UNDERSTANDING ERRORS IN LLM-BASED QUESTION ANSWERING OVER IMPERFECT TABLES

Anonymous authors
Paper under double-blind review

ABSTRACT

We investigate error discovery and handling in question answering over imperfect tables through controlled studies across three large language models (LLMs) on human-reviewed RADAR-T examples. Answering questions over these tables requires handling errors that can affect the answer. We vary row order and compare original, error-marked, and repaired tables to test whether discovery depends on where errors appear and whether providing their locations is sufficient for accurate question answering. First, reordering rows changes error discovery even when the table contents and gold answer remain unchanged. Complete discovery is higher for back than front placements and, averaged over the tested mean positions, for compact than widely spaced layouts. Second, providing verified error locations alone is insufficient for accurate QA, leaving a substantial accuracy gap between error-marked and repaired tables. Providing tables with human-reviewed repairs already applied raises code-assisted QA accuracy by 39.0–59.1 percentage points over the error-marked tables across the three systems. GBDI, a simple workflow, puts these findings into practice by combining error discovery across shuffled table views with explicit guidance for verifying and handling the reported errors. On RADAR-T, GBDI raises observed QA accuracy by 3.8–18.5 percentage points over a code-agent baseline across five systems. These results highlight the importance of both reliable error discovery and effective error handling in question answering over imperfect tables. Our anonymous repository is available at https://anonymous.4open.science/r/GBDI-ICLR-2027-85BD/

1 INTRODUCTION

Tabular data organize records into rows and features into columns, with each cell representing a feature value whose interpretation depends on the column semantics and data type (Wickham, 2014; Hulsebos et al., 2019). This structured representation supports data analysis across domains such as healthcare (Johnson et al., 2023), finance (Chen et al., 2021), and materials design (Choudhary et al., 2020). Table question answering (TQA) (Pasupat & Liang, 2015; Herzig et al., 2020) enables users to access and analyze tabular data through natural-language questions, which often involve operations such as filtering, comparison, and aggregation (Wang et al., 2024; Zhu et al., 2021). Recent methods use large language models (LLMs) to interpret questions and reason over tables, often with the help of executable code or explicit table operations (Jiang et al., 2023; Wang et al., 2024; Gao et al., 2023). Progress in reasoning over a supplied table, however, does not by itself establish reliability when its contents are imperfect. Real-world tables can contain errors such as missing values and inconsistent formats (Gu et al., 2025). Understanding how these errors affect LLM-based analysis is therefore essential for reliable TQA beyond clean inputs.

Answering questions over imperfect tables involves two connected demands: discovering errors that matter to the question and deciding how to handle them before computing an answer. LLMs can critique and revise outputs through iterative feedback (Madaan et al., 2023), but reasoning corrections remain unreliable without external feedback (Huang et al., 2024). Neither result establishes reliable error handling in tables. In prose, word order contributes directly to meaning; in a table, a value must be interpreted through its column, its relations to other values, and the requested computation. For many questions, the displayed order of records is incidental. RADAR shows that data imperfections substantially degrade LLM-based analysis (Gu et al., 2025), motivating a closer question: where do

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

these systems struggle—in discovering relevant errors, or in translating their discovery into correct answers? We investigate two corresponding challenges.

Challenge 1: Discovering Errors. LLMs receive textual table representations as token sequences, raising the possibility that error discovery depends on presentation order even when that order is irrelevant to the answer. Prior work shows that semantically equivalent table arrangements can change final predictions (Liu et al., 2024b; Dong et al., 2026). We ask whether this sensitivity also affects which errors are discovered before answering. Specifically, can the same errors be overlooked when their rows appear earlier in the table or farther apart? We call rows containing errors that can affect the answer *critical rows*. Reliable systems should discover these rows regardless of their placement whenever reordering preserves the meaning of the question and gold answer.

Challenge 2: Handling Errors. Discovering an erroneous value does not determine a valid replacement or how that value should enter the computation. A correction must be supported by the available table evidence; when a required value cannot be recovered, the affected record may need to be excluded under the task rules. An error notification therefore leaves substantive decisions unresolved. We ask whether explicitly marking the relevant errors is enough to recover accurate answers, or whether a substantial gap remains between knowing where the errors are and answering from a table in which they have already been handled.

To investigate these challenges, we conduct two controlled studies on human-reviewed RADAR-T instances across three systems. The first uses answer-preserving changes in critical-row placement, holding all row contents, the question, and the gold answer fixed. On 60 instances, we vary where critical rows appear; on the 59 instances with multiple critical rows, we also vary their spacing at controlled mean positions. Complete discovery is higher when critical rows are placed near the back rather than the front. Averaged over the tested mean positions, Wide layouts yield lower complete discovery rates than Compact layouts. Thus, answer-preserving changes in presentation can change whether relevant errors are discovered.

The second study compares answers on original tables, tables with verified errors marked, and tables with reviewed repairs already applied, using 154 reviewed instances. Error-marked inputs yield smaller and uncertain changes in accuracy, whereas providing repaired tables yields much larger gains for code-assisted QA. Across the three systems, providing repaired rather than marked tables raises accuracy by 39.0–59.1 percentage points. Together, the studies reveal both a sensitivity in error discovery and a substantial difficulty that remains after error locations are supplied.

To illustrate the practical implications of these findings, we combine inspection across shuffled table views with explicit guidance for verifying and handling reported errors in a simple workflow, Geometry-Balanced Discovery and Intervention (GBDI). The workflow aggregates candidate errors from multiple views, checks them against the visible table, and applies supported actions before final QA. On all 313 RADAR-T instances, GBDI raises observed accuracy over the Code Agent baseline by 3.8–18.5 percentage points across five systems; paired 95% confidence intervals exclude zero for four. Separate controls compare shuffled and repeated inspections at the same call budget and vary intervention guidance while holding the discovery evidence fixed.

Our contributions are:

- We show that the discovery of question-relevant errors depends on their placement, even when row permutations preserve the table contents and gold answer.
- We distinguish marked inputs from inputs with completed repairs, showing that providing error locations without repairs leaves a substantial gap in final QA accuracy.
- We demonstrate a practical application of these findings through GBDI, a simple workflow that combines varied table views with explicit error-handling guidance to improve final QA.

2 PROBLEM FORMULATION

Following RADAR (Gu et al., 2025), a base task is a pair (T_s, Q) of a clean source table and a natural-language question. RADAR introduces one of five artifact types—missing values, bad values, outliers, formatting errors, and logical inconsistencies—into a table T derived from T_s; clean variants omit this perturbation. Each TQA instance is then a triple (T, Q, A), where T is the resulting observed table and A is the gold answer computed from the corresponding recovered table. The system receives T and Q and predicts $\hat{A}$, which we evaluate using the RADAR answer-scoring rules.

2.1 CRITICAL CELLS

For instance i, let T_i denote its observed table and c a cell in it. Let E_i denote the set of cells in T_i that contain a human-verifiable error, and let $D_i(Q_i)$ denote the set of cells whose values can affect the answer to Q_i, for example through filtering, comparison, or aggregation. The set of critical cells is their intersection:

$$\mathcal{C}_i = E_i \cap D_i(Q_i). \tag{1}$$

A critical cell is both erroneous and relevant to the current question.

A table may contain errors that do not affect the answer to Q_i. These cells belong to E_i but not to $\mathcal{C}_i$. Conversely, a cell may be relevant to the question but contain no error; such a cell belongs to $D_i(Q_i)$ but not to $\mathcal{C}_i$. Our error-discovery analysis focuses on these critical cells and the rows that contain them.

Because our row-order experiments operate on rows, we also define a critical row as any row containing at least one critical cell:

$$\mathcal{R}_i = \{r \in T_i \ : \ \exists\, c \in r \text{ with } c \in \mathcal{C}_i\}\,. \tag{2}$$

Thus, $\mathcal{C}_i$ identifies the error locations that matter for the question, while $\mathcal{R}_i$ identifies the rows that contain them.

2.2 INTERVENTION

Discovering a critical row does not by itself determine how to handle its errors. For each critical row $r \in \mathcal{R}_i$, let $a_i^{\mathrm{H}}(r)$ denote its human-reviewed intervention:

$$a_i^{\mathrm{H}}(r) = \begin{cases} \textsc{Edit}, & \text{if every required correction can be reliably recovered,} \\ \textsc{Drop}, & \text{otherwise.} \end{cases} \tag{3}$$

EDIT replaces the implicated erroneous values with supported corrections. DROP excludes the affected row from the computation when at least one required correction cannot be reliably recovered.

Human-reviewed reference. To evaluate discovery and intervention against a common reference, we review 154 RADAR-T instances. Reviewers inspect each table and question to identify critical cells and their containing rows, and verify the supported EDIT replacements or DROP decisions. These annotations provide reference rows for discovery evaluation in Section 3 and the error markings and completed repairs compared in Section 4. Appendix A describes the review procedure, coverage, and subset selection.

2.3 ANSWER-PRESERVING ROW PERMUTATIONS

To study the effect of row order, let π denote a permutation of the data rows of T_i with the header fixed. We consider questions that do not refer directly to row positions, such as the first or last row. Values with semantic meaning, such as dates or ranks, remain attached to their original rows.

Let $\mathrm{Answer}(Q, T)$ denote the gold answer for question Q and observed table T, computed from the corresponding recovered table. Under these conditions, changing the displayed row order preserves this gold answer:

$$\mathrm{Answer}(Q_i, \pi(T_i)) = \mathrm{Answer}(Q_i, T_i) = A_i. \tag{4}$$

The critical cells also remain unchanged; only their displayed row positions change. We can therefore vary where critical rows appear while holding the underlying data, question, and gold answer fixed.

2.4 Complete Discovery Rate

For a fixed system–mode pair, let $\widehat{\mathcal{R}}_{i,\pi}$ denote the nominated row-ID set from view π, with valid IDs mapped to canonical row identities from the original table. For a collection of evaluated views $\mathcal{V}$ with nonempty critical-row sets, the complete discovery rate (CDR) is

$$\mathrm{CDR}(\mathcal{V}) = \frac{100}{|\mathcal{V}|} \sum_{(i,\pi)\in\mathcal{V}} \mathbf{1}\left[\mathcal{R}_i \subseteq \widehat{\mathcal{R}}_{i,\pi}\right]. \tag{5}$$

A view counts as successful only if the system discovers every critical row in the reference set. Missing any critical row counts as a discovery failure. CDR measures complete coverage, so additional nominated rows do not reduce the score. Invalid submissions are treated as empty predictions. Appendix B.6 complements CDR with row-level recall, precision, and F1, the number of nominated rows, and the false-positive rate on instances with no critical rows.

In Section 3, we use these controlled row permutations to study whether the position and dispersion of critical rows affect error discovery. In Section 4, we instead compare answers obtained from the original table, a table with reviewed errors marked, and a table with reviewed repairs applied.

3 Discovering Errors: Geometric Sensitivity

Imperfect TQA requires a system to discover the critical rows that contain errors capable of changing the answer. For questions that do not depend on displayed row order, a row permutation changes neither the critical-row set nor the gold answer. We test whether systems respect this invariance by moving the same human-reviewed critical rows through answer-preserving permutations. We vary two properties of their placement: position and, for instances with multiple critical rows, dispersion.

3.1 Setup: Position and Dispersion

We describe critical-row placement by its average position in the table and by the spacing between critical rows. For instance i, let $n_i \geq 2$ be the number of data rows and $k_i = |\mathcal{R}_i| \geq 1$ the number of critical rows. We index the critical rows by $j \in \{1, \ldots, k_i\}$, independently of their displayed order. In a row-permuted view π, let x_{ij}^{π} be the zero-based displayed row index of critical row j divided by $n_i - 1$. The first and last rows thus have normalized positions 0 and 1, respectively.

Position. We measure the average location of the critical rows by their mean normalized position:

$$\mu_{i,\pi} = \frac{1}{k_i} \sum_{j=1}^{k_i} x_{ij}^{\pi}. \tag{6}$$

Smaller values indicate that the critical rows lie nearer the beginning of the table on average; larger values indicate that they lie nearer the end.

Dispersion. For $k_i \geq 2$, we average the distance from each critical row to its nearest other critical row:

$$\sigma_{i,\pi} = \frac{1}{k_i} \sum_{j=1}^{k_i} \min_{\substack{1\leq j'\leq k_i \\ j'\neq j}} \left| x_{ij}^{\pi} - x_{ij'}^{\pi} \right|. \tag{7}$$

Larger values indicate greater average distance to the nearest other critical row. Figure 1 illustrates both quantities. The position experiment varies $\mu_{i,\pi}$, whereas the dispersion experiment varies $\sigma_{i,\pi}$ while holding $\mu_{i,\pi}$ fixed within each comparison.

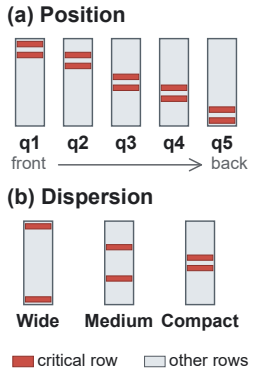

Figure 1: Position and dispersion.

3.2 Experimental Design

We use 60 instances from the human-reviewed reference, with 12 from each of the five error types, covering 29 base tasks and 21 source tables. Instances are selected using annotation metadata and layout feasibility (Appendix A.3). Each comparison uses the same question, prompt, cell values, and critical rows.

