

TriVTab: Fusing Triple-Based Structured Reasoning and Visual Reasoning for Multimodal Table Question Answering

Zhuo Wen^{1,2} Yang Lin³ Weifan Zhao³ Ji Zhang^{4,*} Jun Gao^{1,2,*}

¹Key Laboratory of High Confidence Software Technologies, CS, Peking University, China

²Beijing Key Laboratory of Software and Hardware Cooperative Artificial Intelligence Systems

³ZTE Corporation, China

⁴Nanjing University of Aeronautics and Astronautics, China

zhuowen@gmail.com

{lin.yang, zhao.weifan}@zte.com.cn

j.zhang@nuaa.edu.cn gaojun@pku.edu.cn

*Corresponding authors.

Abstract

Multimodal Table Question Answering requires both accurate perception of cell contents in a table image and precise symbolic reasoning over the extracted values; multimodal large language models (MLLMs) handle the former well, but the latter remains more challenging, motivating the use of an explicit code interpreter for computation. Through a fine-grained partition analysis, we find instance-level complementarity between the two paths: even when the visual path is stronger overall, the structured path correctly answers a non-trivial subset of questions missed by the visual path. To exploit this complementarity, we propose **TriVTab**, a dual-path framework that combines direct visual reasoning with triple-based structured reasoning through agreement-gated multimodal fusion. Rather than seeking a uniformly stronger standalone path, TriVTab reconciles complementary paths to improve a strong visual baseline. TriVTab (i) uses an execution-oriented cell-triple representation to bind each value to an explicit row and column context, making complex table structures directly addressable for code generation; and (ii) employs agreement-gated multimodal fusion to treat cross-path agreement as corroboration and concentrate multimodal fusion on divergent predictions. On five multimodal Table QA benchmarks, TriVTab improves over a strong Qwen3-VL-8B chain-of-thought baseline by +2.53 points on average and over a Self-Reflection baseline by +3.46 points.

1 Introduction

Tables are a pervasive medium for organizing structured information in financial reports, scientific papers, and web pages, making Table Question Answering (Table QA) a central problem in doc-

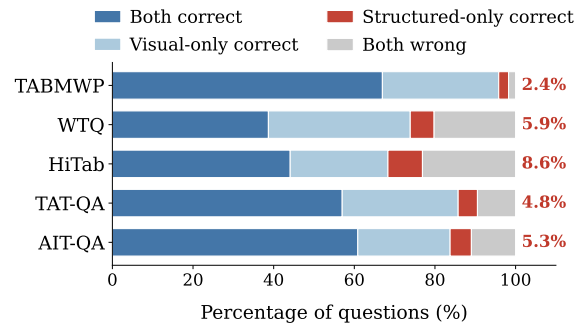


Figure 1: Instance-level complementarity between the visual reasoning path P_v and the structured reasoning path P_s across five multimodal Table QA benchmarks. The two paths correspond to the visual and structured reasoning components of TriVTab. Each bar partitions predictions into four cases according to whether each path answers correctly: both correct, visual-only correct, structured-only correct, and both wrong.

ument understanding. While early Table QA research assumed access to clean machine-readable inputs such as HTML or CSV (Pasupat and Liang, 2015; Herzig et al., 2020; Liu et al., 2022), a growing number of real-world scenarios present tables only as images embedded in PDFs, screenshots, or scanned documents. This multimodal setting (Zheng et al., 2024; Kim et al., 2024) is substantially more challenging, as it simultaneously requires accurate visual perception of cell contents, precise symbolic reasoning over the extracted values, and robustness to complex structures such as hierarchical headers and merged cells.

Multimodal large language models (MLLMs) have become the dominant approach to multimodal Table QA and achieve strong performance across diverse table types (OpenAI et al., 2024; Bai et al., 2025; Chen et al., 2024b). Recent work further

improves this paradigm through table-specialized training on large or synthetic corpora (Zheng et al., 2024; Zhou et al., 2025; Kim et al., 2026) and prompting methods such as chain-of-thought reasoning (Wei et al., 2022). These advances improve cell reading, region localization, and layout interpretation. However, questions that require symbolic computation—aggregating over many cells, chaining arithmetic, or comparing numerical quantities—remain challenging.

Delegating computation to an explicit interpreter provides a natural way to address this limitation. Program-aided approaches (Gao et al., 2023; Chen et al., 2023; Cheng et al., 2023; Wang et al., 2024) parse a table into a structured representation and delegate reasoning to executable code. In multimodal Table QA, this requires first converting the table image into a structured textual representation over which the code generator can express the intended operations. The intermediate representation is therefore a central design choice: it should make cell values directly addressable while preserving the row and column context that downstream computation depends on.

The two paths are complementary at the level of individual instances. Although the structured path is the weaker single-path baseline in our setup, it correctly answers a non-trivial fraction of questions on which the visual path fails (Figure 1). We refer to this property as *instance-level complementarity*: the two paths are jointly valuable because they fail on different instances, not because they are equally strong on average.

Instance-level complementarity can be exploited by early routing—predicting the better path and running only it—or by late fusion, which runs both paths and reconciles them. We adopt late fusion: early routing is efficient but cascades errors whenever the path classifier is wrong, whereas running both paths keeps each candidate available as evidence. The key question is then how to resolve disagreements. Answer-level heuristics such as fixed weighting or token-level confidence decide without checking whether either answer is grounded in the table image, so effective fusion should instead resolve disagreements through a per-instance, image-grounded verification step.

Building on these observations, we propose **TriVTab**. Our contributions are as follows:

- We propose **TriVTab**, a training-free dual-path inference framework for multimodal Ta-

ble QA that combines direct visual reasoning with triple-based structured reasoning through agreement-gated multimodal fusion. Rather than seeking a uniformly stronger standalone path, TriVTab reconciles two instance-level complementary paths to improve a strong visual baseline.

- We introduce an execution-oriented *cell-triple representation* as the interface between visual parsing and code generation. By binding each value to an explicit composite row and column context, cell triples flatten complex table structures—including multi-level headers, row/column hierarchies, and merged cells—into directly addressable records for code generation. A controlled ablation against HTML, Markdown, and linearized 2D representations (Section 4.4.1) shows that this advantage tends to become more pronounced as table complexity increases.
- We design an *agreement-gated multimodal fusion* strategy. Normalized agreement between the two paths is directly adopted, while disagreement is resolved by a multimodal fusion module that re-examines the table image together with both paths’ answers and reasoning traces.

On five multimodal Table QA benchmarks, TriVTab improves over a strong Qwen3-VL-8B baseline with chain-of-thought prompting by +2.53 points on average (5/5 datasets), and over a Self-Reflection baseline by +3.46 points.

2 Related Work

Table QA. Table QA has evolved from semantic parsing over structured tables (Pasupat and Liang, 2015; Herzig et al., 2020; Liu et al., 2022) to multimodal settings where tables appear as images (Zheng et al., 2024; Kim et al., 2024). Recent benchmarks emphasize either structural complexity (Cheng et al., 2022; Katsis et al., 2022) or symbolic computation (Zhu et al., 2021; Lu et al., 2023).

Multimodal table understanding. General-purpose MLLMs (OpenAI et al., 2024; Bai et al., 2025; Chen et al., 2024b) and table-specialized variants (Zheng et al., 2024; Zhou et al., 2025; Kim et al., 2026) process table images end-to-end, but precise symbolic computation remains challenging for them (Nogueira et al., 2021). Program-aided

approaches (Gao et al., 2023; Chen et al., 2023; Cheng et al., 2023; Wang et al., 2024) delegate computation to executable code, but in the multimodal setting they must first convert the table image into a structured textual representation before code can operate on it. An extended discussion is provided in Appendix A.

3 Method

3.1 Overview

TriVTab is a dual-path framework that improves a strong visual reasoning path by pairing it with a complementary structured reasoning path. As illustrated in Figure 2, the framework comprises three components that operate over a table image I and a question Q :

- A *visual reasoning path* $\mathcal{P}_v$ that answers the question by directly attending to the image with an MLLM, well-suited to perception-centric questions (Section 3.2);
- A *triple-based structured reasoning path* $\mathcal{P}_s$ that parses the image into a compact cell-triple representation $\{(r, c, v)\}$ and generates executable code over it, well-suited to computation-centric questions (Section 3.3);
- A *multimodal fusion module* $\mathcal{F}$ that is invoked only on disagreement, jointly examining the table image alongside both paths’ reasoning traces to produce the final answer (Section 3.4).

Formally, the two reasoning paths yield candidate answers along with supporting traces:

$$(a_v, e_v) = \mathcal{P}_v(I, Q), \quad (a_s, e_s) = \mathcal{P}_s(I, Q), \quad (1)$$

where e_v is $\mathcal{P}_v$ ’s reasoning trace and $e_s = (H, T, \pi)$ comprises the parsed HTML table H , the flattened cell-triple representation T , and the generated code π .

The final answer is determined by an agreement-gated rule:

$$\hat{a} = \begin{cases} a_{\text{agree}}, & \text{if } \text{Eq}_{\text{norm}}(a_v, a_s; Q), \\ \mathcal{F}, & \text{otherwise.} \end{cases} \quad (2)$$

Here, Eq_{norm} uses the same answer normalization as evaluation, and a_{agree} denotes the shared answer under this normalization. In the second

case, the fusion module $\mathcal{F}$ is invoked (Section 3.4). Agreement gating concentrates the multimodal fusion cost on the subset of questions where the two paths genuinely disagree.

