Analyzing LLM Instruction Optimization for Tabular Fact Verification

Xiaotang Du[1] **Giwon Hong**[1] **Wai-Chung Kwan**[1] **Rohit Saxena**[1] **Ivan Titov**[1]
Pasquale Minervini[1,2] **Emily Allaway**[1]
[1]University of Edinburgh, United Kingdom [2]Miniml.AI, United Kingdom
{xiaotang.du, ghong, p.minervini, emily.allaway}@ed.ac.uk

Abstract

Instruction optimization provides a lightweight, model-agnostic approach to enhancing the reasoning performance of large language models (LLMs). This paper presents the first systematic comparison of instruction optimization, based on the DSPy optimization framework, for tabular fact verification. We evaluate four out-of-the-box prompting techniques that cover both text-only prompting and code use: direct prediction, Chain-of-Thought (CoT), ReAct with SQL tools, and CodeAct with Python execution. We study three optimizers from the DSPy framework—COPRO, MiPROv2, and SIMBA— across four benchmarks and three model families. We find that instruction optimization consistently improves verification accuracy, with MiPROv2 yielding the most stable gains for CoT, and SIMBA providing the largest benefits for ReAct agents, particularly at larger model scales. Behavioral analyses reveal that SIMBA encourages more direct reasoning paths by applying heuristics, thereby improving numerical comparison abilities in CoT reasoning and helping avoid unnecessary tool calls in ReAct agents. Across different prompting techniques, CoT remains effective for tabular fact checking, especially with smaller models. Although ReAct agents built with larger models can achieve competitive performance, they require careful instruction optimization.

1 Introduction

Verifying natural language claims against structured data is a central capability for trustworthy NLP systems deployed in science, public health, and information quality assurance. While numerous methods have been proposed for tabular fact verification (Yang and Zhu, 2021; Ou and Liu, 2022; Lu et al., 2025; Zhang et al., 2024b, *inter alia*), the resulting systems are often specialized to a particular dataset or fail to outperform simpler prompting approaches.

In this work, we conduct a comparative study of out-of-the-box prompting techniques, paired with instruction optimization, for tabular fact verification. Instruction optimization is a technique that allows for improvements to LLM performance without gradient updates. Since LLMs are known to be sensitive to prompt formulation (Webson and Pavlick, 2022; Leidinger et al., 2023), we analyze the impacts of instruction optimization on practical and generalizable prompting techniques, such as Chain-of-Thought (Wei et al., 2022).

Recent frameworks for instruction optimization (e.g., DSPy; Khattab et al., 2024) treat multi-step LLM pipelines as programs whose textual parameters can be automatically tuned by search or meta-reasoning, yielding large gains on diverse tasks. Despite this progress, a systematic understanding of how instruction optimization affects tabular fact verification is lacking. The following impacts are particularly underexplored: (1) prompting techniques that differ in their intermediate computation (e.g., direct prediction, CoT, and program-aided reasoning via SQL and Python), (2) optimizer families, and (3) model scale and families. Tool-augmented agents (e.g., ReAct; Yao et al., 2023) promise stronger grounding by interleaving thoughts with executable actions, but their end-to-end effectiveness depends critically on the learned tool interface and execution reliability—factors that instruction optimization may help or hinder.

We present the first comparative study of instruction optimization for tabular fact verification using the DSPy optimization framework. Our study focuses on three optimizers within DSPy: COPRO, MiPROv2, and SIMBA[1]. We analyze these optimizers across four benchmarks (TabFact, PubHealthTab, SciTab and MMSci), four prompting techniques (Direct prediction, CoT, ReAct, and

[1]We restrict MiPROv2 and SIMBA to instruction-only tuning to isolate the effect of instructions from few-shot example selection.

CodeAct), and three base LLMs (Qwen3, Gemma3, GPT-4o). We conduct a comprehensive analysis, using both task metrics (accuracy, macro-F1) and behavioral shifts (e.g., frequency and success of SQL/Python calls, error taxonomies), to address the following research questions:

- What is the impact of optimized instructions on CoT reasoning?
- How does the optimized instructions affect the tool calling behavior of ReAct agent?
- Does program-aided reasoning show consistent advantages over CoT in tabular fact checking?

Our experiment results show that the influence of instruction optimization varies depending on the prompting technique and model size. Compared with smaller models, larger models benefit more from the optimized instructions and show greater improvement across four test sets. Among different optimizers, MiPROv2 brings consistent gains for CoT reasoning, while SIMBA proves effective for improving tool use behavior of ReAct.

A closer inspection of the optimized instructions reveal that MiPROv2 and SIMBA optimizer encourage the model to choose a direct reasoning path, avoiding unnecessary tool calls for simple claims. With SIMBA optimizer, the tool use frequency is reduced, with more successful tool execution. In addition, we find SIMBA optimizer includes more heuristics about numerical comparison in the refined instructions, which helps improve the overall accuracy in CoT. Our code is available at `https://github.com/xiaodu0123/tabfact-prompt-optimization`.

2 Related Work

Table-based Fact Checking Verifying claims against structured evidence requires compositional reasoning over diverse table schema. TabFact (Chen et al., 2020) established the first large-scale benchmark for binary fact verification on Wikipedia tables. Later datasets incorporated more nuanced labeling schema (e.g., three labels instead of only two) and more complex claims requiring multi-hop reasoning (Wang et al., 2021). Among these, several domain-specific datasets have been created: PubHealthTab (Akhtar et al., 2022), which targets claims about public health, SciTab (Lu et al., 2023), which includes claims from computer science publications, and SciAtomicBench (Zhang et al., 2025), which covers computer science along with other domains such as finance. While fact verification datasets typically present tabular data in textual form, multi-modal datasets have also been created (Yang et al., 2025b). Additionally, some fact-verification datasets mix tabular evidence with text (Aly et al., 2021; Schlichtkrull et al., 2023; Zhao et al., 2024) and figures (Wang et al., 2025; Chan et al., 2024).

Early methods for tabular fact verification used symbolic or programmatic reasoning (Chen et al., 2020; Zhong et al., 2020; Shi et al., 2020; Zhang et al., 2020; Yang et al., 2020; Yang and Zhu, 2021; Ou and Liu, 2022). While some recent work has also made use of neuro-symbolic systems (Glenn et al., 2024; Aly and Vlachos, 2024; Cheng et al., 2023), there has been an increasing focus on adapting and making use of LLMs. To this end, prior works have developed both pre-training (Eisenschlos et al., 2020; Dong and Smith, 2021; Zhang et al., 2024a) and fine-tuning (Wu and Feng, 2024; Jiang et al., 2025) approaches for table-based fact verification, as well as more general table-based reasoning tasks (Herzig et al., 2020; Liu et al., 2022). Additionally, several works propose prompting techniques for improving model reasoning over tables (Wang et al., 2024b; Zhang et al., 2025; Abhyankar et al., 2025; Zhang et al., 2024b). Recently, work has also begun to investigate agentic systems and tool-use for table-based fact verification (Lu et al., 2025; Zhou et al., 2025). However, despite these advances, many systems are computationally intensive or specialized to a particular dataset. In contrast, our work explores computationally light instruction optimization techniques applied to general prompting strategies.

Most closely related to our work are two recent analyses into the challenges of various table understanding tasks, including fact verification. Bhandari et al. (2025) examine how instruction tuning, in-context examples, and model size impact performance on tabular reasoning tasks, while Wu et al. (2025) survey approaches to table understanding tasks more broadly. In contrast to these analyses, our work compares instruction optimization techniques applied to simple prompting strategies (standard baselines such as CoT as well as simple programmatic reasoning models such as ReAct). Additionally, while Bhandari et al. (2025) cover multiple table understanding tasks, our work focuses only on table-based fact verification, opting instead to cover a wider range of datasets tabular fact verification.

3 Method

3.1 Prompting Techniques

Chain-of-Thought Chain-of-thought reasoning (CoT) (Wei et al., 2022) encourages LLMs to generate intermediate reasoning steps before producing the final answer. With CoT, LLMs can decompose a complex query into sub-problems and progressively build the solution in the reasoning traces.

ReAct ReAct (Yao et al., 2023) serves as a foundational framework for tool-based agents by interleaving reasoning with task-specific actions. ReAct enables LLMs to interact with external tools, allowing them to collect additional evidence and ground their reasoning in the tool execution output. In our experiments, we evaluate a ReAct agent with access to a standard SQL tool that can execute SQL queries on the table data to retrieve relevant information and perform math operations.

CodeAct CodeAct (Wang et al., 2024a) leverages executable Python code as the primary action modality for tool-based agents. Unlike existing paradigms that rely on tool calls in text or JSON formats, CodeAct enables multi-step operations and flexible tool chaining through code execution, allowing the agent to perform sophisticated actions by integrating with Python's control flow and existing libraries. In our experiments, the CodeAgent has no access to pre-defined tools. It generates free-form python codes to process the table data and perform math operations step by step.

3.2 Instruction Optimization

In our analysis, we focus on three LLM-based instruction optimization approaches in the DSPy (Khattab et al., 2024) framework: COPRO, MiPROv2 (Opsahl-Ong et al., 2024) and SIMBA.

DSPy Framework DSPy is a framework for algorithmically optimizing model prompts and weights, treating LLM pipelines as programmes that can be automatically compiled and optimized.

COPRO Cooperative Prompt Optimization (COPRO) systematically explores various candidate instructions in a beam search-like manner and evaluates their performance on the train set. The optimizer iteratively refines the prompt instruction by proposing multiple new candidate instructions based on the N best prompts among previous attempts and their corresponding evaluation scores.

MiPROv2 Multi-Stage Instruction Prompt Optimization (MiPROv2) is an advanced framework that can refine both the instruction and few-shot demonstrations through a three-stage pipeline. First, the optimizer bootstraps multiple candidate sets of few-shot demonstrations from the training data. Then, it generates diverse prompt instructions and demonstrations based on previously evaluated candidates, the properties of the downstream task, and randomly sampled prompting strategies. Finally, MiPROv2 employs Bayesian optimization method to efficiently search the best combination of candidate instruction and demonstration.

Compared with COPRO, MiPROv2 provides a richer context for the generation of new candidate instructions and performs more efficient evaluation on mini-batches of training data.

SIMBA Stochastic Introspective Mini-Batch Ascent (SIMBA) is an introspective prompt optimization algorithm that leverages the language model's capacity for self-reflection to iteratively improve instruction quality. The optimizer identifies challenging training instances where model outputs exhibit high variability, then applies two complementary strategies to refine prompts. One strategy performs contrastive analysis, where the model compares successful and unsuccessful execution traces to generate explicit improvement rules that are appended to the original instruction. Another strategy incorporates successful execution trajectories as few-shot demonstrations.

4 Experiments

4.1 Datasets

We evaluate the performance of various LLMs on four tabular fact checking datasets: TabFact (Chen et al., 2020), PubHealthTab (Akhtar et al., 2022), SciTab (Lu et al., 2023) and MMSci (Yang et al., 2025b). These datasets cover diverse domains and table types, ranging from general knowledge to specialized data, thereby enabling a more comprehensive evaluation of the generalization ability of different approaches. In SciTab, PubHealthTab, and MMSci, there are three labels: *supports*, *refutes* and *not enough info*; TabFact is a binary classification task with only *supports* and *refutes* labels.

SciTab SciTab (Lu et al., 2023) is a benchmark designed for scientific claim verification, leveraging real-world table evidence from scientific publications in machine learning and natural language

Dataset	Train	Dev	Test
TabFact	92,585	12,851	8,609
SciTab	210	429	429
PubHealthTab	1440	177	180
MMSci	-	-	1,038

Table 1: Statistics of the fact checking datasets.

processing domains. The dataset presents unique challenges in claim ambiguity, compositional reasoning and numerical analysis of scientific data.

PubHealthTab PubHealthTab (Akhtar et al., 2022) is a table-based fact checking dataset focusing on public health claims. The evidence tables are extracted from multiple web sources, which exhibit noisy and complex table structure with varying content quality.

TabFact TabFact (Chen et al., 2020) is a large-scale table-based fact verification dataset that consists of human-annotated claims with Wikipedia tables as evidence. TabFact provides two test sets that differ in the claim complexity, and we use the complex test set for evaluation.

MMSci MMSci (Yang et al., 2025b) is a benchmark for multimodal scientific reasoning across three table-based tasks. We use the table fact verification test set, converting table images to textual format, to evaluate generalization to unseen data.

4.2 Optimization

For each considered LLM, we evaluate the performance of different prompting techniques, including direct prompting, CoT, ReAct and CodeAct to study the impact of instruction optimization on both language-based reasoning and program-aided reasoning. We use the same instructions in the system prompt before optimization for different experiments, i.e. `verify the given claim against the provided table data.` All the experiments are conducted in zero-shot setting.

4.3 Evaluation Setup

Models and Baselines We conduct our experiments using Qwen3 (Yang et al., 2025a), Gemma3 (Team et al., 2025) and GPT-4o models, which allows us to systematically investigate the impact of instruction optimization on reasoning and tool-calling behavior across different model families and sizes. The same model is used for proposing candidate instructions and evaluating instruction quality during optimization. To examine the effectiveness of optimized instructions, we compare the model performance in GPT-4o experiments with ReActable (Zhang et al., 2024b), a ReAct framework that uses GPT-4o with human-written instructions and SQL and Python as tools.

Data processing Each fact checking dataset is processed into a unified data format. We then split three of the datasets (TabFact, SciTab, and PubHealthTab) into train, development and test sets; our fourth dataset, *MMSci, is used only for evaluation*. We create a hybrid training set for instruction optimization by randomly sampling 100 instances from the training splits of the three datasets. We sample 40 PubHealthTab instances, 40 SciTab instances and 20 TabFact instances to ensure the label distribution of the hybrid dataset is balanced. Statistics of the processed datasets are in Table 1.

Evaluation metrics We optimize the instructions using the hybrid train data, and evaluate the performance on the development and test sets of all four datasets with accuracy and macro-F1. During instruction optimization, only accuracy is used to measure the quality of different candidate prompts.

5 Results

We report the test performance of different prompting techniques with Qwen3 models on four fact checking datasets in Table 2. For direct prompting and CoT, larger models generally achieve higher accuracy and F1 than their smaller counterparts across most test sets. For program-aided reasoning paradigms (ReAct, CodeAct), increasing model size does not yield significant performance gains. Although larger models have similar baseline performance to smaller versions, they benefit substantially more from instruction optimization and show greater improvement with refined instructions.