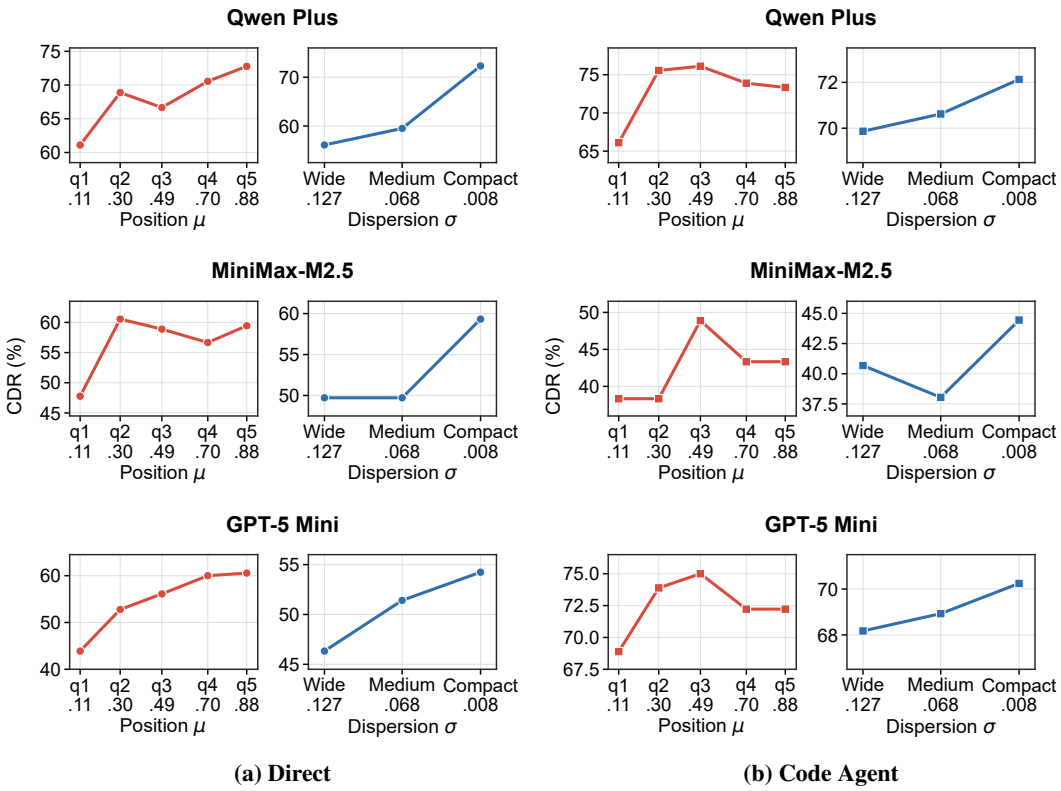

Figure 2: Complete discovery rate under (a) Direct and (b) Code Agent. In each subfigure, rows correspond to systems; the left column varies position (q1–q5, front to back), and the right column varies dispersion (Wide to Compact). Tick labels report mean realized μ or σ.

Position experiment. All 60 instances are used. For each instance, we keep the critical rows together as a contiguous block and vary their mean position across five bands, q1–q5, from the front to the back of the table (Figure 1(a)). The remaining rows retain their relative order as the block moves. We repeat these five placements under three fixed orders of the remaining rows, yielding $60 \times 3 \times 5 = 900$ views per system–mode pair.

Dispersion experiment. The dispersion experiment covers the 59 instances with multiple critical rows. The remaining instance has a single critical row, for which spacing between critical rows cannot be measured. For each instance, we construct three fixed orders of the remaining rows and test three target mean positions: 25%, 50%, and 75%. At each target position and fixed row order, we compare Compact, Medium, and Wide layouts with increasing spacing between critical rows (Figure 1(b)). The individual critical-row positions vary across these layouts, while their mean position remains fixed within each comparison. This gives $59 \times 3 \times 3 \times 3 = 1{,}593$ views per system–mode pair. The mean realized σ is 0.008, 0.068, and 0.127 for Compact, Medium, and Wide, respectively. Appendix B.2 describes the placement procedure.

Systems and inference. We evaluate Qwen Plus, MiniMax-M2.5, and GPT-5 Mini using the Direct and Code Agent modes adapted from RADAR (Gu et al., 2025). In this experiment, both modes identify rows that may contain errors relevant to the question. Direct reports these rows in one response. The Code Agent receives the same complete table and question and may inspect the table with read-only Python, using up to five responses including its final submission.

Figure 2 reports CDR for both experiments and inference modes. Numerical results and uncertainty estimates appear in Appendices C.1 and E. We also evaluate position and dispersion effects on missingness instances constructed from WikiSQL following the RADAR missingness setting (Appendix C.3).

3.3 RESULTS

CDR is higher when critical rows appear near the end of the table (q5) than near the beginning (q1). With Direct, Qwen rises from 61.1% to 72.8%, MiniMax from 47.8% to 59.4%, and GPT from 43.9% to 60.6% (Figure 2(a)). The Code Agent shows the same pattern, with smaller increases for all three systems (Figure 2(b)).

Wide layouts yield lower CDR than Compact layouts for all three systems using Direct (Figure 2(a)). Averaged over the three tested values of μ, the decrease is 16.2 points for Qwen, 9.6 for MiniMax, and 7.9 for GPT. The same direction holds at each tested μ for every system under Direct (Appendix C.2).

These controlled comparisons identify presentation sensitivity in discovering question-relevant errors while holding the question, table contents, and gold answer fixed.

4 HANDLING ERRORS: DISCOVERY ALONE IS INSUFFICIENT

Even when relevant errors are discovered, their locations do not specify how they should be handled. The model must still determine supported corrections or exclusions. We therefore compare error marking with completed, human-reviewed repairs to examine whether supplying error locations is sufficient for accurate QA.

4.1 EXPERIMENTAL SETUP

We evaluate Qwen Plus, MiniMax-M2.5, and GPT-5 Mini in both Direct and Code Agent modes on all 154 instances in the human-reviewed reference. Under each table condition, the model answers the question, and we evaluate final-answer accuracy.

We compare three table states, with their Direct prompts shown in Figure 3. Raw uses the original table. Discovery (Disc, error-marked) wraps each reviewed critical cell in `<<ERROR:value>>`, retaining the original value inside the marker. Action (Act, repaired) applies the reviewed replacements and removes the reviewed DROP rows.

Shared task	Data: {table} Based on the given table, answer the following question: {question}
Raw	*No added notice.*
Discovery	The values marked <<ERROR:...>> have been verified as erroneous and relevant to the question; the text inside each marker is the original erroneous value, not a correction. Treat these markings as facts when answering: correct a marked value when the table supports a reliable replacement, or exclude its row if it cannot be repaired.
Action	The table below has already been repaired for this question. All reviewed errors relevant to the answer have been corrected, and rows that could not be repaired have been removed. Treat the supplied table as final and answer the question without making further changes.

Figure 3: The shared Direct task template (top) and three input conditions (below). Each notice, when present, precedes the task in the actual prompt. `{table}` receives the corresponding table; `{question}` is unchanged. Code Agent uses the same notices with its own task template. Full prompts appear in Appendix F.

Discovery asks the model to handle the marked errors, whereas Action provides a repaired table for answering. All conditions use the same question; none supplies the gold answer.

For instance i, let $\mathcal{P}_i^{\mathrm{H}}$ denote its repair plan: supported replacements for erroneous cell values in EDIT rows and the DROP rows to remove. With Mark and Repair denoting the marking and repair operations, respectively, the three table states are:

$$\begin{aligned} T_i^{\mathrm{Raw}} &= T_i, \\ T_i^{\mathrm{Disc}} &= \mathrm{Mark}(T_i, \mathcal{C}_i), \\ T_i^{\mathrm{Act}} &= \mathrm{Repair}(T_i, \mathcal{P}_i^{\mathrm{H}}). \end{aligned} \tag{8}$$

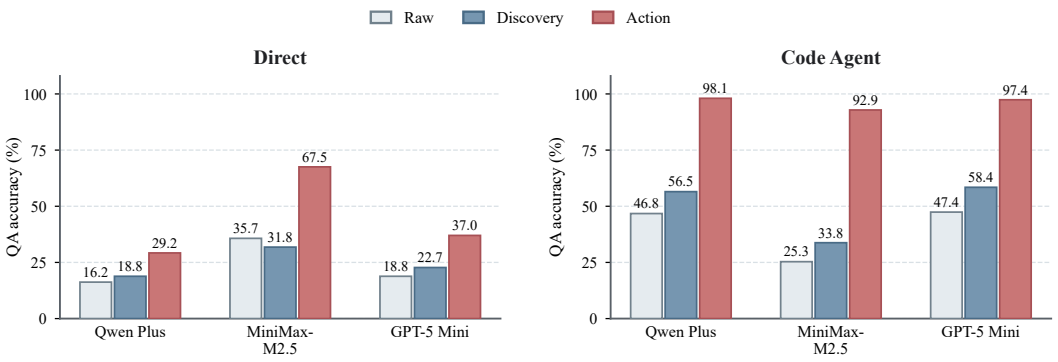

Figure 4: Final-answer accuracy (%) on 154 reviewed RADAR-T instances. RAW uses the original table, DISCOVERY (error-marked) marks critical cells, and ACTION (repaired) provides the table with reviewed cell corrections and DROP row removals already applied. Paired intervals are reported in Appendix E.

4.2 RESULTS

Providing repaired rather than error-marked tables raises observed QA accuracy across all three systems and both inference modes (Figure 4). The increase is particularly pronounced under the Code Agent: 39.0–59.1 percentage points, with all paired intervals excluding zero. By comparison, changes from error marking relative to Raw inputs are smaller and statistically uncertain. Detailed results and paired intervals are reported in Appendix E.3, Table 18.

These results distinguish discovering relevant errors from resolving them for question answering. Providing reviewed repairs yields higher QA accuracy than providing verified error locations alone, motivating an explicit intervention stage after discovery.

5 GEOMETRY-BALANCED DISCOVERY AND INTERVENTION

GBDI puts the two findings into practice by combining discovery across shuffled table views with explicit guidance for verifying and handling candidate errors. We evaluate how view randomization affects discovery, how intervention guidance affects QA with discovery reports held fixed, and how the complete workflow performs on final QA.

5.1 GBDI WORKFLOW

Multi-view discovery. As shown in Figure 5, GBDI constructs M views by independently sampling row permutations uniformly, keeping columns and cell values unchanged. Each view is inspected in a separate Direct context with the same question, prompt, and decoding configuration. Reported candidate errors are mapped back to the original rows and merged into a single report, termed the union ledger. We call this discovery procedure Random-M, where M is a hyperparameter set to 5 by default.

Guided intervention. The Code Agent receives the original table, question, union ledger, and Intervention Skill: prompt instructions for verifying candidates against the question and visible table evidence before choosing actions. The Skill provides guidance on value recovery, formatting, and execution checks. It instructs the agent to use EDIT when the evidence supports a unique correction, DROP when a question-relevant row is verified to be unusable and no supported correction is available, and NO-OP when an intervention is not justified.

The agent applies the selected actions to a working copy and then answers the original question in the same conversation and Python state. Appendices B and F provide the prompts and implementation details.

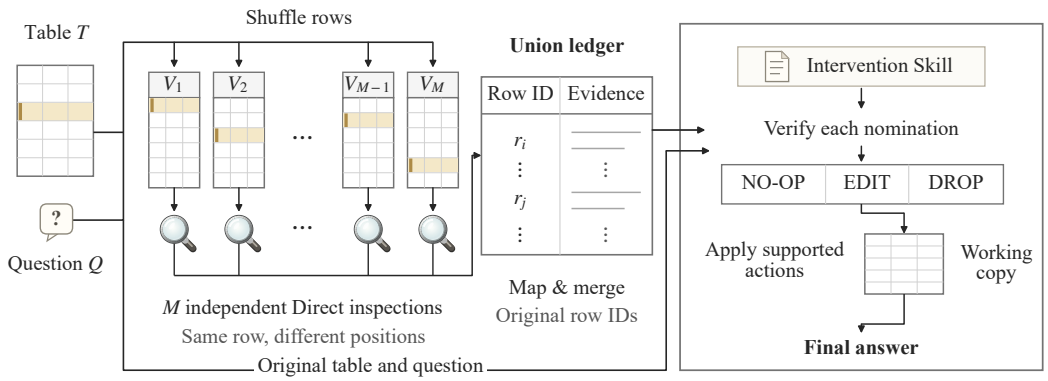

Figure 5: GBDI workflow. M independently shuffled views of the input table are inspected in separate Direct contexts. Candidate errors and supporting evidence are mapped to the original rows and merged. The Code Agent verifies each nomination, applies supported actions to a working table, and computes the final answer. The highlighted row illustrates a change in position across views.

5.2 GBDI Evaluation

We evaluate Qwen Plus, GLM-4.5-Air, MiniMax-M2.5, GPT-5 Mini, and DeepSeek-V4.1-Flash on all 313 RADAR-T instances from 53 base tasks and 27 source tables. The Direct and Code Agent baselines use the released row order. With the default $M = 5$, GBDI adds five inspection calls and reserves one response for intervention. We compare Random-5 with five independent inspections of the original row order (Repeated-5); paired discovery and QA results are reported in Appendices D.2 and E.4. For QA, paired 95% confidence intervals use source-table cluster resampling (Field & Welsh, 2007); prompts and execution details appear in Appendix F.

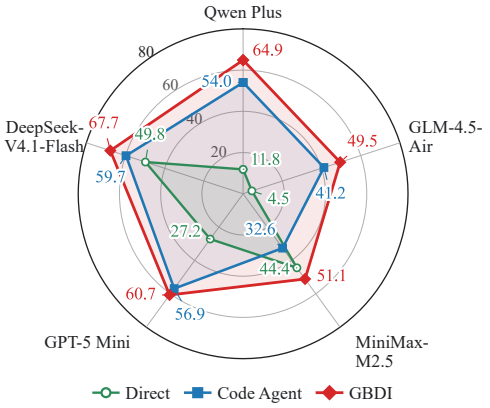

Figure 6: QA accuracy (%) on 313 RADAR-T instances. Numerical results and paired CIs are in Appendix E.4, Table 20.

QA accuracy. Relative to Code Agent, GBDI raises QA accuracy point estimates by 3.8–18.5 percentage points across all five systems (Figure 6).

Intervention guidance. With the Qwen Plus Random-5 discovery reports held fixed, the Intervention Skill improves QA by 6.4 percentage points over general guidance on allowed operations. Removing formatting rules reduces accuracy; the effects of removing recovery rules or execution checks are less certain (Appendices D.2 and D.3).

QA after complete discovery. Even when all critical rows are discovered, incorrect answers remain across Qwen, GLM, and MiniMax. Among instances with complete discovery, replacing generic guidance with Skill while keeping the discovery reports fixed changes more answers from incorrect to correct than from correct to incorrect (Appendix D.4).

Sensitivity to view count. For Qwen Plus, CDR shows diminishing returns as M increases to five, while final QA at $M = 5$ improves over using the same Skill without discovery reports. Individual QA increments remain uncertain (Appendices D.5 and D.6).

6 Related Work

LLMs in TQA. StructGPT retrieves structured information through specialized interfaces, while Chain-of-Table constructs intermediate tables through successive reasoning operations (Jiang et al., 2023; Wang et al., 2024). Earlier models use joint text–table representations, cell selection and aggregation, or execution-oriented pretraining (Herzig et al., 2020; Yin et al., 2020; Liu et al., 2022). These approaches develop representations or reasoning pipelines for deriving answers from supplied tables. We study how question-relevant errors are discovered and handled, using RADAR's Direct and Code Agent baselines for controlled comparisons within the same systems and inference interfaces.