3.2 Visual Reasoning Path

The visual reasoning path applies chain-of-thought prompting (Wei et al., 2022) directly over the table image, a standard and strong protocol for eliciting step-by-step reasoning from MLLMs. Given image I and question Q , we prompt the MLLM to (i) describe the table regions relevant to the question, (ii) extract the specific values needed, and (iii) produce a final answer in a normalized format:

$$(a_v, e_v) = \text{MLLM}(I, Q; P_v), \quad (3)$$

where e_v is the natural-language rationale preceding the answer. We elicit e_v explicitly rather than only the final answer, so that both the answer and its supporting rationale are available to the downstream fusion module.

3.3 Triple-Based Structured Reasoning Path

The structured reasoning path complements the visual path by decoupling perception from computation: it first parses the table image into an explicit symbolic representation, then offloads computation to a deterministic Python interpreter. The central design choice is the intermediate representation that bridges these two stages, since it determines how readily a code generator can locate values and express the required operations. The path consists of two stages: image-to-triples parsing (Section 3.3.1) and code generation with sandboxed execution (Section 3.3.2).

3.3.1 Image-to-Triples Parsing

Reliably extracting a flat, execution-ready representation directly from a table image is difficult, especially for tables with hierarchical headers or merged cells. We therefore target a representation in which every cell is independently addressable by its row and column context, and decompose parsing into two steps that each play to a model’s strengths: first recover the table structure as HTML, then flatten it into such cell triples. We prompt an MLLM to convert the image I into an HTML table H , leveraging its mature tabular generation capability:

$$H = \text{MLLM}(I; P_{\text{html}}). \quad (4)$$

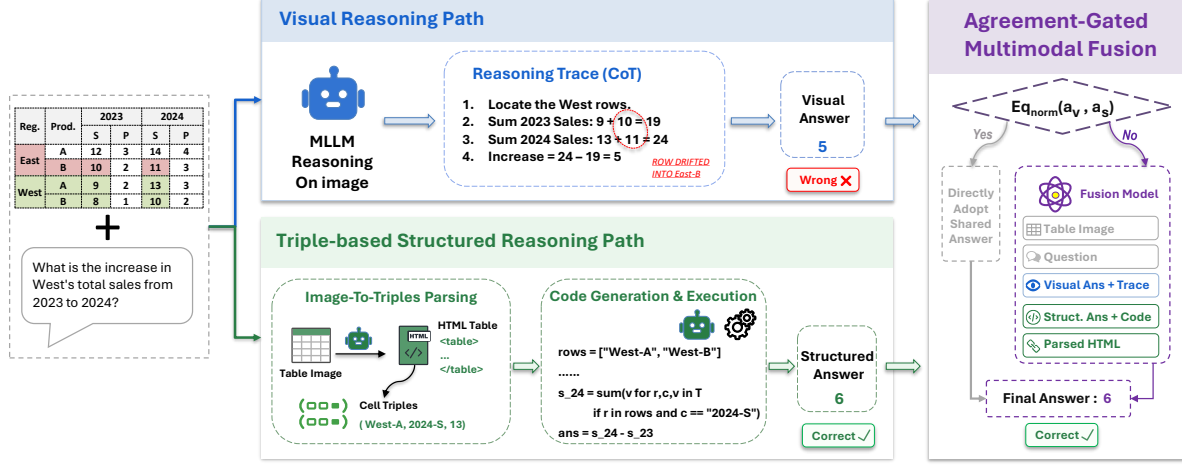


Figure 2: Overview of TriVTab. The visual reasoning path ($\mathcal{P}_v$) answers the question directly from the table image, while the structured reasoning path ($\mathcal{P}_s$) parses the image into cell triples $\{(r, c, v)\}$ and generates executable code over them. When the two paths agree, the answer is directly adopted; otherwise, a multimodal fusion module examines the image, both candidate answers, and their reasoning traces to produce the final answer.

Here HTML serves only as an intermediate bridge, not as the representation handed to the code generator. Feeding raw HTML directly to a code generator creates two problems: (i) HTML tokens consume a large fraction of the LLM’s context, especially for tables with many rows; and (ii) nested `<thead>`, `<tbody>`, `rowspan`, and `colspan` constructs force the code generator to parse structure and reason at the same time, which we empirically find to be error-prone.

We therefore flatten H into an execution-oriented *cell-triple representation* $T = \{(r, c, v)\}$, where each data cell is encoded as a tuple binding its value v to a composite row context r and column context c . The full flattening procedure—identifying the header region, unfolding merged cells, and removing duplicate keys—follows a set of deterministic rules detailed in Appendix G (Algorithm 1).

The cell-triple representation mitigates both problems. First, it is *flat*: each triple independently describes one cell, eliminating the need for the code generator to traverse nested tags. Second, it is *self-contained*: because each value already carries its (r, c) context, downstream code can filter and aggregate by matching on labels rather than relying on positional indexing. We empirically validate both advantages in Section 4.4.1.

3.3.2 Code Generation and Execution

The cell-triple representation reduces code generation to filter-and-aggregate operations over a flat

key-value set, which a text-only LLM can perform without specialized tabular libraries. Given the triple set T and the question Q , we first retrieve a question-relevant subset $T_{\text{code}} \subseteq T$, since the full set can be large for wide or long tables. Because each triple is bound to an explicit column label, this retrieval is targeted: rather than searching the whole table, we match question keywords against cell values within the relevant columns. We then prompt the LLM to produce a Python program π that, when executed over T_{code} , returns the answer:

$$\pi = \text{LLM}(T_{\text{code}}, Q; P_{\text{code}}), \quad (5)$$

where the retrieval procedure is detailed in Appendix J.

The prompt P_{code} specifies the triple schema and restricts π to Python built-ins, explicitly prohibiting tabular libraries such as `pandas` and `numpy`: since each triple already pairs a value with its row and column keys, list comprehensions and dictionary lookups suffice.

The generated program is executed in a sandboxed Python environment with a wall-clock timeout, a memory cap, and disabled filesystem, network, and subprocess access, following standard practice for tool-augmented LLM systems (Wang et al., 2025). Upon a syntactic or runtime error, the traceback is fed back together with the original prompt and a revised program is requested; we allow up to $K = 3$ retries. The path outputs $(a_s, e_s) = (\pi(T_{\text{code}}), (H, T, \pi))$ on success, or a null answer with the last error trace as evidence if

all retries fail.

3.4 Agreement-Gated Multimodal Fusion

TriVTab combines the two paths through an agreement gate: it adopts their shared answer when they agree, and invokes the fusion module only when they disagree. This gating concentrates the cost of multimodal fusion on the subset of questions where the paths actually diverge, sparing computation on questions where both paths already converge.

Agreement gating. Agreement is determined after the same answer normalization used at evaluation (Equation 2), not by raw string matching, so that formatting differences (e.g., thousands separators) do not trigger unnecessary fusion.

Multimodal fusion on disagreement. When the two answers disagree, we instantiate the fusion module $\mathcal{F}$ as a single MLLM call that examines all available evidence in the same multimodal context as the original task:

- the original task input: the table image I and the question Q ;
- the visual path’s output: its answer a_v and reasoning trace e_v ;
- the structured path’s output: its answer a_s (or null if execution failed), the generated code π , and the parsed HTML table H .

It verifies both candidate answers against this evidence and outputs the final answer $\hat{a}$ with a brief justification:

$$\hat{a} = \text{MLLM}(I, Q, a_v, e_v, a_s, \pi, H; P_{\text{fusion}}). \quad (6)$$

Crucially, this step re-grounds the decision in the table image rather than choosing between two answers on textual cues alone: it checks which candidate the visual evidence supports, and whether the generated code matches the intent of Q . The intent of this design is to incorporate the structured path when its answer is supported by the table image, allowing it to provide complementary evidence on questions where the two paths disagree. Section 4.4.2 shows that strategies which decide without this image-grounded re-examination, such as confidence-weighted selection, fall below TriVTab.

4 Experiments

4.1 Experimental Setup

Datasets. We evaluate TriVTab on five multimodal Table QA benchmarks derived from MMTab (Zheng et al., 2024): TABMWP (Lu et al., 2023), WTQ (Pasupat and Liang, 2015), HiTab (Cheng et al., 2022), TAT-QA (Zhu et al., 2021), and AIT-QA (Katsis et al., 2022). Following MMTab, we use rendered table images converted from the original structured tables, so that our methods are evaluated in an image-based Table QA setting. Dataset details and preprocessing choices are provided in Appendix C.

Evaluation Metrics. We use exact-match accuracy as the primary metric, with the following normalizations applied to both predictions and references before comparison: (i) lowercasing and whitespace stripping; (ii) numerical answers matched up to a relative tolerance of 10^{-3} to accommodate formatting differences (e.g., 1,234 vs. 1234); (iii) list answers compared as unordered sets when the question does not specify ordering. The same normalization function is also used to determine agreement between the visual and structured paths before invoking the fusion module.

Baselines. We compare TriVTab against two groups of baselines: prior multimodal table methods and path-only variants.

Prior multimodal table methods. We include prior multimodal table understanding models as contextual reference points, including Table-LLaVA-13B (Zheng et al., 2024), SynTab-LLaVA (Zhou et al., 2025), InternVL-2.5 (Chen et al., 2024a), and TabFlash-3B (Kim et al., 2026).

Path-only variants. To isolate each path’s contribution, we evaluate two single-path variants under the same backbone: Visual Reasoning Only, which uses the visual path $\mathcal{P}_v$ with chain-of-thought prompting; and Structured Reasoning Only, which uses the structured path $\mathcal{P}_s$ and returns the execution result directly.

Implementation Details. Unless stated otherwise, all paths use Qwen3-VL-8B as the multimodal backbone, and Qwen3-Flash (Yang et al., 2025) as the text-only code generator in $\mathcal{P}_s$. The fusion module is instantiated as the same Qwen3-VL-8B used in the visual path. The full sandbox configuration is provided in Appendix H, and all prompts are listed in Appendix N. All experiments

are run on 2 NVIDIA RTX 5880 GPUs with 48GB memory each.