The effectiveness of instruction optimization for tabular reasoning is highly dependent on both model scale and the prompting technique. For optimizing CoT reasoning, MiPROv2 brings the most consistent gains, achieving the highest accuracy and F1 on PubHealthTab, TabFact and MMSci for Qwen3-8B, and showing competitive results across three datasets with Qwen3-32B. For program-based reasoning, SIMBA provides the strongest performance gain on SciTab, particularly for improving ReAct with the Qwen3-32B model. COPRO also offers moderate benefits for Qwen3-

Module	Optimizer	Qwen3-8B								Qwen3-32B							
		PubHealth		SciTab		TabFact		MMSci		PubHealth		SciTab		TabFact		MMSci	
		Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1
Direct	Baseline	**73.3**	**73.4**	**58.7**	56.5	58.0	52.8	57.2	**41.5**	84.4	82.3	52.9	49.6	64.1	62.8	68.2	46.3
	+COPRO	**73.3**	**73.4**	**58.7**	**56.6**	58.1	52.8	57.0	41.2	84.4	82.7	**53.4**	**51.1**	65.3	63.9	69.5	47.7
	+MiPROv2	72.8	72.9	56.4	54.3	**58.5**	**53.4**	56.6	41.3	84.4	82.3	53.1	50.0	64.1	62.8	68.2	46.3
	+SIMBA	**73.3**	**73.4**	**58.7**	56.5	58.1	52.8	**57.3**	41.4	**85.6**	**84.3**	52.7	50.0	**67.6**	**68.7**	**70.6**	**49.4**
CoT	Baseline	83.9	82.3	64.3	64.4	77.6	80.5	81.9	58.0	88.3	87.6	66.4	66.4	84.5	86.6	86.5	61.6
	+COPRO	83.9	82.8	**66.2**	**66.2**	76.8	79.9	79.4	56.3	87.2	86.1	67.4	67.3	85.5	87.6	86.7	61.5
	+MiPROv2	**86.1**	**85.7**	66.0	66.0	**80.3**	**83.1**	**82.5**	**59.4**	87.2	86.5	**68.8**	**68.6**	**86.9**	**88.5**	**87.7**	**65.4**
	+SIMBA	82.2	81.4	62.5	62.2	77.6	80.6	81.6	58.7	**90.0**	**89.6**	**68.8**	**68.6**	85.2	87.1	87.0	64.2
ReAct	Baseline	**86.7**	**86.6**	61.3	61.2	**83.8**	**85.3**	82.5	58.5	87.8	87.4	61.5	60.1	**86.4**	**87.0**	**87.5**	62.6
	+COPRO	84.4	83.9	**62.2**	**62.1**	80.5	81.5	**83.6**	**61.5**	86.1	84.7	62.0	61.5	81.5	84.1	85.0	61.0
	+MiPROv2	81.7	81.1	61.8	61.8	75.5	80.6	82.2	60.0	87.8	87.2	61.5	60.9	84.2	85.2	86.2	63.0
	+SIMBA	86.1	85.2	58.3	58.3	82.9	84.7	80.8	57.3	**90.6**	**90.0**	**66.2**	**65.9**	86.1	**87.0**	85.9	**65.0**
CodeAct	Baseline	**86.1**	**86.0**	57.1	57.1	82.0	83.5	81.2	59.2	85.6	84.9	58.0	57.5	85.9	87.1	87.5	**66.1**
	+COPRO	82.8	82.2	**59.7**	59.1	80.0	82.2	83.3	**60.7**	**87.2**	**86.6**	62.2	61.8	**86.7**	**87.9**	**88.1**	63.3
	+MiPROv2	**86.1**	85.7	56.9	56.6	80.5	82.0	82.1	59.0	83.9	83.5	59.0	58.2	86.4	87.6	86.5	62.7
	+SIMBA	85.0	84.9	**59.7**	**59.5**	**84.8**	**85.5**	**84.3**	59.8	85.6	85.2	**69.2**	**69.3**	85.4	87.0	86.5	62.6

Table 2: Results of Qwen3-8B and Qwen3-32B on test sets. **Bold** is best performance per method and dataset.

32B model but less consistently than SIMBA. This suggests that larger models are better at identifying patterns of successful trajectories through self-reflection and comparative analysis, leading to more effective rules for optimizing tool use in diverse scenarios.

According to Table 3, the general trend observed with the Gemma3 model family is slightly different from Qwen3. The larger Gemma3 model shows consistently higher performance for both CoT reasoning *and* program-aided reasoning. Unlike Qwen3, where the optimizers fail to enhance the performance for ReAct with a smaller model, Gemma3 models respond more positively to instruction optimization across different prompting techniques and show greater improvement with refined instructions at both sizes.

Similar to Qwen3 experiments, MiPROv2 still delivers significant improvements when optimizing CoT. SIMBA performs exceptionally well for improving ReAct and CodeAct, particularly for the larger 27B model. COPRO remains effective for smaller model (12B) but provides smaller incremental gains relative to MiPROv2 and SIMBA. Overall, the Gemma3 model family underperforms Qwen3, even after applying instruction optimization. For both Gemma3 and Qwen3 models, CoT reasoning consistently achieves competitive performance after instruction optimization compared with program-aided reasoning methods on tabular fact checking.

Table 4 summarizes the test performance of GPT-4o models. Due to budget considerations, GPT-4o models and ReActable are evaluated on a smaller TabFact test set (TabFact-mini) with 400 random instances. GPT-4o models demonstrate much stronger baseline performance, and consequently benefit less from instruction optimization than Qwen3 and Gemma3 models. For GPT-4o-mini, MiPROv2 is more effective for improving CoT reasoning, while SIMBA yields greater improvements across the test sets for optimizing ReAct. However, no single optimizer provides consistent performance gains for optimizing CodeAct. For the GPT-4o model, SIMBA performs consistently well and brings improvement to both CoT and ReAct, whereas MiPROv2 is shown to be effective for enhancing CodeAct performance. ReAct with GPT-4o shows slightly worse performance on SciTab and TabFact-mini compared with the ReActable baseline, but it can consistently outperform ReActable across all test sets after SIMBA optimization, which demonstrates the superiority of DSPy-based instruction optimization over manually designed prompts.

According to the test performance on MMSci, we observe that for Qwen3-32B and Gemma3-27B model, the optimized instructions with superior performance on PubHealthTab, SciTab and TabFact often generalize well to MMSci. Specifically, instructions optimized by SIMBA consistently achieves the highest F1 scores on MMSci in both direct prompting and ReAct settings, while CoT instructions learned by MiPROv2 continues to deliver the strongest improvements on MMSci. However, this trend is not observed in GPT-4o models, for which the performance on the other three fact checking datasets is not predictive of test performance on MMSci. Although SIMBA shows strong performance on SciTab and TabFact-mini across direct prompting, CoT and ReAct settings, these performance gains do not consistently transfer to MMSci test data. This may indicate instructions proposed by GPT-4o during SIMBA optimization generalize less effectively on unseen data.

To further examine the effectiveness and generalizability of instruction optimization, we conduct ablation studies using varying random seeds, initial instructions of diverse quality and different training data. We also extend the evaluation of ReAct agents by considering single or multiple most commonly used tools introduced in TART framework (Lu et al., 2025). More detail of our experiments can be found in Section C.

5.1 Effects of Optimizing Instructions on Table Reasoning

Tables 2 and 3 provide evidence that prompt optimization improves table reasoning performance in both the direct and CoT settings. To analyze this in more detail, Tables 5 and 6 present confusion matrices comparing the differences between SIMBA optimization and the baseline.

The most salient pattern observed from Tables 5 and 6 is that optimization increases the proportion of refute predictions, indicating that it leads to more conservative predictions overall. To further explain this effect, we analyzed the optimized instructions for both the direct and CoT modules.

For direct prompting, we observe that the optimized instructions are often tuned towards ordinal information. For Qwen3-32B optimized with SIMBA, the resulting instruction (Table 7) includes an additional prompt encouraging attention to the order of elements in the claim (e.g., *"If the table contradicts the claim's order, the module should return 'refutes.'"*). To validate this effect, we defined a set of ordinal terms (see Section A.1) and analyzed the effectiveness of the optimized instruction depending on whether the claim contained ordinal terms (Table 8). Indeed, when the gold label is 'refute', the optimized instruction shows a notable 4.5% improvement (from 66.1% to 70.6%) for claims containing ordinal terms compared to those without.

For CoT, the instruction (Table 7) optimized by SIMBA with the Qwen3-32B model is particularly specialized to numerical comparisons (e.g., *"the module should focus on comparing the values in the table to determine if the claim is supported or refuted"*). Following a similar procedure as above, we defined a set of comparative terms (Section A.2) and analyzed whether the optimized instruction prompts more explicit comparison behavior during reasoning, and how this affects performance (Table 9). Again, when the gold label is 'refute', the optimized instruction yields a 2.4% improvement (from 83.7% to 86.1%).

Tables 8 and 9 validate that the optimization process encourages the model to focus on specific aspects crucial for table reasoning (element order and numerical comparison for Direct prediction and CoT respectively) and that these behaviors are effectively reflected in performance improvements.

5.2 Analysis of Tool Use Behavior

We investigate the tool use behavior of ReAct with Qwen3 models on the SciTab test data before and after instruction optimization. For the 32B model, COPRO increases the tool calling frequency from 90% to 93%, but leads to more error rates (successful execution is reduced from 42% to 12%). Both MiPROv2 and SIMBA guide the model towards a more direct reasoning path by reducing tool usage frequency from 90% to 70.6% and 35.4% respectively, while increasing the average length of reasoning in the trajectory by 16% and 33%. This implies that MiPROv2 and SIMBA encourage direct table interpretation rather than relying heavily on tool-based verification and only invoke tool calls when necessary. MiPROv2 leads to more tool execution errors after optimization, while SIMBA maintains a similar level of error rate compared to the baseline.

For the smaller 8B model, COPRO and MiPROv2 dramatically reduce tool usage frequency from 67% to around 40%, while SIMBA maintains a similar level of tool use frequency as the baseline with a slightly improved successful execution rate (40.8% to 41.2%).

5.3 Error analysis of ReAct and CodeAct

To assess the validity of ReAct and CodeAct results, we sample 20 error cases from each setting with Qwen3-32B on the SciTab test set and conduct a manual error analysis. We define a high-level error taxonomy consisting of three categories: data-related, code-related, and reasoning errors, and an-

Module	Optimizer	Gemma3-12B								Gemma3-27B							
		PubHealth		SciTab		TabFact		MMSci		PubHealth		SciTab		TabFact		MMSci	
		Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1
Direct	Baseline	77.8	72.8	48.3	43.4	57.6	54.6	64.7	38.9	82.8	80.4	53.6	50.7	58.6	60.3	66.7	45.0
	+COPRO	80.6	77.2	49.9	46.4	58.8	58.9	65.9	**45.3**	82.8	80.2	51.5	48.2	54.4	59.2	65.7	44.9
	+MiPROv2	80.6	79.4	**55.0**	**54.7**	**63.2**	**64.1**	**66.9**	45.0	82.8	80.3	55.9	54.6	59.2	61.6	**67.4**	46.0
	+SIMBA	**81.7**	**79.5**	54.3	52.8	59.2	60.1	64.5	45.0	**85.6**	**83.7**	**60.6**	**60.6**	**62.9**	**62.9**	67.3	**47.4**
CoT	Baseline	87.8	86.4	54.3	52.3	75.5	77.7	79.3	54.6	87.8	86.9	62.2	61.9	78.3	80.8	82.9	58.9
	+COPRO	87.8	86.5	57.3	56.4	74.5	76.6	79.8	54.8	**89.4**	**88.7**	61.5	61.3	78.4	81.6	84.6	59.8
	+MiPROv2	87.2	85.6	58.3	57.8	**80.1**	**82.2**	**84.7**	**60.5**	88.9	87.8	**64.8**	**64.4**	**81.4**	**83.4**	**85.8**	**62.5**
	+SIMBA	**89.4**	**88.8**	**60.1**	**59.6**	77.6	79.3	83.2	57.7	88.9	87.6	63.6	63.8	75.8	79.1	81.9	59.0
ReAct	Baseline	83.9	82.9	49.2	48.7	64.9	72.8	79.9	57.5	87.8	86.8	52.9	52.9	76.3	80.8	82.9	58.7
	+COPRO	**87.2**	**86.5**	**58.3**	**57.1**	77.1	79.4	84.7	**61.0**	85.0	83.6	48.0	47.9	72.9	78.4	69.3	52.5
	+MiPROv2	84.4	83.5	49.0	48.7	64.6	72.5	79.9	57.4	89.4	88.8	**63.6**	**63.2**	82.9	84.4	**86.5**	62.5
	+SIMBA	86.7	85.7	53.4	51.0	**79.8**	**81.1**	**84.8**	59.2	**90.0**	**89.3**	60.4	58.9	**84.0**	**85.0**	85.8	**62.6**
CodeAct	Baseline	86.7	86.0	51.5	49.8	64.7	72.2	83.2	57.7	87.2	86.2	55.9	56.1	73.6	78.7	85.8	61.3
	+COPRO	**89.4**	**88.9**	54.3	53.4	67.0	74.4	**85.0**	61.1	88.9	87.9	**59.2**	**59.5**	79.0	81.5	84.4	61.1
	+MiPROv2	88.3	87.6	49.9	48.6	**78.9**	**81.6**	84.7	59.0	85.6	84.8	55.5	56.0	81.3	83.6	86.5	62.8
	+SIMBA	85.0	84.0	**55.2**	**54.8**	77.5	79.9	83.4	**61.7**	**89.4**	**88.5**	58.3	56.7	**83.1**	**84.4**	**87.6**	**65.6**

Table 3: Results of Gemma3-12B and Gemma3-27B on test sets. **Bold** is best performance per method and dataset.

Module	Optimizer	GPT-4o-mini								GPT-4o							
		PubHealth		SciTab		TabFact-mini		MMSci		PubHealth		SciTab		TabFact-mini		MMSci	
		Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1
ReActable		73.9	73.4	52.7	52.4	85.5	85.5	64.3	48.2	83.3	82.6	67.6	67.6	89.8	89.7	85.4	62.8
Direct	Baseline	85.6	85.3	58.3	58.4	65.0	**66.7**	70.0	51.2	**90.6**	**89.8**	65.0	65.0	73.2	74.8	82.1	60.1
	+COPRO	**86.7**	**87.1**	**61.1**	**61.0**	**65.2**	65.7	70.8	51.8	89.4	88.9	64.8	64.7	76.0	77.0	**84.7**	61.2
	+MiPROv2	85.0	85.2	60.1	59.8	63.5	63.9	**71.6**	**52.9**	90.0	89.1	65.0	65.1	74.5	76.0	84.4	**61.7**
	+SIMBA	86.1	85.6	57.1	56.6	60.5	64.0	69.7	51.1	89.4	88.5	**65.3**	**65.2**	**76.5**	**77.3**	82.8	59.8
CoT	Baseline	**90.6**	**90.1**	62.9	63.0	79.8	82.4	83.0	58.9	87.8	87.2	69.2	69.1	87.8	89.6	87.7	63.7
	+COPRO	90.0	89.6	61.8	61.7	81.0	82.7	83.6	61.3	87.8	87.5	69.7	69.6	88.0	89.8	**88.4**	65.0
	+MiPROv2	89.4	88.9	**64.8**	**64.8**	**81.2**	**83.0**	84.4	60.9	89.4	88.9	**70.6**	**70.5**	88.5	89.9	88.3	**65.8**
	+SIMBA	90.0	89.5	64.3	64.3	78.8	81.1	**84.5**	**62.2**	**90.0**	**89.8**	**70.6**	**70.5**	**90.2**	**91.4**	87.9	64.3
ReAct	Baseline	87.8	87.3	55.0	53.1	84.8	85.4	84.4	61.2	88.3	87.3	64.1	62.8	90.0	90.3	**89.5**	66.2
	+COPRO	89.4	88.9	59.4	58.4	82.8	83.7	**85.7**	61.8	**89.4**	**88.9**	67.8	67.3	90.2	91.0	88.7	67.0
	+MiPROv2	90.0	89.6	**60.1**	**60.0**	82.5	83.2	84.5	60.3	**89.4**	88.4	66.2	65.6	90.8	91.4	88.5	**67.6**
	+SIMBA	**91.7**	**91.1**	**60.1**	59.9	**84.8**	**86.1**	84.0	**62.2**	88.3	87.6	**68.3**	**68.3**	**91.0**	**92.3**	88.1	64.0
CodeAct	Baseline	**84.4**	**83.7**	**59.0**	**58.8**	82.5	83.9	84.5	60.4	87.2	86.7	63.4	62.3	90.2	90.8	89.3	**65.4**
	+COPRO	**84.4**	82.8	53.4	52.2	83.5	84.7	**85.4**	**61.3**	89.4	89.0	62.9	60.7	90.5	91.4	**89.7**	64.9
	+MiPROv2	80.6	77.9	52.2	48.9	**85.2**	**86.6**	82.9	57.8	**91.1**	**90.6**	**65.0**	**63.9**	**91.2**	**91.7**	89.2	62.6
	+SIMBA	**84.4**	83.0	55.7	54.9	81.5	83.0	84.1	58.6	88.3	87.6	61.1	60.5	90.0	91.4	89.0	**65.4**

Table 4: Results of GPT-4o-mini and GPT-4o on test sets. **Bold** is best performance per method and dataset.

alyze which category or categories each erroneous instance belongs to, noting that a single instance can fall into more than one category.