Robustness for LLMs. General LLM robustness studies examine adversarial prompt perturbations (Zhu et al., 2024) and distractions from irrelevant context (Shi et al., 2023). For TQA, RobuT evaluates question and table perturbations and uses adversarial data augmentation for model fine-tuning (Zhao et al., 2023). FREB-TQA tests retrieval and reasoning robustness, including the placement of answer-containing rows, and investigates fine-tuning on transposed tables (Zhou et al., 2024). AutoDCWorkflow generates data-cleaning workflows for a specified analysis purpose; its benchmark evaluates answer, data, and workflow quality (Li et al., 2025). Our controlled studies examine the discovery and handling of erroneous cells relevant to the question. We build on RADAR's imperfect-table setting, which requires supported corrections or record exclusions (Gu et al., 2025), and distinguish the effects of marking verified errors from those of supplying completed repairs.

Order Sensitivity in LLMs. TableFormer addresses permutation robustness through a pretrained table–text encoder (Yang et al., 2022). Rethinking Tabular Data Understanding with Large Language Models (Liu et al., 2024b) examines table structure variations and combines textual and symbolic reasoning through answer aggregation. Long-context studies examine how evidence position and spacing affect retrieval and reasoning (Liu et al., 2024a; Tian et al., 2025). The Power of Order (Dong et al., 2026) evaluates random row and column permutations and introduces a gradient-based attack that searches for permutations that degrade final-answer alignment. Its objective is to expose answer failures through adversarial inputs. Our target is the discovery of erroneous cells that matter to a question: the model must assess both whether a cell value is erroneous and whether that error is relevant to the requested computation. We measure this discovery explicitly and separately examine the QA gap between supplied error locations and completed repairs.

7 Limitations

First, our controlled diagnostics use imperfect tables, where explicit rows and columns support precise error localization. For the questions studied here, row permutations preserve the gold answer, enabling controlled tests of error discovery as row placement varies. Extending these diagnostics to text, images, or multimodal inputs requires definitions of question-relevant errors, answer-preserving transformations, and supported interventions, as well as reliable evaluation annotations.

Second, our experiments access LLMs through black-box interfaces, so connecting the observed effects to specific internal representations requires further investigation. Combining the controlled diagnostics with analyses of internal activations could help explain how models represent error evidence and use it during discovery, intervention, and answer generation.

8 Conclusion

We investigated error discovery and error handling in LLM-based TQA over imperfect tables. We used answer-preserving changes in row position and dispersion to probe error discovery under controlled conditions. Complete discovery is higher for back than front placements and, averaged over the tested mean positions, for Compact than Wide layouts. Providing verified error locations alone leaves a substantial QA accuracy gap relative to tables with reviewed repairs already applied. As a practical application of these findings, GBDI combines discovery across shuffled table views with guidance for verifying and handling reported errors. It achieves higher observed QA accuracy than both Direct and Code Agent across all five systems. These findings highlight two complementary challenges: discovering question-relevant errors and handling them using the available evidence.

AI USE STATEMENT

We used LLMs to check grammar, format LaTeX, and clarify technical descriptions in this manuscript. The scientific contributions and conclusions are based on research conducted by the authors.

ETHICS STATEMENT

We confirm that this research complies with all applicable ethical guidelines and does not present any ethical issues.

REPRODUCIBILITY STATEMENT

Appendix A documents the human-reviewed reference and annotation procedure. Appendix B specifies the experimental protocols, model configurations, and scoring rules. Appendix E describes confidence interval estimation, and Appendix F provides the prompts and output schemas used in our experiments. Code is available at `https://anonymous.4open.science/r/GBDI-ICLR-2027-85BD/`.

REFERENCES

Zhiyu Chen, Wenhu Chen, Charese Smiley, Sameena Shah, Iana Borova, Dylan Langdon, Reema Moussa, Matt Beane, Ting-Hao Huang, Bryan Routledge, and William Yang Wang. FinQA: A dataset of numerical reasoning over financial data. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 3697–3711, 2021. doi: 10.18653/v1/2021.emnlp-main.300.

Kamal Choudhary, Kevin F. Garrity, Andrew C. E. Reid, Brian DeCost, Adam J. Biacchi, Angela R. Hight Walker, Zachary Trautt, Jason Hattrick-Simpers, A. Gilad Kusne, Andrea Centrone, Albert Davydov, Jie Jiang, Ruth Pachter, Gowoon Cheon, Evan Reed, Ankit Agrawal, Xiaofeng Qian, Vinit Sharma, Houlong Zhuang, Sergei V. Kalinin, Bobby G. Sumpter, Ghanshyam Pilania, Pinar Acar, Subhasish Mandal, Kristjan Haule, David Vanderbilt, Karin Rabe, and Francesca Tavazza. The joint automated repository for various integrated simulations (JARVIS) for data-driven materials design. *npj Computational Materials*, 6(1), 2020. doi: 10.1038/s41524-020-00440-1.

Xinshuai Dong, Haifeng Chen, Xuyuan Liu, Shengyu Chen, Haoyu Wang, Shaoan Xie, Kun Zhang, and Zhengzhang Chen. The power of order: Fooling LLMs with adversarial table permutations. *arXiv preprint arXiv:2605.00445*, 2026.

C. A. Field and A. H. Welsh. Bootstrapping clustered data. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 69(3):369–390, 2007. doi: 10.1111/j.1467-9868.2007.00593.x.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: Program-aided language models. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 10764–10799. PMLR, 23–29 Jul 2023. URL `https://proceedings.mlr.press/v202/gao23f.html`.

Ken Gu, Zhihan Zhang, Kate Lin, Yuwei Zhang, Akshay Paruchuri, Hong Yu, Mehran Kazemi, Kumar Ayush, A. Ali Heydari, Maxwell A. Xu, Girish Narayanswamy, Yun Liu, Ming-Zher Poh, Yuzhe Yang, Mark Malhotra, Shwetak Patel, Hamid Palangi, Xuhai Xu, Daniel McDuff, Tim Althoff, and Xin Liu. RADAR: Benchmarking language models on imperfect tabular data. In *Advances in Neural Information Processing Systems*, volume 38. Curran Associates, Inc., 2025. doi: 10.52202/085713-3699.

Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Eisenschlos. TaPas: Weakly supervised table parsing via pre-training. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 4320–4333, 2020. doi: 10.18653/v1/2020.acl-main.398.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. In *International Conference on Learning Representations*, pp. 32808–32824, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/8b4add8b0aa8749d80a34ca5d941c355-Paper-Conference.pdf.

Madelon Hulsebos, Kevin Hu, Michiel Bakker, Emanuel Zgraggen, Arvind Satyanarayan, Tim Kraska, Çağatay Demiralp, and César Hidalgo. Sherlock: A deep learning approach to semantic data type detection. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1500–1508, 2019. doi: 10.1145/3292500.3330993.

Jinhao Jiang, Kun Zhou, Zican Dong, Keming Ye, Xin Zhao, and Ji-Rong Wen. StructGPT: A general framework for large language model to reason over structured data. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 9237–9251, 2023. doi: 10.18653/v1/2023.emnlp-main.574.

Alistair E. W. Johnson, Lucas Bulgarelli, Lu Shen, Alvin Gayles, Ayad Shammout, Steven Horng, Tom J. Pollard, Sicheng Hao, Benjamin Moody, Brian Gow, Li-wei H. Lehman, Leo A. Celi, and Roger G. Mark. MIMIC-IV, a freely accessible electronic health record dataset. *Scientific Data*, 10(1), 2023. doi: 10.1038/s41597-022-01899-x.

Lan Li, Liri Fang, Bertram Ludäscher, and Vetle I. Torvik. AutoDCWorkflow: LLM-based data cleaning workflow auto-generation and benchmark. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 7766–7780, 2025. doi: 10.18653/v1/2025.findings-emnlp.410.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024a. doi: 10.1162/tacl_a_00638.

Qian Liu, Bei Chen, Jiaqi Guo, Morteza Ziyadi, Zeqi Lin, Weizhu Chen, and Jian-Guang Lou. TAPEX: Table pre-training via learning a neural SQL executor. In *International Conference on Learning Representations*, 2022.

Tianyang Liu, Fei Wang, and Muhao Chen. Rethinking tabular data understanding with large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 450–482, 2024b. doi: 10.18653/v1/2024.naacl-long.26.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-Refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://papers.nips.cc/paper_files/paper/2023/hash/91edff07232fb1b55a505a9e9f6c0ff3-Abstract-Conference.html.

Panupong Pasupat and Percy Liang. 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)*, pp. 1470–1480, 2015. doi: 10.3115/v1/P15-1142.

Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed H. Chi, Nathanael Schärli, and Denny Zhou. Large language models can be easily distracted by irrelevant context. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 31210–31227. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/shi23a.html.

Runchu Tian, Yanghao Li, Yuepeng Fu, Siyang Deng, Qinyu Luo, Cheng Qian, Shuo Wang, Xin Cong, Zhong Zhang, Yesai Wu, Yankai Lin, Huadong Wang, and Xiaojiang Liu. Distance between relevant information pieces causes bias in long-context LLMs. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 521–533, 2025. doi: 10.18653/v1/2025.findings-acl.28.

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. Chain-of-Table: Evolving tables in the reasoning chain for table understanding. In *The Twelfth International Conference on Learning Representations*, 2024.

Hadley Wickham. Tidy data. *Journal of Statistical Software*, 59(10), 2014. doi: 10.18637/jss.v059.i10.

Jingfeng Yang, Aditya Gupta, Shyam Upadhyay, Luheng He, Rahul Goel, and Shachi Paul. TableFormer: Robust transformer modeling for table-text encoding. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 528–537, 2022. doi: 10.18653/v1/2022.acl-long.40.

Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. TaBERT: Pretraining for joint understanding of textual and tabular data. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 8413–8426, 2020. doi: 10.18653/v1/2020.acl-main.745.

Yilun Zhao, Chen Zhao, Linyong Nan, Zhenting Qi, Wenlin Zhang, Xiangru Tang, Boyu Mi, and Dragomir Radev. RobuT: A systematic study of table QA robustness against human-annotated adversarial perturbations. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6064–6081, 2023. doi: 10.18653/v1/2023.acl-long.334.

Victor Zhong, Caiming Xiong, and Richard Socher. Seq2SQL: Generating structured queries from natural language using reinforcement learning, 2017. URL https://arxiv.org/abs/1709.00103.

Wei Zhou, Mohsen Mesgar, Heike Adel, and Annemarie Friedrich. FREB-TQA: A fine-grained robustness evaluation benchmark for table question answering. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 2479–2497, 2024. doi: 10.18653/v1/2024.naacl-long.137.

Fengbin Zhu, Wenqiang Lei, Youcheng Huang, Chao Wang, Shuo Zhang, Jiancheng Lv, Fuli Feng, and Tat-Seng Chua. 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)*, pp. 3277–3287, 2021. doi: 10.18653/v1/2021.acl-long.254.

Kaijie Zhu, Jindong Wang, Jiaheng Zhou, Zichen Wang, Hao Chen, Yidong Wang, Linyi Yang, Wei Ye, Yue Zhang, Neil Gong, and Xing Xie. PromptRobust: Towards evaluating the robustness of large language models on adversarial prompts. In *Proceedings of the 1st ACM Workshop on Large AI Systems and Models with Privacy and Safety Analysis*, pp. 57–68. ACM, November 2024. doi: 10.1145/3689217.3690621. URL http://dx.doi.org/10.1145/3689217.3690621.

APPENDIX CONTENTS

A Human-Reviewed Reference **14**
A.1 Reference Coverage . . . 14
A.2 Review Procedure . . . 14
A.3 Geometry Cohort . . . 14
A.4 Task Coverage and Cohort Membership . . . 14

B Experimental Protocols **17**
B.1 Benchmark and Shared Inference Conditions . . . 17
B.2 Geometry Diagnostic . . . 17
B.3 Table-State Diagnostic . . . 17
B.4 GBDI and Matched Controls . . . 18
B.5 System Configurations . . . 18
B.6 Scoring and Reporting . . . 19

C Additional Geometry Results **19**
C.1 Position and Dispersion Results . . . 19
C.2 Joint Position-by-Dispersion Results . . . 20
C.3 Error Discovery on WikiSQL . . . 22

D Additional GBDI Analyses **23**
D.1 Equal-Call View Randomization . . . 23
D.2 View and Guidance Controls . . . 24
D.3 Intervention Skill Rules . . . 25
D.4 Discovery and Answer Recovery . . . 26
D.5 Discovery Budget and Ledger Support . . . 26
D.6 Final QA by View Count . . . 27

E Uncertainty Estimates **28**
E.1 Cluster Bootstrap . . . 28
E.2 Direct vs. Code Agent . . . 29
E.3 Table-State Contrasts . . . 30
E.4 GBDI Discovery and QA . . . 30

F Prompts and Output Schemas **32**
F.1 Geometry Discovery . . . 32
F.2 Table-State Diagnostic . . . 33
F.3 Multi-View Inspection Schemas . . . 34
F.4 Evidence Ledgers . . . 35
F.5 Intervention Guidance . . . 36
F.6 QA Conversation . . . 39

A HUMAN-REVIEWED REFERENCE

This section documents the 154-instance reference introduced in Section 2, including its coverage, review procedure, and the subsets used in the experiments. The annotations identify the critical cells $\mathcal{C}_i$, their containing rows $\mathcal{R}_i$, and supported interventions under the definitions in the main text.

A.1 REFERENCE COVERAGE

The reference comprises 154 non-clean RADAR-T instances from 34 base tasks and 24 source tables. It contains 1,162 distinct instance–row critical locations. Each retained annotation identifies the affected row and records the implicated columns, visible evidence, and an EDIT or DROP action; an EDIT also records the supported replacement values.

Of the 154 instances, 141 have nonempty critical-row sets and 13 have empty sets. The table-state comparison in Section 4 uses all 154 instances. CDR for the multi-view analysis in Section 5 uses the 141 nonempty sets; the other 13 instances are excluded because complete discovery would be vacuous. Their false-positive rates are reported separately in Appendix D.5.

A.2 REVIEW PROCEDURE

Four paid undergraduate annotators worked in two pairs, reviewing 79 and 75 instances, respectively. Within each pair, both annotators assessed each instance independently. An additional reviewer, one of the authors, checked all annotations and adjudicated disagreements.

