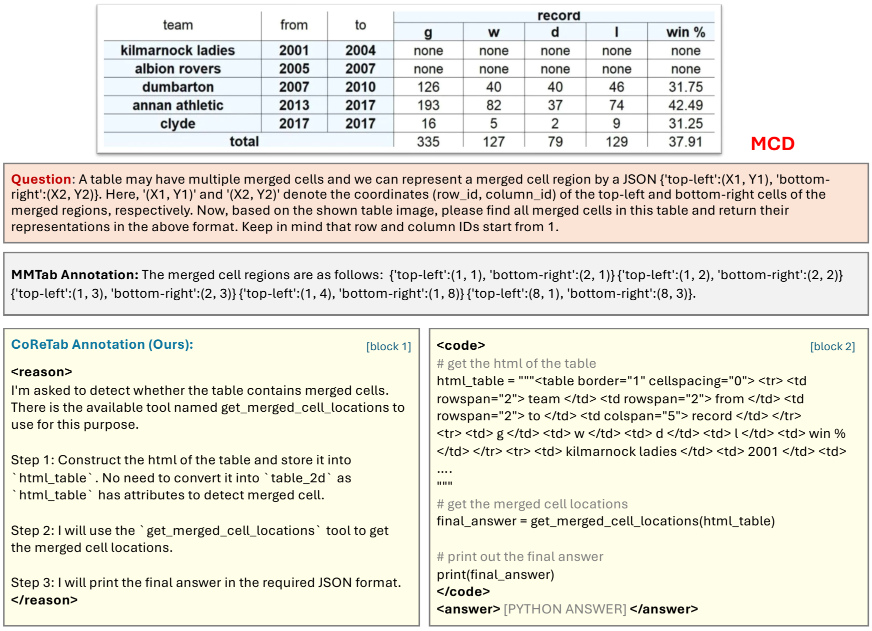
Step3: Use the `get_cell_location` tool to get the cell location for each value.

Step4: Merge the cell locations to get the final answer in the format of `row_id, column_id`.
</reason>

[block 2]

```
<code>
# Extract the list values of the cells to be located
values =['light brown sugar', 'dark brown sugar', 'maple syrup', 'powdered sugar’]
# get the html of the table
html_table = """<table border="1" cellspacing="0"> <tr> <td> powdered sugar </td> <td> $3/kilogram </td> </tr> <tr> <td> light brown sugar </td> <td> $3/kilogram </td> </tr> <tr> <td> dark brown sugar </td> <td> $3/kilogram </td> </tr> <tr> <td> maple syrup </td> <td> $3/kilogram </td> </tr> </table>"""
# convert the html to 2D grid
table_2d =get_table_2d(html_table)
# get the location of the cells for each value
answers =[]
for value in values:
        answer =get_cell_location(table_2d, value)
        answers.append(answer)
final_answer = "\n".join(answers)
# print the final answer in given format
print(final_answer)</code>

<answer>[PYTHON ANSWER]</answer>
```

Figure 16: MMTab and CoReTab annotations for a sample from TCL.



team	from	to	record				
			g	w	d	l	win %
kilmarnock ladies	2001	2004	none	none	none	none	none
albion rovers	2005	2007	none	none	none	none	none
dumbarton	2007	2010	126	40	40	46	31.75
annan athletic	2013	2017	193	82	37	74	42.49
clyde	2017	2017	16	5	2	9	31.25
total			335	127	79	129	37.91

MCD

Question: A table may have multiple merged cells and we can represent a merged cell region by a JSON {'top-left':(X1, Y1), 'bottom-right':(X2, Y2)}. Here, '(X1, Y1)' and '(X2, Y2)' denote the coordinates (row_id, column_id) of the top-left and bottom-right cells of the merged regions, respectively. Now, based on the shown table image, please find all merged cells in this table and return their representations in the above format. Keep in mind that row and column IDs start from 1.

MMTab Annotation: The merged cell regions are as follows: {'top-left':(1, 1), 'bottom-right':(2, 1)} {'top-left':(1, 2), 'bottom-right':(2, 2)} {'top-left':(1, 3), 'bottom-right':(2, 3)} {'top-left':(1, 4), 'bottom-right':(1, 8)} {'top-left':(8, 1), 'bottom-right':(8, 3)}.