4.2 Main Results

Table 1 reports exact-match accuracy on the five multimodal Table QA benchmarks.

TriVTab consistently improves over single-path baselines on the same backbone. On Qwen3-VL-8B, TriVTab achieves 84.04 average accuracy, improving over Visual Reasoning Only (81.51) by +2.53 points on average, with the largest gain of +3.49 on HiTab. The improvement is consistent across all five datasets (5/5). The structured path alone is much weaker (58.92 average), yet it provides complementary signal that the fusion module exploits, as we analyze in Section 4.3.

The gains persist at a smaller model scale. We additionally evaluate TriVTab with Qwen3-VL-4B (Appendix B), while keeping the remaining pipeline configuration unchanged. TriVTab again improves over the corresponding single-path variants on all five datasets. This suggests that the observed benefit is not specific to the Qwen3-VL-8B setting.

Comparison to prior methods. For broader context, Table 1 includes published results from representative multimodal table systems. TriVTab obtains higher reported accuracy than these methods across the five evaluated benchmarks.

4.3 Understanding the Source of Improvement

TriVTab consistently improves over the visual path, despite the structured path trailing it by 22.59 points on average as a standalone baseline. A stage-wise analysis of structured-path failures is provided in Appendix M.1. We next examine why combining such an imbalanced pair can still help, and how much headroom remains for the current fusion strategy.

Instance-level complementarity. Table 2 quantifies, across all five datasets, the complementarity previewed in Figure 1: each prediction is partitioned into four cases by which path answers it correctly. Although the structured path is uniformly weaker on average, the $\mathcal{P}_s$ -only-correct slice is non-empty across all datasets, ranging from 2.4% on TABMWP to 8.6% on HiTab. This instance-level complementarity increases the correct-answer coverage available from the two reasoning paths. For

reference, we report a *path-selection oracle*, which counts a question as correct if either path answers it correctly (Table 2, Oracle column).

Gains span all reasoning categories. To verify that fusion exploits complementarity broadly rather than only on a narrow slice of questions, we partition the test pool into six reasoning categories (Appendix E). TriVTab outperforms the visual path on *every* category—with gains from +1.99 to +2.75—and regresses on none, including categories where the structured path alone is far weaker (Appendix F). This pattern shows that the overall improvement is not concentrated in a single reasoning category and is consistent with complementary evidence being useful across diverse reasoning demands.

A gap to the path-selection oracle remains. TriVTab remains below the path-selection oracle on all five benchmarks (Table 2, Gap column). The gap is smallest on TABMWP (0.56) and largest on HiTab (5.07), indicating that the current fusion strategy does not recover all correct predictions available from the two reasoning paths. This leaves room for improved disagreement resolution.

4.4 Ablation Experiments

4.4.1 Intermediate Representation for Code Generation

The structured reasoning path generates code over an intermediate representation of the table. To isolate the effect of this representation choice, we hold the visual parsing model and the code generator fixed and compare four alternatives for the input handed to the code generator: **HTML** (the raw HTML produced by the MLLM), **Markdown** (rule-based flattening of HTML into Markdown tables), **Linearized 2D** (a row-major serialization where each row is a list of key-value pairs), and **Cell Triples** (our flat $\{(r, c, v)\}$ representation). We conduct this ablation on four datasets with sufficient structural variation, randomly sampling about 500 test questions from each dataset and excluding TABMWP, whose small flat tables offer little signal for comparing structural representations.

Cell triples tend to provide larger gains on more complex tables. To characterize how representation choice interacts with table complexity, we partition each dataset into Easy, Medium, and Hard subsets by table size and the presence of hierarchical or merged headers (procedure in Appendix D).

Method	TABMWP	WTQ	HiTab	TAT-QA	AIT-QA	Avg.
Prior multimodal table methods						
Table-LLaVA-13B (Zheng et al., 2024)	59.8	20.4	10.9	15.7	6.1	22.58
SynTab-LLaVA (Zhou et al., 2025)	88.3	39.6	35.7	51.9	28.6	48.82
InternVL-2.5 (Chen et al., 2024a)	93.1	45.2	53.7	55.7	58.7	61.28
TabFlash-3B (Kim et al., 2026)	93.7	46.4	60.5	59.9	54.9	63.08
Qwen3-VL-8B backbone						
Visual Reasoning Only	<u>95.81</u>	<u>73.87</u>	<u>68.34</u>	<u>85.75</u>	<u>83.76</u>	<u>81.51</u>
Structured Reasoning Only	69.41	44.61	52.66	61.79	66.14	58.92
TriVTab (Ours)	97.70	76.61	71.83	87.95	86.11	84.04

Table 1: Main results on five multimodal Table QA benchmarks, measured by exact-match accuracy (%). Results for prior multimodal table methods follow the comparative evaluation reported by Kim et al. (2026). The best result in each column is shown in bold, and the second-best result is underlined.

Dataset	Prediction partition (counts)				Accuracy (%)		
	Both ✓	$\mathcal{P}_v$ only ✓	$\mathcal{P}_s$ only ✓	Both ✗	TriVTab	Oracle	Gap
TABMWP	5147	2217	188	134	97.70	98.26	0.56
WTQ	1681	1528	257	878	76.61	79.79	3.18
HiTab	695	382	135	364	71.83	76.90	5.07
TAT-QA	440	222	37	73	87.95	90.54	2.59
AIT-QA	311	117	27	56	86.11	89.04	2.93
Avg.	—	—	—	—	84.04	86.91	2.87

Table 2: Per-dataset partition of predictions by which path is correct (left), and TriVTab versus the path-selection oracle (right). The oracle counts a question as correct if either reasoning path is correct; Gap is the difference between the oracle and TriVTab.

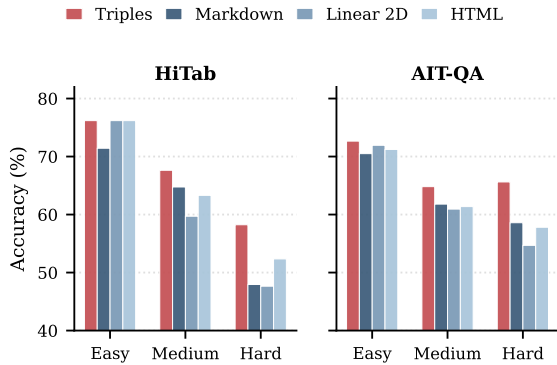


Figure 3: Structured path accuracy under different intermediate representations on HiTab and AIT-QA, broken down by table complexity. Full results for all datasets are in Appendix I.

Figure 3 shows results on HiTab and AIT-QA, our most structurally demanding benchmarks; full results are in Appendix I.

On HiTab, cell triples match the best alternative on Easy questions and lead the next-best representation by +2.88 and +5.89 on Medium and Hard subsets; on AIT-QA, the lead grows from +0.72 (Easy) to +3.01 (Medium) and +7.03 (Hard). This

matches the design motivation: binding each value to explicit row and column contexts flattens hierarchical structure and reduces the positional indexing the code generator must resolve. On easier, flatter tables, representation choice matters less, though cell triples remain competitive.

The main exception is the Hard split of WTQ, where HTML slightly outperforms cell triples (60.87 vs. 56.52). We attribute this to WTQ’s predominantly flat, Wikipedia-style tables: even its Hard split rarely contains the hierarchical headers or merged cells that cell triples are designed to flatten, so the representations are close, and the small split size (46 questions) makes the ordering unstable. Overall, cell triples tend to show stronger relative performance as structural complexity increases, while remaining competitive on simpler tables.

4.4.2 Path-Combination Strategy Comparison

Instance-level complementarity can be exploited in different ways, with different trade-offs. One option is *early routing*: classify the question and send it to a single path, executing only that path. This is efficient, but a mis-classified question is

Dataset	Early routing	Single-path verification	Late fusion		
	Category Routing	Self Ref.	Random tie.	Conf. weighted	TriVTab (Ours)
TABMWP	91.50	95.51	84.79	96.51	97.70
WTQ	68.46	74.08	62.06	73.96	76.61
HiTab	67.13	67.39	63.20	69.92	71.83
TAT-QA	73.19	83.55	74.22	84.20	87.95
AIT-QA	82.58	82.39	76.91	78.47	86.11
Avg.	76.57	80.58	72.24	80.61	84.04

Table 3: Comparison of path-combination strategies, measured by exact-match accuracy (%). Category Routing selects a single path before execution (early routing); Self-Reflection verifies the visual path only; and the late-fusion methods run both paths and resolve disagreement afterward. The best result in each row is shown in bold.

unrecoverable—the unchosen path is never run—so routing errors cascade directly into final accuracy. The alternative is *late fusion*: run both paths and reconcile them afterward, at a higher cost but with access to both candidate answers as evidence. We also include a single-path verification baseline (Self-Reflection) that revisits the visual answer without consulting the structured path. We compare TriVTab against representative strategies of these kinds, all built on $\mathcal{P}_v$ and $\mathcal{P}_s$:

- **Category Routing** (early routing): a question is classified into one of the reasoning categories (Appendix E) and routed to a fixed path—aggregation and arithmetic questions to $\mathcal{P}_s$, the rest to $\mathcal{P}_v$; unrecognized questions default to $\mathcal{P}_v$.
- **Random tiebreaker** (late fusion): on agreement, the shared answer is adopted; otherwise a_v or a_s is chosen uniformly at random.
- **Confidence-weighted** (late fusion): on disagreement, the answer with the higher token-level log-probability is selected.
- **Self-Reflection** (single-path verification): the visual path revisits its own answer in a second pass with the image as context, without ever invoking the structured path.