Reviewers inspected the complete table and question to identify critical cells and supported interventions according to Section 2. The gold answer was used only to check the resulting computation, not to infer error locations or replacement values.

A.3 GEOMETRY COHORT

The geometry study in Section 3 uses 60 of the reviewed instances, with 12 from each of the five error types. Selection requires the answer-preserving row-order assumption and feasible controlled layouts. The cohort spans 29 base tasks and 21 source tables. All 60 instances have at least one critical row and enter the position experiment; 59 have multiple critical rows and also enter the dispersion experiment.

We apply equal quotas across error types and layout feasibility checks using annotation metadata, without using system outputs. The selected cohort is fixed across systems and inference modes. For every retained layout, we verify that the rows moved and scored are exactly $\mathcal{R}_i$. Appendix B.2 describes the layout construction.

A.4 TASK COVERAGE AND COHORT MEMBERSHIP

Table 1 lists all 34 base tasks in the human-reviewed reference and their membership in the position and dispersion cohorts. Task IDs and source identifiers follow RADAR (Gu et al., 2025); Table 2 gives the corresponding source dataset names. The question summaries identify the tasks; the experiments use the original RADAR questions.

Each membership cell lists the included error types: `M` = missing values, `B` = bad values, `O` = outliers, `F` = formatting errors, and `L` = logical inconsistencies. A dash denotes no included instance. Within these cohorts, each task–error-type pair identifies one RADAR-T instance. The reviewed column includes all 154 instances, including the 13 with empty critical-row sets.

The 60-instance position cohort and 59-instance dispersion cohort contain the same 29 base tasks from 21 source tables. Their only membership difference is the logical-inconsistency variant of `olympics-country`: it has one critical row and is therefore included only in the position experiment.

Table 1: Task coverage and exact error-type membership of the three cohorts. Each listed error type identifies one RADAR-T instance.

RADAR task ID and question summary	Source	Reviewed	Position	Dispersion
`actor-age-gaps` Mean age gap for couples with an older male actor.	D12	M B O F L	M O F L	M O F L
`board-games-min-players` Count games meeting both player-count limits.	D11	M B O F L	M B L	M B L
`car-co2` Mean estimated car emissions for 2018–2023.	D06	M B O F L	F	F
`daily-activity` Mean distance share from moderate-or-higher activity.	D20	M B O F L	O F	O F
`employee-years` Count older employees with at least five years of tenure.	D25	M B O F L	—	—
`england-wales-ethnicity` Find the area ranked seventh by Black Caribbean population.	D14	M B O F L	—	—
`england-wales-housing` Share of lone-parent families in one- or two-bedroom homes.	D15	M B O F L	M B O	M B O
`england-wales-housing-bedroom-count` Estimate the total bedrooms across households.	D15	M B O F L	B O	B O
`football-european-league-goal-diff` Mean absolute gap between expected and scored goals for top-five teams.	D26	M B O F L	M	M
`influenza-like-illness` Median weekly case count for ages 25–64.	D07	M B O F L	M O L	M O L
`movies-roi` Mean film return relative to production budget.	D08	M B O F L	F	F
`nba-players-best-shooters` Identify leading players by two shooting-efficiency measures.	D27	M B O F L	F	F
`northern-hemisphere-eelgrass-habitats` Sum salinity readings at US sampling sites.	D17	M B O F	F	F
`nurses-hourly-salary` Mean state-level median hourly nurse pay.	D09	M B O F L	M B O L	M B O L
`nurses-salary-difference` Mean state-level gap between high and median nurse pay.	D09	M B O F	B O	B O
`nyc-green-taxis` Count trips in hours with the longest mean duration.	D01	M B O F L	B	B
`nyc-green-taxis-passengers` Sum passenger counts across taxi trips.	D01	M B O F L	O F	O F
`nyc-green-taxis-rates` Mean total trip charge per mile.	D01	M B O F L	B L	B L
`nyc-regents-exam-scores` Count passing students in the most frequent school group.	D02	M B O F L	M B L	M B L
`nyc-regents-exam-scores-2` Compare counts of high-score-dominant and low-score-dominant rows.	D02	F	—	—
`nyc-regents-exam-scores-borough` Count passing students in the most-tested borough.	D02	M B O F L	L	L
`olympics-country` Find the highest team mean medals per Games.	D13	M B F L	F L	F
`olympics-gold-winners` Mean athlete age across gold-medal performances.	D13	M B O F L	B O F	B O F
`pet-respiratory-motion` Mean speed between consecutive recorded timestamps.	D05	M B O F L	L	L
`physical-health-exam-bmi` Mean BMI for men with no smoking history.	D18	M B O F L	O F L	O F L

Continued on next page.

Table 1: Task coverage and cohort membership (continued).

RADAR task ID and question summary	Source	Reviewed	Position	Dispersion
`physical-health-exam-rhr` Compare low-heart-rate counts across two age groups.	D18	O F L	—	—
`queensland-water-emissions` Average coordinates of a frequency-weighted facility centroid.	D03	O L	—	—
`ross-sea-algal-pigment` Find chlorophyll c2 at the median chlorophyll a value.	D16	M F L	M	M
`sales-2` Sum units ordered during the fourth quarter.	D24	M B F L	B	B
`traffic-violations` Mean vehicle age in violations recorded during 2014–2024.	D04	M B F L	M B L	M B L
`traffic-violations-speeding` Mean excess speed for violations over 20 mph above the limit.	D04	M B O F L	M O L	M O L
`udemy-classes-price` Count high-priced courses discounted by more than 88 percent.	D23	M B O F L	M	M
`udemy-classes-rating` Count courses with a rating above 4.1.	D23	M B O F L	M F	M F
`ultra-trail-races-rank` Mean age among runners finishing in the top five.	D10	M B O F L	B O F	B O F
Instances		**154**	**60**	**59**
Distinct base tasks		34	29	29
Distinct source tables		24	21	21

Table 2: Source datasets represented in the human-reviewed reference. Source identifiers and dataset names follow RADAR.

Source	Dataset
D01	2021 Green Taxi Trip Data
D02	2014–15 to 2017–19 NYC Regents Exam Results – Public
D03	Emissions from Industrial Facilities in Queensland – 2004
D04	Traffic Violations
D05	Tracking data, Subject (a) MC Motion
D06	Fuel Economy Data
D07	Outpatient Illness and Viral Surveillance
D08	Movies Bechdel Test
D09	Registered Nurses
D10	Ultra Trail Running
D11	Board Games
D12	Hollywood Age Gaps
D13	Olympics Athletes and Medals
D14	Ethnic group (England and Wales) 2011
D15	Household Composition by Number of bedrooms 2011
D16	Algal Pigment Concentrations in Ross Sea
D17	Eelgrass Biomass and Diversity (ZEN)
D18	Framingham Heart Study Dataset
D20	FitBit Fitness Tracker Data
D23	Udemy Finance Courses
D24	Sales Data
D25	IBM HR Analytics
D26	Football Expected Goals
D27	2023–2024 NBA Player Stats

B EXPERIMENTAL PROTOCOLS

The protocols below follow the geometry, table-state, and GBDI studies in Sections 3–5. Appendix A defines the reviewed evaluation subsets; Appendix F provides the exact prompts and output schemas.

B.1 BENCHMARK AND SHARED INFERENCE CONDITIONS

RADAR-T contains 313 instances from 53 base tasks and 27 source tables (Gu et al., 2025). A base task specifies a question and source data; instances apply artifact variants at the table-size setting for the split. The records use `example_id` for instances and `task_id` for base tasks. Constructed views are repeated layouts of an instance, not independent tasks. The full benchmark is used for end-to-end QA; the reviewed reference and its geometry subsets are described in Appendix A.

At inference time, each system receives the question, displayed table, condition-specific instructions, and any evidence generated within that condition. Human annotations are used for offline scoring and the controlled table-state study, where errors are marked or repaired before inference. No condition receives the gold answer.

B.2 GEOMETRY DIAGNOSTIC

We adapt the Direct and Code Agent interfaces from RADAR (Gu et al., 2025) to critical-row discovery. Under Direct, systems read the complete table view and return candidate row IDs in one response. Code Agent systems receive the same view and objective and may use up to five responses, including the final candidate submission, with read-only Python inspection. The visible dataframe is restored after each inspection command, and the task terminates at localization rather than final QA. Here, localization is the runtime term for critical-row discovery.

Every view preserves row values and multiplicities. The question–table prompt arrangement is fixed within each comparison. We call a fixed order of the remaining, noncritical rows a background and reuse it across the compared layouts.

Position. The position experiment uses the 60-instance cohort in Appendix A.3. For each instance and each of three backgrounds formed by the remaining rows, the same critical-row block is placed so that its realized μ falls in one of five equal-width bands. The feasible block location is randomized within the assigned band, subject to preserving critical-row identities, critical-row count, row contents, question, and prompt template. Each system–mode pair therefore evaluates $60 \times 3 \times 5 = 900$ views. Complete discovery rate is reported separately for each of the five position bands.

Dispersion. The dispersion cohort excludes the one instance with a single critical row, leaving 59 instances with at least two critical rows. For each instance, three target mean positions μ (25%, 50%, and 75%) and three backgrounds are crossed with Compact, Medium, and Wide layouts. Critical-row identities, values, count, μ, question, and the relative order of the remaining rows are held fixed within each comparison triplet. The construction yields $59 \times 3 \times 3 \times 3 = 1{,}593$ views per system–mode pair. Compact uses the minimum consecutive critical-row span; Medium uses an intermediate symmetric span; Wide uses the largest feasible symmetric span within a normalized radius of 0.2. The audit records realized μ and σ for every constructed view.

B.3 TABLE-STATE DIAGNOSTIC

The diagnostic in Section 4 compares Raw, Discovery, and Action on the 154 instances in Appendix A, under both Direct and Code Agent. Raw uses the canonical table. Discovery wraps each reviewed critical cell value in `<<ERROR:...>>` while retaining the original value; it does not provide a replacement or a separate evidence packet. Action starts from the canonical table, applies the reviewed replacement values for EDIT, and removes rows assigned DROP. The Discovery instruction identifies marked cells as verified errors. The Action instruction states that the reviewed errors relevant to the answer have been repaired. Both notices precede the usual user turn; the system prompt is unchanged within each inference mode. Raw Direct uses the released RADAR Direct prompt. Conditions within a mode share the answer scorer and response allowance. Requests for all three systems and both modes in this diagnostic omit an explicit output-token cap; provider

defaults still apply. Comparisons are paired by instance and clustered by source table for uncertainty estimation.

B.4 GBDI AND MATCHED CONTROLS

GBDI uses the M independently permuted row orders in Section 5.1. We denote these M independent inspections by Random-M. Repeated-M is the call-matched control: it presents the unchanged released row order in M independent inspection contexts. Inspection contexts use the same decoding parameters within a system. Nominations are mapped to the original rows; every valid nomination and its independent-view support count are retained. The final Code Agent receives the canonical table, question, evidence ledger, and Intervention Skill. The intervention turn checks visible evidence and prepares a persistent `work_df`; the same conversation and Python namespace continue to final QA. The primary configuration is Random-5. Visible evidence, rather than nomination frequency alone, determines whether an intervention is supported.

The Qwen, GLM, and MiniMax benchmark collections use the same instance–view permutations, and every view retains all rows. DeepSeek-V4.1-Flash also uses five independently permuted views. Permutation construction does not use human annotations. The union ledger retains every valid nomination. A support threshold of at least two is examined separately in the ledger-support sensitivity analysis; end-to-end QA uses the union ledger.

Within a system, Random-M and Repeated-M share the call allowance, discovery prompt, output schema, parser, and final QA protocol. The guidance factorial crosses the two view conditions with Generic/Skill on the same 313 instances for Qwen, GLM, and MiniMax; GPT and DeepSeek-V4.1-Flash appear only in the complete-system evaluation. Generic specifies general operation boundaries; Skill adds artifact-specific rules and parsing, validation, and submission guidance. Within each layout, both guidance conditions receive the same ledger, so only the guidance text changes. The reserved turn, persistent namespace, and five-response final QA allowance remain fixed.

The blank-ledger diagnostic preserves the canonical table, question, Skill, intervention turn, and final budget while removing ledger coordinates and observations. It measures the contribution of ledger availability within the candidate-guided workflow.

B.5 SYSTEM CONFIGURATIONS

Table 3 lists the systems evaluated on RADAR-T and their request identifiers. The benchmark and component analyses use the same recorded GBDI outcomes (Random-5+Skill). We conducted the API evaluations during August and September 2026.

The Direct baseline uses the unchanged RADAR prompt, canonical RADAR table, original question, and required answer parser on all 313 instances.

Table 3: Systems evaluated on RADAR-T and their API providers. API identifiers are request identifiers rather than checkpoint hashes.

System	API identifier	Provider
Qwen Plus	`qwen-plus`	Bailian
MiniMax-M2.5	`MiniMax-M2.5`	Bailian
GPT-5 Mini	`openai/gpt-5-mini`	OpenRouter/OpenAI
DeepSeek-V4.1-Flash	`deepseek-flash`	DeepSeek native API
GLM-4.5-Air	`glm-4.5-air`	Bailian

The Bailian requests specify temperature 0 and top-p 1. DeepSeek-V4.1-Flash uses temperature 0 with thinking disabled and no submitted response-token cap. GPT-5 Mini omits these sampling parameters. For its GBDI runs, requests specify low reasoning effort, seed 408, a 12,000-token response cap, and disabled provider fallback. Within each system, the prompt, parser, and renderer are shared across view families. The Random-5 and Repeated-5 configurations use five inspection contexts, one intervention turn, and at most five final QA responses.

B.6 Scoring and Reporting

The Direct baseline extracts the answer following the required `The answer is:` prefix. Format retries regenerate a response, count against the three-attempt allowance, and contribute to observed usage when the provider returns usage information. The Code Agent baseline, GBDI, and the view-family controls require a valid `done` submission whose `kwargs.answer` field contains the answer. The extraction rule is shared within each submission interface, while Direct and Code retain their respective output formats. All reported runs were rescored with these rules. RADAR-T discovery predictions are matched to the canonical critical-row sets in Appendix A. Invalid formats, budget exhaustion, tool errors, and transport failures are recorded separately; all count as empty predictions or incorrect answers in the primary denominator.