CoReTab Annotation (Ours): [block 1]

<reason>
I'm asked to detect whether the table contains merged cells. There is the available tool named get_merged_cell_locations to use for this purpose.

Step 1: Construct the html of the table and store it into `html_table`. No need to convert it into `table_2d` as `html_table` has attributes to detect merged cell.

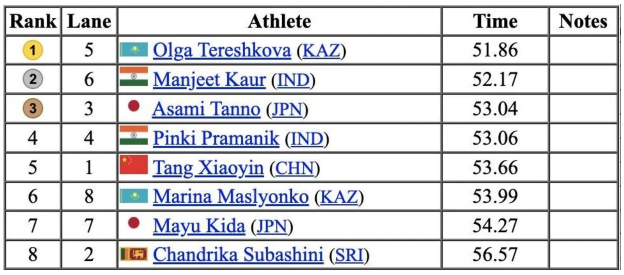
Step 2: I will use the `get_merged_cell_locations` tool to get the merged cell locations.

Step 3: I will print the final answer in the required JSON format.
</reason>

[block 2]

```
<code>
# get the html of the table
html_table = """<table border="1" cellspacing="0"> <tr> <td rowspan="2"> team </td> <td rowspan="2"> from </td> <td rowspan="2"> to </td> <td colspan="5"> record </td> </tr> <tr> <td> g </td> <td> w </td> <td> d </td> <td> l </td> <td> win % </td> </tr> <tr> <td> kilmarnock ladies </td> <td> 2001 </td> <td>
....
"""
# get the merged cell locations
final_answer = get_merged_cell_locations(html_table)

# print out the final answer
print(final_answer)
</code>
<answer> [PYTHON ANSWER] </answer>
```

Figure 17: MMTab and CoReTab annotations for a sample from MCD.



Rank	Lane	Athlete	Time	Notes
①	5	Olga Tereshkova (KAZ)	51.86	
②	6	Manjeet Kaur (IND)	52.17	
③	3	Asami Tanno (JPN)	53.04	
4	4	Pinki Pramanik (IND)	53.06	
5	1	Tang Xiaoyin (CHN)	53.66	
6	8	Marina Maslyonko (KAZ)	53.99	
7	7	Mayu Kida (JPN)	54.27	
8	2	Chandrika Subashini (SRI)	56.57	

RCE

Question: Analyze the provided table and extract the contents of specified rows, identified by row indices. Represent the cells in a row with the JSON format {"row_id":"<row index>", "cell_list":"<a list of cells in this row>"}. For instance, {"row_id":"5", "cell_list": ["152", "UN", "one last kiss"]}. Target rows: the 4th row, the 3rd row,

MMTab Annotation: The extracted row contents are listed below. For the 4th row: {"row_id": "4", "cell_list": ['', '3', 'Asami Tanno\xa0(JPN)', '53.04']} For the 3rd row: {"row_id": "3", "cell_list": ['', '6', 'Manjeet Kaur\xa0(IND)', '52.17']}

CoReTab Annotation (Ours): [block 1]

<reason>
The task is to extract the cells from the table.
With the available tools, will use `get_row_values` and `get_col_values` to do this.

Step1: list row and column indices to be extracted from the text query and save it in the variable `indices`.

Step2: get the html of the table and save it in the variable `html_table`.

Step3: convert html_table to a 2D grid using `get_table_2d` tool and save it in the variable `table_2d`.

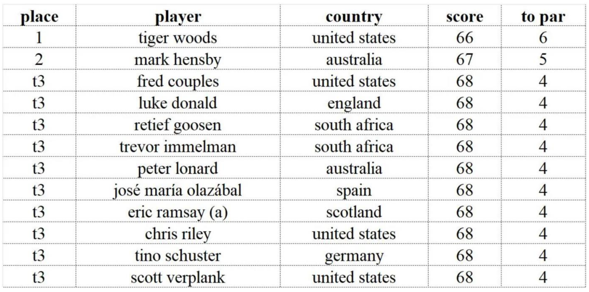
Step4: use the `get_row_values` and `get_col_values` tool to get the row and column values.