The single-path baselines are reported in Table 1, while Table 3 compares strategies for combining the two paths.

Category-based routing is insufficient. Category Routing reaches only 76.57 on average, below Visual Reasoning Only (81.51) on all five datasets.

A fixed path assignment at the reasoning-category level cannot capture the substantial instance-level variation in which path is more useful. TriVTab instead retains both candidate outputs and resolves disagreements after evidence from both paths is available.

Naive combination and verification strategies also hurt. All three simple combination or verification baselines fall below Visual Reasoning Only on average. Random tiebreaker drops to 72.24 (−9.27) by giving the weaker path equal voice on disagreements. Confidence-weighted (80.61, −0.90) and Self-Reflection (80.58, −0.93) are comparable but for different reasons: token-level confidence is unreliable across paths with different answer formats, while Self-Reflection never accesses the structured path and is limited to what re-reading the image can recover.

TriVTab outperforms all alternatives on every benchmark. TriVTab achieves the highest accuracy on all five datasets, improving over the best alternative (Confidence-weighted) by +3.43 on average and over early routing by +7.47 (Table 3). Two factors drive this consistency: agreement gating avoids an additional fusion call when the two paths already converge, and the fusion module, when invoked, conditions on multimodal evidence rather than answer-level cues or a pre-execution path choice.

Additional comparisons with three-sample visual self-consistency and simplified pipeline variants are provided in Appendix L.

4.4.3 Inputs to the Fusion Module

We ablate the auxiliary evidence given to the fusion module—the visual-side reasoning trace, the structured-side evidence (generated code and parsed HTML), or both—while always keeping the table image, the question, and both candidate answers. All variants lie within a narrow range (83.70–84.04 avg), and no single reduced configuration dominates across datasets. Full input achieves the highest macro-average accuracy at 84.04, so we adopt it as the default configuration. The four variants and per-dataset results are in Appendix K.

5 Conclusion

We presented TriVTab, a dual-path framework for multimodal Table QA that combines a visual reasoning path and a triple-based structured reasoning path through agreement-gated multimodal fusion.

The framework is built on the observation that the two paths are complementary at the instance level: rather than seeking a uniformly stronger standalone path, TriVTab reconciles two complementary paths to improve a strong visual baseline. On five benchmarks, TriVTab improves over Visual Reasoning Only by +2.53 points on average and over a Self-Reflection baseline by +3.46 points. A remaining gap to the path-selection oracle highlights room for improved disagreement resolution.

Limitations

Inference cost. The full TriVTab pipeline involves additional model invocations for HTML parsing, code generation, and fusion. Agreement gating mitigates this cost by invoking the fusion module only when the two paths disagree, and efficiency-oriented variants remain a natural extension.

Dependence on upstream visual parsing. While the cell-triple representation reduces structural ambiguity *downstream* of parsing, it cannot recover from cell-level perception errors introduced upstream. Our stage-wise error analysis (Appendix M.1) further shows that reconstruction and triple-conversion errors account for a substantial fraction of the inspected structured-path failures.

Language coverage. Our evaluation is restricted to English Table QA benchmarks. The core framework—dual-path reasoning, agreement gating, and multimodal fusion—is not inherently tied to English, though some components such as answer normalization and keyword-based retrieval may require adaptation for other languages, which differ in numeric and date formatting and header conventions. We leave a multilingual evaluation to future work.

Evaluation scope. Following MMTab, our evaluation uses table images rendered from originally structured tables. The current experiments therefore do not establish robustness to severely degraded scans, strong blur or skew, or highly irregular document layouts, which may further affect table reconstruction and the structured reasoning path.

Acknowledgments

This work was supported in part by NSFC (No. 62272008), ZTE-PKU joint program, and

Peking University Medicine plus X Pilot Program—Artificial Intelligence and Medical Development Initiative (No. BMU2025YXXLHAIYX001).

References

- Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhi-fang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Mingsheng Li, and 45 others. 2025. [Qwen3-vl technical report](#). *Preprint*, arXiv:2511.21631.
- Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023. [Program of Thoughts Prompting: Disentangling computation from reasoning for numerical reasoning tasks](#). *Transactions on Machine Learning Research*.
- Zhe Chen, Weiyun Wang, Yue Cao, Yangzhou Liu, Zhangwei Gao, Erfei Cui, Jinguo Zhu, Shenglong Ye, Hao Tian, Zhaoyang Liu, Lixin Gu, Xuehui Wang, Qingyun Li, Yiming Ren, Zixuan Chen, Jiapeng Luo, Jiahao Wang, Tan Jiang, Bo Wang, and 23 others. 2024a. Expanding performance boundaries of open-source multimodal models with model, data, and test-time scaling. *arXiv preprint arXiv:2412.05271*.
- Zhe Chen, Jiannan Wu, Wenhai Wang, Weijie Su, Guo Chen, Sen Xing, Muyan Zhong, Qinglong Zhang, Xizhou Zhu, Lewei Lu, Bin Li, Ping Luo, Tong Lu, Yu Qiao, and Jifeng Dai. 2024b. [InternVL: Scaling up vision foundation models and aligning for generic visual-linguistic tasks](#). In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 24185–24198.
- Zhoujun Cheng, Haoyu Dong, Zhiruo Wang, Ran Jia, Jiaqi Guo, Yan Gao, Shi Han, Jian-Guang Lou, and Dongmei Zhang. 2022. [HiTab: A hierarchical table dataset for question answering and natural language generation](#). In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1094–1110, Dublin, Ireland. Association for Computational Linguistics.
- 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 *The Eleventh International Conference on Learning Representations, ICLR 2023*. OpenReview.net.
- Jun Gao, Yinjun Han, Yang Lin, Hao Miao, and Mo Xu. 2024. [Learned distributed query optimizer: Architecture and challenges](#). *ZTE Communications*, 22(2):49–54.
- Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. [PAL: Program-aided language models](#). In *Proceedings of the 40th International*

- Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 10764–10799. PMLR.
- 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.
- Mohit Iyyer, Wen-tau Yih, and Ming-Wei Chang. 2017. [Search-based neural structured learning for sequential question answering](#). In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1821–1831, Vancouver, Canada. Association for Computational Linguistics.
- Yannis Katsis, Saneem Chemmengath, Vishwajeet Kumar, Samarth Bharadwaj, Mustafa Canim, Michael Glass, Alfio Gliozzo, Feifei Pan, Jaydeep Sen, Karthik Sankaranarayanan, and Soumen Chakrabarti. 2022. [AIT-QA: Question answering dataset over complex tables in the airline industry](#). In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Industry Track*, pages 305–314, Hybrid: Seattle, Washington + Online. Association for Computational Linguistics.
- Jongha Kim, Minseong Bae, Sanghyeok Lee, Jinsung Yoon, and Hyunwoo J. Kim. 2026. [TabFlash: Efficient table understanding with progressive question conditioning and token focusing](#). In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 22573–22581.
- Yoonsik Kim, Moonbin Yim, and Ka Yeon Song. 2024. [Tablevqa-bench: A visual question answering benchmark on multiple table domains](#). *Preprint*, arXiv:2404.19205.
- Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2023. [Visual instruction tuning](#). In *Advances in Neural Information Processing Systems*, volume 36, pages 34892–34916. Curran Associates, Inc.
- 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 *The Tenth International Conference on Learning Representations, ICLR 2022*. OpenReview.net.
- Pan Lu, Liang Qiu, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, Tanmay Rajpurohit, Peter Clark, and Ashwin Kalyan. 2023. [Dynamic prompt learning via policy gradient for semi-structured mathematical reasoning](#). In *The Eleventh International Conference on Learning Representations, ICLR 2023*. OpenReview.net.
- Ansong Ni, Srini Iyer, Dragomir Radev, Veselin Stoyanov, Wen-tau Yih, Sida I. Wang, and Xi Victoria Lin. 2023. [LEVER: Learning to verify language-to-code generation with execution](#). In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 26106–26128. PMLR.
- Rodrigo Nogueira, Zhiying Jiang, and Jimmy Lin. 2021. [Investigating the limitations of transformers with simple arithmetic tasks](#). *Preprint*, arXiv:2102.13019.
- OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, and 262 others. 2024. [Gpt-4 technical report](#). *Preprint*, arXiv:2303.08774.
- Panupong Pasupat and Percy Liang. 2015. [Compositional semantic parsing on semi-structured tables](#). In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1470–1480, Beijing, China. Association for Computational Linguistics.
- Nils Reimers and Iryna Gurevych. 2019. [Sentence-BERT: Sentence embeddings using Siamese BERT-networks](#). In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China. Association for Computational Linguistics.
- Wei Wang, Shaofeng Li, Tian Dong, Yan Meng, and Haojin Zhu. 2025. [From function calls to MCPs for securing AI agent systems: Architecture, challenges and countermeasures](#). *ZTE Communications*, 23(3):27–37.
- Zilong Wang, Hao Zhang, Chun-Liang Li, Julian Martin Eisenschlos, Vincent Perot, Zifeng Wang, Lesly Miculicich, Yasuhisa Fujii, Jingbo Shang, Chen-Yu Lee, and Tomas Pfister. 2024. [Chain-of-table: Evolving tables in the reasoning chain for table understanding](#). In *The Twelfth International Conference on Learning Representations, ICLR 2024*.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. [Chain-of-thought prompting elicits reasoning in large language models](#). In *Advances in Neural Information Processing Systems*, volume 35.
- 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. 2025. [Qwen3 technical report](#). *arXiv preprint arXiv:2505.09388*.

- Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. [TaBERT: Pretraining for joint understanding of textual and tabular data](#). In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8413–8426, Online. Association for Computational Linguistics.
- Mingyu Zheng, Xinwei Feng, Qingyi Si, Qiaoqiao She, Zheng Lin, Wenbin Jiang, and Weiping Wang. 2024. [Multimodal table understanding](#). In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9102–9124, Bangkok, Thailand. Association for Computational Linguistics.
- Victor Zhong, Caiming Xiong, and Richard Socher. 2017. [Seq2SQL: Generating structured queries from natural language using reinforcement learning](#). *Preprint*, arXiv:1709.00103.
- Bangbang Zhou, Zuan Gao, Zixiao Wang, Boqiang Zhang, Yuxin Wang, Zhineng Chen, and Hongtao Xie. 2025. [SynTab-LLaVA: Enhancing multimodal table understanding with decoupled synthesis](#). In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 24796–24806.
- Fengbin Zhu, Wenqiang Lei, Youcheng Huang, Chao Wang, Shuo Zhang, Jiancheng Lv, Fuli Feng, and Tat-Seng Chua. 2021. [TAT-QA: A question answering benchmark on a hybrid of tabular and textual content in finance](#). In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3277–3287, Online. Association for Computational Linguistics.

A Extended Related Work

Table Question Answering

Table Question Answering has been studied for over a decade as a testbed for grounded reasoning over structured data. Early work assumed clean, machine-readable inputs: Pasupat and Liang (2015) introduced WikiTableQuestions and framed the task as semantic parsing into logical forms, followed by benchmarks emphasizing compositional and conversational reasoning (Iyyer et al., 2017). A subsequent line of work pre-trained language models specifically for tabular inputs, including TAPAS (Herzig et al., 2020), TaBERT (Yin et al., 2020), and TAPEX (Liu et al., 2022).

As the field matured, benchmarks broadened along two complementary axes. Structural complexity increased with HiTab (Cheng et al., 2022) and AIT-QA (Katsis et al., 2022), which feature hierarchical headers and wide domain-specific tables; reasoning complexity increased with TAT-QA (Zhu et al., 2021) and TABMWP (Lu et al., 2023), which emphasize symbolic computation. More recent benchmarks such as MMTAB (Zheng et al., 2024) and TableVQA-Bench (Kim et al., 2024) formalize a multimodal setting in which tables appear as images embedded in PDFs or screenshots. This setting exposes a key limitation of methods designed for structured tables: they assume that table structure is already available, whereas multimodal Table QA requires visual perception, structure recovery, and downstream reasoning to be handled jointly.

End-to-End Multimodal Approaches

A natural approach to multimodal Table QA is to apply multimodal large language models (MLLMs) end-to-end. General-purpose MLLMs such as the GPT-4V family (OpenAI et al., 2024), the Qwen-VL series (Bai et al., 2025), the InternVL series (Chen et al., 2024b), and LLaVA-style models (Liu et al., 2023) have demonstrated strong performance on visually grounded question answering by leveraging large-scale image-text pre-training. Building on these foundations, several lines of work specialize MLLMs for tabular visual understanding: Table-LLaVA (Zheng et al., 2024) fine-tunes a LLaVA backbone on a large multimodal table corpus, SynTab-LLaVA (Zhou et al., 2025) augments training with synthetic tables, and TabFlash (Kim et al., 2026) introduces an efficient table-specialized variant. Across these methods, the dom-

inant paradigm treats the image as the sole reasoning substrate and couples perception and symbolic reasoning end-to-end within the model, without delegating computation to an external engine, which remains challenging for questions involving precise symbolic operations (Nogueira et al., 2021).

Program-Aided Reasoning over Tables

A complementary line of work delegates computation to an explicit symbolic engine. Treating tabular data through query-style operations has a long history in database research, where recent work has increasingly turned to learned models for query planning and optimization (Gao et al., 2024). The semantic parsing tradition (Pasupat and Liang, 2015; Zhong et al., 2017) translates natural language questions into formal queries over structured tables, an idea substantially revitalized by large language models. PAL (Gao et al., 2023) and Program-of-Thoughts (Chen et al., 2023) replace natural-language chain-of-thought with executable code, offloading arithmetic to a Python interpreter. Applied to Table QA, Binder (Cheng et al., 2023) and Lever (Ni et al., 2023) generate SQL or Python over given structured tables, while Chain-of-Table (Wang et al., 2024) iteratively transforms the table through atomic operations before answering. These approaches assume access to structured table inputs; in the multimodal setting, the table structure must instead be recovered from the image before program-aided reasoning can be applied.

B Cross-Scale Evaluation

To verify that TriVTab’s improvements are not tied to a specific MLLM backbone, we evaluate $\mathcal{P}_v$, $\mathcal{P}_s$, and the full pipeline on Qwen3-VL-4B. All other components — code generator, prompts, fusion module configuration — are kept identical to the main experiments. Results are reported in Table 4.

The relative improvement of TriVTab over single-path baselines on Qwen3-VL-4B is comparable to that on Qwen3-VL-8B, indicating that instance-level complementarity and agreement-gated fusion remain effective at smaller scales.

C Dataset Statistics

We evaluate on five widely used Table QA benchmarks that collectively cover diverse table sources, structural complexities, and reasoning types:

- **TABMWP** (Lu et al., 2023) consists of elementary-level math word problems

Dataset	Visual Only	Structured Only	TriVTab
TABMWP	97.11	72.22	97.70
WTQ	71.96	47.83	75.26
HiTab	72.46	49.11	73.37
TAT-QA	85.88	57.60	86.70
AIT-QA	80.24	64.04	83.17
Avg.	81.53	58.16	83.24

Table 4: Cross-scale results on Qwen3-VL-4B, measured by exact-match accuracy (%). “Visual Only” and “Structured Only” denote the visual reasoning path and the structured reasoning path, respectively. TriVTab improves over both single-path variants on every dataset, mirroring the pattern observed with the 8B backbone.

grounded in small tables, predominantly testing arithmetic reasoning.

- **WTQ** (WikiTableQuestions) (Pasupat and Liang, 2015) contains open-domain questions over Wikipedia tables, dominated by multi-hop retrieval and comparison.
- **HiTab** (Cheng et al., 2022) features hierarchical tables with multi-level headers and merged cells, stress-testing structural understanding.
- **TAT-QA** (Zhu et al., 2021) is drawn from financial reports and mixes textual and tabular evidence. Following prior multimodal Table QA evaluation, we restrict evaluation to its table-only subset, which contains a high proportion of symbolic computation questions.
- **AIT-QA** (Katsis et al., 2022) targets airline industry tables, with many lookup and filtering questions over wide, structurally clean tables.

All five datasets originally provide tables in structured form. To construct the multimodal evaluation setting, we adopt the rendered table images released by MMTab (Zheng et al., 2024), which converts tables from these source datasets into a unified image format using a standardized rendering pipeline. This makes our evaluation directly comparable with prior multimodal Table QA work built on MMTab. We use all datasets and model artifacts under their respective research-use licenses or terms of use.

D Table Complexity Categorization

To analyze how intermediate representation choice interacts with table complexity (Section 4.4.1), we

Condition	Contribution
$n_{\text{cells}} > 100$	+2
$30 < n_{\text{cells}} \leq 100$	+1
$n_{\text{merged}} > 5$	+2
$0 < n_{\text{merged}} \leq 5$	+1
$n_{\text{cells}} > 50$ and $r_{\text{num}} > 0.5$	+1

Table 5: Rule-based scoring scheme for table complexity. Each row specifies a condition on the structural features and the corresponding contribution to the complexity score. A table is labeled *Hard* if its total score is ≥ 4 , *Medium* if $2 \leq \text{score} < 4$, and *Easy* otherwise.

automatically assign each table a complexity label of *Easy*, *Medium*, or *Hard* based on structural features extracted directly from its HTML form.

Features. For each table, we extract three structural features from its HTML representation:

- $n_{\text{cells}} = n_{\text{rows}} \times n_{\text{cols}}$: total number of cells, where n_{rows} counts `<tr>` tags and n_{cols} is the maximum number of `<td>/<th>` cells across rows.
- n_{merged} : number of cells with `rowspan > 1` or `colspan > 1`, indicating hierarchical or merged headers.
- r_{num} : the fraction of non-empty cells whose textual content matches a numeric pattern, measuring how data-dense (rather than text-dense) the table is.

Scoring. We compute a complexity score by summing rule-based contributions from the three features, as shown in Table 5:

Label assignment. A table is labeled *Hard* if its score is ≥ 4 , *Medium* if $2 \leq \text{score} < 4$, and *Easy* otherwise. This rule-based scheme jointly accounts for table size (n_{cells}), structural irregularity (n_{merged}), and data density (r_{num}), and produces deterministic splits without requiring manual annotation.

E Question Type Taxonomy

To analyze TriVTab’s fusion behavior across reasoning types (Section 4.3), we classify each question into one of six categories based on the core reasoning steps required to answer it. The categories are designed to span the spectrum from pure cell lookup to multi-step combinatorial reasoning, while remaining structure-agnostic — categorization is determined by the reasoning process

required, not by surface features of the table or the question wording.

Categories.

- **(A) Direct Retrieval:** The answer can be directly located in a single cell without any computation or filtering. *Example:* “What was the sales figure in 2019?”
- **(B) Conditional Filtering:** Rows or columns must first be filtered based on a condition before retrieving the target value. *Example:* “What is the profit corresponding to the product with the highest sales volume?”
- **(C) Aggregation Statistics:** An aggregation operation (sum, count, mean, max, min) is applied to multiple cells. *Example:* “How many countries have won more than 10 gold medals?”
- **(D) Comparative Judgment:** Two or more values must be compared to determine an ordering or relationship. *Example:* “Which quarter had higher profits?”
- **(E) Arithmetic Reasoning:** Mathematical operations such as difference, ratio, growth rate, or multiple are required. *Example:* “What percentage did revenue increase from 2020 to 2021?”
- **(F) Multi-hop Combinatorial:** The question requires composing multiple steps from the previous categories in sequence (e.g., filtering followed by aggregation). *Example:* “What is the average profit of the top two products in terms of sales volume?”