CDR measures complete coverage without penalizing additional nominations. We therefore also report row-level precision, recall, and F1. For an evaluated view or ledger, let $\widehat{\mathcal{R}}$ be the set of nominated row IDs and $\mathcal{R}$ the corresponding nonempty reference critical-row set. Then

$$\mathrm{Prec} = \frac{|\widehat{\mathcal{R}} \cap \mathcal{R}|}{|\widehat{\mathcal{R}}|}, \qquad \mathrm{Rec} = \frac{|\widehat{\mathcal{R}} \cap \mathcal{R}|}{|\mathcal{R}|}, \qquad \mathrm{F1} = \frac{2|\widehat{\mathcal{R}} \cap \mathcal{R}|}{|\widehat{\mathcal{R}}| + |\mathcal{R}|}. \tag{9}$$

Precision is zero for an empty prediction. For parsed geometry Direct submissions, unmatched row IDs remain false-positive nominations. Each metric is computed before macro averaging; F1 is not the harmonic mean of the reported average precision and recall. Geometry metrics average over all views in the stated condition, with equal numbers of views per instance. Multi-view metrics average over the 141 reviewed instances with nonempty critical-row sets; the budget analysis additionally averages over the specified view subsets. Mean rows is the average nomination count over all 154 reviewed instances. For the 13 empty-reference instances, Empty FP is the fraction with at least one nominated row. All rate columns are percentages. These discovery metrics use the same reference and predictions as the corresponding CDR.

Discovery contrasts are paired instance-level CDR differences averaged over the construction backgrounds and, for dispersion, the same values of μ. The dispersion effect is $\mathrm{CDR}(\mathrm{Wide}) - \mathrm{CDR}(\mathrm{Compact})$. Mode differences subtract the Direct effect from the Code Agent effect. All calculations are performed before rounding. Binary QA results are reported as benchmark QA accuracy on the stated evaluation population. Appendix E specifies the paired contrasts and cluster bootstrap procedures.

C Additional Geometry Results

This section supplements the geometry study in Section 3 with the complete marginal results, joint position–dispersion breakdowns, and an external validation on WikiSQL. The RADAR-T cohort and layout construction are specified in Appendices A.3 and B.2; paired uncertainty estimates appear in Appendix E.

C.1 Position and Dispersion Results

Table 4 reports the complete discovery rate (CDR) values plotted in Figure 2, together with macro precision, recall, and F1 as defined in Appendix B.6. The table has two parts: (a) the five position bands and (b) the three dispersion levels, averaged over the same three values of μ. The position experiment uses 60 instances and 900 views per system–mode pair; the dispersion experiment uses 59 instances with multiple critical rows and 1,593 views per pair. Both experiments use three fixed backgrounds, and invalid submissions count as discovery failures.

Table 4: Discovery metrics by critical-row position and dispersion. Precision (Prec.), recall (Rec.), and F1 are macro averages; all entries are percentages. Each position band uses 180 views of 60 instances. Each dispersion level uses 531 views of 59 instances, averaged over three values of μ.

(a) Position μ

		Direct				Code Agent			
System	Layout	CDR	Prec.	Rec.	F1	CDR	Prec.	Rec.	F1
Qwen Plus	q1	61.1	70.6	76.8	70.2	66.1	68.8	72.4	67.5
	q2	68.9	72.4	77.8	71.9	75.6	75.7	80.1	74.8
	q3	66.7	71.7	76.0	70.9	76.1	76.1	82.2	76.9
	q4	70.6	75.5	79.5	74.7	73.9	75.4	79.2	75.1
	q5	72.8	76.1	82.0	75.7	73.3	75.4	78.6	73.9
MiniMax-M2.5	q1	47.8	59.0	59.4	56.6	38.3	44.8	42.3	42.6
	q2	60.6	65.5	69.0	65.0	38.3	44.1	42.9	42.7
	q3	58.9	68.5	67.7	66.2	48.9	51.6	52.1	51.1
	q4	56.7	67.5	67.6	65.8	43.3	46.8	45.9	45.5
	q5	59.4	68.5	68.0	66.2	43.3	50.0	47.7	47.8
GPT-5 Mini	q1	43.9	59.6	56.0	55.8	68.9	75.3	74.1	72.6
	q2	52.8	65.4	62.4	61.7	73.9	79.6	79.7	77.2
	q3	56.1	61.7	61.2	60.1	75.0	77.7	78.7	76.3
	q4	60.0	69.3	66.9	65.9	72.2	75.4	77.2	73.8
	q5	60.6	66.7	66.9	64.9	72.2	77.1	77.6	75.5

(b) Dispersion σ

		Direct				Code Agent			
System	Layout	CDR	Prec.	Rec.	F1	CDR	Prec.	Rec.	F1
Qwen Plus	Compact	72.3	75.7	81.8	75.8	72.1	74.6	78.4	73.8
	Medium	59.5	75.8	78.9	74.1	70.6	74.9	78.9	74.3
	Wide	56.1	76.4	77.0	73.1	69.9	75.5	78.6	74.5
MiniMax-M2.5	Compact	59.3	67.2	68.9	65.6	44.4	50.8	48.8	48.7
	Medium	49.7	66.8	65.2	63.5	38.0	45.6	42.4	42.8
	Wide	49.7	63.6	63.8	61.2	40.7	48.1	45.6	45.7
GPT-5 Mini	Compact	54.2	64.7	62.0	61.3	70.2	76.9	76.8	74.8
	Medium	51.4	64.8	61.4	61.1	68.9	78.4	76.4	75.1
	Wide	46.3	62.8	59.4	58.9	68.2	78.4	75.6	74.8

C.2 Joint Position-by-Dispersion Results

The dispersion experiment contains nine layouts per instance and background. Table 5 reports their realized coordinates. The mean nearest-neighbor dispersion is 0.0081, 0.0677, and 0.1268 for Compact, Medium, and Wide, respectively, at each value of μ. Within each instance–background pair, μ is fixed across dispersion levels and σ is fixed across positions. All systems and both inference modes use the same layouts.

Table 5: Realized geometry of the 3×3 layout design. Each cell gives mean position $\bar{\mu}$ / mean dispersion $\bar{\sigma}$.

Target μ	Compact	Medium	Wide
25%	0.2495 / 0.0081	0.2495 / 0.0677	0.2495 / 0.1268
50%	0.5008 / 0.0081	0.5008 / 0.0677	0.5008 / 0.1268
75%	0.7499 / 0.0081	0.7499 / 0.0677	0.7499 / 0.1268

Tables 6 and 7 report CDR, precision, recall, and F1 for every position–dispersion combination. Every condition uses the same 59 instances and three backgrounds, yielding 177 views. The three

values of μ are distinct from the five bands in the contiguous critical-row position experiment. Within each position group, σ varies while μ remains fixed. Invalid submissions count as discovery failures.

Table 6: Joint position–dispersion discovery metrics under Direct. Each condition uses the same 59 instances and three backgrounds (177 views). Precision, recall, and F1 are macro averages; all entries are percentages.

System	μ	Dispersion	CDR	Prec.	Rec.	F1
Qwen Plus		Compact	71.2	72.7	80.2	73.3
	25%	Medium	53.1	71.9	75.8	70.5
		Wide	52.0	73.0	74.0	70.0
		Compact	70.1	75.9	81.0	75.7
	50%	Medium	59.3	77.2	79.1	74.9
		Wide	54.2	76.9	77.6	73.8
		Compact	75.7	78.3	84.3	78.4
	75%	Medium	66.1	78.2	81.8	76.9
		Wide	62.1	79.4	79.3	75.5
MiniMax-M2.5		Compact	57.1	66.1	67.8	63.7
	25%	Medium	48.6	67.7	64.5	63.7
		Wide	45.2	61.4	61.2	58.4
		Compact	61.0	67.1	69.0	66.0
	50%	Medium	49.2	63.5	63.9	61.3
		Wide	48.6	62.4	62.9	60.9
		Compact	59.9	68.6	69.8	67.1
	75%	Medium	51.4	69.2	67.3	65.5
		Wide	55.4	66.9	67.4	64.4
GPT-5 Mini		Compact	53.1	64.7	62.2	61.3
	25%	Medium	47.5	61.4	57.5	57.2
		Wide	46.9	63.7	59.0	58.7
		Compact	55.9	63.4	62.1	61.1
	50%	Medium	53.7	64.5	62.3	61.6
		Wide	42.9	56.6	55.0	54.0
		Compact	53.7	66.1	61.7	61.5
	75%	Medium	53.1	68.4	64.4	64.6
		Wide	49.2	68.1	64.0	64.1

Under Direct, CDR decreases from Compact to Wide at each μ for every system. Intermediate values show local variation; for MiniMax at $\mu = 75\%$, CDR is 51.4% for Medium and 55.4% for Wide.

Table 7: Joint position–dispersion discovery metrics under Code Agent. Each condition uses the same 59 instances and three backgrounds (177 views). Precision, recall, and F1 are macro averages; all entries are percentages.

System	μ	Dispersion	CDR	Prec.	Rec.	F1
Qwen Plus		Compact	68.9	72.4	76.0	71.3
	25%	Medium	70.6	72.4	76.7	72.0
		Wide	68.9	73.2	76.4	71.8
		Compact	71.8	75.6	78.3	74.8
	50%	Medium	68.4	78.1	80.5	77.2
		Wide	72.9	76.3	80.4	75.9
		Compact	75.7	75.8	80.9	75.2
	75%	Medium	72.9	74.1	79.4	73.6
		Wide	67.8	76.9	79.0	75.6
MiniMax-M2.5		Compact	44.6	48.6	47.9	47.0
	25%	Medium	36.7	43.1	41.6	41.0
		Wide	37.3	44.1	42.3	42.3
		Compact	42.9	47.9	47.1	46.7
	50%	Medium	40.7	46.6	43.9	44.4
		Wide	43.5	50.1	47.6	47.6
		Compact	45.8	55.9	51.4	52.4
	75%	Medium	36.7	46.9	41.6	42.9
		Wide	41.2	50.2	46.9	47.4
GPT-5 Mini		Compact	69.5	76.4	76.8	74.6
	25%	Medium	69.5	77.7	76.3	74.7
		Wide	66.7	78.0	74.9	74.1
		Compact	71.2	78.5	79.0	76.2
	50%	Medium	66.7	79.4	74.9	74.7
		Wide	70.1	78.0	75.7	74.6
		Compact	70.1	75.8	74.7	73.4
	75%	Medium	70.6	78.2	78.0	76.0
		Wide	67.8	79.2	76.1	75.6

Code Agent results also vary across the tested mean positions.

C.3 Error Discovery on WikiSQL

We constructed missingness instances from 41 WikiSQL tables (Zhong et al., 2017), following the RADAR missingness setting (Gu et al., 2025). Each question selects rows by a rank threshold and asks for the sum of another numeric column. We replace three question-relevant values in the rank column with empty cells. We evaluate Direct discovery with Qwen-Turbo and Qwen3-8B using the position and dispersion manipulations in Section 3. Position uses q1–q5 and three fixed backgrounds; dispersion crosses Compact, Medium, and Wide with three mean positions and one fixed background. We score all planned views against the injected critical rows using the metrics in Appendix B.6, with invalid submissions treated as empty predictions.

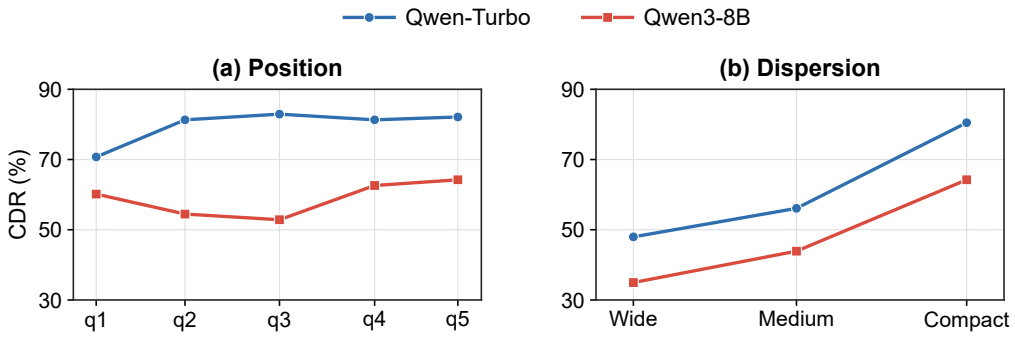

Figure 7: Direct discovery on WikiSQL missingness instances. Position runs from q1 (front) to q5 (back); dispersion runs from Wide to Compact. Each point averages 123 views of the same 41 instances.

Table 8: Discovery metrics on WikiSQL missingness instances. All entries are percentages; precision, recall, and F1 are macro averages. Each system uses 615 position views and 369 dispersion views.

	Qwen-Turbo				Qwen3-8B			
Layout	CDR	Prec.	Rec.	F1	CDR	Prec.	Rec.	F1
(a) Position								
q1	70.7	47.2	74.0	49.5	60.2	20.9	66.1	26.0
q2	81.3	74.6	83.2	75.7	54.5	38.9	58.8	42.1
q3	82.9	77.7	84.0	78.4	52.8	46.1	58.3	48.4
q4	81.3	75.8	83.5	76.8	62.6	57.1	66.1	58.9
q5	82.1	74.8	82.7	76.6	64.2	57.6	64.8	59.5
(b) Dispersion								
Compact	80.5	72.9	82.1	74.0	64.2	56.6	67.5	58.6
Medium	56.1	78.3	78.3	73.8	43.9	50.4	61.8	51.5
Wide	48.0	80.5	73.7	74.2	35.0	49.8	58.3	48.1

Figure 7 and Table 8 show a decrease in CDR from Compact to Wide layouts: from 80.5% to 48.0% for Qwen-Turbo and from 64.2% to 35.0% for Qwen3-8B. Position effects are less uniform. CDR is higher at q5 than q1 for both systems, but neither curve is monotonic; Qwen3-8B recall is slightly lower at q5. In this external missingness setting, the dispersion contrast is more consistent across the two systems than the position contrast.

D Additional GBDI Analyses

The analyses below expand Section 5.2: we first test view randomization under equal call budgets, then examine intervention guidance, the relation between discovery and answer recovery, and sensitivity to view count and ledger support. They use the protocols in Appendix B.4 and the reviewed reference in Appendix A.

D.1 Equal-Call View Randomization

CDR and macro precision, recall, and F1 are evaluated on the 141 reviewed instances with nonempty critical-row sets. Mean rows and Empty FP follow Appendix B.6. Within each system, Repeated-5 and Random-5 use the same five-call allowance, prompt, parser, coordinate mapping, and union rule.