The main sources of errors in ReAct are issues with reasoning (45%) and data-related errors (35 %). ReAct often focuses on part of the statement in a complex claim, neglecting other relevant data in tool planning or reasoning steps. Also, tool traces with execution errors can lead to illogical reasoning that is inconsistent with the provided table data. Data-related issues include underspecified claims and noisy table structure, which makes it difficult for ReAct to construct correct SQL queries.

For CodeAct, the error analysis in Table 11 shows that the largest share of errors (70%) stems from issues intrinsic to the SciTab dataset, including subjective or underspecified claims (e.g. undefined notions of "significance"), claims requiring information not present in the table, or incorrect gold labels. Code-related errors account for over 30% of cases; among 6 such errors, 4 arise from code execution failures due primarily to syntactic issues, indicating that more robust coding agents could substantially reduce this error type. Reasoning errors constitute around 15% of the total and are claim-related, typically involving claim misinterpretation or failure to recognize that the table lacks the required information, where a "not

Direct		Gold	
(Base→SIMBA)		Support	Refute
Predict	Support	3,870→2,941	2,430→1,270
	Refute	311→1,242	1,648→2,880
	Not Enough Info	125→123	225→153

Table 5: Confusion matrices before and after prompt optimization for Direct prediction with the Qwen3-32B model on TabFact. → and → indicate positive and negative changes respectively.

CoT		Gold	
(Base→SIMBA)		Support	Refute
Predict	Support	3,643→3,648	429→403
	Refute	483→499	3,628→3,691
	Not Enough Info	180→159	246→209

Table 6: Confusion matrices before and after prompt optimization for CoT with the Qwen3-32B model on TabFact. → and → indicate positive and negative changes respectively.

enough information" label should have been predicted. Notably, even when code execution failed, the model occasionally compensated during the reasoning phase by approximating the intended computation through inspection of the generated code or table contents.

5.4 Comparison of CoT with ReAct

To understand the performance differences between CoT reasoning and tool-augmented agents, we conduct a detailed analysis of 50 instances from the SciTab test set and evaluate Qwen3-8B and Qwen3-32B predictions after optimizing instructions with MiPROv2. As shown in Table 10, ReAct consistently outperforms CoT across both Qwen3-32B (78% vs. 70%) and Qwen3-8B (60% vs. 54%) on this subset. A closer inspection of these 50 claims reveals complementary failure modes.

The primary advantage of ReAct lies in its iterative verification process, which is effective for verifying complex or multi-clause claims that involve multiple quantitative checks, such as “G2S approaches outperform the S2S baseline”. Even though some tool calls may fail with execution errors, the agent can re-examine the table evidence and switch to a different strategy using language-based reasoning in the following step. This stands in contrast to CoT’s reasoning process, which often fails to aggregate all sub-claims before reaching a final verdict. In addition, CoT often misinterprets metric directionality, such as treating lower error rates (e.g., ADDED/MISS values) as worse performance. For Qwen3-32B, four of the six ReAct-only successes are on claims that CoT incorrectly labels as refutes due to misinterpreting metric directionality. The other are cases where CoT abstains from deciding on multi-clause improvements (e.g. "outperforms baselines by 10.5 F1"). The 8B model exhibits a similar pattern but with a stronger tendency to abstain from making decisions.

However, ReAct’s reliance on tool-based verification can hinder its performance on simple and straightforward claims, where direct table interpretation is enough for making the decision, such as "BI and IS individually outperform the oracle". For the Qwen3-8B model, every instance where CoT succeeded but ReAct failed is due to tool execution errors. The agent is prone to violating the tool schema and representing table data in the wrong format when making tool calls. The resulting execution errors prevent the verifier from revisiting the claim, causing the reasoning process to terminate with either a *not enough info* or *refutes* label. Overall, this comparison confirms that tool planning encourages more systematic evidence checking, but the benefits only appear when the learned tool interface is properly aligned with the tool executor.

5.5 Comparison of ReAct with CodeAct

We analyze 100 random instances from SciTab test data to study the differences between ReAct and CodeAct based on Qwen3-32B predictions. Distribution of the correctness before and after SIMBA optimization is shown in Table 12.

Among 17 ReAct-only correct instances in the baseline experiments, we observe that CodeAct often struggles with parsing table data as well as handling numerical aggregation and comparison. Unlike ReAct that uses SQL tool to extract relevant table data and solves math problems in the following reasoning steps, CodeAct relies on python codes for both table parsing and math reasoning. The generated python code often introduces a table representation or directly assigns relevant table cells to variables, which is prone to mistakes. Also, CodeAct tends to be overly strict when making numerical comparisons and may fail to consider all the relevant entries. After SIMBA optimization, CodeAct gives correct predictions on 11 out of 17 error instances. We find SIMBA is particu-

Module	Instruction
Baseline	Verify the given claim against the provided table data.
Direct (SIMBA)	Verify the given claim against the provided table data.\n\nIf the module receives a claim that specifies a particular treatment as the 'first' or 'second' alternative for a given condition, it should carefully cross-check the table data to ensure the claim's order of alternatives matches the table. [...] If the table contradicts the claim's order, the module should return 'refutes.'[...] Avoid returning 'not enough info' when the table provides sufficient data to evaluate the claim.
CoT (SIMBA)	Verify the given claim against the provided table data.\n\nIf the claim refers to the effectiveness or performance of a method in a specific stage, and the table includes evaluation metrics (e.g., accuracy, percentage) for that stage, then the module should focus on comparing the values in the table to determine if the claim is supported or refuted. Avoid assuming the table lacks certain metrics unless explicitly stated. [...]

Table 7: Baseline and optimized instructions for the Qwen3-32B model using the SIMBA optimizer. The prompt parts that exhibit the characteristics analyzed in Tables 8 and 9 are underlined. Some parts of the instructions have been omitted for clarity.

Metric	With Ordinal Terms		Without Ordinal Terms	
	Baseline	SIMBA	Baseline	SIMBA
Instances	1,499		7,110	
Accuracy by Gold Label				
All (overall)	63.7	**69.0**	64.2	**67.3**
Refute	39.0	**70.6**	38.1	**66.1**
Support	**90.2**	67.4	**89.8**	67.4

Table 8: Comparison of Direct prediction with Qwen3-32B before and after optimization (SIMBA) on instances containing ordinal terms *in the claim* (Section A.1) from TabFact.

Metric	Baseline	SIMBA	Overlap
Instances w/ comparative terms	1,882	**2,016**	2,382
Accuracy on Overlapping Instances by Gold Label			
All (overall)	83.0	**84.1**	–
Refute	83.7	**86.1**	–
Support	**82.2**	81.8	–

Table 9: Comparison of CoT with Qwen3-32B before and after optimization (SIMBA) on instances containing comparative terms *in the reasoning* (Section A.2) from TabFact.

Outcome	Qwen3-32B	Qwen3-8B
Both correct	33	22
CoT only correct	2	5
ReAct only correct	6	8
Both wrong	9	15

Table 10: Comparison of CoT and ReAct on 50 random SciTab test claims.

Taxonomy	ReAct	CodeAct
Data-related Errors	7 (35%)	14 (70.0%)
Code-related Errors	5 (25%)	6 (30.0%)
Reasoning Errors	9 (45%)	3 (15.0%)

Table 11: Number of error instances of each type found in ReAct and CodeAct predictions with Qwen3-32B on SciTab.

Outcome	Baseline	with SIMBA
Both correct	44	52
ReAct only correct	17	12
CodeAct only correct	7	13
Both wrong	32	23

Table 12: Comparison of ReAct and CodeAct with Qwen3-32B on 100 random SciTab test instances

larly effective for improving table parsing and simple numerical comparison by introducing heuristic rules. However, CodeAct still shows limited reasoning ability after optimization in validating complex claims, especially demonstrating weak performance when the claim contains ambiguous descriptions or requires multi-row comparisons.

6 Conclusion

Recent research on tabular fact checking has investigated tool-augmented and agentic approaches, yet it remains unclear whether external tools and program-aided reasoning provide consistent advantages over language-based reasoning for latest LLMs. In this work, we present the first comparative analysis of various prompting techniques for tabular fact checking and examine the impact of instruction optimization on reasoning and tool use behavior across two model families. Our analysis reveals that MiPROv2 optimizer yields substantial gains for CoT reasoning, while SIMBA proves effective in refining the instructions of ReAct and CodeAct agents. Both MiPROv2 and SIMBA encourage more direct reasoning paths and help reduce unnecessary tool calls. Overall, CoT remains a strong choice for tabular fact checking, particularly with smaller models. Meanwhile, ReAct agents built on larger models can achieve competitive performance, but require more careful instruction optimization.

Limitations

While our study provides valuable insights into the impact of instruction optimization on diverse prompting techniques for tabular fact checking, we acknowledge several limitations in our experimental setup. The scope of our experiments is restricted to two specific model families, Qwen3 and Gemma3, each with two sizes, and the generalizability of our findings to other model architectures or substantially larger models remains an open question. Our comparative analysis is limited to three representative instruction optimization methods, and a broader survey of latest techniques such as GEPA (Agrawal et al., 2025) could reveal different model behaviors or performance trade-offs. Our evaluation of ReAct agent is conducted with a single tool for executing SQL queries, which does not capture the complexities of tool selection in multi-tool scenarios. Additionally, we do not account for the potential influence of the quality of the initial instruction on the optimization process. All the experiments are conducted with the same seed instruction in a simple format.

Acknowledgments

We thank the anonymous reviewers for the insightful feedback and comments. Xiaotang Du was partly supported by the UKRI Centre for Doctoral Training in Natural Language Processing, funded by UK Research and Innovation (grant EP/S022481/1) and the University of Edinburgh, School of Informatics. Pasquale Minervini was partially funded by ELIAI (The Edinburgh Laboratory for Integrated Artificial Intelligence), EPSRC (grant no. EP/W002876/1), an industry grant from Cisco, and a donation from Accenture LLP. Giwon Hong was supported by the ILCC PhD program (School of Informatics Funding Package) at the University of Edinburgh, School of Informatics. This work was supported by the Edinburgh International Data Facility (EIDF) and the Data-Driven Innovation Programme at the University of Edinburgh. Some evaluations were performed using resources provided by the Cambridge Service for Data Driven Discovery (CSD3) operated by the University of Cambridge Research Computing Service, provided by Dell EMC and Intel using Tier-2 funding from the Engineering and Physical Sciences Research Council (capital grant EP/T022159/1), and DiRAC funding from the Science and Technology Facilities Council.

References

Nikhil Abhyankar, Vivek Gupta, Dan Roth, and Chandan K. Reddy. 2025. H-STAR: LLM-driven hybrid SQL-text adaptive reasoning on tables. 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 8841–8863, Albuquerque, New Mexico. Association for Computational Linguistics.

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. 2025. Gepa: Reflective prompt evolution can outperform reinforcement learning. *Preprint*, arXiv:2507.19457.

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.

Rami Aly, Zhijiang Guo, Michael Sejr Schlichtkrull, James Thorne, Andreas Vlachos, Christos Christodoulopoulos, Oana Cocarascu, and Arpit Mittal. 2021. The fact extraction and VERification over unstructured and structured information (FEVEROUS) shared task. In *Proceedings of the Fourth Workshop on Fact Extraction and VERification (FEVER)*, pages 1–13, Dominican Republic. Association for Computational Linguistics.

Rami Aly and Andreas Vlachos. 2024. TabVer: Tabular fact verification with natural logic. *Transactions of the Association for Computational Linguistics*, 12:1648–1671.

Kushal Raj Bhandari, Sixue Xing, Soham Dan, and Jianxi Gao. 2025. Exploring the robustness of language models for tabular question answering via attention analysis. *Trans. Mach. Learn. Res.*, 2025.

Chu Sern Joel Chan, Aakanksha Naik, Matthew Akamatsu, Hanna Bekele, Erin Bransom, Ian Campbell, and Jenna Sparks. 2024. Overview of the context24 shared task on contextualizing scientific claims. In *Proceedings of the Fourth Workshop on Scholarly Document Processing (SDP 2024)*, pages 12–21, Bangkok, Thailand. Association for Computational Linguistics.

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

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

Rui Dong and David Smith. 2021. Structural encoding and pre-training matter: Adapting BERT for table-based fact verification. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 2366–2375, Online. Association for Computational Linguistics.

Julian Eisenschlos, Syrine Krichene, and Thomas Müller. 2020. Understanding tables with intermediate pre-training. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 281–296, Online. Association for Computational Linguistics.

Parker Glenn, Parag Dakle, Liang Wang, and Preethi Raghavan. 2024. BlendSQL: A scalable dialect for unifying hybrid question answering in relational algebra. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 453–466, Bangkok, Thailand. Association for Computational Linguistics.

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

Chuang Jiang, Mingyue Cheng, Xiaoyu Tao, Qingyang Mao, Jie Ouyang, and Qi Liu. 2025. Tablemind: An autonomous programmatic agent for tool-augmented table reasoning. *arXiv preprint arXiv:2509.06278*.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. 2024. Dspy: Compiling declarative language model calls into state-of-the-art pipelines. In *ICLR*. OpenReview.net.

Alina Leidinger, Robert van Rooij, and Ekaterina Shutova. 2023. The language of prompting: What linguistic properties make a prompt successful? In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 9210–9232, Singapore. Association for Computational Linguistics.

Jason Liu and Contributors. 2024. Instructor: A library for structured outputs from large language models.

Qian Liu, Bei Chen, Jiaqi Guo, Morteza Ziyadi, Zeqi Lin, Weizhu Chen, and Jian-Guang Lou. 2022. TAPEX: table pre-training via learning a neural SQL executor. In *ICLR*. OpenReview.net.