Step5: merge the row and column values to get the final answer in the required JSON format.
</reason>

[block 2]

```
<code>
# list row and column indices to be extracted indices
=[{'row':4},{'row':3}]# get the html of the table html_table = """<table
border="1" cellspacing="0"> <tr> <th> Rank </th> <th> Lane </th> <th>
Athlete </th> <th> Time </th> <th> ...Mayu Kida </a> <span>( <a> JPN
</a>) </span> </td> <td>54.27 </td> <td> </td> </tr> <tr> <td>8 </td>
<td>2 </td> <td> <a> Chandrika Subashini </a> <span>( <a> SRI </a>)
</span> </td> <td>56.57 </td> <td> </td> </tr> </table>"""

# convert the html to 2D
grid table_2d =get_table_2d(html_table)
# get the values of the cells
answers =[]
for index in indices:
        if 'row' in index:
                row_id = index['row’]
                answer = get_row_values(table_2d, row_id)
        elif 'col' in index:
                col_id = index['col’]
                answer =get_col_values(table_2d, col_id)
        answers.append(answer)
final_answer = "\n".join(answers)
# print out the final answer in the required format
print(final_answer) </code>
<answer>[PYTHON ANSWER]</answer>
```

Figure 18: MMTab and `CoReTab` annotations for a sample from RCE.



place	player	country	score	to par
1	tiger woods	united states	66	6
2	mark hensby	australia	67	5
t3	fred couples	united states	68	4
t3	luke donald	england	68	4
t3	retief goosen	south africa	68	4
t3	trevor immelman	south africa	68	4
t3	peter lonard	australia	68	4
t3	josé maría olazábal	spain	68	4
t3	eric ramsay (a)	scotland	68	4
t3	chris riley	united states	68	4
t3	tino schuster	germany	68	4
t3	scott verplank	united states	68	4

Question: Analyze the table's content, including its title '2005 open championship', and determine whether the given statement is supported or refuted by the table information. Return the result as JSON in the format {"answer": "<YOUR ANSWER>"}, e.g., {"answer": "deputed"}. statement: jose maria olazabal was the only player from united states to score four under par

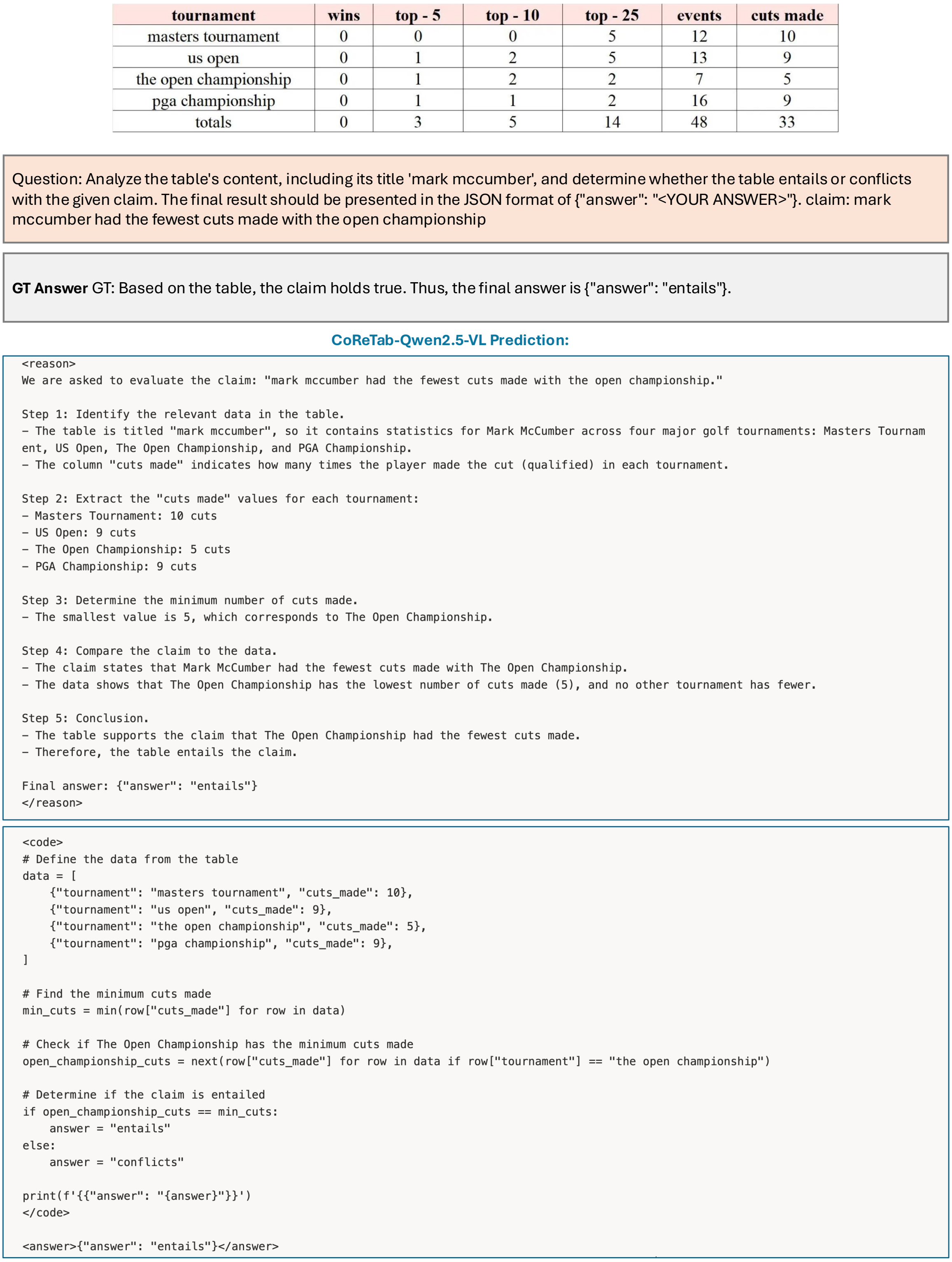
GT Answer: GT: Based on the evidence presented in the table, we can conclude that the statement is not true. Therefore, the answer is {"answer": "refuted"}.