Table 9: Discovery under an equal five-call allowance. Repeated-5 uses the released row order; Random-5 permutes all rows independently in each call. CDR and macro precision, recall, and F1 use 141 nonempty references. Mean rows uses all 154 reviewed instances; Empty FP uses 13 empty references. All rate columns are percentages.

System	Views	CDR	Prec.	Rec.	F1	Mean rows	Empty FP
Qwen Plus	Repeated-5	58.9	71.0	77.1	68.9	9.11	61.5
	Random-5	78.7	72.5	85.6	74.6	10.03	76.9
GLM-4.5-Air	Repeated-5	61.0	60.5	74.6	62.5	9.11	84.6
	Random-5	66.7	59.7	76.7	62.3	10.24	76.9
MiniMax-M2.5	Repeated-5	38.3	47.1	48.9	45.4	4.98	38.5
	Random-5	58.2	65.1	69.4	64.3	7.14	23.1

Random-5 yields higher CDR point estimates for all three systems. Paired intervals exclude zero for Qwen and MiniMax, but not GLM (Table 19). The final QA ledger retains all valid nominations, which the agent must verify against visible table evidence. Random-5 also raises recall for all three systems, but precision and F1 do not improve uniformly: for GLM, they change from 60.5% to 59.7% and from 62.5% to 62.3%, respectively. The mean ledger size increases for all three systems.

D.2 View and Guidance Controls

The Qwen Plus factorial analysis crosses Random-5 and Repeated-5 with Generic and Skill guidance on all 313 RADAR-T instances. Both view conditions share the discovery prompt, five-call allowance, aggregation rule, and final QA runtime. Within each view condition, Generic and Skill receive identical ledgers, tables, questions, and Code Agent response allowances. Their discovery metrics are therefore shared and reported in Table 9. In Table 10(a), each guidance cell reports the number of correct answers among all 313 instances, followed by accuracy. CDR in panel (b) is computed on the 141 reviewed instances with nonempty critical-row sets. We define $\Delta_{\text{Skill}} = \text{Skill} - \text{Generic}$ and $\Delta_{\text{view}} = \text{Random-5} - \text{Repeated-5}$ using unrounded rates.

Table 10: Controlled tests of the two findings with Qwen Plus. (a) QA accuracy on all 313 instances when intervention guidance is varied while the discovery ledger is held fixed within each view family. (b) Random-5 varies row order across five inspections, whereas Repeated-5 repeats the released order at the same call allowance; the QA comparison holds Skill guidance fixed. Differences and intervals are reported in percentage points.

(a) View family × guidance

Views	Generic	Skill	Δ_{Skill}	95% CI
Repeated-5	172 (55.0%)	190 (60.7%)	+5.8	$[-0.7, 12.0]$
Random-5	183 (58.5%)	203 (64.9%)	+6.4	$[1.6, 11.2]$

(b) Random-5 vs. Repeated-5

Metric	Repeated-5	Random-5	Δ_{view}	95% CI
Complete discovery rate	58.9%	78.7%	+19.9	$[11.1, 29.2]$
QA accuracy	60.7%	64.9%	+4.2	$[-0.3, 9.0]$

With the same five-call allowance, Random-5 raises CDR from 58.9% to 78.7%. Its estimated QA accuracy is 4.2 points higher than that of Repeated-5, although the paired interval includes zero. Relative to Repeated-5, Random-5 corrects 39 previously incorrect answers and changes 26 correct answers to incorrect answers, yielding a net gain of 13/313.

With each ledger held fixed, Skill raises QA accuracy by 5.8 points under Repeated-5 and 6.4 points under Random-5; the latter interval excludes zero. The blank-ledger diagnostic uses the same Skill-guided QA protocol (180/313). Relative to this diagnostic, Repeated-5 yields 10 additional correct

answers and Random-5 yields 23. This comparison measures the effect of ledger availability within the candidate-guided workflow.

Table 11: Counts of correct QA answers for the five error types and clean inputs in the Qwen Repeated-5/Random-5 guidance factorial.

		Generic		Skill	
Input type	n	Repeated-5	Random-5	Repeated-5	Random-5
Bad values	53	26	28	26	25
Clean	53	46	49	47	49
Logic	53	18	23	17	28
Formatting	53	29	28	40	38
Missingness	53	25	25	31	30
Outliers	48	28	30	29	33
All	313	172	183	190	203

With Skill fixed, Random has its largest advantage over Repeated on Logic (28 versus 17 correct). Skill provides its largest gain over Generic on Formatting in both layouts: 11 additional correct answers with Repeated and 10 with Random. Under Random, Skill also yields five additional correct answers each on Logic and Missingness and three on Outliers, while yielding three fewer on Bad values and leaving Clean unchanged. The aggregate gain is concentrated in Logic, Formatting, and Missingness.

The factorial experiment compares detailed guidance with generic operation boundaries under the same intervention-turn budget. The Skill combines artifact-specific action rules with parsing, validation, and submission guidance, so this contrast evaluates the complete guidance text.

Under Random-5, the Skill corrects 47 answers and changes 27 correct answers to incorrect answers, yielding a net gain of 6.39 points. The view-by-guidance interaction is 0.64 points (95% CI $[-7.92, 9.20]$); the Skill gain alone does not establish an interaction.

The Skill-gain interval also excludes zero under base-task clustering ($[0.96, 11.65]$).

D.3 Intervention Skill Rules

We further remove three instruction groups from the Qwen Plus Skill, one at a time, while keeping the Random-5 ledger, 313 instances, and final QA protocol fixed. The recovery variant omits explicit procedures for deriving missing or invalid values from table evidence. The formatting variant omits rules for parsing inconsistent formats. The execution variant omits the multi-turn checks in the Skill; the intervention turn required in every condition remains in place. These variants delete instructions rather than paraphrasing the same guidance.

Table 12: Qwen Plus final-answer accuracy on all 313 RADAR-T instances when one part of the Intervention Skill is removed. Differences are variant minus Full Skill, in percentage points; intervals use paired source-table cluster resampling.

Guidance	Correct	QA (%)	Δ (pp)	95% CI (pp)
Full Skill	203/313	64.9	–	–
Without recovery rules	186/313	59.4	-5.4	$[-11.3, 0.0]$
Without formatting rules	190/313	60.7	-4.2	$[-7.6, -0.4]$
Without execution checks	197/313	62.9	-1.9	$[-6.7, 3.0]$

Removing formatting rules reduces overall accuracy from 64.9% to 60.7%, with the paired interval for the change below zero. On Formatting instances, correct answers fall from 38/53 to 33/53. Removing recovery rules also lowers the point estimate, but the corresponding interval reaches zero.

The execution-check contrast is uncertain. Because the shared intervention-turn protocol remains in place, the last variant does not test a workflow without execution constraints.

D.4 Discovery and Answer Recovery

Let D^+ denote complete discovery and A^+ a correct final answer. The underlying ledger precision, recall, and F1 are reported in Table 9.

Table 13: Discovery completeness and final-answer outcomes on the 141 reviewed instances with at least one critical row. Complete means that the union of five inspections contains every critical row. Counts are descriptive paired outcomes; the 13 empty-critical-row instances are excluded because completeness is vacuous for them.

System	Views	Guidance	D^+A^+	D^+A^-	D^-A^+	D^-A^-
Qwen Plus	Repeated	Generic	37	46	27	31
	Repeated	Skill	51	32	28	30
	Random	Generic	57	54	16	14
	Random	Skill	72	39	16	14
GLM-4.5-Air	Repeated	Generic	32	54	7	48
	Repeated	Skill	38	48	16	39
	Random	Generic	37	57	10	37
	Random	Skill	40	54	13	34
MiniMax-M2.5	Repeated	Generic	19	35	29	58
	Repeated	Skill	24	30	20	67
	Random	Generic	30	52	11	48
	Random	Skill	40	42	12	47

We examine final QA among instances with complete discovery. Randomized views increase complete discovery for all three systems, yet a substantial share of instances with complete discovery still receive incorrect answers. Conversely, some incomplete ledgers still yield the gold answer. These counts alone do not explain why: the final agent can inspect the complete table, including rows absent from the ledger. The paired guidance comparison is most informative within the complete-discovery stratum: with the ledger held fixed, Skill produces more corrections than regressions for every system. Complete discovery alone therefore does not determine answer recovery; intervention remains consequential after all reviewed critical rows have been identified.

D.5 Discovery Budget and Ledger Support

Table 14 evaluates the view budget and support threshold using five independently permuted Qwen views. For each M, it averages over all $\binom{5}{M}$ subsets rather than selecting an arbitrary prefix. A row is retained when nominated in at least k selected views. Precision, recall, and F1 follow Appendix B.6 and are macro averaged over the 141 nonempty references and the relevant view subsets. Mean rows uses all 154 reviewed instances; Empty FP uses the 13 empty references.

Table 14: Qwen Plus sensitivity to the Random-view budget M and ledger support k. Panel (a) averages over all subsets of size M from five views. CDR and macro precision, recall, and F1 use 141 nonempty references; Mean rows uses all 154 reviewed instances; Empty FP uses 13 empty references. Rate columns are percentages. The deployed setting is bold.

(a) View budget with union ledger ($k = 1$)

M	CDR	Prec.	Rec.	F1	Mean rows	Empty FP
1	45.5	71.8	71.8	68.6	6.92	64.6
2	65.9	72.6	80.5	72.9	8.41	70.0
3	73.5	72.7	83.2	73.9	9.11	73.1
4	76.7	72.7	84.6	74.4	9.61	75.4
5	**78.7**	**72.5**	**85.6**	**74.6**	**10.03**	**76.9**

(b) Support threshold with five views ($M = 5$)

k	CDR	Prec.	Rec.	F1	Mean rows	Empty FP
1	**78.7**	**72.5**	**85.6**	**74.6**	**10.03**	**76.9**
2	68.8	73.6	80.6	73.4	7.97	69.2
3	61.0	71.9	76.2	70.9	7.09	61.5
4	44.0	72.4	68.7	67.0	5.88	53.8
5	22.0	67.5	48.0	52.8	3.66	53.8

Under the union rule ($k = 1$), CDR rises from 45.5% with one view to 78.7% with five. The largest increment occurs between one and two views; beyond three views, the remaining gain is 5.2 points. Recall rises from 71.8% to 85.6%, while precision remains between 71.8% and 72.7%. The mean number of nominated rows rises from 6.92 to 10.03, and the empty-set false-positive rate rises from 64.6% to 76.9%. Thus, the added views recover more required rows but also expand the candidate set.

Requiring repeated support produces a different operating point. At $M = 5$, raising k from one to two removes about two rows per ledger but reduces CDR by 9.9 points. The same change raises precision from 72.5% to 73.6% but reduces recall from 85.6% to 80.6%. Higher thresholds continue to shrink the ledger and reduce recall, without consistently improving macro precision. The end-to-end condition retains the union rule because downstream intervention verifies each nominated row. The analysis also shows diminishing returns by five views; it does not extrapolate to larger budgets.

D.6 Final QA by View Count

We evaluate Qwen Plus final QA with the same Skill at $M \in \{0, 1, 3, 5\}$ (Table 15). Each nonzero budget uses the first M of these five permuted views; $M = 0$ has no discovery ledger. QA is evaluated on all 313 instances at each budget. Unlike the all-subset averages in Table 14, these fixed prefixes need not have the same discovery metrics at a given M. Panel (a) reports the metrics of these exact prefixes; panel (b) reports their final QA outcomes.

Table 15: Qwen Plus discovery and final QA using the first M of five permuted views; $M = 0$ supplies no ledger. In panel (a), CDR and macro precision, recall, and F1 use 141 nonempty references, Mean rows uses all 154 reviewed instances, and Empty FP uses 13 empty references. Rate columns are percentages. Panel (b) evaluates all 313 instances; QA gains are relative to the preceding row, with paired source-table cluster 95% intervals.

(a) Discovery with fixed prefixes

M	CDR	Prec.	Rec.	F1	Mean rows	Empty FP
0	0.0	0.0	0.0	0.0	0.00	0.0
1	44.7	72.9	71.1	68.6	6.51	61.5
3	73.0	72.1	82.6	73.1	9.05	76.9
5	78.7	72.5	85.6	74.6	10.03	76.9

(b) Final QA

M	Correct	QA (%)	Gain (pp)	95% CI
0	180/313	57.5	–	–
1	187/313	59.7	$+2.2$	$[-3.0, 7.9]$
3	191/313	61.0	$+1.3$	$[-4.0, 6.3]$
5	203/313	64.9	$+3.8$	$[-0.9, 8.3]$

QA accuracy rises from 57.5% at $M = 0$ to 64.9% at $M = 5$, yielding a paired gain of 7.3 points (95% CI $[3.2, 11.9]$). The intermediate point estimates are 59.7% at $M = 1$ and 61.0% at $M = 3$; intervals for the individual increments include zero.

E Uncertainty Estimates

We report paired confidence intervals for the geometry study (Section 3), the table-state comparison (Section 4), and GBDI (Section 5). The resampling unit follows the shared source of dependence within each evaluation.

E.1 Cluster Bootstrap

We resample clusters to retain dependence among related observations (Field & Welsh, 2007). The reviewed geometry analysis uses 20,000 base-task cluster-bootstrap samples over 29 base tasks. All instances and backgrounds associated with a sampled task are retained together. Complete discovery rate (CDR) contrasts use an instance-weighted estimand.

The 154-instance table-state comparison uses 10,000 source-table cluster bootstrap samples over 24 source tables. End-to-end QA comparisons use 10,000 source-table cluster bootstrap samples with the task-to-source mapping from RADAR (Gu et al., 2025). None of these procedures treats repeated views as independent instances.

For a contrast between conditions A and B, let a_i denote the paired difference in binary outcomes and $b_i = 1$. For QA, the outcome is answer correctness; for discovery, it is the indicator in Equation 5. For geometry, paired contrasts are first averaged within each instance. With $m_s^{(b)}$ denoting the number of times cluster s appears in bootstrap sample b, the estimate is

$$\widehat{\Delta}^{(b)} = 100 \, \frac{\sum_s m_s^{(b)} \sum_{i \in s} a_i}{\sum_s m_s^{(b)} \sum_{i \in s} b_i}. \tag{10}$$

The reported endpoints are the 2.5th and 97.5th percentiles of these estimates. Clustering changes the resampling unit, not the instance or reference-row weighting of the reported estimand. Intervals are pointwise rather than simultaneous across all systems and contrasts. For position, the q5–q1 contrast pairs the two endpoint layouts within each instance and background, using all 60 instances. We also report a descriptive linear trend: for each instance and background, we regress the binary complete-discovery outcome on the ordinal band index (1 through 5). Endpoint differences and slopes are averaged over the three backgrounds within each instance before cluster resampling. For

dispersion, the Wide–Compact contrast uses 59 instances and averages paired differences over the three mean positions and three backgrounds within each instance.