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

Xinyuan Lu, Liangming Pan, Yubo Ma, Preslav Nakov, and Min-Yen Kan. 2025. TART: An open-source tool-augmented framework for explainable table-based reasoning. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 4323–4339, Albuquerque, New Mexico. Association for Computational Linguistics.

Krista Opsahl-Ong, Michael J Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. 2024. Optimizing instructions and demonstrations for multi-stage language model programs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 9340–9366, Miami, Florida, USA. Association for Computational Linguistics.

Suixin Ou and Yongmei Liu. 2022. Learning to generate programs for table fact verification via structure-aware semantic parsing. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7624–7638, Dublin, Ireland. Association for Computational Linguistics.

Michael Schlichtkrull, Zhijiang Guo, and Andreas Vlachos. 2023. AVeriTeC: A dataset for real-world claim verification with evidence from the web. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Qi Shi, Yu Zhang, Qingyu Yin, and Ting Liu. 2020. Learn to combine linguistic and symbolic information for table-based fact verification. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 5335–5346, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, Louis Rouillard, Thomas Mesnard, Geoffrey Cideron, Jean bastien Grill, Sabela Ramos, Edouard Yvinec, Michelle Casbon, Etienne Pot, Ivo Penchev, and 197 others. 2025. Gemma 3 technical report. *Preprint*, arXiv:2503.19786.

Chengye Wang, Yifei Shen, Zexi Kuang, Arman Cohan, and Yilun Zhao. 2025. SciVer: Evaluating foundation models for multimodal scientific claim verification. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8562–8579, Vienna, Austria. Association for Computational Linguistics.

Nancy X. R. Wang, Diwakar Mahajan, Marina Danilevsky, and Sara Rosenthal. 2021. SemEval-2021 task 9: Fact verification and evidence finding for tabular data in scientific documents (SEM-TAB-FACTS). In *Proceedings of the 15th International Workshop on Semantic Evaluation (SemEval-2021)*, pages 317–326, Online. Association for Computational Linguistics.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. 2024a. Executable code actions elicit better llm agents. In *ICML*.

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

Albert Webson and Ellie Pavlick. 2022. Do prompt-based models really understand the meaning of their prompts? In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2300–2344, Seattle, United States. Association for Computational Linguistics.

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.

Xiaofeng Wu, Alan Ritter, and Wei Xu. 2025. Tabular data understanding with llms: A survey of recent advances and challenges. *arXiv preprint arXiv:2508.00217*.

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

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025a. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Bohao Yang, Yingji Zhang, Dong Liu, André Freitas, and Chenghua Lin. 2025b. Does table source matter? benchmarking and improving multimodal scientific table understanding and reasoning. *arXiv preprint arXiv:2501.13042*.

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

Xiaoyu Yang and Xiaodan Zhu. 2021. Exploring decomposition for table-based fact verification. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 1045–1052, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing reasoning and acting in language models. In *Proceedings of the Eleventh International Conference on Learning Representations*. ICLR.

Hongzhi Zhang, Yingyao Wang, Sirui Wang, Xuezhi Cao, Fuzheng Zhang, and Zhongyuan Wang. 2020. Table fact verification with structure-aware transformer. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1624–1629, Online. Association for Computational Linguistics.

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

Yuji Zhang, Qingyun Wang, Cheng Qian, Jiateng Liu, Chenkai Sun, Denghui Zhang, Tarek Abdelzaher, Chengxiang Zhai, Preslav Nakov, and Heng Ji. 2025. Atomic reasoning for scientific table claim verification. *arXiv preprint arXiv:2506.06972*.

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

Yilun Zhao, Yitao Long, Yuru Jiang, Chengye Wang, Weiyuan Chen, Hongjun Liu, Yiming Zhang, Xiangru Tang, Chen Zhao, and Arman Cohan. 2024. FinDVer: Explainable claim verification over long and hybrid-content financial documents.

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

Wei Zhou, Mohsen Mesgar, Annemarie Friedrich, and Heike Adel. 2025. Efficient multi-agent collaboration with tool use for online planning in complex table question answering. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 945–968, Albuquerque, New Mexico. Association for Computational Linguistics.

A Implementation Details

A.1 List of Ordinal Terms

Table 13 shows the list of ordinal terms used in the experiments in Section 5.1. These terms are pre-determined prior to the experiments and used to identify cases where the claim involves ordinal information, such as ordering or ranking relations between the claim and the table values or among the table values themselves.

A.2 List of Comparative Terms

Table 14 shows the list of comparative terms used in the experiments in Section 5.1. These terms are pre-determined prior to the experiments and served as a proxy indicator for whether the model's reasoning process involves comparisons either between the claim and the table values or among the table values themselves.

A.3 Hyperparameter setting

We conduct our analysis with the instruction optimizers from DSPy.

COPRO For COPRO optimization, we set the breadth to 6 and depth to 3. The initial temperature for instruction generation is set to 1.2.

MiPROv2 To ensure a consistent comparison among the optimizers, MiPROv2 optimizer is configured to refine only the prompt instructions. Specifically, we set the number of bootstrapped demonstrations and number of labeled demonstrations to 0 and use the medium optimization mode.

SIMBA We set the number of generated candidates per iteration to 6 and the number of iteration steps to 8. The maximal number of few-shot demonstrations is set to 0 to ensure only the instructions are optimized. We use a random seed of 0 for sampling candidate programs during optimization.

A.4 Dataset Details

All the evaluation datasets (PubHealthTab, SciTab, TabFact, MMSci) used in our experiments are available under the MIT License. For each dataset, we adopt the original version from the official Github repository.

A.5 Computation Details

Experiments with Qwen3-8B, Gemma3-12B and Gemma3-27B were run on a single NVIDIA A100 GPU with 80GB of GPU memory. Experiments with Qwen3-32B were run on two NVIDIA A100 GPUs. The GPU hours depend on the model size, prompting method, optimizer type and size of test dataset. For example, optimizing the instructions in ReAct or CodeAct with COPRO and SIMBA optimizer on the hybrid train data takes up to 10 hours. Evaluation on TabFact test data with ReAct or CodeAct framework using a large backbone model (27B/32B) usually takes up to 2 to 3 days.

A.6 Implementation of ReActable

The official ReActable implementation is incompatible with GPT-4o models, since the models do not consistently follow the output format defined in the in-context samples, which leads to parsing failure when extracting code blocks from generated content. To evaluate ReActable with GPT-4o models, we reproduce the pipeline by enforcing structured output generation using `instructor` library (Liu and Contributors, 2024). At each iteration step, the model can choose from three actions, including executing SQL command, executing Python snippet and predicting the final verdict. The code is executed on the most recent intermediate table that can be successfully processed. We set the maximum number of iterations to 5, use a temperature of 0 and limit the generation to 3,500 tokens per iteration step. The model is forced to output the final label if it generates repeated code blocks in consecutive iterations or if code execution fails.

We extend the prompt templates by adapting the instructions to our tasks and design system prompts with more detailed instructions to provide guidance on structured output generation, as shown in Figure 1, 2, 3 and 4. We use the same in-context demonstrations across all three-class fact checking experiments, which are built from 5 random instances in SciTab training set. We use 3 out of these in-context samples annotated with binary labels in TabFact experiments.

B Optimized Instructions

Table 15 shows the optimized instructions with different optimizers for Qwen3-32B. With COPRO optimization, the refined instruction describes a detailed procedural workflow, where "design an SQL query to extract relevant data" is included as a mandatory reasoning step. The agent is forced to use the tool for simple queries, which can lead to a high volume of low-quality or unnecessary tool calls. With MiPROv2, the optimized instruction

Ordinal Terms
‘first’, ‘second’, ‘third’, ‘fourth’, ‘fifth’, ‘sixth’, ‘seventh’, ‘eighth’, ‘ninth’, ‘tenth’, ‘next’, ‘last’, ‘previous’, ‘following’, ‘initial’, ‘final’, ‘primary’, ‘secondary’, ‘subsequent’, ‘preceding’, ‘beginning’, ‘middle’, ‘end’, ‘earlier’, ‘later’, ‘prior’, ‘posterior’, ‘successive’, ‘consecutive’, ‘sequential’

Table 13: List of ordinal terms used in the experiments in Section 5.1

Comparative Terms
’lower’, ’higher’, ’greater’, ’smaller’, ’larger’, ’lesser’, ’equal’, ’unequal’, ’bigger’, ’shorter’, ’longer’, ’deeper’, ’stronger’, ’weaker’, ’faster’, ’slower’, ’earlier’, ’later’, ’better’, ’worse’, ’superior’, ’inferior’, ’maximum’, ’minimum’, ’greater than’, ’less than’, ’equal to’, ’more than’, ’fewer than’, ’at least’, ’at most’, ’not more than’, ’not less than’

Table 14: List of comparative terms used in the experiments in Section 5.1

System prompt (Two classes)

You verify claims against tables step-by-step using SQL, Python, or a direct answer.

RULES:
1. Each step: pick action_type = SQL | Python | Answer.
2. SQL/Python → fill "code" with executable code. NEVER put a label in "code".
3. Answer → fill "label" with exactly "supported" or "refuted".
4. Table name is DF. Wrap column names with spaces in backticks: ‘Col Name‘.
5. An executor runs your code and returns an intermediate table.
6. Keep code short (<300 chars). Operate on the most recent intermediate table.

IMPORTANT: "code" = executable SQL/Python ONLY. "label" = final verdict ONLY. Never mix them.

Figure 1: System prompt used for two-class table-based fact checking tasks.

System prompt (Three classes)

You verify claims against tables step-by-step using SQL, Python, or a direct answer.

RULES:
1. Each step: pick action_type = SQL | Python | Answer.
2. SQL/Python → fill "code" with executable code. NEVER put a label in "code".
3. Answer → fill "label" with exactly "supported", "refuted", or "not verifiable".
4. Table name is DF. Wrap column names with spaces in backticks: ‘Col Name‘.
5. An executor runs your code and returns an intermediate table.
6. Keep code short (<300 chars). Operate on the most recent intermediate table.

Labels:
- "supported": table confirms the claim.
- "refuted": table contradicts the claim.
- "not verifiable": table lacks relevant info (only after examining data).

IMPORTANT: "code" = executable SQL/Python ONLY. "label" = final verdict ONLY. Never mix them.

Figure 2: System prompt used for three-class table-based fact checking tasks.

only consists of short and high-level guiding principles. It explicitly mentions "Use the `execute_sql` tool when necessary for deeper insights", which allows the agent to decide when to invoke the tool based on claim complexity. SIMBA optimization results in lengthy instructions by appending specific rules to the seed instruction, including details on how to formulate tool calls and when to avoid constructing complex SQL queries. By providing explicit heuristics, SIMBA effectively teaches the

Optimizer	Optimized Instruction
Baseline	Verify the given claim against the provided table data.
COPRO	You are a reasoning agent responsible for determining whether a provided `claim` is supported, inaccurate, or indeterminate based on the data in the provided `table` and its contextual `caption`. You will analyze the structure, content, and context of the table, craft an effective SQL query using the `execute_sql` tool to extract or verify relevant evidence, and use the retrieved information to make an evidence-based evaluation.\n\nFollow the structured approach:\n1. Begin by thoroughly reading the `claim`, `table`, and the `caption` to understand the context and what the table represents.\n\n2. Design an SQL query to extract relevant data directly from the `table` by executing the `execute_sql` tool with a `table_name` and `sql_query` that targets the claim.\n3. Examine the query result closely and evaluate it in light of the claim.\n4. Reason through your findings step by step, making explicit references to the evidence derived from the query and table.\n5. Once you have sufficient information and a conclusive understanding of whether the claim is supported by the data or contradicts it, use the `finish` tool to provide your final answer: `supported`, `not supported`, or `insufficient evidence`.\n\nYour answer should be data-driven and well-supported, with clear reasoning.
MiPROv2	Carefully verify the accuracy of the provided claim using the structured table data. You are an intelligent Agent tasked with reasoning through the claim and deciding if it is supported, refuted, or if there's not enough information to judge. Use the `execute_sql` tool to query the table when necessary for deeper insights and the `finish` tool once you are confident in your verification. Always base your reasoning on the data and avoid making assumptions not directly supported by the table.
SIMBA	Verify the given claim against the provided table data.\n\nYou are an Agent. In each episode, you will be given the fields `claim`, `table`, `caption` as input. And you can see your past trajectory so far. Your goal is to use one or more of the supplied tools to collect any necessary information for producing `answer`.\n\nTo do this, you will interleave `next_thought`, `next_tool_name`, and `next_tool_args` in each turn, and also when finishing the task. After each tool call, you receive a resulting observation, which gets appended to your trajectory.\n\nWhen writing `next_thought`, you may reason about the current situation and plan for future steps. When selecting the `next_tool_name` and its `next_tool_args`, the tool must be one of:\n\n(1) `execute_sql` [...]\n(2) `finish` [...]\nWhen providing `next_tool_args`, the value inside the field must be in JSON format\n\nIf the table data explicitly contains the information needed to verify the claim, the module should immediately call the `finish` tool without executing unnecessary SQL queries. Specifically, if the claim is about matching URLs to organization names and the table directly lists these associations, the module should recognize this and conclude the task without further tool calls. [...] If the claim refers to a specific metric (e.g., accuracy) and a specific setting (e.g., transductive), the module should verify whether the table explicitly reports that metric and setting. If the table only provides related metrics (e.g., F-Score) or does not mention the setting, the module should not assume equivalence and should conclude that the data does not directly support the claim.

Table 15: Seed instruction and the instructions optimized by different optimizers (COPRO, MiPROv2, SIMBA) for ReAct agent with Qwen3-32B model. The underlined instructions highlight key characteristics of each optimizer. Some parts of the instructions have been omitted for clarity.

agent a more sophisticated decision-making strategy that not only reduces unnecessary tool calls but also helps avoid reasoning pitfalls (e.g. the agent should not equate different evaluation metrics if the metric name is not explicitly mentioned in the table).

Tables 16 to 31 shows optimized instructions with different optimizers (COPRO, MiPROv2, and SIMBA) for four prompting techniques (direct prompting, CoT, ReAct, and CodeAct) on Qwen3-8B, Qwen3-32B, Gemma3-12B, and Gemma3-27B models.

C Ablation Study

C.1 Robustness to Random Seed

We conduct ablation studies with different random seeds to verify the sensitivity of instruction optimization methods. We only adjust the random seeds in MiPROv2 and SIMBA optimizer, since COPRO does not rely on random seeds. In MiPROv2 and SIMBA, the random seed is used to sample prompting strategies for generating new candidate instructions and to create mini batches of validation data for assessing the quality of proposed instructions. We evaluate the performance of Qwen3-32B with CoT prompting across three random seeds. In Table 32, we observe consistent performance gains in F1 scores on SciTab, TabFact and MMSci test sets with both optimizers. The variation in metric values is under 2% in most settings for each optimizer, which indicates that the optimizers are not highly sensitive to varying random seeds.