CoReTab-Qwen2.5-VL Prediction:

```
<reason>
We are asked to verify the statement: "jose maria olazabal was the only player from united states to score four under par."

Step 1: Identify the relevant column – "to par" – which indicates how many under par a player is.

Step 2: Look for players from the "united states" and check their "to par" values.

From the table:
- tiger woods: to par = 6 → not four under par
- fred couples: to par = 4 → from united states
- luke donald: to par = 4 → from england
- retief goosen: to par = 4 → from south africa
- trevor immelman: to par = 4 → from south africa
- peter lonard: to par = 4 → from australia
- josé maría olazábal: to par = 4 → from spain
- eric ramsay (a): to par = 4 → from scotland
- chris riley: to par = 4 → from united states
- tino schuster: to par = 4 → from germany
- scott verplank: to par = 4 → from united states

So, players from the United States with "to par" = 4:
- fred couples
- chris riley
- scott verplank

Thus, there are three players from the United States who scored four under par, not just one.

Step 3: The statement claims that "jose maria olazabal was the only player from united states to score four under par" – but this is f
alse because there are two others (chris riley and scott verplank).

Therefore, the statement is refuted by the table.

Final answer: {"answer": "refuted"}
</reason>
```

```
<code>
# Define the data from the table
players = [
    {"place": "1", "player": "tiger woods", "country": "united states", "score": 66, "to par": 6},
    {"place": "2", "player": "mark hensby", "country": "australia", "score": 67, "to par": 5},
    {"place": "t3", "player": "fred couples", "country": "united states", "score": 68, "to par": 4},
    {"place": "t3", "player": "luke donald", "country": "england", "score": 68, "to par": 4},
    {"place": "t3", "player": "retief goosen", "country": "south africa", "score": 68, "to par": 4},
    {"place": "t3", "player": "trevor immelman", "country": "south africa", "score": 68, "to par": 4},
    {"place": "t3", "player": "peter lonard", "country": "australia", "score": 68, "to par": 4},
    {"place": "t3", "player": "josé maría olazábal", "country": "spain", "score": 68, "to par": 4},
    {"place": "t3", "player": "eric ramsay (a)", "country": "scotland", "score": 68, "to par": 4},
    {"place": "t3", "player": "chris riley", "country": "united states", "score": 68, "to par": 4},
    {"place": "t3", "player": "tino schuster", "country": "germany", "score": 68, "to par": 4},
    {"place": "t3", "player": "scott verplank", "country": "united states", "score": 68, "to par": 4},
]

# Filter players from United States with to par = 4
us_players_four_under = [p for p in players if p["country"] == "united states" and p["to par"] == 4]

# Check if Jose Maria Olazabal is the only one
is_only_one = len(us_players_four_under) == 1 and us_players_four_under[0]["player"] == "josé maría olazábal"

# Determine the answer
answer = "refuted" if not is_only_one else "supported"
print(f'{{"answer": "{answer}"}}')
</code>

<answer>{"answer": "refuted"}</answer>
```

