

QUIETT: Query-Independent Table Transformation for Robust Reasoning

Gaurav Najpande Tampu Ravi Kumar* Manan Roy Choudhury*
Neha Valeti* Yanjie Fu Vivek Gupta

Arizona State University

{gnajpand, traviku2, mroycho1, nvaleti1, yanjie.fu, vgupt140}@asu.edu

*Equal contribution (co-second authors)

Abstract

Real-world tables often contain schema inconsistencies, heterogeneous value formats, and implicit relational structures that degrade table reasoning and question answering. Existing approaches address these issues at query time, repeating normalization and restructuring for every new question and producing representations that do not generalize to unseen queries. We introduce **QUIETT**, a *transform-first, query-later* framework that converts each raw table into a single SQL-ready canonical representation before any evaluation query is observed, under three constraints: no evaluation query access, information preservation, and single-table reuse. QUIETT operates through a three-stage pipeline: issue probing, where synthetic queries surface structural deficiencies such as ambiguous schemas and heterogeneous formats; plan generation; and structured execution. Experiments on four benchmarks across five model families show consistent improvements over the evaluated baselines, which span direct prompting and multi-step reasoning methods, with particularly strong gains on a manually annotated challenge set of structurally diverse unseen questions.

1 Introduction

Table based reasoning is a central component of many natural language understanding tasks, including question answering, information retrieval and decision support (Zhu et al., 2021; Herzig et al., 2020), yet real-world tables rarely arrive in a form suitable for natural-language or symbolic querying. For example, result columns embed outcome and score in a single string ("W 24-17"), columns mix scales and currencies ("1.2M", "EUR 850K", "1,500,000"), symbols stand in for missing values ("†", "-"), and hierarchical headers collapse ambiguously when flattened (Cheng et al., 2022).

Code and data are available at <https://github.com/CoRAL-ASU/QUIETT>.

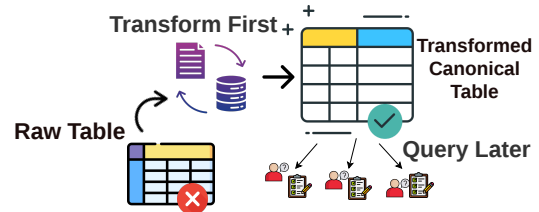


Figure 1: QUIETT transforms a raw table into a single SQL-ready canonical representation before any evaluation query is observed. The transformed table is reused across all queries without modification.

Normalization methods such as NormTab (Nahid and Rafiei, 2024a) and E5 (Zhang et al., 2024) reduce surface-form variability by standardizing values and applying lightweight structural fixes, but leave schema-level issues including heterogeneous layouts, implicit relations, and undecomposed attributes to downstream reasoning. Because these methods assume mostly fixed schemas, they cannot resolve the structural deficiencies that degrade reasoning on real-world tables.

Reasoning-coupled methods integrate transformation directly into the query pipeline. Chain-of-Table (Wang et al., 2024) and TableReasoner (Xiong et al., 2025b) perform query-driven schema selection and iterative restructuring; Binder (Cheng et al., 2023) and Dater (Ye et al., 2023) condition execution and decomposition on the observed question. These methods improve accuracy on seen queries but couple table representation to the query at hand: the same table yields different transformed views for different questions, transformation work repeats for every new query, and representation quality cannot be evaluated independently of reasoning behavior. In every case, reuse across questions, models, and tasks is structurally precluded.

Standard data engineering offers a useful analogy. In the widely adopted medallion architecture, data engineers prepare analytics-ready representations before any downstream query is posed (Chu

et al., 2016), and that single representation serves diverse analytical workloads without query-specific adaptation. Table QA lacks an equivalent. QUIETT draws on this intuition, motivated by the medalion architecture’s principle of preparing analytics-ready representations before queries arrive. However, applying this principle to open-domain table QA introduces challenges that ETL does not face. ETL assumes a known target schema and a fixed query workload. In open-domain table QA, tables arrive with unknown structure and no future queries are observed at transformation time. QUIETT addresses this by treating schema discovery as intrinsic to transformation: it derives the canonical representation from the raw table’s own structural properties through issue probing. The result is a framework evaluated by a criterion that ETL systems never impose: a single canonical representation, constructed without query access and without a predefined schema, must improve downstream QA across diverse datasets, model families, and reasoning strategies.

We introduce QUIETT, a *transform-first, query-later* framework that converts each raw table into a single SQL-ready canonical representation before any evaluation query is observed. QUIETT operates under three explicit constraints: it accesses no downstream evaluation queries during transformation, retains grounding and provenance for transformed content, and produces exactly one canonical table reused across all queries. These constraints isolate the effect of table representation from model and prompt variation. When the same reasoning model and prompting strategy are held fixed, performance differences between raw and canonical tables reflect representation quality alone. QUIETT targets the regime where tables are preprocessed once and reused across arbitrary future queries, complementing rather than replacing query-adaptive methods that condition transformation on observed queries.

Our main contributions are: (1) We introduce QUIETT, a three-stage transform-first framework that converts raw tables into a single SQL-ready canonical representation before any evaluation query is observed, decoupling table preprocessing from query-time reasoning. (2) Across four benchmarks and five model families, QUIETT consistently outperforms the evaluated baselines by 4–8 F1 and 3–6 accuracy points. These baselines span direct prompting and multi-step reasoning methods. (3) We construct a manually annotated

challenge set of 2,500 questions over 529 tables (Fleiss’ $\kappa = 0.93$) targeting structural diversity absent from standard benchmarks. (4) Ablation studies show that issue probing is the primary driver of gains, with transformation quality mattering more than QA model capacity.

2 QUIETT Framework

2.1 Problem Setup

Let $T_{\text{raw}} \in \mathcal{T}$ denote a raw, semi-structured table and $\mathcal{Q}$ the space of possible downstream queries. We construct a transformation function $f : \mathcal{T} \rightarrow \mathcal{T}$ that maps T_{raw} to a canonical, SQL-ready table $T_C = f(T_{\text{raw}})$, reused across all subsequent queries without modification, under three constraints:

(a) No evaluation query access. f depends only on T_{raw} , not on any downstream evaluation query $q \in \mathcal{Q}$.

(b) Information preservation. Values in T_C remain grounded in source-table content, with provenance retained for transformed values. This constraint concerns grounding and recoverability rather than exact reconstruction of the original surface representation.

(c) Single-table reuse. T_C is produced exactly once and reused unchanged, so performance differences between T_{raw} and T_C reflect representation quality alone.

Canonicalization objective. Without a known future workload, a canonical representation need not be unique. QUIETT resolves the table-intrinsic issues found during probing using a fixed operator vocabulary, with table-specific cases handled through a validated custom operator. A source column is retained whenever derived columns capture only part of its content. The goal is to produce one reusable SQL-ready representation rather than anticipate every future question. This process may under-transform issues missed during probing or over-transform useful surface forms, and multiple valid decompositions may exist.

Given a fixed reasoning model M and prompting strategy S , a query q is answered as $\hat{a} = M(S(q, T_C))$, isolating the effect of table representation from model or prompt variation.

2.2 Pipeline Overview

QUIETT instantiates this setting as a three-stage pipeline (Algorithm 1): (1) probe the raw table to identify structural issues, (2) generate a transformation plan that resolves them, and (3) execute the plan to produce the canonical table T_C .

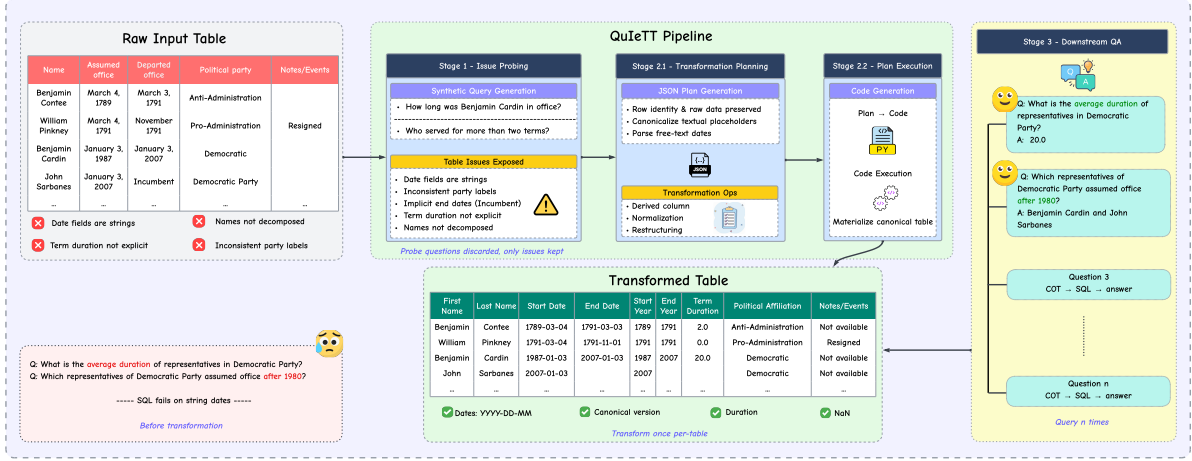


Figure 2: QUIETT framework for robust table reasoning. **Stage 1: Issue Probing.** Synthetic queries are generated from the raw table to expose structural issues such as inconsistent formats, implicit relations, missing derived fields, and undecomposed attributes. **Stage 2: Transformation.** Detected issues guide a transformation plan that applies normalization, parsing, restructuring, and derived column creation, executed via code to materialize a canonical, SQL-ready table. **Stage 3: QA.** The transformed table is reused across all downstream queries, enabling accurate and consistent reasoning without modifying the underlying model or prompting strategy.

Algorithm 1 QUIETT Framework

```

Input: raw table  $T_{\text{raw}}$ 
Output: canonical table  $T_C$ , answers  $\{\hat{a}\}$ 
Stage 1: Issue Probing
 $Q_{\text{probe}} \leftarrow \text{GenerateSyntheticQueries}(T_{\text{raw}})$ 
 $\mathcal{I} \leftarrow \text{ExtractIssues}(Q_{\text{probe}})$   $\triangleright$  probes discarded
 $\pi \leftarrow \text{GeneratePlan}(T_{\text{raw}}, \mathcal{I})$ 
 $\text{Validate}(\pi)$   $\triangleright$  format, completeness
Stage 2: Transformation
 $T \leftarrow T_{\text{raw}}$ 
for operation  $o \in \pi$  do
  if  $o$  is deterministic then
     $T \leftarrow \text{ApplyOp}(o, T)$ 
  else
     $T \leftarrow \text{GenerateAndExecuteCode}(o, T)$ 
  end if
end for
 $T_C \leftarrow T$   $\triangleright$  information preserved
Stage 3: Downstream QA
for query  $q$  do
   $r \leftarrow \text{Reason}(q, T_C)$   $\triangleright$  CoT  $\rightarrow$  SQL
   $\hat{a} \leftarrow \text{Execute}(r, T_C)$ 
end for
Return  $T_C, \{\hat{a}\}$ 

```

2.3 Issue Probing

Issue probing generates a set of synthetic queries $Q_{\text{probe}}(T_{\text{raw}}) = \{q_{\text{syn}}^{(1)}, \dots, q_{\text{syn}}^{(M)}\}$ directly from T_{raw} , where each query targets a structural pattern that would hinder SQL or general-purpose querying, such as ambiguous column schemas, heterogeneous value formats, type inconsistencies, packed multi-value cells, or implicit relational structure (see Appendix I.1 for prompt design and issue aggregation). Each probe query carries a `failure_reason` field that records why the raw table cannot answer it reliably; grouping queries

by identical failure reasons yields the issue set $\mathcal{I}(T_{\text{raw}}) = \{i_1, i_2, \dots, i_N\}$ that drives transformation planning. The probe queries themselves are discarded after issue extraction and later stages access only $\mathcal{I}(T_{\text{raw}})$, preserving constraint (a). We generate 12-20 probe queries per table. Appendix F shows this budget achieves 98.5% structural coverage. QUIETT is therefore independent of downstream evaluation queries but not fully query-agnostic: the synthetic probes serve as table-intrinsic diagnostic seeds that are discarded after issue extraction, and no later stage accesses their content.