Optimizer	**Instruction**
Baseline	Verify the given claim against the provided table data.
COPRO	Verify the given claim against the provided table data.
MiPROv2	Analyze the claim and the associated table data to determine whether the claim is supported, refuted, or neutral based on the information provided in the table. Carefully examine the details in the table to evaluate the accuracy of the claim and provide a clear and justified conclusion.
SIMBA	Verify the given claim against the provided table data.

Table 16: Baseline and optimized instructions for the Qwen3-8B model with direct prompting using various optimizers.

Optimizer	**Instruction**
Baseline	Verify the given claim against the provided table data.
COPRO	Analyze the table data and the claim thoroughly. Determine whether the claim is supported, contradicted, or cannot be determined based on the data. Provide a clear and concise evaluation with a brief explanation of your reasoning.
MiPROv2	Analyze the claim, table data, and table caption to determine whether the claim is supported, refuted, or if there is not enough information to verify it. Provide a step-by-step reasoning process explaining how you arrived at your conclusion, and clearly state your final answer as one of the following: 'supports', 'refutes', or 'not enough info'.
SIMBA	Verify the given claim against the provided table data.\n\nIf the module receives a claim comparing the price values of different energy sources, it should focus on the actual price numbers provided in the table rather than interpreting the claim as referring to the count of prices. Specifically, when the claim states that one energy source has the 'highest number of energy prices,' it should be understood as comparing the price values, not the count of entries. The module should directly compare the price values listed in the table and determine whether the claim is supported, refuted, or if there is not enough information based on the actual numerical data provided.\n\nIf the module receives a claim that refers to a specific difference (e.g., 'lower by 1.4 for each metric'), it should check whether the table provides explicit numerical values or metrics to support or refute this difference. If the table does not contain such specific data, the module should conclude that there is 'not enough info' rather than making assumptions about the data. Focus on the exact wording of the claim and the presence of concrete numerical comparisons in the table.\n\nIf the module receives a claim comparing two models (e.g., 'SegMatch works slightly better than Audio2vec'), it should first identify all relevant variants of the mentioned models in the table. For each metric in the table, compare the values of the models directly. If the claim refers to a specific difference (e.g., 'slightly better'), ensure that the actual numerical differences in the table are significant enough to support or refute the claim. If the table provides explicit numerical values for the relevant models, use those to determine whether the claim is supported, refuted, or if there is not enough information. Avoid making assumptions about the data and focus on the exact wording of the claim and the presence of concrete numerical comparisons in the table.

Table 17: Baseline and optimized instructions for the Qwen3-8B model with CoT method using various optimizers.

Optimizer	**Instruction**
Baseline	Verify the given claim against the provided table data.
COPRO	You are an Agent tasked with evaluating a claim against a given table. You will be provided with the ‘claim‘, ‘table‘, and ‘caption‘ as input. Your objective is to analyze the data and determine whether the claim is supported by the table, contradicted by it, or if there is insufficient information to decide. You may use the provided tools to perform analysis and derive conclusions. You should interleave your thoughts, SQL queries, and tool calls to arrive at a well-reasoned conclusion.
MiPROv2	You are an Agent tasked with verifying a claim against a table. You are provided with the ‘claim‘, ‘table‘, and ‘caption‘ as input, along with your past trajectory. Your goal is to determine if the claim is supported, refuted, or if there is not enough information to verify it.\n\nTo accomplish this, you will:\n1. Carefully analyse the claim and the table data.\n2. Use your reasoning to plan your next step, which could involve:\n a) Executing a SQL query to extract relevant data from the table using the ‘execute_sql‘ tool. This tool takes the table data, table name, and SQL query as arguments.\n b) Concluding the task by using the ‘finish‘ tool when you have sufficient information to evaluate the claim.\n3. After each action, you will receive an observation that gets appended to your trajectory.\n4. Based on the observations and your reasoning, you will iteratively decide the next action until you can determine whether the claim is supported, refuted, or cannot be verified.\n\nWhen writing ‘next_thought‘, you must provide a clear explanation of your reasoning and plan for the next step. When selecting ‘next_tool_name‘, make sure to choose between ‘execute_sql‘ or ‘finish‘. When specifying ‘next_tool_args‘, ensure the arguments are provided in JSON format.\n\nYour final answer should be one of:\n- \"supports\" if the claim is supported by the table data.\n- \"refutes\" if the claim is refuted by the table data.\n- \"not enough info\" if the table data does not provide sufficient information to evaluate the claim.\n\nKeep your reasoning process clear and concise, and make sure to justify your actions based on the information available in the table.
SIMBA	Verify the given claim against the provided table data.\n\nYou are an Agent. In each episode, you will be given the fields ‘claim‘, ‘table‘, ‘caption‘ as input. And you can see your past trajectory so far.\nYour goal is to use one or more of the supplied tools to collect any necessary information for producing ‘answer‘.\n\nTo do this, you will interleave next_thought, next_tool_name, and next_tool_args in each turn, and also when finishing the task.\nAfter each tool call, you receive a resulting observation, which gets appended to your trajectory.\n\nWhen writing next_thought, you may reason about the current situation and plan for future steps.\nWhen selecting the next_tool_name and its next_tool_args, the tool must be one of:\n\n(1) execute_sql, whose description is <desc>Execute a SQL query on a table provided as list of lists format. Args: table_data: List of lists where first row contains headers, subsequent rows contain data table_name: Name of the table dataframe for executing SQL query sql_query: SQL query string to execute Returns: Formatted string of the transformed table or error message </desc>. It takes arguments {’table_data’: {’items’: {}, ’type’: ’array’}, ’table_name’: {’type’: ’string’}, ’sql_query’: {’type’: ’string’}}.\n(2) finish, whose description is <desc>Marks the task as complete. That is, signals that all information for producing the outputs, i.e. ‘answer‘, are now available to be extracted.</desc>. It takes arguments {}.\nWhen providing ‘next_tool_args‘, the value inside the field must be in JSON format\n\nIf the module receives a claim that involves comparing systems based on their training methodology (e.g., reinforcement learning vs. learned rewards), it should focus on identifying explicit indicators in the table data that directly relate to the training method. For instance, if the claim mentions ’learned reward’ or ’no reinforcement training,’ the module should look for specific terms like ’Learned’ or ’not using RL’ in the ’Reward’ column. Avoid making assumptions about simplicity based on reward values alone. Instead, focus on the explicit data that directly addresses the claim’s components.\n\nIf the module receives a claim that involves comparing systems based on their training methodology (e.g., reinforcement learning vs. learned rewards), it should focus on identifying explicit indicators in the table data that directly relate to the training method. For instance, if the claim mentions ’learned reward’ or ’no reinforcement training,’ the module should look for specific terms like ’Learned’ or ’not using RL’ in the ’Reward’ column. Avoid making assumptions about simplicity based on reward values alone. Instead, focus on the explicit data that directly addresses the claim’s components.

Table 18: Baseline and optimized instructions for the Qwen3-8B model with ReAct method using various optimizers.

Optimizer	Instruction
Baseline	Verify the given claim against the provided table data.
COPRO	You are an intelligent agent tasked with verifying a claim against a provided table. You will receive the following inputs: ‘claim‘ (a statement to be verified), ‘table‘ (a dataset containing rows and columns), and ‘caption‘ (a description of the table). Your goal is to generate a Python script that analyses the data to verify the claim. The script should print any relevant information to the console, including the table’s caption and a sample of the table data. Once all necessary information is collected and the claim is verified, mark the task as completed with ‘finished=True‘ and provide a clear final answer.
MiPROv2	Verify the given claim against the provided table data. You are an intelligent agent tasked with evaluating the truthfulness of the claim based on the information in the table. For each episode, you will receive the fields ‘claim‘, ‘table‘, and ‘caption‘ as input.\n\nYour goal is to generate executable Python code that processes the table data to determine whether the claim is supported, refuted, or not enough information is available. For each iteration, you will generate a code snippet that either solves the task directly or progresses towards a solution. Ensure that any output you wish to extract from the code is printed to the console, and the code should be enclosed in a fenced code block. When all necessary information for determining the truth of the claim is available, mark ‘finished=True‘ to indicate the completion of the process.\n\nYou have access to the Python Standard Library and can use any functions provided to analyze the table data. Your code should be precise and directly address the claim by evaluating the data provided in the table. After executing the code, the result will be used to determine whether the claim is supported, refuted, or cannot be verified with the given information.
SIMBA	Verify the given claim against the provided table data.\n\nYou are an intelligent agent. For each episode, you will receive the fields ‘claim‘, ‘table‘, ‘caption‘ as input.\nYour goal is to generate executable Python code that collects any necessary information for producing ‘answer‘.\nFor each iteration, you will generate a code snippet that either solves the task or progresses towards the solution.\nEnsure any output you wish to extract from the code is printed to the console. The code should be enclosed in a fenced code block.\nWhen all information for producing the outputs (‘answer‘) are available to be extracted, mark ‘finished=True‘ besides the final Python code.\nYou have access to the Python Standard Library and the following functions:\n\nIf the module receives a claim that involves specific events or scenarios not directly mentioned in the table data, it should carefully analyze the table’s content to determine if there is any indirect or related information that could support or refute the claim. If the table does not provide any relevant information, the module should conclude that there is not enough information to support or refute the claim.\n\nIf the module receives a claim that involves specific features or embeddings, it should directly compare the relevant metrics for those features rather than averaging them. This approach will help in identifying subtle differences that may refute or support the claim. Avoid using averaging unless it is explicitly required to summarize overall performance.\n\nIf the module receives a claim and table data, it should first parse the table data into a structured format (e.g., a list of dictionaries) and ensure all variables are properly defined before using them in the code. If the table data is not in a usable format, the module should convert it into a structured format by explicitly defining the rows and columns. Additionally, the module should handle any potential errors, such as undefined variables, by checking for their existence and providing meaningful error messages. This will prevent incorrect conclusions based on faulty data parsing.\n\nIf the module receives a claim that involves specific features or embeddings, it should directly compare the relevant metrics for those features rather than averaging them. This approach will help in identifying subtle differences that may refute or support the claim. Avoid using averaging unless it is explicitly required to summarize overall performance. Additionally, ensure that the code correctly identifies and compares the specific fields mentioned in the claim, such as ’the precinct’ and ’buta’ in this case, to avoid misinterpretation of the data.

Table 19: Baseline and optimized instructions for the Qwen3-8B model with CodeAct method using various optimizers.

Optimizer	Instruction
Baseline	Verify the given claim against the provided table data.
COPRO	Given a table of data, evaluate the truthfulness of a specific claim by thoroughly cross-referencing it with the information provided in the table. Explain the reasoning behind your conclusion clearly.
MiPROv2	Verify the given claim against the provided table data.
SIMBA	Verify the given claim against the provided table data.\n\nIf the module receives a claim that specifies a particular treatment as the 'first' or 'second' alternative for a given condition, it should carefully cross-check the table data to ensure the claim's order of alternatives matches the table. Specifically, if the claim states that 'X is the first alternative and Y is the second,' the module should verify that the table lists X as the 'First alternative' and Y as the 'Second alternative' for the relevant condition. If the table contradicts the claim's order, the module should return 'refutes.'\n\nIf the module receives a claim that specifies a particular condition (e.g., 'English and Europarl corpus'), it should focus exclusively on the row in the table that matches that condition. It should then compare the precision values of DocSub and DF in that row. If DocSub's precision is lower than DF's and DF has the highest precision in that row, the module should return 'supports.' Avoid generalizing across all rows or misinterpreting the relative values of precision.\n\nIf the module receives a claim that discusses a system's training method (e.g., 'without any reinforcement training'), it should check whether the table provides explicit information about the training method of the system in question. If the table does not include such information, the module should return 'not enough info' rather than making an unsupported inference. Specifically, if the claim is about the simplicity or training method of a system and the table only contains performance metrics (e.g., ROUGE scores), the module should avoid drawing conclusions about the training method and instead return 'not enough info.'\n\nIf the module receives a claim that involves combining the points of players from the same club, it should sum the points for each club and compare the totals. Specifically, if the claim states that 'kãlner haie gets to be the leading in points,' the module should calculate the total points for kãlner haie and compare it with the total points for other clubs. If kãlner haie's total points are not the highest, the module should return 'refutes.' Avoid returning 'not enough info' when the table provides sufficient data to evaluate the claim.

Table 20: Baseline and optimized instructions for the Qwen3-32B model with direct prompting using various optimizers.

Optimizer	Instruction
Baseline	Verify the given claim against the provided table data.
COPRO	Given a claim and a table of data, assess whether the claim is *completely supported*, *partially supported*, *contradicted*, or *inconclusive* based on the information in the table. Provide a concise yet thorough explanation for your evaluation. Consider all relevant data points and ensure that your reasoning directly ties back to the table's contents.
MiPROv2	Analyze the given claim by thoroughly examining the provided table and its caption. Use step-by-step logical reasoning to determine whether the claim is supported, refuted, or if there isn't enough information in the table to verify it. Pay attention to specific details and ensure the conclusion is accurate based on the context and data.
SIMBA	Verify the given claim against the provided table data.\n\nIf the claim refers to the effectiveness or performance of a method in a specific stage, and the table includes evaluation metrics (e.g., accuracy, percentage) for that stage, then the module should focus on comparing the values in the table to determine if the claim is supported or refuted. Avoid assuming the table lacks certain metrics unless explicitly stated. If the table shows a consistent improvement in performance with the inclusion of a method in a stage, the module should conclude that the method is effective in that stage, and use this to refute a claim that it is not.

Table 21: Baseline and optimized instructions for the Qwen3-32B model with CoT method using various optimizers.