Figure 19: Generated response of CoReTab-Qwen2.5-VL on the test sample.



tournament	wins	top - 5	top - 10	top - 25	events	cuts made
masters tournament	0	0	0	5	12	10
us open	0	1	2	5	13	9
the open championship	0	1	2	2	7	5
pga championship	0	1	1	2	16	9
totals	0	3	5	14	48	33

Question: Analyze the table's content, including its title 'mark mccumber', and determine whether the table entails or conflicts with the given claim. The final result should be presented in the JSON format of {"answer": "<YOUR ANSWER>"}. claim: mark mccumber had the fewest cuts made with the open championship

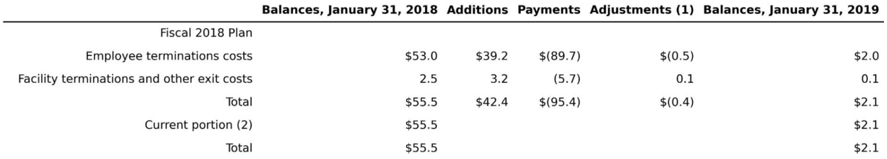
GT Answer GT: Based on the table, the claim holds true. Thus, the final answer is {"answer": "entails"}.

CoReTab-Qwen2.5-VL Prediction:

```
<reason>
We are asked to evaluate the claim: "mark mccumber had the fewest cuts made with the open championship."

Step 1: Identify the relevant data in the table.
- The table is titled "mark mccumber", so it contains statistics for Mark McCumber across four major golf tournaments: Masters Tournament, US Open, The Open Championship, and PGA Championship.
- The column "cuts made" indicates how many times the player made the cut (qualified) in each tournament.

Step 2: Extract the "cuts made" values for each tournament:
- Masters Tournament: 10 cuts
- US Open: 9 cuts
- The Open Championship: 5 cuts
- PGA Championship: 9 cuts

Step 3: Determine the minimum number of cuts made.
- The smallest value is 5, which corresponds to The Open Championship.

Step 4: Compare the claim to the data.
- The claim states that Mark McCumber had the fewest cuts made with The Open Championship.
- The data shows that The Open Championship has the lowest number of cuts made (5), and no other tournament has fewer.

Step 5: Conclusion.
- The table supports the claim that The Open Championship had the fewest cuts made.
- Therefore, the table entails the claim.

Final answer: {"answer": "entails"}
</reason>
```

```
<code>
# Define the data from the table
data = [
    {"tournament": "masters tournament", "cuts_made": 10},
    {"tournament": "us open", "cuts_made": 9},
    {"tournament": "the open championship", "cuts_made": 5},
    {"tournament": "pga championship", "cuts_made": 9},
]

# Find the minimum cuts made
min_cuts = min(row["cuts_made"] for row in data)

# Check if The Open Championship has the minimum cuts made
open_championship_cuts = next(row["cuts_made"] for row in data if row["tournament"] == "the open championship")

# Determine if the claim is entailed
if open_championship_cuts == min_cuts:
    answer = "entails"
else:
    answer = "conflicts"

print(f'{{"answer": "{answer}"}}')
</code>

<answer>{"answer": "entails"}</answer>
```

Figure 20: Generated response of CoReTab-Qwen2.5-VL on the test sample.