Handling hierarchical tables. Hierarchical tables encode information across implicit parent-child levels rather than explicit columns (Zhao et al., 2023), which can cause issue probing to miss cross-level dependencies. Before probing, we apply a deterministic flattening step that collapses multi-level headers into composite column names by concatenating parent and child labels with a separator (e.g., eastern ontario / french-language workers), as illustrated in Appendix H. While this produces a uniform columnar representation that issue probing can operate over, the underlying semantics remain implicitly encoded in column names rather than explicit fields. Issue probing then surfaces these residual structural deficiencies, and the transformation plan resolves them by decomposing composite headers into atomic attributes such as region and worker type.

2.4 Transformation Planning

A transformation plan is an ordered sequence $\pi = \langle o_1, \dots, o_K \rangle$, where the plan length K and the specific operations are determined dynamically from the issue set $\mathcal{I}(T_{\text{raw}}) = \{i_1, i_2, \dots, i_N\}$ detected in Stage 1. Each operation o_i declares its input columns, output columns, and dependencies, yielding an auditable transformation graph. Collectively, π induces an explicit target schema $\mathcal{S}_\pi$ that fixes the column set, names, types, and derived fields of the transformed table before any execution begins. The plan is produced by a function $\mathcal{P} : \mathcal{T} \times \mathcal{I} \rightarrow \Pi$, giving $\pi = \mathcal{P}(T_{\text{raw}}, \mathcal{I}(T_{\text{raw}}))$. Constraining operation semantics while allowing variable plan length and composition lets QUIETT produce reproducible plans while limiting hallucinated or unsupported transformations.

2.5 Plan Execution

The plan π is executed by a structured engine that separates symbolic planning from code-level implementation, following the design principle of Khoja et al. (2025). Given π and T_{raw} , the executor processes each operation $o_i \in \pi$ according to its declared inputs, outputs, and dependencies, producing the canonical table $T_C = E(\pi, T_{\text{raw}})$.

Execution follows two paths depending on the operation type. Operations with registered implementations (see Appendix A) are applied directly through a fixed operator library. Operations requiring table-specific logic are handled by LLM-generated Python code executed against the current table state. In both paths, each step is restricted to its declared inputs and outputs in π . Steps that fail validation trigger a retry: the validator appends the error message to the prompt and re-queries the model up to two additional times. If all attempts fail, the original column values are preserved and execution continues with the next plan step. This fallback prevents invalid outputs from being passed to later steps, limiting cascading errors.

The executor enforces schema conformance throughout, so the final table conforms to $\mathcal{S}_\pi$ regardless of which path individual steps take. Whenever a transformation changes the surface form of a value, such as parsing a date or splitting a packed cell into separate columns, the resulting values remain grounded in the corresponding source content. Columns that require no transformation are carried over to T_C unchanged; exact reconstruction of normalized surface forms is not guaranteed (see Appendix G). Because T_C is produced once and

cached, the pre-processing cost is amortized across all downstream queries (see Appendix B.3).

2.6 Downstream Question Answering

Once T_C is produced, downstream question answering proceeds over the canonical table without further modification. Given a query q , the reasoning model M generates a chain-of-thought reasoning trace that decomposes the question into sub-steps, then translates it into a SQL query executed against T_C to produce the final answer $\hat{a}$. Because T_C already conforms to $\mathcal{S}_\pi$, this CoT→SQL pipeline operates over a consistent representation regardless of the original table’s structural complexity.

3 QUIETT Challenge Set

To evaluate generalization under increased structural complexity, we construct a challenge set from WikiTQ (Pasupat and Liang, 2015) and NQ-Table (Kwiatkowski et al., 2019), two datasets whose structural issues, heterogeneous formats, implicit relations, and irregular schemas are latent and unlabeled, making them the most suitable sources for stress-testing table-intrinsic representation.

Dataset Creation. For each table, we generate structurally challenging question-answer pairs directly from the raw table, targeting structural patterns that hinder reliable answering without transformation, following prior LLM-based table QA benchmarks (Choudhury et al., 2025; Tian et al., 2026; Choudhury et al., 2026). We acknowledge that generating questions from raw tables may not surface all structural complexity compared to canonical representations, but ensures question generation is independent of QUIETT’s transformation. To ensure evaluation neutrality, we apply two controls. First, all answers must be derivable exclusively from the source raw table. Second, all question-answer pairs are manually verified by three independent NLP annotators against the raw table, without access to the canonical representation or model outputs. Because all methods share the same fixed question set and QUIETT has no access to these questions during transformation, any representational alignment introduced during question generation cannot advantage QUIETT at evaluation time.

Dataset Verification. Three independent NLP annotators validate each question-answer pair against the raw source table. Table 1 reports inter-annotator agreement, which is consistently high across all

Expert Pair	Cohen’s Kappa	Jaccard Coeff.
$\alpha_{1,2}$	92.9	96.9
$\alpha_{2,3}$	92.0	96.6
$\alpha_{1,3}$	94.9	97.7
$\alpha_{1,2,3}$ (Fleiss’ κ)	0.9328	0.9562

Table 1: Inter-rater agreement for challenge set annotation.

Statistic	Values	Table Dimension
Total tables	529	Mean: (20, 11)
Total questions	2,500	Median: (8, 8)
Questions per table	4 / 5 / 7	Min: (1, 1)
(mean / median / max)		Max: (380, 76)

Table 2: **Challenge set composition and table structure statistics.** The set comprises 2,500 questions over 529 tables. Tuple-valued entries in the Table Dimension column report (rows, columns); the wide size range reflects the structural diversity the challenge set targets.

pairs, supporting the reliability of the challenge set.

3.1 Dataset Statistics.

Table 2 details scale and question type distribution.

We evaluate QUIETT across four benchmarks, five model families, and two reasoning paradigms, holding downstream QA models and prompting strategies fixed across all conditions.

4 Experiments

4.1 Benchmark Datasets.

We conduct experiments on four widely used table QA benchmarks covering diverse table structures and reasoning requirements. WikiTQ (Pasupat and Liang, 2015) consists of semi-structured Wikipedia tables paired with natural language questions, emphasizing lookup and aggregation. NQ-Table (Kwiatkowski et al., 2019) is derived from Natural Questions and contains noisier web tables with implicit schemas. SequentialQA (Iyyer et al., 2017) requires multi-turn reasoning where later questions depend on earlier answers. HiTab (Cheng et al., 2022) focuses on hierarchical and numerical reasoning over complex table structures. We additionally evaluate on the challenge set constructed from WikiTQ and NQ-Table (Section 3) to assess robustness on structurally harder cases. WikiTQ and NQ-Table are selected for the challenge set because their structural issues, heterogeneous formats, implicit relations, and irregular schemas are latent and unlabeled, making them the most suitable sources for stress testing table-intrinsic representation quality. HiTab’s difficulty is well characterized by its explicit hierarchical structure and SequentialQA’s by conversational context, so neither isolates the schema-level variation the challenge set targets.

4.2 Evaluation Metrics.

We report both token-level F1 and accuracy across all datasets. Before scoring, we apply lightweight surface normalization to both predictions and gold answers such as date strings are converted to a canonical format (*e.g.*, “Jan 5, 2018” → “2018-01-05”), non-semantic characters such as @, ;, and | are removed, and Roman numerals are mapped to integers (*e.g.*, “III” → “3”). Token-level F1 is then computed over the normalized tokens, and accuracy is computed as exact match after the same normalization. We report both metrics because most table QA benchmarks and prior methods report accuracy as their primary metric, while F1 additionally captures graded recovery for list-valued and compositional answers where a prediction may be partially correct. All methods are evaluated using the same normalization and scoring scripts.

4.3 Models and Baselines.

We evaluate across five large language models: Gemini-2.0-flash and Gemini-2.5-flash (Google, 2023), DeepSeek-V3.1 (DeepSeek-AI, 2024), Qwen3-80B, and GPT-OSS-120B¹, spanning both proprietary and open-weight settings. **Baselines** include prompting-based methods (CoT, CoT+SQL, E5 (Zhang et al., 2024), EEDP (Srivastava et al., 2024)), normalization and restructuring methods (NormTab (Nahid and Rafiei, 2024a), TabSQLify (Nahid and Rafiei, 2024b), TableReasoner (Xiong et al., 2025b), GraphOTTER (Li et al., 2024)), an execution-based system (Binder (Cheng et al., 2023)), and multi-step agentic methods (Chain-of-Table (Wang et al., 2024), Program-of-Thoughts). All baselines are implemented following their original papers and publicly released code, using default hyperparameters and prompting strategies without additional tuning. Each baseline receives the input format and preprocessing specified by its original method. For CoT and CoT+SQL, we use the prompt structures summarized in Appendix J, all other baselines use the authors’ publicly released implementations and prompts with default settings. All methods use the same answer-normalization and scoring pipeline. Appendix J documents baseline inputs, preprocessing, and execution-failure handling.

¹Qwen3-80B and GPT-OSS-120B are open-weight models accessed via Vertex AI Model-as-a-Service. GPT-OSS-120B is a 120B-parameter model from OpenAI; Qwen3-80B is an 80B-parameter model.