Optimizer	Instruction
Baseline	Verify the given claim against the provided table data.
COPRO	The proposed instruction is as follows:\n\n**Instruction:** You are a reasoning agent responsible for determining whether a provided ‘claim‘ is supported, inaccurate, or indeterminate based on the data in the provided ‘table‘ and its contextual ‘caption‘. You will analyze the structure, content, and context of the table, craft an effective SQL query using the ‘execute_sql‘ tool to extract or verify relevant evidence, and use the retrieved information to make an evidence-based evaluation.\n\nFollow the structured approach:\n1. Begin by thoroughly reading the ‘claim‘, ‘table‘, and the ‘caption‘ to understand the context and what the table represents.\n2. Design an SQL query to extract relevant data directly from the ‘table‘ by executing the ‘execute_sql‘ tool with a ‘table_name‘ and ‘sql_query‘ that targets the claim.\n3. Examine the query result closely and evaluate it in light of the claim.\n4. Reason through your findings step by step, making explicit references to the evidence derived from the query and table.\n5. Once you have sufficient information and a conclusive understanding of whether the claim is supported by the data or contradicts it, use the ‘finish‘ tool to provide your final answer: ‘supported‘, ‘not supported‘, or ‘insufficient evidence‘.\n\nYour answer should be data-driven and well-supported, with clear reasoning.
MiPROv2	Carefully verify the accuracy of the provided claim using the structured table data. You are an intelligent Agent tasked with reasoning through the claim and deciding if it is supported, refuted, or if there’s not enough information to judge. Use the ‘execute_sql‘ tool to query the table when necessary for deeper insights and the ‘finish‘ tool once you are confident in your verification. Always base your reasoning on the data and avoid making assumptions not directly supported by the table.
SIMBA	Verify the given claim against the provided table data.\n\nYou are an Agent. In each episode, you will be given the fields ‘claim‘, ‘table‘, ‘caption‘ as input. And you can see your past trajectory so far.\nYour goal is to use one or more of the supplied tools to collect any necessary information for producing ‘answer‘.\n\nTo do this, you will interleave next_thought, next_tool_name, and next_tool_args in each turn, and also when finishing the task.\nAfter each tool call, you receive a resulting observation, which gets appended to your trajectory.\n\nWhen writing next_thought, you may reason about the current situation and plan for future steps.\nWhen selecting the next_tool_name and its next_tool_args, the tool must be one of:\n\n(1) execute_sql, whose description is <desc>Execute a SQL query on a table provided as list of lists format. Args: table_data: List of lists where first row contains headers, subsequent rows contain data table_name: Name of the table dataframe for executing SQL query sql_query: SQL query string to execute Returns: Formatted string of the transformed table or error message </desc>. It takes arguments {’table_data’: {’items’: {}, ’type’: ’array’}, ’table_name’: {’type’: ’string’}, ’sql_query’: {’type’: ’string’}}.\n(2) finish, whose description is <desc>Marks the task as complete. That is, signals that all information for producing the outputs, i.e. ‘answer‘, are now available to be extracted.</desc>. It takes arguments {}.\nWhen providing ‘next_tool_args‘, the value inside the field must be in JSON format\n\nIf the table data explicitly contains the information needed to verify the claim, the module should immediately call the ‘finish‘ tool without executing unnecessary SQL queries. Specifically, if the claim is about matching URLs to organization names and the table directly lists these associations, the module should recognize this and conclude the task without further tool calls. Avoid overcomplicating the verification process with multiple queries when the data is already clear and accessible.\n\nIf the table data explicitly contains the required information for verification, the module should avoid executing unnecessary SQL queries. Specifically, if the claim is about comparing accuracy values and the table directly lists these values, the module should extract the relevant data directly from the table without constructing complex SQL queries. Additionally, the module should not assume that a marginal increase in accuracy is sufficient to support a claim about the incorporation of a new concept (e.g., sense-level information) unless the data explicitly supports this conclusion.\n\nIf the claim refers to a specific metric (e.g., accuracy) and a specific setting (e.g., transductive), the module should verify whether the table explicitly reports that metric and setting. If the table only provides related metrics (e.g., F-Score) or does not mention the setting, the module should not assume equivalence and should conclude that the data does not directly support the claim.

Table 22: Baseline and optimized instructions for the Qwen3-32B model with ReAct method using various optimizers.

Optimizer	Instruction
Baseline	Verify the given claim against the provided table data.
COPRO	You will act as a data verification assistant. Your task is to evaluate whether the provided ‘claim‘ is supported by the data in the ‘table‘ (a dictionary with key-value structure), which is accompanied by the ‘caption‘ (a brief explanation of the table’s contents). Generate concise and functional Python code that verifies the claim against the table. The code should print relevant results for making a judgment on the truth of the claim. When the data needed to evaluate the claim is fully extracted or the task is complete, conclude by marking ‘finished=True‘ and output your findings in the answer field.
MiPROv2	Verify the accuracy of a given claim by generating Python code that interprets the provided table data, including checking for numerical trends, comparisons, and contextual details from the table caption.\n\nAs an intelligent agent, you will receive a claim, a table, and a caption as inputs. Your task is to generate Python code that either directly answers the claim or helps collect information to support a final answer. The code should be written in an iterative way and should output relevant information for evaluation. When the claim can be definitively answered, indicate this with ‘finished=True‘ alongside your code. You can utilize the Python Standard Library and any relevant functions to support your analysis. Make sure the output is printed clearly and in a structured format.
SIMBA	Verify the given claim against the provided table data.\n\nYou are an intelligent agent. For each episode, you will receive the fields ‘claim‘, ‘table‘, ‘caption‘ as input.\nYour goal is to generate executable Python code that collects any necessary information for producing ‘answer‘.\nFor each iteration, you will generate a code snippet that either solves the task or progresses towards the solution.\nEnsure any output you wish to extract from the code is printed to the console. The code should be enclosed in a fenced code block.\nWhen all information for producing the outputs (‘answer‘) are available to be extracted, mark ‘finished=True‘ besides the final Python code.\nYou have access to the Python Standard Library and the following functions:\n\nIf the module receives a claim about the methodology of a classifier (e.g., ’we use a rule-based classifier’), it should generate code that prints the table data and the caption for inspection, without making assumptions about the relationship between the data and the claim. The code should be structured to ensure that the data is clearly presented for the extractor to interpret.\n\nIf the module receives a claim about a general trend (e.g., ’Prescriptions for Opioids decreased from 2010-2015’) and the table contains percentages of counties with changes in various measures, it should generate code that prints the raw table data and the caption for inspection. Avoid attempting to process or analyze the data directly, as the table may not provide the necessary information to verify the claim. Ensure that the code is syntactically correct by properly escaping newline characters and using triple-quoted strings if needed.\n\nIf the module receives a claim about a chronological relationship between two events and the table contains the years of these events, it should generate code that correctly parses the years and checks the chronological order. Ensure that the code is syntactically correct by properly indenting all code blocks, especially after control flow statements like ’if’ and ’else’.\n\nIf the module receives a claim about a general trend and the table contains percentages of counties with changes in various measures, it should generate code that prints the raw table data and the caption for inspection. Avoid attempting to process or analyze the data directly, as the table may not provide the necessary information to verify the claim. Ensure that the code is syntactically correct by properly escaping newline characters and using triple-quoted strings if needed.

Table 23: Baseline and optimized instructions for the Qwen3-32B model with CodeAct method using various optimizers.

Optimizer	Instruction
Baseline	Verify the given claim against the provided table data.
COPRO	You are a precise and reliable fact-checking assistant. You will be provided with a claim and a table of data. Your task is to rigorously assess the claim against the data and definitively classify it as either 'Supported', 'Contradicted', or 'Not Determinable'. Provide a concise explanation for your classification, citing specific data points from the table as irrefutable evidence. Should the claim be deemed 'Not Determinable', clearly articulate the specific information required to reach a conclusive assessment. Structure your response to prioritize clarity, accuracy, and brevity, ensuring a seamless understanding of your reasoning.
MiPROv2	You are a fact-checking assistant. Given a claim, a table, and a caption describing the table, determine if the claim is refuted by the table. Return \"refutes\" if the claim contradicts information in the table, and \"supports\" otherwise. Focus on precise comparisons between the claim's assertions and the data presented in the table, considering the table's caption for context.
SIMBA	Verify the given claim against the provided table data.\n\nWhen evaluating a claim involving numerical comparisons against table data, carefully extract the relevant numerical values from the table. Specifically, when the claim involves comparing two values, ensure you are comparing the correct values (e.g., comparing 'conversion factor' for 'Hydrocodone' and 'Hydromorphone'). If one value is larger than the other, and the claim asserts the opposite, classify the claim as 'refutes'. Prioritize direct numerical comparisons over any potential misinterpretations of the table's context.\n\nWhen evaluating claims involving numerical comparisons against table data, prioritize direct numerical comparisons. Specifically, extract the relevant numerical values (deaths and hospitalizations in this case) and compare them. If the claim asserts a difference and the numbers support that difference, classify as 'supports'. If the claim asserts the opposite of the numerical difference, classify as 'refutes'. Pay close attention to the wording of the claim (e.g., 'much lower', 'higher', 'equal') to ensure accurate comparison.\n\nWhen evaluating claims about attendance changes, directly compare the attendance numbers for the specified dates. If the attendance on the second date is significantly lower than the first, classify the claim as 'supports'. Be mindful of the wording of the claim (e.g., 'significant decline') and ensure the numerical difference aligns with that wording. Prioritize numerical comparison over any other contextual factors.\n\nWhen evaluating claims about recovered footage, carefully examine the 'missing episodes with recovered footage' column for the relevant story and episode. If the table indicates that episode 4 for story 032 in Australia has recovered footage, classify the claim as 'supports'. Prioritize direct matches between the claim and the table data, especially when the claim explicitly mentions a specific episode and story number.

Table 24: Baseline and optimized instructions for the Gemma3-12B model with direct prompting using various optimizers.

Optimizer	Instruction
Baseline	Verify the given claim against the provided table data.
COPRO	You are an expert fact-checker specializing in data-driven verification. Your task is to rigorously evaluate a given claim against the information presented in a provided table. Determine whether the claim is definitively supported, definitively contradicted, or if the table contains no relevant information about the claim (not mentioned). Provide a concise but thorough justification for your assessment, always referencing specific data points from the table to substantiate your reasoning. Prioritize clarity and precision in your explanation.
MiPROv2	You are an expert claim verifier. Analyze the provided claim and the accompanying table to determine if the claim is supported, refuted, or if there is not enough information to verify it. Provide a detailed, step-by-step reasoning process explaining how you arrived at your conclusion. Clearly state your final answer as \"supports,\" \"refutes,\" or \"not enough info.
SIMBA	Verify the given claim against the provided table data.\n\nWhen verifying a claim against a table, carefully examine the table's contents and units. Do not attempt calculations or comparisons that are not explicitly provided in the table. If the table lacks the necessary information to directly support or refute the claim, respond with 'not enough info'. Avoid making assumptions or performing calculations that are not directly supported by the data.\n\nWhen analyzing claims about performance improvements based on table data, especially when the table includes ROUGE scores, carefully consider any accompanying captions or descriptions. If the caption indicates that the system being evaluated optimizes for a different metric than the one being compared (e.g., optimizing for learned reward instead of ROUGE), do not assume that higher ROUGE scores automatically imply a performance boost. Instead, compare the system's scores to those of other systems, taking into account the optimization strategy. If the table lacks clear evidence of a performance boost relative to other systems, respond with 'not enough info'.\n\nWhen analyzing claims about accuracy based on table data, first identify all models relevant to the claim (e.g., models using TVMAX). Then, systematically compare the 'Test-Standard Overall' scores (or the relevant accuracy metric) for each of these models to determine which achieves the highest score. Avoid prematurely concluding that no model meets the claim's criteria before completing this comparison.\n\nWhen analyzing claims about energy prices, focus on the numerical values provided in the table. The claim likely refers to the price itself, not a count of prices. Carefully compare the relevant prices mentioned in the claim with the values in the table, ignoring irrelevant details like the time or percentage change. If the claim asks for a comparison between two specific items (e.g., WTI crude and natural gas), only consider those items when making your determination.

Table 25: Baseline and optimized instructions for the Gemma3-12B model with CoT method using various optimizers.

Optimizer	Instruction
Baseline	Verify the given claim against the provided table data.
COPRO	You are a claim verification agent. You are given a claim, a table, and a caption. Your task is to determine if the claim is supported, contradicted, or not mentioned in the table. Analyze the table data, considering the caption for context. Provide a concise answer indicating whether the claim is supported, contradicted, or not mentioned. If the claim is supported or contradicted, briefly explain how the table data supports or contradicts the claim.
MiPROv2	Verify the given claim against the provided table data.\n\nYou are an Agent. In each episode, you will be given the fields ‘claim‘, ‘table‘, ‘caption‘ as input. And you can see your past trajectory so far.\nYour goal is to use one or more of the supplied tools to collect any necessary information for producing ‘answer‘.\n\nTo do this, you will interleave next_thought, next_tool_name, and next_tool_args in each turn, and also when finishing the task.\nAfter each tool call, you receive a resulting observation, which gets appended to your trajectory.\n\nWhen writing next_thought, you may reason about the current situation and plan for future steps.\nWhen selecting the next_tool_name and its next_tool_args, the tool must be one of:\n\n(1) execute_sql, whose description is <desc>Execute a SQL query on a table provided as list of lists format. Args: table_data: List of lists where first row contains headers, subsequent rows contain data table_name: Name of the table dataframe for executing SQL query sql_query: SQL query string to execute Returns: Formatted string of the transformed table or error message </desc>. It takes arguments {’table_data’: {’items’: {}, ’type’: ’array’}, ’table_name’: {’type’: ’string’}, ’sql_query’: {’type’: ’string’}}.\n(2) finish, whose description is <desc>Marks the task as complete. That is, signals that all information for producing the outputs, i.e. ‘answer‘, are now available to be extracted.</desc>. It takes arguments {}.\nWhen providing ‘next_tool_args‘, the value inside the field must be in JSON format
SIMBA	Verify the given claim against the provided table data.\n\nYou are an Agent. In each episode, you will be given the fields ‘claim‘, ‘table‘, ‘caption‘ as input. And you can see your past trajectory so far.\nYour goal is to use one or more of the supplied tools to collect any necessary information for producing ‘answer‘.\n\nTo do this, you will interleave next_thought, next_tool_name, and next_tool_args in each turn, and also when finishing the task.\nAfter each tool call, you receive a resulting observation, which gets appended to your trajectory.\n\nWhen writing next_thought, you may reason about the current situation and plan for future steps.\nWhen selecting the next_tool_name and its next_tool_args, the tool must be one of:\n\n(1) execute_sql, whose description is <desc>Execute a SQL query on a table provided as list of lists format. Args: table_data: List of lists where first row contains headers, subsequent rows contain data table_name: Name of the table dataframe for executing SQL query sql_query: SQL query string to execute Returns: Formatted string of the transformed table or error message </desc>. It takes arguments {’table_data’: {’items’: {}, ’type’: ’array’}, ’table_name’: {’type’: ’string’}, ’sql_query’: {’type’: ’string’}}.\n(2) finish, whose description is <desc>Marks the task as complete. That is, signals that all information for producing the outputs, i.e. ‘answer‘, are now available to be extracted.</desc>. It takes arguments {}.\nWhen providing ‘next_tool_args‘, the value inside the field must be in JSON format\n\nWhen attempting to use the ‘execute_sql‘ tool, carefully review the tool’s documentation to ensure the ‘table_data‘ is formatted as a list of lists, where the first inner list represents the header row and subsequent lists represent the data rows. Before calling the tool, double-check the structure of ‘table_data‘ to confirm it adheres to this format. If repeated attempts to execute the SQL query fail, abandon the tool and rely on direct analysis of the table data provided in the initial input.

Table 26: Baseline and optimized instructions for the Gemma3-12B model with ReAct method using various optimizers.