Table 16: Position and dispersion contrasts in complete discovery rate. Entries give estimates with pointwise 95% confidence intervals in brackets. Endpoint is q5−q1; trend is the slope per ordinal position band. Dispersion is Wide−Compact. Differences are in percentage points; trends are in percentage points per band.

		Position (60 instances)		Dispersion (59 instances)
System	Mode	Endpoint	Trend	Wide−Compact
Qwen Plus	Direct	$+11.7$ $[+4.0, +20.2]$	$+2.5$ $[+0.7, +4.5]$	-16.2 $[-22.4, -10.5]$
	Code Agent	$+7.2$ $[-0.6, +15.8]$	$+1.3$ $[-0.4, +3.1]$	-2.3 $[-6.6, +1.8]$
MiniMax-M2.5	Direct	$+11.7$ $[+4.6, +19.1]$	$+1.9$ $[+0.3, +3.7]$	-9.6 $[-13.0, -6.2]$
	Code Agent	$+5.0$ $[-2.6, +13.6]$	$+1.5$ $[-0.1, +3.4]$	-3.8 $[-9.3, +1.3]$
GPT-5 Mini	Direct	$+16.7$ $[+8.1, +26.9]$	$+4.1$ $[+2.0, +6.3]$	-7.9 $[-13.0, -2.5]$
	Code Agent	$+3.3$ $[-3.5, +10.3]$	$+0.5$ $[-0.6, +1.7]$	-2.1 $[-5.0, +1.3]$

Invalid submission counts for position (out of 900 per mode) are 0/0 for Qwen, 8/207 for MiniMax, and 0/0 for GPT, with Direct listed first and Code Agent second. For dispersion (out of 1,593 per mode), the corresponding counts are 1/0, 11/386, and 2/1, respectively. Invalid outputs are scored as empty predictions and remain in the denominators; all planned trajectories are present.

E.2 Direct vs. Code Agent

For each instance, value of μ, and background, we pair Direct and Code Agent outputs on the same layout and compute $\mathrm{CDR}(\text{Wide}) - \mathrm{CDR}(\text{Compact})$. The Code Agent effect minus the Direct effect is averaged over backgrounds and values of μ within each instance before base-task resampling. This paired contrast compares the effects across modes without relying on whether the separate within-mode intervals include zero. A positive value indicates less Compact-to-Wide degradation under the Code Agent. Means are computed over μ values and backgrounds within each of the 59 instances, preserving paired outcomes from both modes.

Table 17: Code Agent vs. Direct differences in the Wide vs. Compact CDR effect, in percentage points.

System	Mean	95% CI	$\mu = 25\%$	$\mu = 50\%$	$\mu = 75\%$
Qwen Plus	+13.9	[+8.4, +20.0]	+19.2	+16.9	+5.6
MiniMax-M2.5	+5.8	[-0.2, +11.7]	+4.5	+13.0	+0.0
GPT-5 Mini	+5.8	[+0.4, +11.5]	+3.4	+11.9	+2.3

The mean CDR contrast favors the Code Agent for all three systems. The paired intervals exclude zero for Qwen and GPT; the MiniMax interval includes zero. Position-specific differences are positive for Qwen and GPT; MiniMax varies across values of μ. These comparisons retain invalid outputs in their original denominators and compare the complete modes, which differ in both programmatic access and response allowance.

E.3 TABLE-STATE CONTRASTS

Table 18: Final-answer accuracy (%) and paired table-state contrasts on the 154 reviewed instances. Differences and source-table cluster 95% intervals are reported in percentage points. Raw uses the original table, Discovery uses the error-marked table, and Action uses the repaired table.

System	Mode	Raw	Discovery	Action	Discovery vs. Raw	Action vs. Discovery
Qwen Plus	Direct	16.2	18.8	29.2	$+2.6\ [-5.5, 10.1]$	$+10.4\ [-1.2, 23.3]$
	Code Agent	46.8	56.5	98.1	$+9.7\ [0.0, 19.5]$	$+41.6\ [34.1, 49.2]$
MiniMax-M2.5	Direct	35.7	31.8	67.5	$-3.9\ [-16.4, 9.0]$	$+35.7\ [22.4, 47.4]$
	Code Agent	25.3	33.8	92.9	$+8.4\ [-0.6, 16.8]$	$+59.1\ [49.5, 68.9]$
GPT-5 Mini	Direct	18.8	22.7	37.0	$+3.9\ [-3.3, 11.5]$	$+14.3\ [5.7, 24.1]$
	Code Agent	47.4	58.4	97.4	$+11.0\ [-2.4, 24.1]$	$+39.0\ [29.0, 49.3]$

Within each mode, the three states use the same instances, questions, system prompt, response allowance, and answer scorer. The Discovery–Raw intervals include or touch zero for all three systems in both modes. Action–Discovery intervals exclude zero for MiniMax and GPT under Direct and for all three systems under the Code Agent, but not for Qwen under Direct. Because Action provides an already repaired table, this contrast does not isolate the effect of an action label.

E.4 GBDI DISCOVERY AND QA

CDR contrasts use 141 reviewed instances and resample 34 base tasks; QA contrasts resample 27 source tables. Random-5 uses five independently permuted views. Repeated-5 performs five independent inspections of the released row order, with the inspection and final QA budgets held fixed. GPT has no Repeated-5 arm.

Table 19: Random-5 vs. Repeated-5 CDR on 141 reviewed instances. Differences are in percentage points.

System	Δ CDR	95% CI
Qwen Plus	$+19.86$	$[+11.11, +29.23]$
GLM-4.5-Air	$+5.67$	$[-1.44, +13.48]$
MiniMax-M2.5	$+19.86$	$[+9.59, +29.73]$

Table 20: End-to-end QA accuracy (%) on all 313 RADAR-T instances. GBDI combines five randomized inspections with intervention guidance before final Code Agent QA. Δ_{Direct} and Δ_{Code} denote the gains over each baseline; the paired 95% interval is for Δ_{Code}. All differences are reported in percentage points.

	Baseline		Test condition	Gain (pp)		
System	Direct	Code Agent	GBDI	Δ_{Direct}	Δ_{Code}	95% CI
Qwen Plus	11.8	54.0	64.9	+53.0	+10.9	[5.3, 16.5]
GLM-4.5-Air	4.5	41.2	49.5	+45.0	+8.3	[2.9, 14.0]
MiniMax-M2.5	44.4	32.6	51.1	+6.7	+18.5	[11.9, 25.6]
GPT-5 Mini	27.2	56.9	60.7	+33.5	+3.8	[−2.5, 9.8]
DeepSeek-V4.1-Flash	49.8	59.7	67.7	+17.9	+8.0	[2.9, 12.5]

Table 21: Paired GBDI QA gains on 313 instances (percentage points).

System	Reference	Δ	95% CI
Qwen Plus	Code Agent	$+10.86$	$[+5.26, +16.51]$
	Repeated + Skill	$+4.15$	$[-0.32, +9.03]$
MiniMax-M2.5	Code Agent	$+18.53$	$[+11.87, +25.62]$
	Repeated + Skill	$+4.79$	$[+0.00, +9.59]$
GPT-5 Mini	Code Agent	$+3.83$	$[-2.50, +9.80]$
DeepSeek-V4.1-Flash	Code Agent	$+7.99$	$[+2.94, +12.54]$
GLM-4.5-Air	Code Agent	$+8.31$	$[+2.92, +13.97]$
	Repeated + Skill	$+1.28$	$[-3.61, +6.17]$

The QA intervals for GBDI relative to the Code Agent baseline exclude zero for Qwen, GLM, MiniMax, and DeepSeek-V4.1-Flash, but not GPT. QA differences between Random-5 and Repeated-5 under equal call budgets are less certain. The corresponding CDR intervals exclude zero for Qwen and MiniMax, but not GLM.

F PROMPTS AND OUTPUT SCHEMAS

The prompt components below follow the geometry diagnostic (Section 3), the table-state comparison (Section 4), and GBDI (Section 5). The boxes reproduce the text used in the reported evaluations. Role labels distinguish system instructions from user inputs; bracketed uppercase fields denote instance-specific content. We omit redundant internal titles, format headings, lists, and command examples, and normalize minor encoding and typographical artifacts for readability.

F.1 GEOMETRY DISCOVERY

Direct discovery requests a JSON object with `candidates` and `unresolved`. Each candidate has `row_id`, `columns`, and `evidence`; there are no separate observed-value or relevance fields. The Code Agent uses the same discovery objective with the command interface shown below.

System | **Direct discovery**

SETTING

You are a diagnostic table inspector. Inspect the visible table only for the downstream question. Do not repair the table and do not answer the question.

Identify only rows for which the visible table provides concrete evidence that a query-relevant value or relation is invalid or unusable. Do not list a row merely because it is unusual, irrelevant, extreme, or difficult to interpret. When evidence suggests a problem but does not identify one row, record it as unresolved instead of listing every possible row.

Return exactly one JSON object and no markdown:

```
{
 "candidates": [
  {
   "row_id": "opaque __row_id__",
   "columns": ["visible column name"],
   "evidence": "short visible-table reason"
  }
 ],
 "unresolved": ["short unresolved issue"]
}
```

An empty candidates list is valid. Gold answers, hidden references, artifact labels, and external facts are unavailable.

User | **Direct discovery input**

Downstream question:

[QUESTION]

Visible table (also available as pandas dataframe df in repair mode):

[TABLE VIEW]

All cells are object dtype. __row_id__ is runtime metadata, not a source-table field.

Code Agent discovery uses two YAML commands. The `python` command supports inspection, and `submit_localization` submits the JSON payload. The runtime allows at most five response rounds, maintains a persistent namespace, and restores the visible `df` after each inspection command. This discovery endpoint differs from the `done` submission used by the QA agent.

System | Code Agent discovery

SETTING

You are a diagnostic table inspector. Inspect the visible table only for the downstream question. Do not repair the table and do not answer the question.

Identify only rows for which the visible table provides concrete evidence that a query-relevant value or relation is invalid or unusable. Do not list a row merely because it is unusual, irrelevant, extreme, or difficult to interpret. When evidence suggests a problem but does not identify one row, record it as unresolved instead of listing every possible row.

You may inspect the complete pandas dataframe df with read-only Python. It contains the exact visible CSV and all cells are object dtype. Use Python when it helps check complete-column patterns, cross-column relations, masks, or candidate coverage. pd, np, json, math, re, and statistics are already available. Variables persist across inspection turns, while df is restored after each command.

On every turn return exactly one YAML command and no prose.

To inspect the table:

```
command: python
kwargs:
  code: |
    print(df.head())
```

To finish:

```
command: submit_localization
kwargs:
  payload: |
    {"candidates":[{"row_id":"opaque __row_id__","columns":["visible column
        name"],"evidence":"short visible-table reason"}],"unresolved":[]}
```

An empty candidates list is valid. Candidate row ids and columns must exist in the visible table. Gold answers, hidden references, artifact labels, and external facts are unavailable.

User | Code Agent discovery input

Downstream question:

[QUESTION]

Visible table (also available as pandas dataframe df):

[TABLE VIEW]

All cells are object dtype. __row_id__ is runtime metadata, not a source-table field.

F.2 TABLE-STATE DIAGNOSTIC

Raw uses the standard table–question prompt for each mode. Discovery and Action retain the mode-specific system prompt and prepend the following notices to the user turn. Discovery marks erroneous table values as `<<ERROR:value>>`; Action supplies the table after reviewed edits and drops have been applied. The notices are identical across Direct and Code Agent.

User | Discovery condition

The values marked <<ERROR:...>> have been verified as erroneous and relevant to the question; the text inside each marker is the original erroneous value, not a correction. Treat these markings as facts when answering: correct a marked value when the table supports a reliable replacement, or exclude its row if it cannot be repaired.

User | **Action condition**

The table below has already been repaired for this question. All reviewed errors relevant to the answer have been corrected, and rows that could not be repaired have been removed. Treat the supplied table as final and answer the question without making further changes.

F.3 Multi-View Inspection Schemas

Each view is processed in a fresh context under Random-M or Repeated-M, as specified in Appendix B.4. Within a system, the two conditions share the inspection prompt and parser. The evaluation uses the following two schemas.

Qwen Plus, GLM, MiniMax, and DeepSeek-V4.1-Flash. The output contains `row_id`, a `columns` list, and `evidence`, together with the top-level `unresolved` list.

System | **Multi-view discovery**

SETTING

You are a diagnostic table inspector. Inspect the visible table only for the downstream question. Do not repair the table and do not answer the question.

Identify only rows for which the visible table provides concrete evidence that a query-relevant value or relation is invalid or unusable. Do not list a row merely because it is unusual, irrelevant, extreme, or difficult to interpret. When evidence suggests a problem but does not identify one row, record it as unresolved instead of listing every possible row.

Return exactly one JSON object and no markdown:

```
{
 "candidates": [
  {
   "row_id": "canonical __row_id__",
   "columns": ["visible column name"],
   "evidence": "short visible-table reason"
  }
 ],
 "unresolved": ["short unresolved issue"]
}
```

An empty candidates list is valid. Gold answers, hidden references, artifact labels, and external facts are unavailable.

User | **Multi-view discovery input**

Downstream question:

[QUESTION]

Visible table:

[TABLE VIEW]

All cells are object dtype. __row_id__ is runtime metadata identifying the canonical source row; it is not a source-table field.

GPT-5 Mini. The prompt requests observed values and query relevance alongside the candidate rows, columns, and evidence. The parser accepts blank observed values for missing cells and requires nonempty relevance and evidence text. It normalizes the singular `column` field to a column list before aggregation.

System | GPT-5 Mini discovery

You are a diagnostic table inspector. You are given a downstream question and a complete table view. The __row_id__ column contains stable, nonordinal row identifiers used to map records across views. It is not a source-table field and does not encode row order.

Identify only rows whose visible contents provide concrete evidence of an unreliable value or relation that can affect the requested computation. Do not nominate a row merely because it is unusual, extreme, irrelevant, excluded by an ordinary query predicate, or difficult to interpret.

For each supported candidate, report the row ID, one affected visible column, the displayed value in that cell, why that field can enter the requested computation, and one concise observation that can be checked in the displayed table. Emit a separate record for each affected column when needed.