Category	Method	Gemini 2.5								GPT-OSS							
		WikiTQ		NQ-Table		SequentialQA		HiTab		WikiTQ		NQ-Table		SequentialQA		HiTab	
		F1	EM	F1	EM	F1	EM	F1	EM	F1	EM	F1	EM	F1	EM	F1	EM
Direct	CoT	74.31	72.85	71.22	66.36	61.98	57.21	75.21	71.00	69.40	66.56	66.60	63.54	60.26	58.29	53.90	50.80
	CoT + SQL	54.66	52.49	36.64	30.96	58.23	53.37	69.01	67.00	43.00	42.29	47.90	33.95	55.20	49.60	31.90	30.70
	NormTab	58.70	56.19	53.00	51.05	50.00	49.92	56.81	54.60	65.91	62.61	50.30	48.34	48.60	43.10	54.67	52.00
	E5	74.35	70.30	55.90	52.92	31.64	29.81	78.89	75.10	56.90	52.15	54.00	51.76	43.80	40.46	77.47	75.00
	EEDP	65.00	51.10	72.79	70.17	60.30	57.13	77.60	73.00	60.10	59.54	71.80	58.28	54.90	50.31	55.48	52.40
	Binder	51.89	49.69	67.24	59.59	22.49	19.84	59.03	56.30	61.50	58.47	66.50	64.30	21.50	17.23	58.55	55.03
	TabSQLify	53.02	49.50	34.78	27.77	26.78	24.44	69.92	67.90	73.90	68.98	46.99	41.52	26.73	23.87	53.00	40.41
	TableReasoner	54.32	50.05	64.71	57.58	42.51	39.15	61.68	59.54	69.79	64.64	62.70	59.45	66.22	63.37	56.81	52.78
Agentic	Chain-of-Table	74.30	73.20	69.56	67.45	22.27	20.93	70.12	65.80	62.11	57.53	60.21	53.47	23.20	19.14	65.60	53.80
	POT	60.21	58.39	67.70	65.69	63.52	54.58	69.12	65.40	69.90	68.16	57.60	50.08	44.50	41.84	43.50	38.23
	GraphOTTER	49.29	42.75	61.79	56.09	42.76	43.98	81.05	79.71	57.50	52.50	55.88	48.47	44.26	42.26	78.60	74.61
	QUIETT	79.80	78.56	80.10	76.19	72.32	66.69	84.41	83.10	76.21	73.23	72.75	68.50	73.00	64.58	79.50	78.70
		DeepSeek V3.1								Qwen 3							
		WikiTQ		NQ-Table		SequentialQA		HiTab		WikiTQ		NQ-Table		SequentialQA		HiTab	
		F1	EM	F1	EM	F1	EM	F1	EM	F1	EM	F1	EM	F1	EM	F1	EM
Direct	CoT	60.92	59.62	63.60	59.12	59.50	50.36	71.13	68.00	58.95	57.55	61.34	59.91	57.12	54.03	72.97	69.00
	CoT + SQL	51.21	47.39	34.21	29.28	56.30	55.12	65.45	61.00	51.55	48.95	34.54	33.36	59.64	58.11	56.23	51.00
	NormTab	53.19	51.33	55.40	52.34	45.57	43.24	43.35	42.27	51.12	47.72	46.80	41.92	48.68	46.69	43.30	42.78
	E5	73.00	68.20	54.55	49.45	42.75	35.11	80.50	76.80	62.30	57.30	53.16	50.23	42.69	33.97	72.21	71.20
	EEDP	62.20	52.40	72.29	69.42	60.25	52.79	74.18	69.60	62.40	53.70	62.21	61.31	55.90	53.18	71.27	67.70
	Binder	50.62	47.22	67.44	61.20	20.19	16.63	58.66	55.90	61.68	58.48	61.56	59.38	20.54	20.34	56.29	53.50
	TabSQLify	73.00	71.45	56.94	48.03	30.01	28.58	67.62	63.40	48.40	45.45	51.34	47.67	29.18	27.27	62.30	60.40
	TableReasoner	47.63	41.38	33.33	32.36	56.89	50.36	48.63	46.42	62.49	61.53	63.77	60.11	60.59	58.99	52.95	49.18
Agentic	Chain-of-Table	63.67	61.29	70.10	61.23	44.53	41.66	75.55	70.80	61.39	59.17	64.33	61.20	44.28	38.88	76.18	71.80
	POT	48.39	45.72	65.33	58.68	59.79	56.47	56.55	53.30	46.56	44.33	62.10	59.68	59.28	53.12	32.30	30.70
	GraphOTTER	34.80	30.30	35.29	32.63	44.05	39.73	61.30	58.82	50.84	46.94	55.80	51.41	42.93	40.53	77.05	73.63
	QUIETT	74.56	72.21	79.20	72.29	66.15	59.57	81.30	79.40	64.20	60.96	72.60	65.23	62.54	58.39	81.22	81.00

Table 3: **Main results.** QUIETT is compared against direct prompting methods and agentic methods across four benchmarks and four model families. Top half: Gemini 2.5 and GPT-OSS; bottom half: DeepSeek V3.1 and Qwen 3. Bold indicates best result per column. Gemini 2.0 results are in Table 12.

5 Results and Analysis

We structure our analysis around four questions: (1) Does QUIETT improve consistently across model families and scales? (2) Do gains generalize across diverse table structures and reasoning settings? (3) Is transformation quality a stronger driver of performance or QA model capacity? (4) How does QUIETT perform on structurally challenging unseen queries?

Does QUIETT generalize across model families?

QUIETT consistently improves over the strongest baseline for each model across both F1 and accuracy, with F1 gains of approximately 4-8 points and corresponding accuracy gains of 3-6 points across datasets. These improvements hold for both proprietary models (Gemini 2.0, Gemini 2.5) and open-weight models (DeepSeek-V3.1, Qwen3-80B, GPT-OSS-120B), indicating that the benefits of improved table representation generalize across model families and scales.

Does QUIETT generalize across table structures? QUIETT generalizes across diverse table types and reasoning settings. On WikiTQ and NQ-Table, which contain semi-structured web tables, QUIETT reaches F1 of 79.80 and 80.10 with accuracy of 78.56 and 76.19 respectively with Gemini 2.5. On SequentialQA, where questions are inter-dependent across turns, QUIETT achieves up to 72.32 F1 and 66.69 accuracy, showing that pre-

processing benefits extend beyond single-turn answering. On HiTab, which emphasizes hierarchical numerical reasoning, QUIETT achieves peak F1 of 84.41 and accuracy of 83.10, the highest across all methods and models. Together, these results suggest that table-intrinsic structural issues are a consistent bottleneck across table types, and that resolving them before reasoning is broadly beneficial.

Does transformation quality matter more or model capacity? Table 15 shows that transformation quality is a stronger driver of downstream performance than QA model capacity. On WikiTQ, tables transformed with Gemini 2.5 achieve 65.12 F1 and 62.23 accuracy with Mistral, compared to 63.33 F1 and 59.35 accuracy when Qwen 3 performs the transformation with the same QA model. Similar trends hold on NQ-Table (63.28 & 58.12 vs. 62.56 & 60.23) and SequentialQA (47.24 & 44.84 vs. 42.81 & 41.12). The pipeline is also not critically dependent on large models for issue detection, replacing the issue probing model with LLaMA-14B Scout (Table 13) achieves 68.34 F1 and 68.14 accuracy on WikiTQ and 70.11 F1 and 69.34 accuracy on NQ-Table with Gemini 2.5, within 6-10 scores across both metrics, suggesting that structural deficiencies are detectable even by smaller models.

How are gains on structurally challenging unseen queries? Table 4 reports performance on the

challenge set described in Section 3. QUIETT achieves 74.41 F1 and 71.37 accuracy on WikiTQ and 77.16 F1 and 76.50 accuracy on NQ-Table with Gemini 2.5, outperforming all baselines by 5-8 F1 and 4-7 accuracy points. These gains are larger than those on the standard benchmarks, consistent with the challenge set targeting structurally diverse cases where table-intrinsic representation issues are most acute. The rationale for selecting WikiTQ and NQ-Table for the challenge set is discussed in Section 3.

Dataset	Method	Gemini 2.5		GPT-OSS	
		F1	EM	F1	EM
WikiTQ	CoT	67.11	65.50	63.58	60.36
	NormTab	62.28	61.90	61.71	57.23
	EEDP	67.60	66.34	65.12	61.86
	Binder	65.56	63.39	65.12	64.48
	QUIETT	74.41	71.37	70.06	68.59
NQ-Table	CoT	65.63	63.35	61.38	58.49
	NormTab	66.84	66.12	65.68	64.23
	EEDP	71.94	68.50	67.18	63.87
	Binder	72.28	67.22	68.33	67.30
	QUIETT	77.16	76.50	71.19	70.66

Table 4: Challenge set results

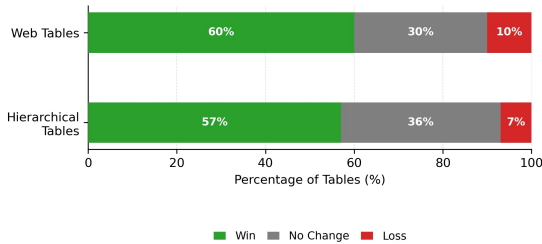


Figure 3: **Table-level performance outcomes.** Win, No Change, and Loss indicate whether QUIETT improves, leaves unchanged, or degrades per-table QA performance relative to the raw table baseline.

How often does QUIETT improve vs. degrade per table? Figure 3 shows per-table outcomes after applying QUIETT. On web tables, 60% improve, 31% show no change, and 9% degrade. On HiTab, 57% improve, 36% show no change, and 7% degrade. The higher proportion of unchanged cases on HiTab reflects that hierarchical tables often already encode relational structure explicitly. Provenance columns are introduced in only 13% of web tables and 9% of HiTab tables, confirming that most canonical representations do not rely on auxiliary columns for information completeness.

When does transformation hurt? To characterize the 7-9% of tables where performance degrades (Figure 3), we analyze all degraded cases across both web and hierarchical tables (Figure 4). The

dominant failure mode (62.6%) is ambiguous multi-value cells or packed cells that are difficult to split cleanly, causing answer spans to become partial or harder to recover. The remaining failures arise from surface-form changes like type casting (18.0%) alters matching behavior, and value replacement (17.5%) occasionally normalizes a value that was useful for answer extraction. Over-normalization, downstream QA errors on valid tables, and unnecessary transformation of already-clean tables together account for under 2% of degraded cases. These failure modes are localized and interpretable, reflecting inherent ambiguities in specific tables rather than a systematic weakness of the transform-first approach. The 7-9% degradation also provides a natural signal for identifying tables that may benefit from complementary query-conditioned methods.

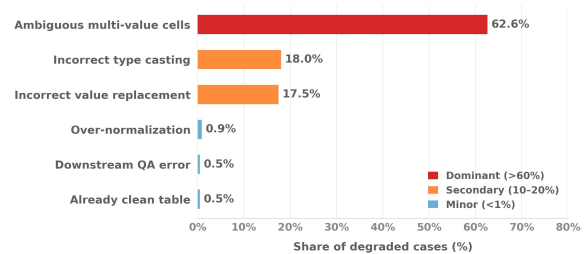


Figure 4: **Error Analysis for degraded tables.**

5.1 Ablation studies.

We conduct three targeted ablations to isolate the contribution of individual pipeline components.

Removing issue probing. Skipping the issue probing stage entirely (Table 5) causes the largest performance drops: from 79.80 to 53.22 F1 and 78.56 to 52.06 accuracy on WikiTQ, and from 80.10 to 56.80 F1 and 76.19 to 54.37 accuracy on NQ-Table with Gemini 2.5, decreases exceeding 25 F1 and accuracy points. Similar drops are observed with GPT-OSS, falling to 51.40 F1 and 51.15 accuracy on WikiTQ and 54.90 F1 and 51.82 accuracy on NQ-Table. Without issue detection, the transformation plan has no structured basis for deciding what to normalize or derive, and the pipeline reduces to undirected table restructuring.