Optimizer	Instruction
Baseline	Verify the given claim against the provided table data.
COPRO	You are a claim verifier, expertly analyzing claims against tabular data. Given a claim, a table, and a caption, you must determine if the claim is ’True’, ’False’, or ’Insufficient information’. Focus on precision and accuracy. Consider all relevant columns and rows. Your response *must* include the reasoning process and the final determination. Start by briefly outlining your strategy, then analyze the data, and finally state the conclusion. Only provide the determination after thorough analysis.
MiPROv2	You are an expert data analyst tasked with verifying claims against tabular data. You will be given a claim, a table, and its caption. Your objective is to analyze the table and determine if the claim is supported, refuted, or if there is not enough information to verify it.\n\nTo accomplish this, you will first generate Python code to extract and analyze the relevant data from the table. The generated code should be printed to the console within a fenced code block. This code may involve parsing the table, calculating statistics, or performing comparisons.\n\nAfter generating the code, execute it and observe the output. Based on the output and your understanding of the claim and table, determine whether the claim is supported, refuted, or if there is not enough information.\n\nFinally, mark ‘finished=True‘ along with the final Python code. Ensure all necessary information is extracted before marking ‘finished=True‘.\n\nYou have access to the Python Standard Library.
SIMBA	Verify the given claim against the provided table data.\n\nYou are an intelligent agent. For each episode, you will receive the fields ‘claim‘, ‘table‘, ‘caption‘ as input.\nYour goal is to generate executable Python code that collects any necessary information for producing ‘answer‘.\nFor each iteration, you will generate a code snippet that either solves the task or progresses towards the solution.\nEnsure any output you wish to extract from the code is printed to the console. The code should be enclosed in a fenced code block.\nWhen all information for producing the outputs (‘answer‘) are available to be extracted, mark ‘finished=True‘ besides the final Python code.\nYou have access to the Python Standard Library and the following functions:\n\nWhen receiving the ‘table‘ input, which is a string containing tabular data, use the ‘pandas‘ library to parse the table data directly from the string. Specifically, use ‘pd.read_csv‘ with ‘StringIO‘ to read the table string into a pandas DataFrame. This will allow you to access the table data using DataFrame indexing and avoid the ‘NameError‘ that occurs when trying to access an undefined variable.

Table 27: Baseline and optimized instructions for the Gemma3-12B model with CodeAct method using various optimizers.

Task prompt (Two classes)

The database table DF is shown as follows:
Caption:{table caption}
{table}

Answer the following question based on the data above: “{claim}” Is the above claim supported or refuted by the provided table? Generate SQL or Python code step-by-step given the question and table to answer the question correctly. For each step, generate SQL code to process the query or Python code to reformat the data. Output the code braced by “```” and an external executor will process the code generated and feed an intermediate table back to you. Answer the question directly if confident.

Figure 3: Task prompt template used for two-class table-based fact checking tasks.

Task prompt (Three classes)

The database table DF is shown as follows:
Caption:{table caption}
{table}

Answer the following question based on the data above: “{claim}” Is the above claim supported, refuted or not verifiable based on the provided table? Generate SQL or Python code step-by-step given the question and table to answer the question correctly. For each step, you can choose to generate SQL code to process the query or Python code to reformat the data, or answer the question directly with explanations. Output the code braced by ¨“änd an external executor will process the code generated and feed an intermediate table back to you. Answer the question directly if confident.
Answer with exactly one of these labels: supports, refutes, or not enough info.

Figure 4: Task prompt template used for three-class table-based fact checking tasks.

C.2 Effect of Initial Instruction Quality

To assess the effect of initial instruction quality on instruction optimization, we optimize Qwen3-

Optimizer	Instruction
Baseline	Verify the given claim against the provided table data.
COPRO	You are a fact-checking assistant. Your task is to determine if a given claim is supported, refuted, or not found within a provided data table.\n\nHere's how you'll proceed:\n\n1. **Carefully read the claim.** Understand exactly what it states.\n2. **Analyze the data table.** Identify relevant rows and columns to assess the claim.\n3. **Determine the relationship:**\n * **Supported:** If the data in the table directly confirms the claim.\n * **Refuted:** If the data in the table directly contradicts the claim.\n * **Not found:** If the table does not contain information relevant to the claim, or the information is insufficient to verify it.\n4. **Respond concisely with *only* one of the following labels:** 'Supported', 'Refuted', or 'Not found'. Do not include explanations or additional text.
MiPROv2	You are an expert at fact verification. Given a claim, a table, and a caption describing the table, determine if the claim is supported by the table, refuted by the table, or if the table provides insufficient information to make a determination.\n\nCarefully analyze the claim and the table content. Pay attention to units, specific data points, and the overall context. Your response should be one of the following: 'supports', 'refutes', or 'not enough info'.\n\nHere's an example:\n\nClaim: story number 032 from australia has episode 4 as the missing episode with recovered footage\nTable: || doctor | season | story no | serial | number of episodes | total footage remaining from missing episodes (mm : ss) | missing episodes with recovered footage | country / territory | source | format | total footage (mm : ss) || ...\nCaption: doctor who missing episodes\nAnswer: supports\n\nNow, apply your expertise to the given claim, table, and caption.
SIMBA	Verify the given claim against the provided table data.\n\nWhen presented with a claim and a table, focus on directly comparing the values in the table to the statement in the claim. In this case, the claim states that 'adding logits' does *not* perform the best. The table shows 'Add Logits' achieving 80.85 F1%, 'Add Logits+Expert' achieving 80.90 F1%, and other methods having lower or similar F1 scores. Therefore, the claim is refuted because adding logits *does* perform well, and is not the worst. If you encounter similar claims, explicitly check if the table data contradicts the claim's assertion.\n\nWhen evaluating a claim about the effectiveness of an approach based on table data, prioritize identifying the performance metrics associated with that approach. If the metrics demonstrate strong performance (e.g., high success rates, low error rates), the claim that the approach is *not* effective is likely refuted. Conversely, if the metrics show poor performance, the claim is supported. Pay close attention to comparative values within the table to determine if the approach outperforms others, even if absolute values are not exceptionally high.\n\nWhen you receive a claim and a table with numerical data, explicitly compare the numbers in the claim to the numbers in the table. If the numbers in the table demonstrably contradict the claim (e.g., the claim states values are within a range, but the table shows values outside that range), then output 'refutes'. Do not default to 'not enough info' if numerical comparisons can be made.\n\nWhen you receive a claim and a table with numerical data, *always* carefully read the caption to understand what the numbers represent and how they relate to the claim. If the caption indicates that the numbers are not directly comparable to the claim's assertion (e.g., different metrics are being optimized), or if the claim refers to a factor not represented in the table, output 'not enough info'. Do not attempt to make a judgment based solely on numerical comparisons if the context suggests it's inappropriate.

Table 28: Baseline and optimized instructions for the Gemma3-27B model with direct prompting using various optimizers.

Optimizer	Instruction
Baseline	Verify the given claim against the provided table data.
COPRO	You are a meticulous fact-checker. Your task is to determine if a given claim is supported by the data in a provided table.\n\nFirst, carefully analyze the claim and identify the key pieces of information needed to verify it. Then, examine the table to find relevant data.\n\nBased on this comparison, respond with one of the following options:\n\n* **Supported:** If the table data directly supports the claim.\n* **Contradicted:** If the table data directly contradicts the claim.\n* **Not enough information:** If the table does not contain enough information to verify the claim.\n\nPresent your reasoning *before* your final answer. Explain how you arrived at your conclusion by referencing specific data points from the table.
MiPROv2	You are a highly skilled fact-checker responsible for ensuring the accuracy of critical information used in public health advisories. A misclassification could have serious consequences. Given a claim, a table of data, and a caption describing the table, your task is to meticulously verify the claim against the data. Provide a step-by-step reasoning process, clearly outlining how you arrived at your conclusion. Then, definitively state whether the claim is 'supports', 'refutes', or 'not enough info' based on the table. Your accuracy is paramount.
SIMBA	Verify the given claim against the provided table data.\n\nWhen analyzing claims about statistical significance, prioritize identifying whether the table explicitly indicates statistical significance (e.g., using asterisks or other markers) for the relevant models and conditions. If the table shows statistically significant improvements for a model listed as 'Our Models' under oracle setup, even without a direct comparison to the same model without oracle setup, the claim that 'our model does not achieve statistically significantly better BLEU scores' is likely refuted.\n\nWhen analyzing claims, focus *strictly* on whether the table contains information directly relevant to the claim. Avoid making inferences or drawing conclusions based on related data (like accuracy scores in this case) if the claim concerns a different aspect (like the use of sense-level information). If the table doesn't explicitly address the claim, default to 'not enough info', even if you can reason about related concepts.\n\nWhen analyzing claims, strictly adhere to the information presented in the table. If the table defines ranges or boundaries (like 'less than 17%'), avoid interpreting the claim as being 'inaccurate' simply because a value falls within that range. Instead, focus on whether the table *directly* supports, refutes, or lacks information regarding the claim. If the table only provides definitions or categorizations without addressing the claim's truthfulness, default to 'not enough info'.

Table 29: Baseline and optimized instructions for the Gemma3-27B model with CoT method using various optimizers.

32B with CoT prompting using three different seed instructions. We consider seed instructions of diverse informativeness, including an empty instruction, a paraphrase of the default instruction used in previous analysis, and a more detailed instruction. Table 33 demonstrates that different optimizers exhibit varying sensitivity to initial instruction quality. COPRO benefits from more informative and detailed initial instructions, showing steadily increasing F1 scores on TabFact and MMSci as the seed instruction contains more information. This is mainly because COPRO proposes new candidate instructions solely based on previous candidates and their evaluation scores. A detailed seed instruction can provide more guidance for instruction generation in early iterations.

SIMBA reaches peak performance with concise seed instructions, whereas introducing additional detail in the seed can lead to degradation. Since SIMBA appends heuristic rules to the initial instructions, starting from a detailed instruction may restrict the search space, which negatively impacts the generalisability of optimized instructions. Given semantically equivalent seed instructions, SIMBA is least sensitive to variations in surface form compared with COPRO and MiPROv2. The model performance after SIMBA optimization with the paraphrased initial instruction remains similar to the baseline across all test sets.

C.3 Optimizing ReAct with TART tools

In addition to SQL tool, we investigate ReAct agents equipped with single or multiple tools that are most commonly used in TART framework (Lu et al., 2025). We optimize the instructions in ReAct agents with Qwen3-32B as the backbone model using MiPROv2 and SIMBA, since these optimizers are more effective for improving ReAct performance based on Table 2. As shown in Table 35, increasing the number of TART tools does not necessarily lead to better optimized performance for ReAct agents. In most experiment settings, F1 scores on the test sets after optimization follow a U-shaped trend as the number of tool functions is increased from 3 to 10. ReAct with a single tool such as `get_column_cell_value` can achieve stronger performance before and after instruction optimization than using top 10 TART tools. The `get_row_by_name` tool is particularly effective on PubHealthTab and MMSci, while

Optimizer	**Instruction**
Baseline	Verify the given claim against the provided table data.
COPRO	You are a highly accurate claim verification agent. Your task is to determine whether a given claim is supported by data within a provided table. You will be given a ‘claim‘, a ‘table‘ (in list of lists format including a header row), and a ‘caption‘ describing the table.\n\nFollow these steps:\n\n1. **Understand the Claim:** Carefully read the ‘claim‘ to identify the key entities and relationship being asserted.\n2. **Analyze the Table:** Examine the ‘table‘ structure, header, and data to understand the information it contains. Consider the ‘caption‘ for helpful context.\n3. **Formulate a SQL Query:** Construct a SQL query that, when executed against the ‘table‘, will directly answer the question posed by the ‘claim‘. The SQL query should be as simple and direct as possible.\n4. **Execute the SQL Query:** Use the ‘execute_sql‘ tool with the ‘table_data‘, a chosen ‘table_name‘ (e.g., \"my_table\"), and your crafted ‘sql_query‘.\n5. **Interpret the Results:** Analyze the results returned by the ‘execute_sql‘ tool. \n6. **Verify the Claim:** Based on the query results, determine if the claim is supported (TRUE), contradicted (FALSE), or if the table does not contain sufficient information to determine (UNKNOWN).\n7. **Finish:** Use the ‘finish‘ tool to signal task completion.\n\nYour response should be accurate and concise, relying solely on the information presented in the ‘table‘. Prioritize SQL queries that directly address the claim’s core assertion.
MiPROv2	You are a fact-checking agent tasked with verifying claims against tabular data. You will be given a claim, a table, and a table caption. Your goal is to determine if the table supports, refutes, or provides insufficient information to verify the claim.\n\n**Here’s how you should operate:**\n\n1. **Reasoning:** Carefully read the claim, table caption, and table data. Formulate a clear plan to determine the answer. Consider potential issues like units, data types, and ambiguous language.\n2. **Tool Use (execute_sql):** If the table is complex or requires specific data extraction, use the ‘execute_sql‘ tool to query the table. Craft SQL queries that directly address the claim. Remember to handle potential errors (e.g., invalid column names, data types). If SQL fails, proceed to manual inspection.\n3. **Manual Inspection:** If ‘execute_sql‘ fails or isn’t necessary, manually inspect the table to find relevant information.\n4. **Final Decision:** Based on your reasoning and the extracted evidence, determine whether the table *supports* the claim, *refutes* the claim, or provides *not enough info* to make a determination.\n5. **Output:** Provide your reasoning step-by-step, then state your final answer as either ’supports’, ’refutes’, or ’not enough info’.\n\n**Important Considerations:**\n\n* Prioritize accuracy. Double-check your reasoning and evidence.\n* Be mindful of the table caption; it often provides crucial context.\n* If the claim refers to a comparison, ensure you are comparing the correct data points.\n* If the table does not contain the information necessary to answer the claim, select \"not enough info\".
SIMBA	Verify the given claim against the provided table data.\n\nYou are an Agent. In each episode, you will be given the fields ‘claim‘, ‘table‘, ‘caption‘ as input. And you can see your past trajectory so far.\nYour goal is to use one or more of the supplied tools to collect any necessary information for producing ‘answer‘.\n\nTo do this, you will interleave next_thought, next_tool_name, and next_tool_args in each turn, and also when finishing the task.\nAfter each tool call, you receive a resulting observation, which gets appended to your trajectory.\n\nWhen writing next_thought, you may reason about the current situation and plan for future steps.\nWhen selecting the next_tool_name and its next_tool_args, the tool must be one of:\n\n(1) execute_sql, whose description is <desc>Execute a SQL query on a table provided as list of lists format. Args: table_data: List of lists where first row contains headers, subsequent rows contain data table_name: Name of the table dataframe for executing SQL query sql_query: SQL query string to execute Returns: Formatted string of the transformed table or error message </desc>. It takes arguments {’table_data’: {’items’: {}, ’type’: ’array’}, ’table_name’: {’type’: ’string’}, ’sql_query’: {’type’: ’string’}}.\n(2) finish, whose description is <desc>Marks the task as complete. That is, signals that all information for producing the outputs, i.e. ‘answer‘, are now available to be extracted.</desc>. It takes arguments {}.\nWhen providing ‘next_tool_args‘, the value inside the field must be in JSON format\n\nWhen you receive table data directly as a list of lists, avoid attempting to use the ‘execute_sql‘ tool. Instead, directly analyze the provided data to answer the question. Prioritize direct analysis of the provided data before resorting to SQL queries, especially if initial SQL attempts fail.\n\nWhen the table data is provided directly as a list of lists, prioritize analyzing the data directly instead of attempting to use SQL. SQL is useful for complex queries on large datasets, but for small, directly provided tables, direct analysis is more efficient and avoids potential errors. Look for keywords like ’list of lists’ or ’table’ in the input to determine if direct analysis is appropriate.\n\nWhen encountering an Odds Ratio (OR) of 1.00, especially in the context of a claim about a baseline percentage, consider explicitly stating the assumption that the baseline rate is equal to the claimed percentage. Avoid immediately concluding ’not enough info’ simply because the table doesn’t directly state the percentage; instead, explore reasonable interpretations of the OR value in relation to the claim.\n\nWhen analyzing tables, prioritize the most directly relevant value to the claim. In this case, the claim asks for the sensitivity *percentage* of the acute phase. While 98% appears in the table, it’s a specificity value, not a sensitivity value. Focus on the ’Sensitivity (%)’ column and the ’Acute phase’ row to find the correct value (17%). Avoid getting distracted by other numbers in the table that aren’t directly responsive to the question.\n\nWhen the input ‘table‘ is provided as a list of lists (rather than a database connection or similar), prioritize direct analysis of the data instead of attempting to use SQL. SQL is more appropriate for large datasets or complex queries, but for small, directly provided tables, direct analysis is more efficient and avoids potential errors. Look for keywords like ’list of lists’ or ’table’ in the input to determine if direct analysis is appropriate. If SQL queries repeatedly fail, immediately switch to direct data analysis.