Classification procedure. We use Qwen3-VL-Flash as an automatic classifier. For each question, the classifier receives the table image and the question, and is prompted with the category definitions above together with one illustrative example per category. The classifier outputs a single category label in JSON format. When multiple categories apply, the prompt instructs it to choose the most core or highest-difficulty step in the reasoning chain; if the question is a genuine composition that cannot be reduced to a single category, it is assigned to (F). The full classification prompt is provided in Appendix N.

F Per-Category Accuracy Results

Table 6 reports accuracy across the six reasoning categories defined in Appendix E. A small number of questions (186 of 14,889) for which the classifier failed to produce a valid category label are excluded from this breakdown; all other experiments use the full evaluation set. TriVTab improves over the visual reasoning path in all six categories, with gains ranging from +1.99 to +2.75 points.

The results show that TriVTab’s improvement is not driven by a single reasoning type. Even direct retrieval questions benefit from fusion, while aggregation statistics obtains the largest gain. Overall, the gains extend across diverse reasoning categories rather than being concentrated in a single type.

G Image-to-Triples Parsing

We flatten the parsed HTML table H into the cell-triple set $T = \{(r, c, v)\}$ with the deterministic procedure in Algorithm 1. We first expand all merged cells into a dense grid, then locate the header region (left columns acting as row headers, top rows acting as column headers), and finally assign each data cell a composite row and column label. The number of row-header columns is found by scanning for vertically merged spans; the number of column-header rows uses a width-dependent rule, since the sparsity and span cues are unreliable when a table has at most two columns.

Type	Reasoning category	Question count	Visual Reasoning Only	Structured Reasoning Only	TriVTab	Gain over Visual
A	Direct retrieval	5065	85.15	50.40	87.40	+2.25
B	Conditional filtering	574	75.78	43.21	78.05	+2.27
C	Aggregation statistics	2367	82.17	63.46	84.92	+2.75
D	Comparative judgment	1893	86.27	65.93	88.75	+2.48
E	Arithmetic reasoning	3700	90.32	77.41	92.65	+2.33
F	Multi-hop combinatorial	1104	88.68	41.94	90.67	+1.99

Table 6: Accuracy by reasoning category. “Question count” denotes the number of evaluated questions in each category, and all accuracy values are exact-match scores (%). “Gain over Visual” denotes the improvement of TriVTab over Visual Reasoning Only. TriVTab improves over the visual path in all six categories, indicating that agreement-gated fusion provides consistent gains across diverse reasoning types.

Algorithm 1 Image-to-Triples Parsing

Require: Parsed HTML table H , header separator SEP
Ensure: Cell-triple set $T = \{(r, c, v)\}$

```

1:  $G \leftarrow \text{EXPANDMERGEDCELLS}(H)$   $\triangleright$  dense grid
2:  $n_r, n_c \leftarrow \text{SHAPE}(G)$ 

3: // Row-header columns
4:  $L \leftarrow 0$ 
5: for  $j = 0$  to  $n_c - 1$  do
6:   if col  $j$  has a vertical merged span (data region) then
7:      $L \leftarrow j + 1$ 
8:   else
9:     break
10:  end if
11: end for
12:  $L \leftarrow \max(L, 1)$ 

13: // Column-header rows
14: if  $n_c \leq 2$  then  $\triangleright$  narrow: numeric rule
15:    $K \leftarrow$  first row with a numeric data cell,
    or the first empty row
16: else  $\triangleright$  wide: sparsity/span rule
17:    $K \leftarrow$  first row with neither  $\geq 2$  empty
    cells nor a horizontal span
18:   complete partial header rows (left/forward fill)
19: end if

20: // Composite labels and triples
21:  $T \leftarrow \emptyset$ 
22: for each data cell  $G[i][j]$  with  $i \geq K, j \geq 0$  do
23:    $r \leftarrow \text{JOIN}(\text{row headers of row } i, SEP)$ 
24:    $c \leftarrow \text{JOIN}(\text{col headers of col } j, SEP)$ 
25:    $T \leftarrow T \cup \{(r, c, G[i][j])\}$ 
26: end for
27: remove duplicate  $(r, c)$  keys, keeping first occurrence
28: return  $T$ 

```

H Sandbox Configuration

The code generated by $\mathcal{P}_s$ is executed in an isolated Python sandbox with the following restrictions: (i) a wall-clock execution timeout; (ii) a per-process memory cap; (iii) no filesystem access (read or write) outside the working directory; (iv) no network access; and (v) no subprocess spawning. Allowed imports are restricted to selected modules from the Python standard library (notably `math` and `re`); third-party tabular libraries such as `pandas`

and `numpy` are intentionally excluded so that the code generator operates over the cell-triple representation using only language-native constructs (list comprehensions, dictionary lookups, and basic arithmetic). Other imports are blocked at module-load time. On a syntax error, runtime exception, or timeout, the traceback is fed back to the code generator and the program is regenerated, up to $K = 3$ retries (Section 3.3).

I Full Representation Ablation

Table 7 reports the full representation ablation across all datasets and table-complexity splits. We note that the Hard splits of TAT-QA and WTQ contain very few questions (10 and 46, respectively), as these datasets consist predominantly of small, flat tables; their Hard-split numbers are therefore based on small samples and should be read as indicative rather than conclusive. The trends discussed in the main text are drawn primarily from HiTab and AIT-QA, whose tables exhibit the structural complexity these splits are designed to probe.

J Triple Subset Retrieval for Code Generation

For each question, we retrieve a question-relevant subset $T_{\text{code}} \subseteq T$ from the flattened triple set and provide it to the code generation prompt, since the full set can be large for wide or long tables. The retrieval is performed via the following two-stage procedure:

Stage 1: Query decomposition. A text-only LLM is prompted with the question Q and the list of column headers extracted from T . It is instructed to decompose Q into a set of *search filters*, where each filter consists of a target column and a list of candidate keywords. The LLM is asked to per-

Dataset	Complexity	Count	Triples	Markdown	Linear 2D	HTML
AIT-QA	Easy	139	72.66	70.50	71.94	71.22
	Medium	233	64.81	61.80	60.94	61.37
	Hard	128	65.62	58.59	54.69	57.81
HiTab	Easy	21	76.19	71.43	76.19	76.19
	Medium	139	67.63	64.75	59.71	63.31
	Hard	340	58.24	47.94	47.65	52.35
TAT-QA	Easy	366	72.13	66.94	64.21	68.03
	Medium	124	58.06	53.23	48.39	51.61
	Hard	10	30.00	30.00	30.00	30.00
WTQ	Easy	155	63.87	59.35	58.71	56.77
	Medium	298	58.39	52.01	52.35	55.37
	Hard	46	56.52	58.70	56.52	60.87

Table 7: Full representation ablation across datasets and table-complexity splits. Count denotes the number of evaluated questions in each split. Accuracy is measured on the structured reasoning path using different intermediate representations. The best result in each row is shown in bold.

form synonym expansion — for instance, mapping “took the gold” to candidate values {“gold”, “champion”, “winner”, “1st place”, “gold medal”} — so that the downstream retrieval is robust to surface-form mismatches between the question and the cell values. The full prompt template is provided in Appendix N.

Stage 2: Dense semantic matching. For each (column, keywords) filter from Stage 1, we encode all distinct cell values in the target column using a sentence encoder¹ and compute cosine similarity between each candidate keyword and each cell value. The cell value with the highest similarity is kept as a match, subject to a similarity threshold of 0.4. Triples whose row or column key contains a matched cell value are retained in T_{code} ; the rest are discarded. This procedure typically reduces the number of triples supplied to the code generator while preserving the cells most relevant to the question.

K Inputs to the fusion module

To understand whether each input signal supplied to the fusion module contributes meaningfully to its final decisions, we compare the full configuration against three reduced variants: (i) **Structured-side only**, which provides only the structured path’s answer a_s , generated code π , and the parsed HTML table H ; (ii) **Visual-side only**, which provides only the visual path’s answer a_v and reasoning trace e_v ; and (iii) **Answers only**, which provides both

¹We use all-MiniLM-L6-v2 (Reimers and Gurevych, 2019).

Dataset	Full input	Structured side only	Visual side only	Answers only
TABMWP	97.70	97.78	97.70	97.62
WTQ	76.61	75.76	76.20	75.74
HiTab	71.83	73.29	71.89	72.91
TAT-QA	87.95	84.20	87.18	86.92
AIT-QA	86.11	87.48	86.11	86.50
Avg.	84.04	83.70	83.82	83.94

Table 8: Ablation on the inputs to the fusion module, measured by exact-match accuracy (%). The best result in each row is shown in bold.

candidate answers (a_v, a_s) but no reasoning traces, no generated code, and no parsed HTML table. The image I and question Q are always included. Results are reported in Table 8.

Full input as the default configuration. The four variants are within a narrow range on average (83.70–84.04), with heterogeneous per-dataset behavior: Structured-side only performs best on TABMWP, HiTab, and AIT-QA, while Full input performs best on WTQ and TAT-QA. No single reduced configuration consistently dominates across benchmarks. Full input achieves the highest macro-average accuracy at 84.04, and we therefore use it as the default configuration.

L Additional Controlled Comparisons

We provide additional controlled comparisons to examine whether the improvement of TriVTab can be recovered through repeated visual inference or through simpler variants of the structured pipeline.