Probe coverage. To explain the large drop when issue probing is removed, we manually examined whether the probes surfaced transformations relevant to answering the evaluation questions. They identified relevant structural transformations for 79% of the analyzed questions. The remaining 21% were simple lookup or pattern-matching ques-

Dataset	Gemini 2.5		GPT-OSS	
	F1	EM	F1	EM
WikiTQ	53.22	52.06	51.40	51.15
NQ-Table	56.80	54.37	54.90	51.82

Table 5: QUIETT performance without issue probing.

Dataset	Gemini 2.5		GPT-OSS	
	F1	EM	F1	EM
WikiTQ	60.57	58.26	58.33	57.29
NQ-Table	63.20	60.56	61.08	58.66

Table 6: QUIETT performance when planning and execution are merged into a single-agent transformation pipeline.

tions that required no structural transformation, indicating that the uncovered cases were not probing failures but questions already answerable from the raw table.

Merging planning and execution. Collapsing plan generation and code execution into a single-agent pipeline (Table 6) degrades performance consistently across both models and metrics. On WikiTQ, performance drops from 79.80 to 60.57 F1 and 78.56 to 58.26 accuracy with Gemini 2.5, and from 76.21 to 58.33 F1 and 73.23 to 57.29 accuracy with GPT-OSS. NQ-Table follows the same pattern, falling from 80.10 to 63.20 F1 and 76.19 to 60.56 accuracy with Gemini 2.5, and from 72.75 to 61.08 F1 and 68.50 to 58.66 accuracy with GPT-OSS. This confirms that separating symbolic planning from code-level execution provides meaningful structure. The plan constrains *what* is transformed, and the executor enforces *how*.

Performance under constrained inputs. Table 7 reports performance when QA models receive only the schema, data types, and two representative rows ($n = 2$) of the transformed table, without access to the full table, raw tables, or execution traces. Performance decreases relative to full-context settings, as expected, but QUIETT retains a substantial fraction of its gains across all datasets and models, with HiTab showing the strongest retention (62.30 F1 and 59.60 accuracy with Gemini 2.5) while WikiTQ and NQ-Table drop more sharply, suggesting that web tables benefit more from full table access. Nevertheless, the canonical representation remains more useful than raw tables under the same constraints, indicating that structural normalization provides value beyond simply exposing more content.

Dataset	Gemini 2.0		Gemini 2.5		DeepSeek V3.1		Qwen 3		GPT-OSS	
	F1	EM	F1	EM	F1	EM	F1	EM	F1	EM
WikiTQ	37.61	36.34	38.92	35.12	27.65	26.78	27.19	23.50	28.21	24.59
NQ-Table	35.42	33.32	32.49	32.03	25.38	24.23	35.31	31.21	32.44	30.46
SequentialQA	40.71	38.90	42.97	36.32	38.98	35.33	36.67	31.12	41.98	38.48
HiTab	59.60	56.80	62.30	59.60	58.12	58.30	46.27	45.30	53.60	52.50

Table 7: QUIETT with limited table representations ($n=2$).

6 Related Work

Table QA methods differ primarily in *when* and *how* they address structural issues in raw tables. We organize prior work along three axes: lightweight normalization, programmatic restructuring, and transformation coupled with reasoning, and clarify where QUIETT sits relative to each.

Lightweight normalization for table QA. Some methods apply one-time normalization to reduce surface-form variability while preserving the original structure. NormTab (Nahid and Rafiei, 2024a) standardizes values such as dates and numbers and applies limited structural fixes, but leaves hierarchical structure, implicit relations, and heterogeneous layouts to downstream reasoning. Related approaches similarly assume mostly fixed schemas and rely on query-time reasoning to resolve remaining ambiguity (Pasupat and Liang, 2015; Nahid and Rafiei, 2024b). Unlike these methods, QUIETT resolves structural issues proactively, before any query is observed, and enforces a single reusable canonical representation.

Programmatic table restructuring. A second line of work performs restructuring via explicit transformation programs. Auto-Tables (Li et al., 2023) synthesizes multi-step pipelines from a fixed operator vocabulary but is limited to predefined transformations and known target schemas. DataMorpher (Sharma et al., 2025) and RelationalCoder (Dong et al., 2025b) generate Python or SQL programs for ETL or schema matching, but remain coupled to user-specified target schemas and do not enforce reusable, query-independent representations. QUIETT differs in that no target schema is predefined, and transformation is evaluated by downstream QA improvement across arbitrary queries and models.

Transformation coupled with reasoning. Other approaches integrate transformation directly into reasoning. TabFormer (Dong et al., 2025a) jointly applies restructuring and SQL answering within a single prompted process, making transformations dependent on the query set. Chain-of-Table (Wang

et al., 2024) and TableReasoner (Xiong et al., 2025a) adopt query-driven schema selection or iterative reasoning, improving efficiency but coupling table structure to query-time decisions. In contrast, QUIETT treats transformation as an evaluation-query-independent preprocessing step that produces a single canonical representation reused across queries, tasks, and models, cleanly separating representation quality from downstream reasoning. A detailed discussion of the limitations of query-conditioned transformation is provided in Appendix E.

7 Conclusion

QUIETT converts raw, semi-structured tables into a single SQL-ready canonical representation before any evaluation query is observed, decoupling table pre-processing from downstream reasoning. Experiments across four benchmarks and five model families show consistent gains in F1 and accuracy under identical prompting conditions, including with smaller models and constrained inputs, indicating that structural normalization of the table itself, independent of the query, is a reliable and underexplored source of improvement in table QA. These results suggest that the field has focused heavily on query-adaptive reasoning while leaving systematic table preparation largely unaddressed. A single reusable canonical representation, constructed without evaluation query access, can serve as a stable foundation for diverse reasoning strategies, models, and tasks. Future work should extend this direction to large tables, multimodal formats, and multi-table reasoning settings.

Limitations

QUIETT has five limitations. First, the pipeline targets tables processable within current context windows; very large tables remain challenging due to input length constraints and the difficulty of producing compact canonical representations. Second, QUIETT operates on structured tabular data only and does not handle tables interleaving free-form text or images. Third, while the canonical representation enables effective QA with smaller downstream models, the transformation stage itself requires sufficiently capable models for issue probing and plan generation. Fourth, a single fixed transformation cannot anticipate every future query; cases requiring latent relations not surfaced during probing may still benefit from query-conditioned methods such as Chain-of-Table (Wang et al., 2024), and

QUIETT works best alongside rather than instead of such approaches. Finally, ambiguous packed cells remain the dominant failure mode (62.6% of degraded cases, Figure 4); more principled alternatives such as child-table relationalization (Li et al., 2023; Dong et al., 2025b) would handle these more robustly but require relaxing the single-table reuse constraint 2.1 (c).

Ethics Statement

This work studies table transformation and reasoning over publicly available benchmark datasets, including WikiTQ, HiTab, NQ-Table, and SequentialQA. All datasets consist of tabular data derived from publicly accessible sources (e.g., Wikipedia) and do not contain personally identifiable information beyond what is already publicly disclosed. The proposed framework, QUIETT, operates on tabular structure and content and does not introduce unsupported values. Transformed values remain grounded in source-table content through normalization and restructuring. The challenge set introduced in this work is automatically generated from existing tables and manually verified by expert annotators to ensure correctness and faithfulness; no human subjects are involved beyond standard dataset verification, and no personal or sensitive attributes are inferred or labeled. We do not anticipate any harmful or unethical use of the proposed method. However, as with all large language model-based systems, care should be taken when applying QUIETT to proprietary, sensitive, or private data, and appropriate data governance and privacy safeguards should be followed. AI tools were used for manuscript and code editing.

Acknowledgments

We gratefully acknowledge the Complex Data Analysis and Reasoning Lab at Arizona State University for providing computational resources and institutional support. We also thank the reviewers for their valuable feedback.

References

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

- Zhoujun Cheng, Tianbao Xie, Peng Shi, Chengzu Li, Rahul Nadkarni, Yushi Hu, Caiming Xiong, Dragomir Radev, Mari Ostendorf, Luke Zettlemoyer, Noah A. Smith, and Tao Yu. 2023. Binding language models in symbolic languages. *arXiv preprint arXiv:2210.02875*. Version a.
- Manan Roy Choudhury, Adithya Chandramouli, Manan Anand, and Vivek Gupta. 2026. [Better call clause: A discrepancy benchmark for auditing llms legal reasoning capabilities](#).
- Manan Roy Choudhury, Anirudh Iyengar Kaniyar Narayana Iyengar, Shikhar Singh, Sugeeth Puranam, and Vivek Gupta. 2025. [TABARD: A novel benchmark for tabular anomaly analysis, reasoning and detection](#). In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 21783–21817, Suzhou, China. Association for Computational Linguistics.
- Xu Chu, Ihab F. Ilyas, Sanjay Krishnan, and Jiannan Wang. 2016. [Data cleaning: Overview and emerging challenges](#). In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD '16*, page 2201–2206, New York, NY, USA. Association for Computing Machinery.
- DeepSeek-AI. 2024. [Deepseek-v3 technical report](#). Technical report, DeepSeek.
- Haoyu Dong, Yue Hu, and Yanan Cao. 2025a. Reasoning and retrieval for complex semi-structured tables via reinforced relational data transformation. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1382–1391, Padua, Italy. ACM.
- Haoyu Dong, Yue Hu, Huailiang Peng, and Yanan Cao. 2025b. [RelationalCoder: Rethinking complex tables via programmatic relational transformation](#). In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1771–1784, Vienna, Austria. Association for Computational Linguistics.
- Google. 2023. [Gemini: A family of highly capable multimodal models](#). Technical report, Google DeepMind.
- Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Eisenschlos. 2020. [TaPas: Weakly supervised table parsing via pre-training](#). In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4320–4333, Online. Association for Computational Linguistics.
- Mohit Iyyer, Wen-tau Yih, and Ming-Wei Chang. 2017. [Search-based neural structured learning for sequential question answering](#). In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1821–1831, Vancouver, Canada. Association for Computational Linguistics.
- Rohit Khoja, Devanshu Gupta, Yanjie Fu, Dan Roth, and Vivek Gupta. 2025. [Weaver: Interweaving SQL and LLM for table reasoning](#). In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 28282–28308, Suzhou, China. Association for Computational Linguistics.
- Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. 2019. [Natural questions: A benchmark for question answering research](#). *Transactions of the Association for Computational Linguistics*, 7:452–466.
- Peng Li, Yeye He, Cong Yan, Yue Wang, and Surajit Chaudhuri. 2023. [Auto-tables: Synthesizing multi-step transformations to relationalize tables without using examples](#). *Proc. VLDB Endow.*, 16(11):3391–3403.
- Qianlong Li, Chen Huang, Shuai Li, Yuanxin Xiang, Deng Xiong, and Wenqiang Lei. 2024. [Graphotter: Evolving llm-based graph reasoning for complex table question answering](#).
- Md Mahadi Hasan Nahid and Davood Rafiei. 2024a. [NormTab: Improving symbolic reasoning in LLMs through tabular data normalization](#). In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 3569–3585, Miami, Florida, USA. Association for Computational Linguistics.
- Md Mahadi Hasan Nahid and Davood Rafiei. 2024b. [TabSQLify: Enhancing reasoning capabilities of LLMs through table decomposition](#). In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 5725–5737, Mexico City, Mexico. Association for Computational Linguistics.
- Panupong Pasupat and Percy Liang. 2015. [Compositional semantic parsing on semi-structured tables](#). In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1470–1480, Beijing, China. Association for Computational Linguistics.
- Ankita Sharma, Jaykumar Tandel, Xuanmao Li, Lanjun Wang, Anna Fariha, Liang Zhang, Syed Arsalan Ahmed Naqvi, Irbaz Bin Riaz, Lei Cao, and Jia Zou. 2025. [Datamorpher: Automatic data transformation using llm-based zero-shot code generation](#). In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*, pages 4536–4539.
- Pragya Srivastava, Manuj Malik, Vivek Gupta, Tanuja Ganu, and Dan Roth. 2024. [Evaluating LLMs’ mathematical reasoning in financial document question answering](#). In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 3853–3878,