Do not repair the table or answer the question. Do not propose a replacement or choose EDIT, DROP, or NO-OP. Gold answers, recovered tables, artifact labels, task identifiers, and predictions from other views are unavailable.

Return exactly one JSON object and no Markdown:

```
{
 "candidates": [
  {
   "row_id": "rid_4f09c2a17b6e",
   "column": "visible column name",
   "observed_value": "value displayed in this cell",
   "query_relevance": "why this field can enter the requested computation",
   "visible_evidence": "one concise checkable observation from this table"
  }
 ]
}
```

An empty candidates list is valid.

User | GPT-5 Mini discovery input

Downstream question:

[QUESTION]

Complete visible table view:

[TABLE VIEW]

All source-table cells are displayed as strings. __row_id__ is stable, nonordinal runtime metadata used only to identify rows across views.

F.4 Evidence Ledgers

For a fixed instance and row r, let $c_M(r) = \sum_{t=1}^{M} \mathbf{1}[r \in \widehat{\mathcal{R}}_t]$ count the views that nominate the row. A ledger entry has the conceptual form

$$L_M(r) = \big[r,\ \mathcal{J}_r,\ \mathcal{X}_r,\ c_M(r),\ \mathcal{E}_r\big], \tag{11}$$

where $\mathcal{J}_r$ contains the named columns, $\mathcal{X}_r$ their visible values, and $\mathcal{E}_r$ the retained observations and any supplied relevance claims. Entries are included whenever $c_M(r) \geq 1$. The values in $\mathcal{X}_r$ are copied from the canonical table; statements in $\mathcal{E}_r$ remain system-generated claims that the final agent must verify.

Both implementations retain the union of valid row nominations and count each row at most once per view. Ledger construction uses only system nominations and requires no support beyond one valid nomination. The text-processing differences described below are held constant across view families within each system.

Qwen, GLM, MiniMax, and DeepSeek-V4.1-Flash. Before constructing the final agent's input, the program unions valid column names, normalizes whitespace, and removes duplicate observations

in view order. It also removes observations that match predefined, case-insensitive expressions suggesting repair actions or replacement values, such as "edit," "drop," "should be," or "correct value." This filtering removes the matching observation, while retaining the row nomination, named columns, and support count for verification. The `unresolved` field is not forwarded, and rows are ordered by decreasing support.

The resulting ledger is passed to the final agent in the user-message block below. Each entry identifies the nominated row and includes its named columns, per-view vote mask, visible cell values, and retained observations. A vote mask uses `1` for nomination, `0` for absence, and `?` for an unparseable view. Empty candidate sets produce no appended ledger.

User | **Discovery ledger**

Five independent inspections examined semantically equivalent views of this same table. Every nominated canonical row is retained below, including single-view nominations. Consensus is an inspection-priority signal, not proof that a row is wrong and not authorization to change or exclude data.

Before answering:

1. establish the exact query operation and answer scope;
2. inspect every ledger row in the complete canonical table;
3. use the vote pattern and observations only to prioritize verification;
4. ignore nominations not supported by the visible table; and
5. directly compute and submit the final answer.

The ledger contains no Gold labels, prescribed operations, replacements, recovered table, Repair Skill, or answer.

GPT-5 Mini. GPT merges valid nominations for the same row, unions columns, and deduplicates relevance and evidence text. The resulting ledger includes visible cell values and retains all evidence text. Rows are ordered by decreasing support. The ledger presents nominations as claims to verify and reports when no row was nominated. Protocol matching therefore applies within each system; ledger format differs between GPT-5 Mini and the other four systems.

F.5 Intervention Guidance

Both Random-5 and Repeated-5 use the Intervention Skill below. The guidance ablation replaces it with the generic guidance box while retaining the same intervention turn and final QA budget.

System | **Generic intervention guidance**

- Treat every ledger coordinate and observation as a lead to verify against the complete visible table, not as proof of corruption or a prescribed operation.
- Preserve the literal answer program required by the question.
- EDIT only when visible evidence determines one exact replacement.
- DROP only when a record required by the answer program is demonstrably unusable and no unique replacement is justified.
- Otherwise use NO-OP, including for usable, irrelevant, unusual-but-valid, or ambiguous data.
- Apply justified operations to work_df before rerunning the unchanged answer program.

System | **Intervention Skill**

Entry Gate

- Complete the original question-answering task in one code-agent context.
- Treat the balanced-view ledger as a fallible inspection aid. A coordinate or observation nominates a candidate but never proves corruption or prescribes an operation.
- Before using an artifact rule, verify one of its direct visible preconditions: an answer-active missing marker, a hard invalid value, or a representation that blocks parsing of an otherwise recoverable value.
- If none of those preconditions is visible, skip the artifact rules and solve the question with the ordinary code-agent procedure. Do not spend an extra round trying to manufacture an intervention.
- Preserve the literal answer program: returned entity, population, filters, grouping unit, ordering, tie rule, selected field, units, aggregation, and rounding.

Operation Boundary

- For each verified answer-relevant problem choose one outcome:
  - **EDIT** only when independently usable visible evidence determines one exact replacement;
  - **DROP** only when the record is demonstrably unusable for the question and no unique replacement is justified;
  - **NO-OP** when the value is usable, irrelevant, unusual but valid, or supported only ambiguously.
- Candidate recurrence across rotations changes inspection priority, not authorization.
- Apply supported interventions before filtering, selection, ranking, adjacency, grouping, aggregation, or rounding, then rerun the unchanged answer program from the beginning.

Missingness

- Activate only for a null, blank, whitespace-only string, or explicit missing marker in a field consumed by the answer program or required to validate it.
- Inspect returned values, filters, selectors, rank/order fields, ratio operands, and only their necessary companions. Do not bulk-fill unrelated columns.
- Test these visible recovery routes:
  - an exact same-row total/parts, difference, product, ratio, average, or start/end/duration identity;
  - an equivalent machine/display field or redundant representation;
  - a complete-key mapping with exactly one value throughout comparable rows.
- Verify a proposed relation on every row where target and sources coexist. Reject it when a required source is missing, malformed, or contradictory.
- Use an alternate representation only to replace a missing answer-active value. Preserve every observed usable value in the primary question-specified field; never recompute the entire column from its alternate representation.
- EDIT only the missing field with its uniquely implied value.
- DROP only when the unrecoverable field is required to decide participation or contribution. Use NO-OP when it cannot affect the answer.

Bad Values

- Activate only for a spreadsheet error, impossible token, explicit sentinel, or value that fails an answer-active field's necessary type or exact visible relation.
- Numeric sentinels remain numeric after pd.to_numeric; detect and mask an established sentinel explicitly before testing missingness or recovery.
- Establish a sentinel from its visible role and repeated conflict with a necessary relation, not merely from being negative, large, rare, or zero-padded.
- Preserve identifiers, codes, names, labels, titles, URLs, descriptions, categories, types, schools, and locations even when they contain digits or unusual text.

System | **Intervention Skill** continued

- Before DROP, test all applicable exact recovery routes:
  - a redundant machine/display or category/code representation;
  - a same-row total/parts, product/factors, rate/base, annual/hourly, or start/end/duration identity;
  - a date-derived field or selector fixed by the question's grouping and ordering;
  - a complete-key mapping with one value throughout comparable rows.
- Validate each relation on unaffected rows and verify every source. If one route yields exactly one value, EDIT before any query operation.
- DROP only when an answer-active record remains unusable and no independent anchor survives. Use NO-OP when evidence or repair direction is ambiguous.

Inconsistent Formatting

- Activate only when representation blocks parsing or comparison of an answer-relevant value while its semantic value remains recoverable.
- Inspect the complete set of distinct answer-active token patterns before writing the parser; do not infer a delimiter or wrapper from one example.
- Match parsing to field role: numeric wrappers for measures, ordinal parsing for ranks, dates for time fields, and categories only when visible redundant fields establish the mapping.
- Parse the complete token with a role-specific pattern. Do not use the first numeric substring when another digit, sign, decimal mark, exponent, percent, or unit remains unexplained.
- Preserve every digit, sign, decimal position, exponent, percentage interpretation, and declared unit scale.
- Remove currency marks, grouping separators, units, percent notation, or text wrappers only when the complete token has one interpretation.
- Parse ordinal or wrapped rank values before filtering the requested ranks. Do not silently coerce malformed rank tokens to NaN and drop them.
- Collapse one duplicated decimal mark only when that deletion yields a unique valid number matching the column's observed grammar. Never invent a decimal position.
- Edit only visibly formatted cells. Do not rewrite an already usable column merely to change its Python dtype.
- Standardize a category only when visible variants clearly denote the same entity and a canonical form is already present. Formatting alone never justifies DROP.

Multi-Turn Execution Contract

- Keep canonical df read-only. Use parsed Series, masks, or a local work_df = df.copy().
- Prefer one compact inspection command that prints the exact active columns, malformed token patterns, and required relation diagnostics, followed by one complete answer command.
- Use full-table masks and original DataFrame indices; never copy indices from a sorted, filtered, or truncated preview.
- Before submission, verify that parsing found the expected records and that the computation produced the requested finite scalar, entity, or list.
- A failed assertion, empty parse, diagnostic count, or exception message is not an answer. Correct the parser or computation and execute it again.
- Once a valid result is observed, submit it immediately with the official done(answer) command; do not spend later rounds re-deriving it.

F.6 QA CONVERSATION

The system message combines the RADAR Code Agent base prompt, the selected guidance, and the intervention-turn contract, in that order. The command schema and permissions remain those of the base prompt, while the selected guidance specifies the evidence required for allowed actions. The base prompt includes the `python` and `done` schemas.

System | Code Agent base prompt

SETTING: You are an expert-level data scientist. Your job is to answer a data driven question in rigorous manner given a data table.

In your analysis:

- Carefully address

1) missing data: empty or null entries simulating incomplete information
2) bad values: clearly erroneous or placeholder entries (e.g., -1, 9999, TEST, #REF! etc.)
3) outliers: implausible extreme values that distort analysis (e.g., 220 breathing rate per minute)
4) inconsistent formatting: variations in representing the same value (e.g., 22 lbs, 22 pounds, weight = 22)
5) inconsistent logic: cross-field contradictions violating common-sense logic (e.g., end time before start time)

- Attempt to safely recover or correct flawed data when reasonable based on the existing data. If data is irrecoverable or suspect, discard the row.

You will be working within a Python shell and can use the following commands to answer the question.

AVAILABLE COMMANDS:

```
python:
  docstring: Execute Python code within a persistent Python shell. The shell maintains
    state across executions, so variables and imports from previous runs remain available.
    When first using this command, the data table is provided as a global variable
    named `df`, and `pandas` has already been imported as `pd`.
  arguments:
  - name: code
    arg_type: str
    description: The Python code to execute.
    required: true
  demonstration:
    command: python
    kwargs:
      code: <arg value>
```

```
done:
  docstring: Indicate that we arrived at the final answer and provide the answer.
    Use this command only when you have arrived at the final answer.
  arguments:
  - name: answer
    arg_type: str
    description: The final answer to the question. Do not apply any formatting, bolding,
      or markup. If the question asks for a list of values, then the answer should
      be a comma-separated list of values (e.g., '42, 43, 44')
    required: true
  demonstration:
    command: done
    kwargs:
      answer: <arg value>
```

Please enclose all command kwargs values in ""

System | **Code Agent base prompt** continued

RESPONSE_FORMAT:

Each response must include:

1. A DISCUSSION field — where you will methodically break down the reasoning process, illustrating how you arrive at conclusions and decide what to do next.
2. A command field — properly formatted YAML within triple backticks and following the structure from COMMANDS.

Important rules:

- Always include exactly one DISCUSSION and one command block.
- Ensure the command block is properly formatted YAML with proper indents and newlines (see the example below).

For example, given a question asking for the average income. You might respond:

DISCUSSION

Let's think step by step. We need to first find the average income of the population. We can do this by summing up the income column and dividing by the number of rows.

```
command: "python"
kwargs:
  code: |-
    income_avg = df['income'].sum() / len(df)
    income_avg
```

System | **Intervention-turn contract**

The first assistant turn is reserved for query-conditioned intervention. Do not answer the downstream question and do not return done in this turn. Return exactly one ordinary YAML python command.

The Python namespace already contains the canonical object-typed df and a

separate work_df = df.copy(). In one executable block:

1. identify the literal answer program required by the question;
2. re-check only query-relevant ledger candidates and any directly necessary companion cells against the complete visible table;
3. decide EDIT, DROP, or NO-OP under the operation boundaries;
4. apply every justified EDIT or DROP to work_df, leaving df unchanged;
5. set intervention_log to a list of concise dictionaries containing the candidate coordinate, visible evidence, chosen action, and reason;
6. print a compact audit summary, but do not compute or print the final answer.

An empty modification set is valid only after the applicable candidates have been checked; in that case retain work_df unchanged and record the relevant NO-OP decisions. The next user turn will contain the Python observation and open final QA. The same message history and Python namespace will persist.

User | Initial QA input

Begin!
Data table (stored in a pandas dataframe named df):
[CANONICAL TABLE]
All cells in the df are object data type, regardless of their appearance.
Question:
[QUESTION]
[RENDERED LEDGER]

BEGIN RESERVED INTERVENTION TURN

Perform the intervention preturn now. Do not answer the question in this turn.

The blank-ledger diagnostic omits the ledger block while retaining the guidance, contract, and canonical table. It therefore measures the effect of ledger availability within the same candidate-guided workflow. Before the intervention turn, the runtime initializes `work_df = df.copy()` and an empty `intervention_log`. It executes the returned Python command, reports its observation and contract status, and appends the continuation below. The runtime would restore the initial value if either object had an invalid type, while retaining the trajectory in the primary score. This fallback was not triggered in any of the five reported 313-instance Random runs: both objects had valid types after every reserved turn. Final QA continues in the same conversation and namespace.

User | Continuation after intervention

Continue as the same agent in the same context. Answer the original question using the persistent work_df and the unchanged literal answer program. The intervention ledger is fallible: retain only operations justified by visible evidence. You may use ordinary Python inspection, then submit exactly one final answer with the official done(answer) command. Do not export a repaired table.

Up to five further responses may use ordinary `python` commands or submit `done` with `kwargs.answer`. Python observations are returned using the benchmark observation template. Benchmark QA scoring credits only a valid final submission, not an answer reconstructed from code or stdout. Neither the ledger nor the internal working table is an additional benchmark output.

Our anonymous repository is available at `https://anonymous.4open.science/r/GBDI-ICLR-2027-85BD/`.