Table 30: Baseline and optimized instructions for the Gemma3-27B model with ReAct method using various optimizers.

Optimizer	Instruction
Baseline	Verify the given claim against the provided table data.
COPRO	You are a highly skilled agent designed to verify claims based on data presented in tables. You will be given a ‘claim‘, a ‘table‘ (represented as a string), and a ‘caption‘ describing the table. Your task is to determine whether the claim is supported by the information in the table. \n\nHere’s how you will proceed:\n\n1. **Understand the Table:** Parse the table string to understand its structure (rows, columns, headers).\n2. **Extract Relevant Data:** Identify the data within the table that relates to the ‘claim‘.\n3. **Verify the Claim:** Compare the information in the ‘claim‘ with the extracted data to determine if the claim is TRUE, FALSE, or UNKNOWN (if the table doesn’t contain enough information to verify the claim).\n4. **Generate Python Code:** Write Python code to accomplish steps 1-3. The code must print the final answer (‘TRUE‘, ‘FALSE‘, or ‘UNKNOWN‘) to standard output. Focus on extracting and comparing the key pieces of information needed to answer the question directly.\n5. **Mark Completion:** Include ‘finished=True‘ after the final code block when the answer is ready to be extracted.\n\nDo not include any conversational text or explanations in your output; only the Python code and the ‘finished=True‘ marker.
MiPROv2	You are a highly skilled data analyst tasked with verifying claims against tabular data. You will receive a ‘claim‘ (a statement to be verified), a ‘table‘ (containing the data in a delimited string format), and a ‘caption‘ (describing the table). Your objective is to determine if the ‘claim‘ is supported, refuted, or if the ‘table‘ provides insufficient information (‘not enough info‘).\n\nYou must generate Python code to parse the ‘table‘ data using the ‘pandas‘ library. The table data is delimited by ‘||‘. Extract the relevant information from the table to evaluate the claim. Ensure your code handles potential errors gracefully, such as missing data or incorrect data types. Print the final boolean result (True or False) representing whether the claim is supported by the table data to the console.\n\nYour response must follow these steps:\n\n1. **Parse the Table:** Use ‘pandas‘ to read the table data from the provided string. Use ‘sep=’||’‘ to correctly parse the data. Clean the data by removing leading/trailing whitespace from column names and cell values.\n2. **Analyze the Claim:** Carefully read and understand the claim. Identify the specific data points needed from the table to verify the claim.\n3. **Extract and Compare:** Extract the relevant data points from the ‘pandas‘ DataFrame. Perform the necessary comparison or calculation to determine if the claim is true or false based on the table data.\n4. **Print the Result:** Print either ‘True‘ or ‘False‘ to the console, representing the result of your analysis.\n5. **Mark Completion:** After printing the result, add ‘Finished: True‘ to signify the task is complete.\n\nYou should only print the boolean value (True or False) and ‘Finished: True‘ to the console; no other text or explanations.
SIMBA	Verify the given claim against the provided table data.\n\nYou are an intelligent agent. For each episode, you will receive the fields ‘claim‘, ‘table‘, ‘caption‘ as input.\nYour goal is to generate executable Python code that collects any necessary information for producing ‘answer‘.\nFor each iteration, you will generate a code snippet that either solves the task or progresses towards the solution.\nEnsure any output you wish to extract from the code is printed to the console. The code should be enclosed in a fenced code block.\nWhen all information for producing the outputs (‘answer‘) are available to be extracted, mark ‘finished=True‘ besides the final Python code.\nYou have access to the Python Standard Library and the following functions:\n\nWhen generating code to parse tables, pay close attention to syntax, especially indentation. Ensure that all ‘else‘ statements have corresponding indented blocks. Before finalizing the code, consider adding a simple test case to verify that the parsing logic works correctly with a small subset of the table data. Also, consider adding error handling to catch potential parsing issues and provide informative error messages.\n\nWhen parsing tables, be extremely careful with the separator. The table uses ’||’ as a separator, but you initially used ’\\\|’. Ensure the separator matches the table’s format exactly. Also, double-check the column names you are trying to access after parsing the table to avoid KeyErrors. Consider adding a small test case to verify the parsing logic with a subset of the table data before proceeding with the full analysis.\n\nWhen parsing the table, double-check the separator used in ‘pd.read_csv‘. The table uses ’||’ as a separator. Ensure this is correctly specified. Also, after creating the DataFrame, verify the number of columns and the assigned column names to ensure they match the table structure. Add a small test case to verify the parsing logic with a subset of the table data before proceeding with the full analysis. If parsing fails, return a simple error message instead of crashing.\n\nWhen generating Python code, pay extremely close attention to indentation. Incorrect indentation is a common source of errors. Before finalizing the code, use a linter or run the code locally to catch these errors. Specifically, ensure that all blocks following control flow statements (e.g., ‘if‘, ‘else‘, ‘try‘, ‘except‘) are properly indented.

Table 31: Baseline and optimized instructions for the Gemma3-27B model with CodeAct method using various optimizers.

Random Seed	Optimizer	PubHealth		SciTab		TabFact		MMSci	
		Acc	F1	Acc	F1	Acc	F1	Acc	F1
0	Baseline	88.3	87.6	66.4	66.4	84.5	86.6	86.5	61.6
	+MiPROv2	87.8	87.3	67.4	67.2	85.9	87.8	87.4	63.6
	+SIMBA	87.2	86.2	**68.8**	**68.6**	85.2	87.1	86.4	63.1
9	+MiPROv2	88.3	87.2	68.5	68.5	85.2	87.4	86.6	62.4
	+SIMBA	88.3	87.4	67.8	67.5	86.1	87.4	87.8	**65.1**
42	+MiPROv2	87.8	87.1	67.8	67.8	**86.5**	**88.3**	87.3	**65.1**
	+SIMBA	**88.9**	**88.0**	**68.8**	**68.6**	**86.5**	87.9	**88.5**	64.7

Table 32: Ablation study on random seeds for Qwen3-32B with CoT. **Bold** indicates best performance per dataset.

Seed Instruction	Optimizer	PubHealth		SciTab		TabFact		MMSci	
		Acc	F1	Acc	F1	Acc	F1	Acc	F1
Verify the given claim against the provided table data. (Default)	Baseline	88.3	87.6	66.4	66.4	84.5	86.6	86.5	61.6
	+COPRO	87.2	86.1	67.4	67.3	85.5	87.6	86.7	61.5
	+MiPROv2	87.2	86.5	**68.8**	**68.6**	**86.9**	**88.5**	**87.7**	**65.4**
	+SIMBA	**90.0**	**89.6**	**68.8**	**68.6**	85.2	87.1	87.0	64.2
Empty string	Baseline	85.6	84.5	68.1	68.2	83.5	85.9	86.4	63.0
	+COPRO	87.2	85.9	67.1	67.0	83.3	85.4	85.5	60.8
	+MiPROv2	85.0	83.8	**69.2**	**69.0**	**86.5**	**88.4**	**87.7**	62.2
	+SIMBA	**87.8**	**86.8**	66.7	66.4	84.1	86.1	86.7	**63.1**
Check whether the table data supports the claim.	Baseline	89.4	88.8	67.1	67.2	84.0	86.1	86.3	62.3
	+COPRO	**90.0**	**89.5**	66.4	66.4	84.4	86.6	85.4	62.2
	+MiPROv2	88.3	87.5	66.4	66.2	84.2	86.4	**86.9**	63.2
	+SIMBA	89.4	88.5	**68.5**	**68.2**	**86.2**	**87.6**	86.6	**63.8**
Examine the claim provided below and systematically compare each component of the statement against the corresponding values, entries, and relationships present in the table. Determine whether the claim is supported by, contradicts, or cannot be verified from the available tabular information.	Baseline	83.9	83.0	**68.8**	**68.8**	85.9	87.9	**87.0**	62.9
	+COPRO	**87.8**	**86.9**	**68.8**	**68.8**	85.5	87.6	86.9	**63.4**
	+MiPROv2	85.6	84.8	66.2	66.2	**87.0**	**88.7**	**87.0**	63.1
	+SIMBA	**87.8**	86.8	65.7	65.8	85.9	87.5	85.8	62.7

Table 33: Ablation study on seed instructions for Qwen3-32B with CoT. **Bold** indicates best performance per seed instruction and dataset.

Training Data	Optimizer	PubHealth		SciTab		TabFact		MMSci	
		Acc	F1	Acc	F1	Acc	F1	Acc	F1
Hybrid	Baseline	88.3	87.6	66.4	66.4	84.5	86.6	86.5	61.6
	+COPRO	87.2	86.1	67.4	67.3	85.5	87.6	86.7	61.5
	+MiPROv2	87.2	86.5	**68.8**	**68.6**	**86.9**	**88.5**	**87.7**	**65.4**
	+SIMBA	**90.0**	**89.6**	**68.8**	**68.6**	85.2	87.1	87.0	64.2
SciTab	+COPRO	87.2	86.5	**69.7**	**69.8**	83.2	85.4	85.5	62.4
	+MiPROv2	**88.9**	**88.0**	68.3	68.2	**86.5**	**88.3**	87.4	62.1
	+SIMBA	87.8	87.3	64.1	63.7	84.8	86.5	**88.3**	**67.4**
TabFact	+COPRO	88.3	88.8	67.4	69.2	85.4	87.0	86.4	87.7
	+MiPROv2	87.2	88.3	**70.2**	**71.3**	86.1	88.1	87.0	88.9
	+SIMBA	**90.0**	**89.3**	64.3	67.5	**87.0**	**88.2**	**88.2**	**89.0**
PubHealth	+COPRO	87.8	87.3	**68.5**	**68.4**	**86.6**	**88.1**	**87.8**	**65.8**
	+MiPROv2	87.2	86.4	66.0	66.0	85.6	87.6	87.4	**65.8**
	+SIMBA	**88.9**	**88.3**	67.4	67.3	85.4	87.1	86.6	62.8

Table 34: Ablation study on training data for Qwen3-32B with CoT. **Bold** indicates best performance per training data configuration and dataset.

`get_row_index_by_value` demonstrates strong performance on SciTab. Despite showing effectiveness in some evaluation settings, ReAct with TART tools or their combinations does not outperform the counterpart with SQL tool after SIMBA optimization.

C.4 Training Data Configuration

We investigate the impact of training data on instruction optimization by conducting the optimization with three training configurations. Instead of using hybrid train data, we randomly sample 100 instances from each of the SciTab, TabFact and PubHealthTab training sets to construct single-

		PubHealth		**SciTab**		**TabFact**		**MMSci**	
Tool Functions	**Optimizer**	Acc	F1	Acc	F1	Acc	F1	Acc	F1
execute_sql	Baseline	87.8	87.4	61.5	60.1	82.8	83.3	**87.5**	62.6
	+MiPROv2	87.8	87.2	61.5	60.9	84.2	85.2	86.2	63.0
	+SIMBA	**90.6**	**90.0**	**66.2**	**65.9**	**86.1**	**87.0**	85.9	**65.0**
Top 10 TART tools	Baseline	85.0	84.3	59.4	58.1	83.4	84.8	84.7	61.8
	+MiPROv2	**87.8**	**87.1**	60.1	59.3	**86.0**	**87.4**	**87.8**	**62.4**
	+SIMBA	84.4	83.9	**62.7**	**62.7**	81.6	84.4	81.6	57.8
Top 3 TART tools	Baseline	85.0	84.1	58.0	56.6	83.4	85.0	83.2	59.1
	+MiPROv2	**86.7**	**85.9**	**61.3**	**60.2**	**83.5**	**85.9**	**86.6**	**62.5**
	+SIMBA	84.4	83.9	58.7	58.5	82.1	84.2	82.8	58.6
Top 5 TART tools	Baseline	84.4	83.6	**59.7**	58.0	**84.1**	85.6	83.8	59.7
	+MiPROv2	**85.0**	**84.2**	58.3	57.2	83.9	**85.8**	**86.3**	**62.6**
	+SIMBA	82.8	81.6	58.7	**58.1**	82.0	84.0	82.2	58.1
equal_to	Baseline	**86.7**	**85.9**	59.9	59.6	83.6	84.5	86.1	62.8
	+MiPROv2	86.1	85.0	59.4	58.5	**84.3**	**85.3**	**87.3**	**63.4**
	+SIMBA	85.0	84.7	**66.0**	**65.8**	83.5	84.8	85.8	61.4
get_column_by_name	Baseline	87.2	86.6	61.8	60.7	**85.2**	**86.6**	**85.8**	60.8
	+MiPROv2	**87.8**	**87.2**	59.2	58.6	81.6	84.4	84.8	**61.7**
	+SIMBA	83.9	82.5	**63.9**	**64.1**	83.4	85.7	83.7	60.2
get_column_cell_value	Baseline	87.8	86.6	59.9	58.5	83.3	85.1	84.4	**62.1**
	+MiPROv2	**88.3**	**87.8**	60.4	59.7	**84.9**	**86.2**	**86.8**	61.8
	+SIMBA	**88.3**	87.6	**65.3**	**65.1**	81.7	84.0	81.4	59.4
get_row_by_name	Baseline	88.3	87.7	57.8	55.6	85.2	86.6	87.0	63.6
	+MiPROv2	**88.9**	**88.2**	60.1	58.5	**85.4**	86.7	**88.1**	63.0
	+SIMBA	87.2	86.3	**61.1**	**60.7**	85.0	**86.8**	87.5	**66.6**
get_row_index_by_value	Baseline	**87.8**	**87.0**	62.2	61.6	**84.6**	**86.5**	86.5	**63.1**
	+MiPROv2	87.2	86.5	64.1	63.8	82.6	84.4	**86.8**	61.6
	+SIMBA	87.2	86.7	**66.0**	**66.1**	82.4	85.6	86.5	61.2

Table 35: Ablation study on tool functions used in TART for Qwen3-32B with ReAct. **Bold** indicates best performance per tool function and dataset.

source training sets. As shown in Table 34, SIMBA benefits more from using a hybrid train set, while MiPROv2 is less sensitive to the distribution of train data. This indicates hybrid data can help SIMBA learn more generalized rules across different table-based fact checking tasks. In contrast, MiPROv2 follows human-written heuristic rules to propose new candidate instructions and mainly relies on the train data to understand the task format. Also, we find optimized instructions learned on TabFact data can generalize surprisingly well on SciTab and MMSci, probably due to cleaner data and binary label distribution in TabFact.