- Bangkok, Thailand. Association for Computational Linguistics.
- Shi-Yu Tian, Zhi Zhou, Wei Dong, Kun-Yang Yu, Ming Yang, Zi-Jian Cheng, Lan-Zhe Guo, and Yu-Feng Li. 2026. [TabularMath: Understanding math reasoning over tables with large language models](#). In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 15469–15498, San Diego, California, United States. Association for Computational Linguistics.
- Zilong Wang, Hao Zhang, Chun-Liang Li, Julian Martin Eisenschlos, Vincent Perot, Zifeng Wang, Lesly Miculicich, Yasuhisa Fujii, Jingbo Shang, Chen-Yu Lee, and Tomas Pfister. 2024. [Chain-of-table: Evolving tables in the reasoning chain for table understanding](#). In *The Twelfth International Conference on Learning Representations*.
- Tianbao Xie, Chen Henry Wu, Peng Shi, Ruiqi Zhong, Torsten Scholak, Michihiro Yasunaga, Chien-Sheng Wu, Ming Zhong, Pengcheng Yin, Sida I. Wang, Victor Zhong, Bailin Wang, Chengzu Li, Connor Boyle, Ansong Ni, Ziyu Yao, Dragomir Radev, Caiming Xiong, Lingpeng Kong, Rui Zhang, Noah A. Smith, Luke Zettlemoyer, and Tao Yu. 2022. [UnifiedSKG: Unifying and multi-tasking structured knowledge grounding with text-to-text language models](#). In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 602–631, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.
- Sishi Xiong, Mengxiang Li, Dakai Wang, Yu Zhao, Jie Zhang, Changzai Pan, Haowei He, Xiangyu Li, Wenhan Chang, Zhongjiang He, Shuangyong Song, and Yongxiang Li. 2025a. [TeleAI at SemEval-2025 task 8: Advancing table reasoning framework with large language models](#). In *Proceedings of the 19th International Workshop on Semantic Evaluation (SemEval-2025)*, pages 1828–1841, Vienna, Austria. Association for Computational Linguistics.
- Sishi Xiong, Dakai Wang, Yu Zhao, Jie Zhang, Changzai Pan, Haowei He, Xiangyu Li, Wenhan Chang, Zhongjiang He, Shuangyong Song, and Yongxiang Li. 2025b. [Tablereasoner: Advancing table reasoning framework with large language models](#).
- Yunhu Ye, Binyuan Hui, Min Yang, Binhua Li, Fei Huang, and Yongbin Li. 2023. [Large language models are versatile decomposers: Decomposing evidence and questions for table-based reasoning](#). In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '23*, page 174–184, New York, NY, USA. Association for Computing Machinery.
- Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. [TaBERT: Pretraining for joint understanding of textual and tabular data](#). In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8413–8426, Online. Association for Computational Linguistics.
- Zhehao Zhang, Yan Gao, and Jian-Guang Lou. 2024. [e⁵: Zero-shot hierarchical table analysis using augmented LLMs via explain, extract, execute, exhibit and extrapolate](#). In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 1244–1258, Mexico City, Mexico. Association for Computational Linguistics.
- Bowen Zhao, Changkai Ji, Yuejie Zhang, Wen He, Yingwen Wang, Qing Wang, Rui Feng, and Xiaobo Zhang. 2023. [Large language models are complex table parsers](#). In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 14786–14802, Singapore. Association for Computational Linguistics.
- Fengbin Zhu, Wenqiang Lei, Chao Wang, Jianming Zheng, Soujanya Poria, and Tat-Seng Chua. 2021. [Retrieving and reading: A comprehensive survey on open-domain question answering](#).

Appendix

A Transformation Operators

QUIETT represents transformation plans using a fixed vocabulary of atomic operators for normalization, restructuring, and enrichment. Registered operators have deterministic, schema-aware, and composable implementations. The vocabulary also contains a custom operator that routes table-specific transformations to LLM-generated code under the same declared input, output, and validation constraints. The planner cannot introduce new operator names; every plan step must match an entry in the operator registry.

Table 8 summarizes the operator vocabulary used in our implementation.

B Resource Usage

QUIETT incurs language model cost exclusively during a one-time, *query-independent* preprocessing stage that is executed once per table. For each table, the pipeline invokes a language model exactly three times: (i) issue generation, (ii) transformation plan generation, and (iii) executor code generation. Registered operator execution is deterministic and does not require any language model calls. LLM-generated code steps are bounded by the plan structure.

B.1 Models and Context Limits

We evaluate the preprocessing pipeline using multiple large language models, including DeepSeek-V3.1, Qwen-3-80B, Gemini-2.0, Gemini-2.5, and GPT-OSS-120B. These models support long-context inference, which is necessary to process table schemas together with representative cell samples within a single prompt.

Table 9 summarizes the documented context window limits for each model. Effective throughput and request rates are governed by provider- and project-level quota policies, which may vary across deployments, regions, and time.

B.2 Token Usage Accounting

For reproducibility, we log token usage at the granularity of individual tables and preprocessing stages (issue generation, plan generation, and code generation). For each language model invocation, we record the number of input tokens, output tokens, and total tokens consumed.

Total preprocessing token usage over a corpus

of tables is computed as:

$$\text{Tokens} = \sum_{i=1}^T \sum_{s \in \{\text{issue, plan, code}\}} (\text{in}_{i,s} + \text{out}_{i,s}),$$

where T denotes the total number of tables.

B.3 Amortization Across Queries

Because the pipeline preprocesses each table once and caches the result, the marginal preprocessing token cost per downstream query is zero. All downstream question answering is performed on the canonical transformed table using an unchanged QA model, without incurring any additional preprocessing-related language model cost.

C Hyperparameters

This section reports the inference and execution hyperparameters used in the QUIETT pipeline. Unless otherwise noted, all LLM calls use deterministic decoding to improve reproducibility.

C.1 Language Model Inference Settings

Table 10 summarizes the decoding and retry configuration used across pipeline stages. We use fixed decoding (temperature = 0.2) for all stages. Maximum output tokens are stage-specific and follow the values used in our implementation.

C.2 Input Construction and Truncation

For prompt construction, we serialize each table into markdown that includes the header and up to 50 rows (uniformly truncated to the first 50 rows when the table is longer). We do not apply explicit cell-level character truncation; long cell values are passed through as-is.

C.3 Executor and Determinism Settings

A deterministic table executor runs each transformation plan produced in Step 2. Determinism is enforced by fixed operator semantics, stable row referencing (via row identifiers when required), and deterministic implementations for any operation that may reorder rows (*e.g.*, sorting) or drop duplicates (via a fixed deduplication policy when enabled). The executor introduces no stochasticity.

C.4 LLM-Generated Code: Safety, Sandboxing, and Determinism

LLM-generated code steps in Stage 2 operate over the in-memory pandas DataFrame produced by the preceding deterministic step. The execution scope exposes only pandas, numpy, re, and the operator runtime library, with no direct filesystem or

Operator	Description
add_row_id	Adds a unique, stable row identifier column to the table.
rename	Renames columns according to a provided name mapping.
select	Selects and retains a specified subset of columns.
parse_date_text	Parses free-text date strings into a normalized date representation.
parse_number	Extracts numeric values (and optional units) from text fields.
extract_regex	Extracts structured groups from a column using a regular expression.
derive_conditional	Creates a new column based on conditional rules applied row-wise.
derive_math	Derives a new column by applying a mathematical expression over existing columns.
map_values	Maps raw values in a column to normalized or canonical values.
replace_value	Replaces exact cell values with specified alternatives.
replace_string	Performs string replacement within column values (optionally regex-based).
cast_column	Casts a column to a specified data type.
fillna_static	Fills missing values using a fixed constant.
fillna_dynamic	Fills missing values using a dynamic rule (<i>e.g.</i> , forward fill).
combine_columns	Combines multiple columns into a single column using a separator.
trim_whitespace	Removes leading and trailing whitespace from column values.
filter_rows	Filters rows based on a boolean condition.
sort	Sorts rows according to one or more columns.
deduplicate_rows	Removes duplicate rows, optionally using a column subset.
keep_raw_snapshot	Preserves original raw values in a snapshot column.
bin_numeric	Discretizes numeric values into bins with optional labels.
one_hot	Expands a categorical column into one-hot encoded indicator columns.
split_multi_value	Splits multi-valued cells and optionally expands the resulting values into separate rows.
pivot_longer	Reshapes selected columns into key-value rows.
custom	Applies a user-defined transformation not covered by built-in operators.

Table 8: Atomic table transformation operators used in the transformation plan.

Model	Context window
DeepSeek-V3.1	Up to 128K tokens
Qwen-3-80B	Up to 128K tokens
GPT-OSS-120B	Up to 131K tokens
Gemini-2.0	Up to $\sim 1,048,576$ tokens
Gemini-2.5	Up to $\sim 1,048,576$ tokens

Table 9: Documented context window limits for models used during QUIETT preprocessing. Actual rate limits depend on provider- and project-specific quota configurations.

network access from generated code. The executor restricts each generated snippet to the declared input and output columns specified in the plan, and validates that: (i) all declared writes columns are present in the output, (ii) no all-NaN columns are introduced, and (iii) row count is preserved unless the operation legitimately modifies table length (*e.g.*, `split_multi_value`, `filter_rows`, or `pivot_longer`). Snippets that fail any check are rejected and the model is re-queried with the error appended to the prompt, up to two additional attempts.