	Balances, January 31, 2018	Additions	Payments	Adjustments (1)	Balances, January 31, 2019
Fiscal 2018 Plan					
Employee terminations costs	$53.0	$39.2	$(89.7)	$(0.5)	$2.0
Facility terminations and other exit costs	2.5	3.2	(5.7)	0.1	0.1
Total	$55.5	$42.4	$(95.4)	$(0.4)	$2.1
Current portion (2)	$55.5				$2.1
Total	$55.5				$2.1

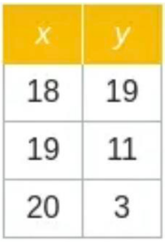
Given the table, answer the following question in detail. Present the final answer in a JSON format {"answer": [<a list of answer strings>]}. What was the employee termination costs as a proportion of total costs in 2018?

GT: Based on the table, we need to conduct the following computations: 53.0/55.5 = 0.95 Therefore, the final answer: {"answer": ["0.95"]}.

CoReTab-Qwen2.5-VL Prediction:

```
<reason>
Step 1: Identify the relevant row for "Employee terminations costs" in the table.
- The value for "Employee terminations costs" is $2.0 (in millions).

Step 2: Identify the total costs for 2018.
- The "Total" row under the "Fiscal 2018 Plan" section shows the total costs as $2.1 (in millions).

Step 3: Calculate the proportion of employee termination costs relative to total costs.
- Proportion = (Employee termination costs) / (Total costs)
- Proportion = 2.0 / 2.1 ≈ 0.95238

Step 4: Format the result as a string with two decimal places and return in the required JSON format.
- The final answer is {"answer": ["0.95238"]}.
</reason>

<code>
# Define the values from the table
employee_termination_costs = 2.0  # in millions
total_costs = 2.1  # in millions

# Calculate the proportion
proportion = employee_termination_costs / total_costs

# Print the final answer in the required JSON format with 4 decimal places
print(f'{{"answer": ["{proportion:.4f}"]}}')
</code>

<answer>{"answer": ["0.95238"]}</answer>
```

Figure 21: Generated response of `CoReTab`-Qwen2.5-VL on the test sample.