L.1 Visual Self-Consistency

We first examine whether repeated inference along the visual path can account for the improvement of TriVTab. We construct Visual-SC@3 by generating three visual-path responses and applying majority voting to their normalized final answers. As shown in Table 9, Visual-SC@3 improves the average accuracy from 81.51 to 82.33, a gain of 0.82 points over a single visual response. TriVTab reaches 84.04, outperforming Visual-SC@3 by 1.71 points. Thus, repeated visual inference provides a measurable improvement, but does not recover the full gain of the dual-path framework.

L.2 Simplified Pipeline Variants

We further examine whether comparable performance can be obtained with simpler variants of the structured pipeline. As shown in Table 9, the strongest simplified variant is *Triples + Visual Fusion*, which removes code generation and execution and achieves an average accuracy of 82.28. Full TriVTab reaches 84.04, retaining a 1.76-point advantage over this variant. Directly answering from HTML, triples, or the image-triple combination also falls below the full framework, while direct image-to-code execution performs substantially worse at 61.11. Overall, these results show that simpler uses of structured information capture part of the benefit, but do not match the performance of the complete TriVTab pipeline under the same evaluation setting.

M Structured-Path Diagnostics

M.1 Stage-Wise Error Analysis

To better understand the relatively low standalone accuracy of the structured reasoning path, we manually inspect 20 structured-path failures sampled across the five datasets. For each example, we examine the table image, reconstructed HTML, cell triples, generated program, and execution result, and assign the failure to the earliest stage at which question-relevant information becomes incorrect or unusable.

We distinguish three mutually exclusive failure stages. *Image-to-HTML reconstruction* errors occur when question-relevant cell content or table structure is already incorrect in the reconstructed HTML. *HTML-to-triple conversion* errors occur when the HTML preserves the required information, but relevant row, column, section, or value context is lost or incorrectly bound during triple

construction. *Downstream code/reasoning* errors occur when the required evidence is preserved in the triples, but retrieval, program generation, execution, or answer extraction produces an incorrect result.

Table 10 shows that failures arise both before and after program generation. Nine of the 20 errors occur upstream: six originate from image-to-HTML reconstruction and three from HTML-to-triple conversion. In these cases, question-relevant content or structural context is already missing or incorrectly represented before it reaches the code reasoning stage. The remaining 11 errors occur downstream, despite the required evidence being preserved in the cell triples, and primarily involve incorrect cell selection, operations, type handling, or answer matching.

Two representative cases illustrate these failure modes. In one TAT-QA example, the reconstructed HTML correctly preserves two table sections, but the HTML-to-triple conversion drops the parent-section context. Consequently, two distinct “Domestic-state” entries receive the same key, causing the generated program to select 57,299 instead of 3,313 and return 1.440 rather than 0.08. In AIT-QA_q-117, both the HTML and cell triples correctly preserve an absolute increase of 66 and a percentage increase of 23.2%, but the generated program selects the absolute-increase column and returns 66. The former illustrates an intermediate-representation error, whereas the latter is a downstream code-selection error.

Overall, the analysis suggests that the structured path does not have a single dominant bottleneck: both upstream table reconstruction and downstream program reasoning contribute substantially to its failures.

M.2 Retrieval-Threshold Sensitivity

The structured path uses a similarity threshold of 0.4 when retrieving question-relevant triples for code generation. To assess sensitivity to this choice, we evaluate thresholds of 0.3, 0.4, 0.5, and 0.6 on the complete evaluation sets while keeping all other components unchanged. Across all runs, the LLM calls for both keyword extraction and code generation use Qwen-Flash with temperature 0.2 and top-p 0.8 (maximum generation length at the API default).

As shown in Table 11, the macro-average structured-path accuracy varies only from 59.12 to 59.57 across the tested thresholds, a maximum

Variant	Main Modification	Avg.
Visual self-consistency		
Visual Reasoning Only	Single visual reasoning pass	81.51
Visual-SC@3	Three visual samples with majority voting	82.33
Simplified pipeline variants		
HTML Direct	Answer directly from reconstructed HTML	79.61
Triples Direct	Answer directly from cell triples	71.16
Image + Triples Direct	Joint multimodal direct answering	80.26
Triples + Visual Fusion	Remove code generation and execution	82.28
Image-to-Code Execution	Direct image-to-code generation	61.11
TriVTab (Ours)	Full structured execution and fusion	84.04

Table 9: Additional controlled comparisons. We report the unweighted average exact-match accuracy (%) across the five benchmarks.

Failure Stage	Cases	Share	Main Causes
Image-to-HTML reconstruction	6	30%	Headers, merged cells, or empty parsing
HTML-to-triple conversion	3	15%	Loss of section, year, or column context
Downstream code/reasoning	11	55%	Selection, operation, type, and matching errors

Table 10: Stage-wise manual analysis of 20 structured-path failures. Each example is assigned to the earliest stage sufficient to explain the final error. Main causes summarize recurring error patterns observed within each stage.

Retrieval Threshold	Avg. Accuracy
0.3	59.12
0.4	59.57
0.5	59.44
0.6	59.53

Table 11: Sensitivity of the structured reasoning path to the triple retrieval threshold on the complete evaluation sets. Accuracy is the unweighted average across the five datasets (%).

difference of 0.45 percentage points. The default threshold of 0.4 achieves the highest average accuracy (59.57), while the nearby settings remain close in performance. These results indicate that aggregate structured-path performance is relatively insensitive to the retrieval threshold within the evaluated range.

N Prompts

N.1 Triple Retrieval Prompt

```
# Role
You are an advanced Database Schema Linker and Query Optimizer. Your goal is to bridge the gap between a user's natural language query and a list of triple data.

# Data Structure
1. **Data Source (`data`)**: A list containing all the data, with each element being a tuple of `(row_key, col_key, value)`.
2. **Row Index List (`all_rows`)**: A list containing all possible `row_key` values in the data. Note: Each `row_key` itself may be a list containing hierarchical information (e.g., `["Header A", "Sub-header B"]`).
3. **Column Index List (`all_cols`)**: A list containing all
```

```
possible `col_key` values in the data. Note: Each `col_key` may also be a list.

# Your Task
You must analyze the query and extract "Search Filters" to retrieve relevant rows.
Perform the following steps using a **Chain of Thought** process:

1. **Analyze Column Semantics**: Look at the provided column names. Infer what kind of data each column likely holds (e.g., "Result" likely holds "Won/Lost", "Event" holds sport names).
2. **Decompose the Query**: Break the user's query into key constraints.
3. **Map & Expand (Crucial)**:
  - Map each constraint to the most appropriate column.
  - **Synonym Expansion**: The user often uses casual language (e.g., "gold medal", "best movie"). You MUST predict how this value might be stored in a standard database (e.g., "Champion", "Winner", "1st Place", "Academy Award").
  - Generate 2-3 likely database values for each keyword.

# Output Format
You must output a single block of text containing your **Reasoning** followed by the **JSON**.

Format:
```text
Reasoning: <Your step-by-step analysis here>

###JSON_START###
```JSON
{
  "filters": [
    {
      "column": "Target_Column",
      "keywords": ["original_term", "synonym1", "synonym2"]
    }
  ]
}
...

Constraints:
Do not make up columns. Use ONLY the provided list.

If the query mentions a specific value (e.g., "2007"), include it.

If the query implies a condition (e.g., "who won"), focus on the 'won' condition.

Few-Shot Examples

Example 1 (Handling Semantic Gap)

Table Columns: [["Event"], ["Athlete"], ["Result"], ["Time"]]
User Query: "Who took the gold in the 100m dash?"
```

Output:

Reasoning:

1. Analyze Columns: "Event" is for sports, "Result" is for rankings/medals.
2. Analyze Query: User wants "gold" in "100m dash".
3. Map and Expand:
 - (1) "100m dash": Maps to column "Event". Database value might be "100m", "100m Sprint".
 - (2) "took the gold": Maps to column "Result". User said "gold", but databases often store this as "Champion", "1st", "Winner", or "Gold Medal". I will include these synonyms to ensure retrieval success.

###JSON_START###

```JSON

```
{
 "filters": [
 {
 "column": ["Event"],
 "keywords": ["100m dash", "100m", "100m Sprint"]
 },
 {
 "column": ["Result"],
 "keywords": ["gold", "Champion", "Winner", "1st Place", "Gold Medal"]
 }
]
}
```

Example 2 (Handling Ambiguity)

Table Columns: [{"Year"}, {"Organization"}, {"Award"}, {"Work"}]

User Query: "Did she get a nomination for Hana Yori Dango?"

Output:

Reasoning:

1. **Analyze Column Semantics**:
  - "Work": The name of the drama or movie.
  - "Award": The specific name of the prize (e.g., "Best Actress").
  - "Organization": The entity giving the award.
  - Observation: There is NO specific "Result" or "Status" column (like "Won/Nominated") in this provided schema. This is tricky.
2. **Decompose the Query**:
  - Condition 1: The work is "Hana Yori Dango".
  - Condition 2: The status is "nomination".
3. **Map & Expand**:
  - "Hana Yori Dango" -> Maps to column "Work".
  - Expansion: I will search for the exact title "Hana Yori Dango".
  - "nomination" -> Where does this go?
    - Analysis: Since there is no "Status" column, the concept of "nomination" might be embedded within the "Award" column itself (e.g., "Best Actress (Nominated)") or the "Organization" column. Or, perhaps the user is asking to check if a row simply exists for this work.
    - Decision: The safest bet is to search for "nomination" or "nominee" within the "Award" column, as that's where award descriptors usually live.

###JSON\_START###

```JSON

```
{
  "filters": [
    {
      "column": ["Work"],
      "keywords": ["Hana Yori Dango"]
    },
    {
      "column": ["Award"],
      "keywords": ["nomination", "Nominee", "Nominated"]
    }
  ]
}
```

Current Task

Given Cols (all_cols): {col_names_json}

User Question: {user_question}

N.2 Code Generation Prompt (P_{code})

Role

You are a data analysis expert capable of generating precise Python code based on natural language questions. Your task is to generate query code for the given "list of triple data".