The custom operator serves as an explicit fallback for transformations that cannot be expressed

Hyperparameter	Value
Decoding	temperature = 0.2
Max output tokens (Step 1: Issue)	8000
Max output tokens (Step 2: Plan)	6000
Max output tokens (Step 3: Code)	1024
Max output tokens (Step 4: Q&A)	4096
Stop sequences:	none (generation is post-processed/validated)
Retries on rate limit/errors	3 attempts
Backoff schedule	exponential backoff, initial delay = 2.0s, doubling each retry

Table 10: Inference hyperparameters used in the QUIETT pipeline. Step 1-3 are query-independent preprocessing; Step 4 performs downstream QA on the canonical table.

within the fixed operator vocabulary. When a plan step declares `op: custom`, the executor unconditionally routes to LLM code generation rather than attempting a registered implementation. All safety and validation checks described above apply equally to custom steps: the generated snippet is restricted to the declared input and output columns, validated against the same three criteria, and re-queried on failure.

D Human Verification Guidelines and Annotation Rubric

To assess the correctness, faithfulness, and clarity of the automatically generated QUIETT challenge set, we conduct a structured human verification study. This section documents the annotation setup, verification objectives, and rubric provided to annotators.

D.1 Annotation Setup

Each annotator is presented with the following materials: (i) the *original raw table* drawn from the source dataset, (ii) a generated natural-language question, and (iii) the corresponding proposed ground-truth answer. Annotators perform verification independently and without access to model predictions or outputs. All judgments are based solely on the provided table content.

D.2 Verification Objectives

Annotators are instructed to evaluate each question-answer pair according to the following criteria:

1. **Well-formedness:** The question is clearly phrased and free of ambiguity.
2. **Answerability:** The question can be answered using only the table content.
3. **Answer correctness:** The provided answer is fully supported by the table.
4. **Structural reasoning:** The question requires non-trivial reasoning, such as aggregation, comparison, filtering, or reasoning over normalized or derived fields, rather than simple cell lookup.

Annotators are explicitly instructed not to assess model difficulty, linguistic style beyond clarity, or subjective preferences.

D.3 Annotation Guidelines

The following guidelines are provided to annotators to ensure consistent judgments:

- **Table faithfulness:** All answers must be derivable exclusively from the table content, including normalized or derived columns introduced by QUIETT. External knowledge may not be used.
- **Table representation awareness:** Annotators should treat the raw table as authoritative. Questions relying on derived or normalized

Criterion	Description
Answer Correctness	The provided answer exactly matches the value(s) supported by the table.
Answerability	The question can be answered using only the table content.
Clarity	The question is unambiguous and clearly specifies the required reasoning.
Structural Reasoning	The question requires aggregation, comparison, filtering, or reasoning over derived or normalized fields.

Table 11: Annotation rubric used for human verification of the QUIETT challenge set.

fields (e.g., parsed dates or normalized units) are considered valid.

- **Ambiguity handling:** If a question admits multiple reasonable interpretations or more than one valid answer, it must be marked incorrect.
- **Incomplete answers:** Answers that partially satisfy the question (e.g., missing entities in list-valued outputs or incorrect aggregation results) are marked incorrect.
- **Assumption prohibition:** Questions requiring assumptions not grounded in the table (e.g., inferred semantics, unstated ordering, or missing values) are marked incorrect.

D.4 Annotation Rubric

Each question-answer pair is evaluated using the rubric shown in Table 11. A pair is marked *valid* if and only if all criteria are satisfied.

D.5 Disagreement Analysis

All annotations are performed independently without discussion or adjudication during the annotation phase. Inter-annotator agreement is reported using pairwise Cohen’s Kappa and multi-rater agreement metrics (Fleiss’ Kappa and Jaccard coefficient), as described in Section 4.

D.6 Annotator Qualifications

All annotators are NLP researchers with prior experience in table-based question answering, semantic parsing, or structured data reasoning. We briefed all annotators using written instructions and example annotations before the study.

D.7 Rationale

This verification protocol ensures that the QUIETT challenge set evaluates faithful, well-defined, and structurally grounded reasoning over tables, rather than exploiting annotation artifacts or superficial

Category	Method	Gemini 2.0							
		WikiTQ		NQ-Table		SequentialQA		HiTab	
		F1	EM	F1	EM	F1	EM	F1	EM
Direct	CoT	61.76	60.67	68.70	67.50	62.33	59.87	76.11	72.00
	CoT + SQL	44.43	44.02	43.29	41.45	50.06	46.83	52.10	50.00
	NormTab	55.95	52.42	51.80	49.76	47.95	42.14	56.24	53.60
	E5	66.60	65.90	48.40	46.34	42.75	38.18	75.83	71.40
	EEDP	61.80	59.30	70.31	63.81	61.10	54.27	76.53	72.00
	Binder	60.81	57.88	62.25	54.90	21.82	20.91	55.62	53.00
	TabSQLify	64.50	61.91	47.51	38.10	10.71	9.74	64.20	60.10
	TableReasoner	60.19	57.89	56.52	53.45	59.52	51.23	60.84	54.31
Agentic	Chain-of-Table	63.27	61.58	62.30	59.28	41.24	39.79	69.43	65.10
	POT	52.11	50.32	61.70	52.07	56.10	52.43	64.00	61.00
	GraphOTTER	55.39	51.16	69.29	65.69	50.36	49.35	64.64	61.62
	QUIETT	67.60	66.56	73.25	68.75	70.11	65.69	77.82	76.50

Table 12: **Gemini 2.0 results (F1 / EM)**. QUIETT compared against direct prompting methods (CoT, CoT+SQL, NormTab, E5, EEDP, Binder, TabSQLify, TableReasoner) and agentic methods (Chain-of-Table, POT, GraphOTTER) across four benchmarks. Bold indicates best result per column.

Dataset	Gemini 2.5		GPT-OSS	
	F1	EM	F1	EM
WikiTQ	68.34	68.14	66.72	63.57
NQ-Table	70.11	69.34	68.20	65.56

Table 13: QUIETT performance when a smaller model (LLaMA-14B Scout) is used for issue generation.

patterns. The high inter-annotator agreement reported in Section 4 indicates strong consistency in judgments and supports the reliability of the challenge set.

E Limitations of Query-Conditioned Transformation

Several methods perform table restructuring conditioned on the input query or query set, where schema selection, decomposition, or transformation is guided by the reasoning objective (e.g., Chain-of-Table (Wang et al., 2024), TableReasoner (Xiong et al., 2025a), TabFormer (Dong et al., 2025a), Dater (Ye et al., 2023), Binder (Cheng et al., 2023)). While effective for improving efficiency and accuracy on observed queries, this design introduces a fundamental coupling between table representation and query distribution.

First, query-conditioned transformation limits generalization to unseen queries. A representation tailored to a specific query or narrow query set may omit information irrelevant to those queries, reducing robustness when new questions arrive, as observed in prior table QA and semantic parsing work (Yin et al., 2020; Herzig et al., 2020).

Second, query-conditioned methods assume query access at transformation time, which does not hold in common data processing settings where tables are preprocessed once and reused across mul-

tiple tasks and users (Chu et al., 2016).

Third, coupling transformation with reasoning introduces representation instability: the same table yields different transformed views depending on the query, making it harder to characterize when and why a method fails, since representation quality and reasoning behavior are entangled by design (Xie et al., 2022; Pasupat and Liang, 2015).

QUIETT addresses these limitations by converting each table once into a single, information-preserving canonical representation before any downstream queries arrive, cleanly separating representation quality from reasoning behavior.

F Probe Budget Analysis

QUIETT generates a finite set of synthetic probe queries during issue generation to expose structural deficiencies in the input table. In our implementation we generate 12-20 probe queries per table. This range is treated as a probe budget chosen to balance structural coverage and redundancy.

To empirically validate this design choice, we analyze the number of issues discovered during probing across two benchmark datasets (WikiTQ and NQ-Table) using two backbone models (Gemini 2.5 and GPT-OSS-120B). Across 3,491 table-model pairs, issue discovery naturally concentrates within this range.

Table 14 summarizes the distribution of detected issues per table. We observe that 98.5% of WikiTQ tables and 99.4% of NQ-Table tables produce between 11-20 issues, with mean ≈ 16.1 and median ≈ 15 .

Figure 5 visualizes the distribution of detected structural issues per table across datasets. The distribution is concentrated around ≈ 16 issues, indicating that most structural deficiencies are uncov-

Dataset	Mean Issues	Median	% in 11-20
WikiTQ	16.1	15	98.5%
NQ-Table	16.1	15	99.4%

Table 14: Issue statistics across datasets demonstrating that most tables naturally fall within the 11-20 issue range.

Transform model: Gemini 2.5				
QA model →	LLaMA		Mistral	
	F1	EM	F1	EM
WikiTQ	59.87	57.50	65.12	62.23
NQ-Table	59.01	53.78	63.28	58.12
SequentialQA	46.72	43.12	47.24	44.84
HiTab	62.30	61.20	59.61	58.40
Transform model: Qwen 3				
QA model →	LLaMA		Mistral	
	F1	EM	F1	EM
WikiTQ	53.67	50.12	63.33	59.35
NQ-Table	51.60	50.70	62.56	60.23
SequentialQA	37.70	36.50	42.81	41.12
HiTab	46.22	45.60	51.30	47.83

Table 15: F1 / EM performance across transformation model and QA model combinations. Stronger transformations consistently improve downstream QA regardless of QA model size.

ered within this probing regime.

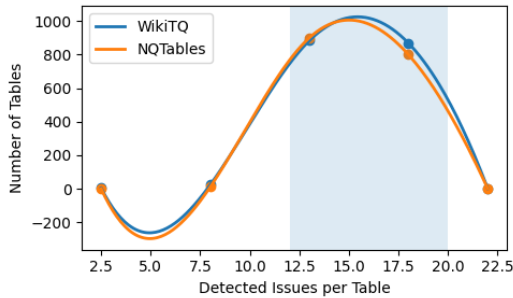


Figure 5: Distribution of detected structural issues per table across WikiTQ and NQ-Table. Nearly all tables exhibit between 11-20 issues, supporting the probe budget used by QUIETT.

We further observe that probe count correlates strongly with issue discovery ($r \approx 0.72$), suggesting that each probe typically surfaces a distinct structural deficiency. At the same time, no tables exceed 20 detected issues, indicating diminishing returns beyond this range.

Together, these observations suggest that the 12-20 probe range provides sufficient structural coverage while avoiding redundant probing.

G Grounding Analysis of Transformed Values

To assess whether QUIETT preserves source-table content during transformation, we conduct a post-

hoc cell-level grounding analysis over transformed tables. For each output value in the canonical table, we determine whether it is (a) an exact string match to a value appearing in the original table or (b) a deterministic derivation of source content. Deterministic derivations include operations such as splitting “John Doe” into first and last name columns, extracting year components from date strings, or casting string values to numeric types.

Our analysis shows that 86.0% of output values are exact string matches to source cells, while the remaining 14.0% arise from deterministic derivations grounded in the original table. We do not observe unsupported values that lack a source-table justification. These findings provide additional empirical evidence that QUIETT preserves table content while making structure more explicit for downstream reasoning.

This analysis is intended as supporting evidence for grounding and traceability. It is not an additional runtime constraint of the executor; rather, it is a post-hoc measurement of how transformed values relate to the source table.