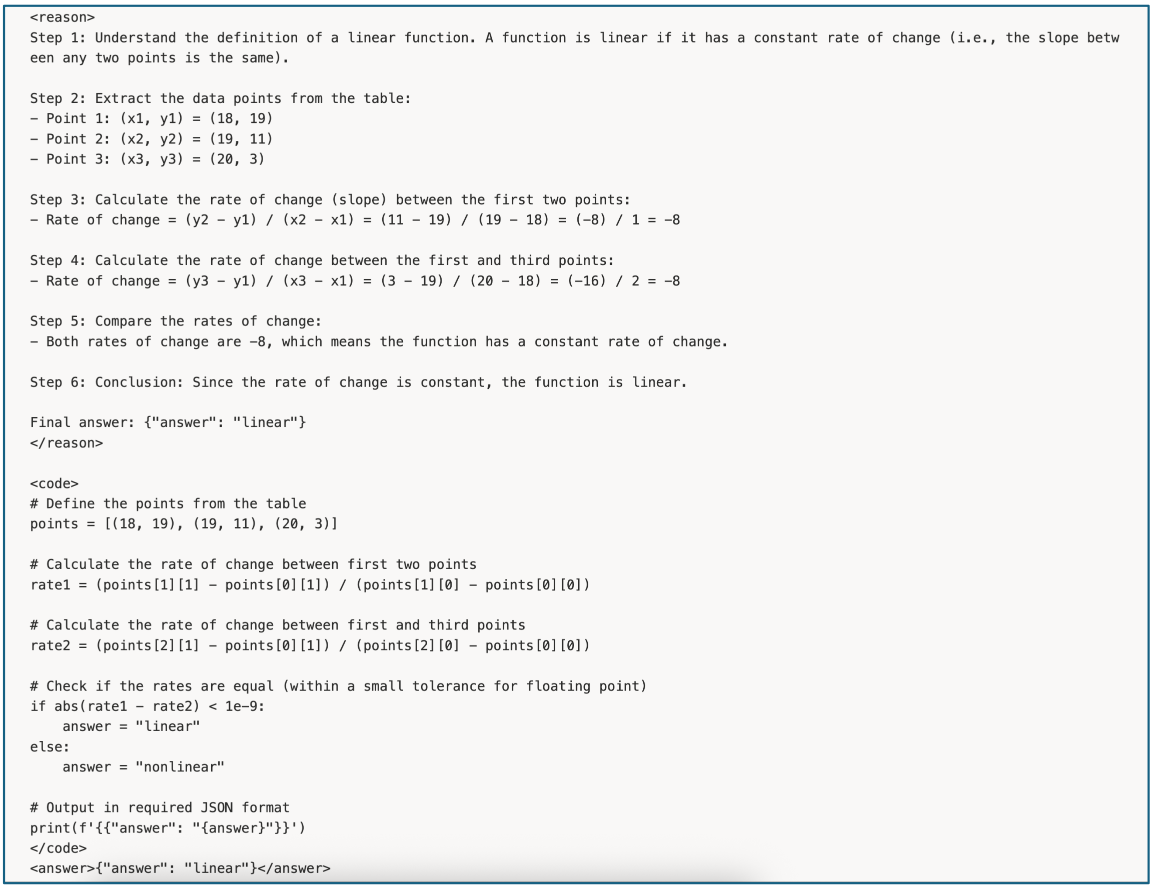


x	y
18	19
19	11
20	3

Question: With the table image in mind, formulate a detail solution for the math problem. Show specific steps towards the answer. Output the final answer as JSON in the format {"answer": "<YOUR ANSWER>"}. The table shows a function. Is the function linear or nonlinear? (A) linear (B) nonlinear

GT: To determine whether the function is linear or nonlinear, see whether it has a constant rate of change. Pick the points in any two rows of the table and calculate the rate of change between them. The first two rows are a good place to start. Call the values in the first row x1 and y1. Call the values in the second row x2 and y2. Rate of change = \frac{y2 - y1}{x2 - x1} = \frac{11 - 19}{19 - 18} = \frac{-8}{1} = -8 Now pick any other two rows and calculate the rate of change between them. Call the values in the second row x1 and y1. Call the values in the third row x2 and y2. Rate of change = \frac{y2 - y1}{x2 - x1} = \frac{3 - 11}{20 - 19} = \frac{-8}{1} = -8 The two rates of change are the same. 8. This means the rate of change is the same for each pair of points. So, the function has a constant rate of change. The function is linear. Therefore, the final answer is {"answer": "linear"}.

CoReTab-Qwen2.5-VL Prediction:



```
<reason>
Step 1: Understand the definition of a linear function. A function is linear if it has a constant rate of change (i.e., the slope betw
een any two points is the same).

Step 2: Extract the data points from the table:
- Point 1: (x1, y1) = (18, 19)
- Point 2: (x2, y2) = (19, 11)
- Point 3: (x3, y3) = (20, 3)

Step 3: Calculate the rate of change (slope) between the first two points:
- Rate of change = (y2 - y1) / (x2 - x1) = (11 - 19) / (19 - 18) = (-8) / 1 = -8

Step 4: Calculate the rate of change between the first and third points:
- Rate of change = (y3 - y1) / (x3 - x1) = (3 - 19) / (20 - 18) = (-16) / 2 = -8

Step 5: Compare the rates of change:
- Both rates of change are -8, which means the function has a constant rate of change.

Step 6: Conclusion: Since the rate of change is constant, the function is linear.

Final answer: {"answer": "linear"}
</reason>

<code>
# Define the points from the table
points = [(18, 19), (19, 11), (20, 3)]

# Calculate the rate of change between first two points
rate1 = (points[1][1] - points[0][1]) / (points[1][0] - points[0][0])

# Calculate the rate of change between first and third points
rate2 = (points[2][1] - points[0][1]) / (points[2][0] - points[0][0])

# Check if the rates are equal (within a small tolerance for floating point)
if abs(rate1 - rate2) < 1e-9:
    answer = "linear"
else:
    answer = "nonlinear"

# Output in required JSON format
print(f'{{"answer": "{answer}"}}')
</code>
<answer>{"answer": "linear"}</answer>
```

Figure 22: Generated response of CoReTab-Qwen2.5-VL on the test sample.