Data Structure

1. **Data Source** (`data`): A list containing all the data, with each element being a tuple of `(row\_key, col\_key, value)`.
2. **Row Index List** (`all\_rows`): A list containing all possible `row\_key` strings. Hierarchical levels are joined by a hyphen (e.g., "Header A-Sub-header B").
3. **Column Index List** (`all\_cols`): A list containing all possible `col\_key` strings (e.g., "Year 2024-Q1").

Goal

Answer the user's natural language question (`User Question`).

Critical Constraints (Must be adhered to)

1. **Prohibition of Pandas and numpy**: Only use Python native list comprehensions or loops.
2. **Exact Key-Value Match**: The `target\_row` and `target\_col` strings must **exactly match** one of the entries in `all\_rows` or `all\_cols`.
3. **Chain of Thought (CoT)**: Before writing the code, you must:
 - Analyze the metric/entity and the time/category dimension in the question.
 - Scan `all\_rows` and `all\_cols` to identify the specific concatenated strings.
 - Explicitly state the selected `target\_row` and `target\_col`.
4. **Strict Type Handling**:
 - Direct Math: All numerical values in data are stored as native types (int or float). Use them directly for mathematical operations (sum, average, etc.) without conversion.
 - Comparison & String Logic: For any equality checks (==) or string-based filtering, always wrap the value in str() first. This ensures consistency when comparing data against string-based row/column keys or user inputs.

Few-Shot Example

Given Rows (`all\_rows`):

```
[
  "Consolidated-Revenue",
  "Consolidated-Cost",
  "Consolidated-Average stage length (miles) (i)"
]
```

Given Cols (`all\_cols`):

```
[
  "Year 2014",
  "Year 2015"
]
```

Data Example:

```
[
  ("Consolidated-Revenue", "Year 2014", 5762),
  ("Consolidated-Average stage length (miles) (i)", "Year 2014", 1581)
]
```

User Question:

"What is the average stage length in 2014?"

Model Output:

Thinking Process

1. **Analyze Question**: The user is looking for "average stage length" for the period "2014".
2. **Identify Keys**:
 - Scanning `all\_rows`: The string "Consolidated-Average stage length (miles) (i)" contains the target metric.
 - Scanning `all\_cols`: The string "Year 2014" matches the target year.
3. **Conclusion**: I will filter the `data` list where the first element matches the row string and the second element matches the column string.

Python Code

```python

```
Defined target keys based on the metadata
target_row = "Consolidated-Average stage length (miles) (i)"
target_col = "Year 2014"
```

```
Filter data to find the value (data is a list of tuples)
Each tuple is (row_key, col_key, value)
answers = [x[2] for x in data if x[0] == target_row and x[1] == target_col]
```

if answers:

```
Print the first matching result
print(answers[0])
```

else:

```
print("No data found.")
```

Current Task

Given Rows (all\_rows): {row\_names\_json}

Given Cols (all\_cols): {col\_names\_json}

Figure 4: Triple retrieval prompt. Used in Stage 1 of triple subset retrieval to decompose the question into search filters with synonym-expanded keywords for downstream dense matching, as described in Appendix J.

```
Given Values (`The first row of data in the table as an example.`):
{data_example_json}
User Question: {user_question}
Please start thinking and writing code.
```

Figure 5: Code generation prompt ( $P_{code}$ ). Used by the code generator in the structured reasoning path to produce an executable Python program over the cell-triple representation (Section 3.3).

### N.3 Visual Path Prompt ( $P_v$ )

```
You are given an image of a table. Carefully analyze the data and
answer the question step by step.

First, read the table and extract all relevant information.
- Identify all column and row headers, and note any merged or multi-
 level cells.
- If the table has a complex structure (e.g., hierarchical headers),
 explicitly describe how you parse it.

Then, reason through the problem logically, showing your complete
thought process.
- If calculation is needed, show the exact formula or arithmetic
 steps, not just the final number.

Finally, provide the precise answer in a specific format.

Your response must contain two parts:
1. Reasoning: Your step-by-step thinking process.
2. Final Answer: The exact answer, enclosed within the tags <
 answer> and </answer>.

Rules for the final answer:
- If the answer is a number, just put the number (e.g., <answer>42</
 answer>).
- If the answer is a short phrase or entity, put it directly (e.g.,
 <answer>John Doe</answer>).
- If the answer requires multiple values, use a comma-separated list
 inside the tags (e.g., <answer>Paris, London, Berlin</answer
 >).
- If no answer can be found, output <answer>N/A</answer>.

Now, here is the question: {QUESTION}
```

Figure 6: Visual path prompt ( $P_v$ ). Used by the visual reasoning path to produce a chain-of-thought rationale and a normalized final answer directly from the table image, as described in Section 3.2.

### N.4 Image-to-HTML Prompt ( $P_{html}$ )

```
Recognize the table in this image and return an HTML-style
reconstructed table in text.

<image>
```

Figure 7: Image-to-HTML parsing prompt ( $P_{html}$ ). Used in the first stage of the structured reasoning path to convert the table image into an HTML representation, which is subsequently flattened into cell triples, as described in Section 3.3.

### N.5 Fusion Module Prompt ( $P_{fusion}$ )

```
You are an expert in answering questions based on tables. Please
carefully observe the table image and answer the following
question.

Question: {question}

To help you answer more accurately, two reference systems provide
the following analyses:
```

```
[Reference System A - Direct Visual Inference]
This system directly answers the question based on the table image.
Its answer is:
{vl_answer}

Its reasoning is:
{vl_reasoning}

[Reference System B - Code Inference]
This system parses the table into the following structured
representation. Note that the parsed structure may contain
errors, so you should rely on the table image as the primary
evidence:
{parsed_table_html}

The system then generates the following code to solve the question:
{code}

The result of code execution is:
{code_answer}

Please carefully verify the two reference systems against the table
image. Determine whether either system contains errors, and
then provide the final answer you think is correct.

Requirements:
1. Check whether the direct visual answer is consistent with the
 table image.
2. Check whether the parsed table correctly reflects the image.
3. Check whether the generated code correctly answers the question.
4. Check whether the code execution result is consistent with the
 table image.
5. Compare the two systems and identify possible errors in each.
6. Provide your final judgment.
7. Place the final answer within the <answer></answer> tags.
```

Figure 8: Multimodal fusion prompt ( $P_{fusion}$ ). Invoked when the two paths disagree; it instructs the fusion module to jointly examine the table image, both candidate answers, and their reasoning traces, and then output the final answer with a brief justification, as described in Section 3.4.

### N.6 Self-Reflection Baseline Prompt

```
You are an expert in answering questions based on tables. Please
carefully observe the table image and answer the following
question.

Question: {question}

To help you answer more accurately, one reference system provides
the following analysis:

[Reference System - Direct Visual Inference]
This system directly answers the question based on the table image.

Its answer is:
{vl_answer}

Its reasoning is:
{vl_reasoning}

Please carefully verify the reference system's reasoning and answer
against the table image. Determine whether the reasoning or
answer contains any errors, and then provide the final answer
you think is correct.

Requirements:
1. Check whether the reasoning is consistent with the table image.
2. Check whether the predicted answer is supported by the table
 image.
3. Identify possible errors in the reasoning process, if any.
4. Provide your final judgment.
5. Place the final answer within the <answer></answer> tags.
```

Figure 9: Self-Reflection baseline prompt. Used in the second pass of the Self-Reflection baseline, which asks the visual path to revisit its first-pass answer given the table image and its prior reasoning trace, as described in Appendix N.6.

## N.7 Question Type Classification Prompt

You are a multimodal table question-answering classifier. You will receive a table image and a question. Please classify the question into one of the following categories based on the reasoning process required to solve it:

A. Direct retrieval type: The answer can be directly located in the cell without the need for calculation or filtering.  
Example: "What was the sales figure in 2019?"

B. Conditional Filtering Type: Rows/columns need to be filtered first based on conditions, and then values are taken.  
Example: "What is the profit corresponding to the product with the highest sales volume?"

C. Aggregation statistics type: Requires aggregation operations such as sum, count, average, max, and min.  
Example: "How many countries have won more than 10 gold medals?"

D. Comparative Judgment Type: It requires comparing the size relationship between two or more values.  
Example: "Which quarter had higher profits?"

E. Arithmetic reasoning type: It requires mathematical operations such as difference, ratio, growth rate, and multiple.  
Example: "What percentage did it increase from 2020 to 2021?"

F. Multi-hop Combinatorial Type: It requires multiple consecutive steps (such as filtering + retrieval, filtering + aggregation, comparison + arithmetic, etc. in combination).  
Example: "What is the average profit of the top two products in terms of sales volume?"

Classification rules:

1. Determine based on the core reasoning steps required from the question to the answer, rather than the surface wording of the question.
2. If multiple types are involved, select the most core or highest difficulty type in the reasoning chain. However, if it is truly a combination of multiple steps and cannot be classified into a single type, choose F.

Please strictly follow the following JSON format for output. Do not output any other content:

```
{"category": "A", "reason": "A one-sentence explanation of the reason."}
```

Question: {question}  
Table\_Image:

Figure 10: Question type classification prompt. Used by Qwen3-VL-Flash as an automatic classifier to assign each question to one reasoning category, as described in [Appendix E](#).