We note that information preservation, as defined in constraint (b) of Section 2, refers to grounding and provenance rather than strict invertibility. The canonical table retains all source content in an accessible, structured form, though exact reconstruction of the original surface representation is not guaranteed for values that undergo parsing or normalization (e.g., date strings converted to ISO format).

H Illustrative Examples

This section presents representative examples highlighting common failure modes in table-based reasoning and how QUIETT transformations resolve them.

H.1 Example 1: Hierarchical Table

Title: agri-food industry sub-groups for workers aged 15 years and over, two agricultural regions of ontario, 2011

		Eastern Ontario		Northern Ontario	
percent		French Language Workers	Other Workers	French Language Workers	Other Workers
	input and service supply	2.9	2.1	2.9	1.3
	food, beverage, and tobacco processing	9.7	6	3	3.3
	food retail and wholesale	35.3	31.3	39.1	37.3
	food service	52.1	60.6	55	58.1

Figure 6: Original hierarchical table with multi-level headers for agri-food employment distribution across two Ontario regions (2011).

Flattened Representation. The original table (Figure 6) contains hierarchical headers encoding attributes such as region and worker category. We

flatten this structure by collapsing multi-level headers into composite column names (e.g., “eastern ontario / french-language workers”), as shown in Figure 7.

While this removes the visual hierarchy, the underlying semantics remain implicitly encoded in the column names. As a result, models must infer both structure and meaning from text, often leading to reasoning and SQL generation errors.

level_0	level_1	eastern ontario / french-language workers	eastern ontario / other workers	northern ontario / french-language workers	northern ontario / other workers
percent	input and service supply	2.9	2.1	2.9	1.3
percent	food, beverage, and tobacco processing	9.7	6	3	3.3
percent	food retail and wholesale	35.3	31.3	39.1	37.3
percent	food service	52.1	60.6	55	58.1

Figure 7: Flattened table obtained by collapsing hierarchical column headers into composite columns combining region and worker type.

Question and Reasoning Failure

Question:

In Northern Ontario, what proportion of workers in the restaurant and food services sector were French-language workers?

Generated CoT (Incorrect):

The question refers to the “food service” row in Figure 7. For Northern Ontario, the value for French-language workers is 55.0 and for other workers is 58.1. To compute the proportion of French-language workers, we divide 55.0 by (55.0 + 58.1), resulting in 0.486.

Predicted Answer: 48.6

Gold Answer: 55.0

Reasoning Context:

The query requires interpreting composite column semantics in Figure 7.

Issue:

The reasoning misinterprets the values in Figure 7 as counts that can be combined, whereas they are already percentages representing the distribution of workers within each category. The correct answer corresponds directly to the value in the “northern ontario / french-language workers” column for the “food service” row (55.0), without requiring any further computation.

Relational Transformation (QUIETT). QUIETT transforms the flattened table into a normalized relational schema by decomposing composite headers into explicit attributes such as *region* and *worker type*. Each row represents a single observation with well-defined fields, as shown in Figure 8.

Correct SQL After Transformation

Generated SQL (Correct):

```
SELECT value_percent
FROM transformed_table
WHERE region_area = 'Northern'
      AND region_province = 'Ontario'
      AND industry_subgroup = 'Food service'
      AND worker_language = 'French';
```

Predicted Answer: 55.0

Gold Answer: 55.0

Observation:

The transformation (Figure 8) eliminates ambiguity by representing attributes such as *region* and *worker language* as explicit columns. This allows the query to operate directly on atomic fields, enabling accurate mapping from natural language to schema and correct retrieval of the expected value.

H.2 Example 2: WebTables

Initial Representation. The original WebTable (Figure 9) presents Ohio State Buckeyes football season data in a flat format with columns such as *date*, *opponent*, *rank*, *site*, *result*, and *attendance*. Although the structure is not hierarchical, multiple semantic attributes are embedded within individual columns.

For example, the *result* column in Figure 9 combines outcome and score (e.g., “W 21-14”), while the *site* column merges stadium and location, and the *rank* column uses textual annotations (e.g., “#14”). These implicit encodings, as shown in Figure 9, require string parsing and can lead to errors in SQL generation.

Question and Reasoning Failure

Question:

Which opponent had the largest amount of points scored?

Generated CoT (Incorrect):

The question asks to find the opponent against whom the Buckeyes scored the largest number of

row_id	region_area	region_province	worker_language	industry_subgroup	food_related	service_related	retail_related	processing_related	wholesale_related	value_percent
1	Eastern	Ontario	French	Input and service supply	FALSE	TRUE	FALSE	FALSE	FALSE	2.9
2	Eastern	Ontario	French	Food, beverage, and tobacco processing	TRUE	FALSE	FALSE	TRUE	FALSE	9.7
3	Eastern	Ontario	French	Food retail and wholesale	TRUE	FALSE	TRUE	FALSE	TRUE	35.3
4	Eastern	Ontario	French	Food service	TRUE	TRUE	FALSE	FALSE	FALSE	52.1
5	Eastern	Ontario	Other	Input and service supply	FALSE	TRUE	FALSE	FALSE	FALSE	2.1
6	Eastern	Ontario	Other	Food, beverage, and tobacco processing	TRUE	FALSE	FALSE	TRUE	FALSE	6
7	Eastern	Ontario	Other	Food retail and wholesale	TRUE	FALSE	TRUE	FALSE	TRUE	31.3
8	Eastern	Ontario	Other	Food service	TRUE	TRUE	FALSE	FALSE	FALSE	60.6
9	Northern	Ontario	French	Input and service supply	FALSE	TRUE	FALSE	FALSE	FALSE	2.9
10	Northern	Ontario	French	Food, beverage, and tobacco processing	TRUE	FALSE	FALSE	TRUE	FALSE	3
11	Northern	Ontario	French	Food retail and wholesale	TRUE	FALSE	TRUE	FALSE	TRUE	39.1
12	Northern	Ontario	French	Food service	TRUE	TRUE	FALSE	FALSE	FALSE	55
13	Northern	Ontario	Other	Input and service supply	FALSE	TRUE	FALSE	FALSE	FALSE	1.3
14	Northern	Ontario	Other	Food, beverage, and tobacco processing	TRUE	FALSE	FALSE	TRUE	FALSE	3.3
15	Northern	Ontario	Other	Food retail and wholesale	TRUE	FALSE	TRUE	FALSE	TRUE	37.3
16	Northern	Ontario	Other	Food service	TRUE	TRUE	FALSE	FALSE	FALSE	58.1

Figure 8: Relational table produced by QUIETT after decomposing composite headers into atomic attributes such as region and worker type. Each row represents a single observation suitable for SQL generation.

Title : Ohio State Buckeyes Football Season					
Date	Opponent	Rank	Site	Result	Attendance
September 11	Baylor	#14	Ohio Stadium, Columbus, OH	W 21-14	88,622
September 18	at Michigan State	#12	Spartan Stadium, East Lansing, MI	W 31-10	73,483
September 25	Stanford	#13	Ohio Stadium, Columbus, OH	L 20-23	89,436
October 2	Florida State		Ohio Stadium, Columbus, OH	L 17-34	89,491
October 9	Wisconsin		Ohio Stadium, Columbus, OH	L 0-6	88,344
October 16	at Illinois		Memorial Stadium, Champaign, IL	W 26-21	73,488
October 23	at Indiana		Memorial Stadium, Bloomington, IN	W 49-25	52,040
October 30	Purdue		Ohio Stadium, Columbus, OH	W 38-6	89,341
November 6	Minnesota		Ohio Stadium, Columbus, OH	W 35-10	87,666
November 13	at Northwestern		Dyche Stadium, Evanston, IL	W 40-28	30,074
November 20	Michigan		Ohio Stadium, Columbus, OH	W 24-14	90,252
December 17	vs. BYU	#17	Jack Murphy Stadium, San Diego, CA	W 47-17	52,533

Figure 9: Original WebTable containing Ohio State Buckeyes football season data. The table is already flat but contains implicit structure such as game location, ranking, and match outcomes embedded in textual fields.

(49) and selects the wrong opponent.

Relational Transformation (QUIETT). The transformed table (Figure 10) converts the original WebTable (Figure 9) into a normalized schema by decomposing composite fields into atomic attributes. In particular, the *result* column is split into *Is_Win*, *Team_Score*, and *Opponent_Score*, and the *site* column into *Stadium*, *City*, and *State*.

This representation, as shown in Figure 10, removes ambiguity in textual encodings and enables direct, reliable SQL querying over structured fields.

points. Looking at the *result* column in Figure 9, we identify the largest number to the left of the dash. The maximum value is 49, which corresponds to Indiana.

Predicted Answer: Indiana

Gold Answer: Florida State

Reasoning Context:

The query requires extracting structured information from the *result* column in Figure 9.

Issue:

The reasoning incorrectly interprets the *result* column in Figure 9 by assuming that the left-hand value always represents the Buckeyes' score. However, the ordering of scores varies depending on whether the game was played at home or away. As a result, the model extracts an incorrect maximum value

Correct SQL After Transformation

Generated SQL (Correct):

```
SELECT Opponent_Name
FROM games
ORDER BY Opponent_Score DESC
LIMIT 1;
```

Predicted Answer: Florida State

Gold Answer: Florida State

Observation:

Using the transformed table (Figure 10), the query directly operates on the structured attribute *Opponent_Score*, enabling correct identification of the maximum value without requiring string parsing.

I Prompt Templates

For readability, we summarize the fixed stage-wise prompts here. The released code contains the exact prompt strings used in all experiments.

	Month	Day	Opponent_Name	Is_Away	Opponent_Ranked	Opponent_Rank	Is_Win	Team_Score	Opponent_Score	Point_Diff	Stadium	City	State	Is_Bowl_Game	Attendance
9	11		Baylor	FALSE	TRUE	14	TRUE	21	14	7	Ohio Stadium	Columbus	OH	FALSE	88622
9	18		Michigan State	TRUE	TRUE	12	TRUE	31	10	21	Spartan Stadium	East Lansing	MI	FALSE	73483
9	25		Stanford	FALSE	TRUE	13	FALSE	20	23	-3	Ohio Stadium	Columbus	OH	FALSE	89436
10	2		Florida State	FALSE	FALSE	NULL	FALSE	17	34	-17	Ohio Stadium	Columbus	OH	FALSE	89491
10	9		Wisconsin	FALSE	FALSE	NULL	FALSE	0	6	-6	Ohio Stadium	Columbus	OH	FALSE	88344
10	16		Illinois	TRUE	TRUE	15	TRUE	26	21	5	Memorial Stadium	Champaign	IL	FALSE	73488
10	23		Indiana	TRUE	FALSE	NULL	TRUE	49	25	24	Memorial Stadium	Bloomington	IN	FALSE	52040
10	30		Purdue	FALSE	FALSE	NULL	TRUE	38	6	32	Ohio Stadium	Columbus	OH	FALSE	89341
11	6		Minnesota	FALSE	FALSE	NULL	TRUE	35	10	25	Ohio Stadium	Columbus	OH	FALSE	87666
11	13		Northwestern	TRUE	FALSE	NULL	TRUE	40	28	12	Dyche Stadium	Evanston	IL	FALSE	30074
11	20		Michigan	FALSE	TRUE	13	TRUE	24	14	10	Ohio Stadium	Columbus	OH	FALSE	90252
12	17		BYU	NULL	TRUE	17	TRUE	47	17	30	Jack Murphy Stadium	San Diego	CA	TRUE	52533

Figure 10: QUIETT-transformed relational representation of the original WebTable (Figure 9). Composite fields such as *result* and *site* are decomposed into atomic attributes including *Is_Win*, *Team_Score*, *Opponent_Score*, *Stadium*, *City*, and *State*. The resulting schema enables direct and unambiguous SQL querying.

I.1 Issue Generation (Step 1)

Issue Generation Prompt

You are generating analysis questions from a RAW table

Table Title: {table_title}
Column Descriptions: {column_descriptions}
Table (Markdown): {raw_markdown_table}

Objective:

- Produce 12-20 diverse, realistic questions that expose structural or semantic deficiencies in the raw table.
- Questions should not be reliably answerable from this RAW table without cleaning, structuring, or transforming the data.

Output format:

- Return ONLY a JSON ARRAY (no prose, no markdown, no backticks).
- Each item MUST be a JSON OBJECT with these exact keys:
 - "qid": string like "Q1", "Q2", ... (contiguous, no gaps)
 - "text": natural language question
 - "depends_on": list of raw column names, or ["unknown"] if unclear
 - "requires": list of data-prep needs
 - "failure_reason": short phrase on why naive analysis fails
- Return ONLY raw JSON. No text before or after the array.
- All strings must use double quotes. No trailing commas.

Issue Aggregation. Step-1 questions are post-processed by grouping on identical *failure_reason* strings. Each unique failure reason becomes one *issue* (*issue_id*, *description*, union of referenced cols, and the list of blocking_questions). The resulting issues array is the input to Step 2.

I.2 Plan Generation (Step 2)

Plan Generation Prompt

You are a transformation planner for arbitrary tabular data.

You NEVER write code. Output only a MACHINE-EXECUTABLE PLAN in JSON.

Inputs:

- TABLE_TITLE : short description of the table.
- COLUMN_DESCRIPTIONS: optional text describing columns.
- TABLE_PREVIEW : first 3 rows in Markdown.
- ISSUES_JSON : issues detected in Step 1.

Design a transformation plan that:

1. FIXES ALL ISSUES from ISSUES_JSON
2. OPENS UP THE TABLE with derived columns
3. PRESERVES original information without redundancy

Output ONE JSON OBJECT:

```
{
  "table_id": string,
  "strategy": string,
  "steps": [
    {
      "step_id": string,
      "op": string,
      "description": string,
      "reads": [string],
      "writes": [string],
      "params": { ... },
      "fixes_issues": [string],
      "depends_on": [string]
    }
  ],
  "final_output": {
    "primary_key": [string],
    "columns": [
      {
        "name": string,
        "role": "canonical" | "derived" | "helper" | "raw_snapshot"
      }
    ]
  }
}
```

Column roles:

- canonical : original column retained in final table
- derived : new column created; always in final table
- helper : intermediate column; dropped via select
- raw_snapshot: original column dropped from final table

Use only operators from the fixed operator vocabulary (Table~\ref{tab:operator_vocab}). Each op must match a registry entry and its params must satisfy required-parameter constraints.

CRITICAL CONSTRAINTS:

1. derive_conditional - use EXACT structure:

```
{ "conditions": [{"condition": "col contains 'v'", "value": true}], "default": false }
```
2. derive_math:

```
"len(col)", "year(col)", "col_a + col_b"
```
3. extract_regex / parse_number - include commas in numeric patterns:

```
"[0-9,]+" not "[0-9]+"

4. Row count may increase only when split_multi_value
   with
   explode=true is explicitly declared in the plan
   strategy.

COLUMN RETENTION:
KEEP (canonical) when derived columns capture only
PART of the info.
DROP (raw_snapshot) when derived columns capture ALL
info, e.g.:
"W 27-7" -> is_win + score_for + score_against
-> raw_snapshot
"45.2%" -> pct_numeric
-> raw_snapshot
"1990-2005"-> start_year + end_year
-> raw_snapshot

WORKED EXAMPLES:

EXAMPLE A - Athletics Table (numeric, ordinal, boolean
):
Plan: add_row_id -> parse_number(Best) -> map_values
(Position) ->
cast_column(pos_numeric) -> derive_conditional
(is_podium) ->
select(...)

EXAMPLE B - Race Results with Multi-Driver Rows (
explode):
Plan: split_multi_value(Co-Drivers, explode=true) ->
trim_whitespace -> add_row_id ->
derive_conditional(is_classified) -> select
(...)

ANTI-PATTERNS (NEVER DO):
1. Writing raw "W"/"L" instead of is_win boolean.
2. cast_column without out= parameter.
3. Skipping the final select step.
4. Exploding a notes or description column.

MANDATORY: Every plan MUST end with a select step
containing:
_row_id, all derived columns, all canonical columns.
NO raw_snapshot or helper columns.

Return ONLY the JSON object. No markdown, no backticks
, no comments.
```

Plan Validation. The emitted plan is validated against a fixed operator schema before execution. Unknown operators, missing required parameters, undefined reads, dangling depends_on, or duplicate writes are all rejected. A failed validation triggers a single regeneration attempt with the validator’s error list appended to the prompt.

I.3 Code Generation (Step 3)

```
Code Generation Prompt

You are a Python Data Engineer. Write a snippet to
transform
DataFrame 'df'.

CONTEXT:
- Current columns: {columns}
- DataFrame shape: {n_rows} rows x {n_cols} columns
- Sample data (first 3 rows of input columns): {
sample_rows}

TASK: {task_description}
OPERATION: {operation}
Parameters: {parameters}
Input columns: {input_columns}
Output columns: {output_columns}
```

```
REQUIREMENTS:
1. All output columns MUST be created.
2. On extraction/parsing failure, fall back to the
   original value.
3. Do NOT create all-NaN columns.
4. Handle missing data gracefully.
5. Do NOT drop rows unless explicitly required.

EXAMPLE PATTERNS:
df['new'] = df['src'].str.extract(r'pattern')[0]
df['new'] = df['new'].fillna(df['src']) # fallback
df['parsed'] = pd.to_datetime(df['date'], errors='
coerce')

Return ONLY valid Python code. Assume df, pd, np, re
exist.
```

Execution Loop. For every step in the plan, Step 3 first attempts deterministic execution via the operator library. Only when the operation is custom or the deterministic call raises an exception does the system fall back to LLM code generation. The generated snippet is checked for: (i) all declared writes columns existing, (ii) no all-NaN columns, and (iii) row-count preservation unless split_multi_value with explode=true is declared. Validation failures are appended to the prompt and the model is re-queried up to two additional times.

I.4 SQL Query Generation for Q/A

```
Question Answering Prompt

You are answering a question about a structured table.
Analyze the table and find the answer.

## TABLE: {table_name}

### Columns:
{column_description}

### Data:
{transformed_table}

-

## QUESTION: {question}

-

1. **Interpret the question** - Identify what
   information is being requested
2. **Identify relevant columns** - Determine which
   column(s) contain the needed data
3. **Determine filtering criteria** - Identify any row
   -level conditions (e.g., specific entity, date,
   value)
4. **Identify required operation** - Determine if
   aggregation is needed:
   - Counting: COUNT
   - Maximum/minimum: MAX/MIN or ORDER BY with LIMIT
   - Summation: SUM
   - Lookup: Direct selection
5. **Extract the answer** - Locate the answer in the
   table data
```

Baseline	Input to reasoning model	Preprocessing and implementation
CoT	Full table serialized in Markdown with the question	No additional table preprocessing; uses our prompt summarized below.
CoT+SQL	Table name, schema, full table serialized in CSV, and the question	The table is loaded into SQLite; uses our SQL-generation prompt summarized below.
E5	Serialized table and question	Self-explanation, code generation and execution, and final-answer reasoning from the released implementation.
EEDP	Serialized table and question	Released Elicit-Extract-Decompose-Predict prompting pipeline with default settings.
NormTab	Normalized table and question	Table normalization produced by the authors' released implementation.
TabSQLify	Query-relevant decomposed table and question	SQL-based table decomposition from the authors' released implementation.
TableReasoner	Table and question in the released method's input format	Query-conditioned schema selection and reasoning from the released implementation.
GraphOTTER	Graph representation of the table and question	Graph construction and iterative reasoning from the released implementation.
Binder	Serialized table and question	Released API-augmented program-generation and execution pipeline.
Chain-of-Table	Table, question, and intermediate transformed views	Iterative table operations from the authors' released implementation.
Program-of-Thoughts	Serialized table and question	Released program-generation and execution pipeline with default settings.

Table 16: Baseline input and preprocessing configurations. Methods other than CoT and CoT+SQL use their publicly released implementations and default settings.

```

<reasoning>
[Step-by-step analysis]
</reasoning>

<answer>
[Extracted answer value]
</answer>

<sql_plan>
SELECT: [target column]
FROM: {table_name}
WHERE: [filter condition if applicable]
ORDER BY: [ordering if applicable]
AGGREGATION: [function if applicable]
</sql_plan>

```

Reason step by step internally using the table, but do not include the reasoning in the response.

Return exactly one line:
Final Answer: <answer value only>

J.2 Chain-of-Thought with SQL Prompt

CoT+SQL receives the same complete table used for downstream execution, together with its SQLite schema. The generated query is executed against this table to obtain the final prediction.

J Baseline Implementation Details

Each baseline is evaluated using the input representation, preprocessing stages, prompts, and default settings specified by its original implementation. We define only the CoT and CoT+SQL prompts; all remaining methods use the prompts and processing pipelines from their publicly released repositories without additional tuning. Table 16 summarizes these configurations.

J.1 Chain-of-Thought Prompt

For readability, we report the prompt structure below. The released code contains the exact system and user prompt strings used in our experiments.

CoT Prompt

You are answering a question using the provided table.

```

## TABLE (Markdown)
{table_text}

## QUESTION
{question}

```

CoT+SQL Prompt

You are answering a question by writing a SQLite query over the provided table.

```

## TABLE NAME
{table_name}

```

```

## SCHEMA
{schema_text}

```

```

## TABLE (CSV)
{table_text}

```

```

## QUESTION
{question}

```

Analyze the question and table, then write one SQLite SELECT query that returns the answer.

Requirements:

1. Use the provided table name exactly.
2. Output only one valid SQLite SELECT statement.
3. Do not output explanations, markdown, or multiple queries.
4. Quote identifiers with double quotes when necessary.
5. Do not use statements that modify the database.

Return only the SQL query.

J.3 Scoring and Execution Failures

All methods use the answer-normalization and scoring procedure described in Section 4.2. For CoT+SQL, the generated query is executed against the complete SQLite table, and the returned value is treated as the model prediction. A query that cannot be parsed or executed produces an empty prediction and is scored as incorrect. The same rule is applied to execution failures from other program-generating baselines; no gold-answer-assisted repair is used.