# TabScope: Question-Adaptive Scope Selection for Table Question Answering

**Yuxiang Wang, Junhao Gan, Jianzhong Qi**
The University of Melbourne
yuxiang.wang8@student.unimelb.edu.au
{junhao.gan, jianzhong.qi}@unimelb.edu.au

## Abstract

Large Language Models (LLMs) have shown strong performance on table question answering, yet their accuracy often degrades as table size increases. We find that this degradation is not uniform across question types. Localization-sensitive questions are particularly affected by irrelevant table content, while questions requiring broader evidence may still benefit from full-table reasoning. Based on this observation, we propose a question-adaptive framework that dynamically selects between localized and full-table reasoning. The framework constructs question-specific sub-tables through operation-aware table decomposition and uses the predicted question type to determine the appropriate reasoning mode. We further introduce silver reference sub-tables for evaluating evidence selection and construct SLQA, a benchmark based on real-world long tables. Experiments on WikiTQ and SLQA show that localization is particularly effective for lookup and local reasoning questions, while adaptive selection between localized and full-table reasoning achieves the best overall performance. These results highlight that long-table QA requires deciding not only how to localize, but also when to localize. Our code and datasets are publicly available at https://github.com/yxw-11/TabScope.

## 1 Introduction

Large Language Models (LLMs) have demonstrated strong reasoning capabilities for table question answering (TableQA), particularly when combined with chain-of-thought (CoT) prompting (Wei et al., 2022; Chen, 2023; Wang et al., 2025b, 2026). However, their performance remains sensitive to table size. As tables grow longer, LLMs struggle to identify the small set of rows and columns needed to support the answer from noisy inputs, leading to lower answer accuracy (Ye et al., 2023; Liu et al., 2024; Wang et al., 2025a; Ye et al., 2026).

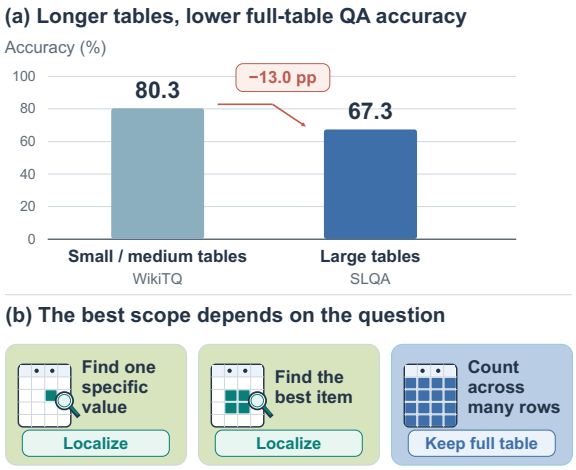


Figure 1: Motivation for question-adaptive table scope selection. (a) Full-table CoT exact-match accuracy on WikiTQ and SLQA, averaged across GPT-5-mini and LLaMA-3.3-70B. The benchmarks differ in both table size and data distribution. (b) Different questions require different evidence scopes, motivating selection between localized and full-table reasoning.

Our analysis shows that localization does not always improve accuracy. It removes irrelevant content but may omit necessary evidence. A lookup may need only one record, while counting requires all matching rows. Missing even one row can change the answer. Full-table reasoning preserves evidence but becomes harder as tables grow. Thus, the preferred scope depends on question type and table size. Figure 1 illustrates these challenges and the motivation for question-adaptive scope selection. This leads to the central question of our work: *when should a TableQA model localize the table, and when should it reason over the full context?*

We regard TableQA as a question-adaptive reasoning problem rather than a fixed decomposition pipeline, especially for long tables. We propose TABSCOPE, which first determines whether a question is better served by the full table or a compact sub-table. When localization is expected to

be beneficial, TABSCOPE uses an operation-aware decomposer with refinement to select the relevant rows and columns before reasoning. Otherwise, the model reasons directly over the full table. This design uses compact table regions when helpful while preserving broader context when necessary.

Existing TableQA benchmarks present two additional limitations for studying evidence localization. First, datasets such as WikiTableQuestions (Pasupat and Liang, 2015) provide only tables, questions, and final answers, without explicit annotations of the supporting sub-tables. This makes it difficult to evaluate the quality of intermediate evidence selection. Second, most existing benchmarks (Pasupat and Liang, 2015; Chen et al., 2020; Wu et al., 2025) contain relatively small tables and provide limited coverage of long-table settings. We address the first limitation by constructing silver reference sub-tables for evaluating decomposition quality. For the second, we develop an automatic pipeline for generating question–answer pairs from real-world tables and use it to construct SLQA, a benchmark for long-table question answering.

Our main contributions are as follows:

- We construct silver reference sub-tables for directly evaluating decomposition quality and propose an operation-aware decomposition method that improves evidence selection.
- We analyze when to localize and introduce a question-adaptive framework that selects between localized and full-table reasoning.
- We develop an automatic pipeline for generating question–answer pairs from real-world tables and use it to build SLQA, a benchmark for long-table question answering.

## 2 Related Work

Recent LLM-based TableQA methods increasingly address the difficulty of reasoning over large tables by reducing, retrieving, or transforming relevant rows and columns before answer generation. We first review two main lines of localization-oriented methods: *semantic evidence selection* and *operation-aware table reduction*. We then review existing TableQA benchmarks and evidence annotations, which provide the context for our silver sub-table construction and long-table benchmark.

**Evidence localization and retrieval.** One line of work selects a smaller sub-table containing relevant evidence before reasoning. DATER (Ye et al., 2023) uses LLMs to decompose both table and questions, selecting relevant rows and columns for downstream reasoning. H-STAR (Abhyankar et al., 2025) similarly extracts relevant table regions within a hybrid symbolic–textual framework. Others use retrieval-augmented methods for table reasoning. TableRAG (Chen et al., 2024) retrieves relevant schema and cell information for large-scale table understanding, T-RAG (Pan et al., 2022) combines dense table retrieval with generation for open-domain TableQA, and GTR (Zou et al., 2025) performs graph-based hierarchical retrieval for cross-table question answering. While these methods mainly improve how relevant evidence is retrieved or selected, our approach makes evidence selection operation-aware and treats localization as a question-dependent decision.

**Program-guided table decomposition.** Another direction obtains intermediate tables through explicit operations or executable programs. Chain-of-Table (Wang et al., 2024) iteratively transforms the table during reasoning and treats intermediate tables as evolving reasoning states. Table-Critic (Yu et al., 2025) iteratively critiques and refines the intermediate table before answer generation. ReAcTable (Zhang et al., 2024) combines LLM reasoning with tools such as SQL and Python executors to manipulate tabular data step by step. TabSQLify (Nahid and Rafiei, 2024) constructs question-relevant sub-tables through SQL queries, while Plan-of-SQLs (Nguyen et al., 2025) executes a sequence of SQL steps to provide interpretable intermediate reasoning traces. In contrast, TABSCOPE does not require a full operation chain or executable program, and applies decomposition only when localized reasoning is helpful.

**Evidence supervision and long-table evaluation.** Most TableQA benchmarks, such as WikiTableQuestions, provide tables, questions, and final answers, but no explicit annotations of the supporting rows and columns (Pasupat and Liang, 2015). This limits direct evaluation of intermediate sub-tables. In addition, many commonly used benchmarks contain relatively small tables, typically under 4K table tokens, making them less suited for evaluating long-table reasoning (Chen et al., 2020; Wu et al., 2025; Cheng et al., 2022). We address these gaps by constructing silver reference sub-tables for evaluating decomposition quality and by introducing SLQA, a benchmark built from real-world long tables.

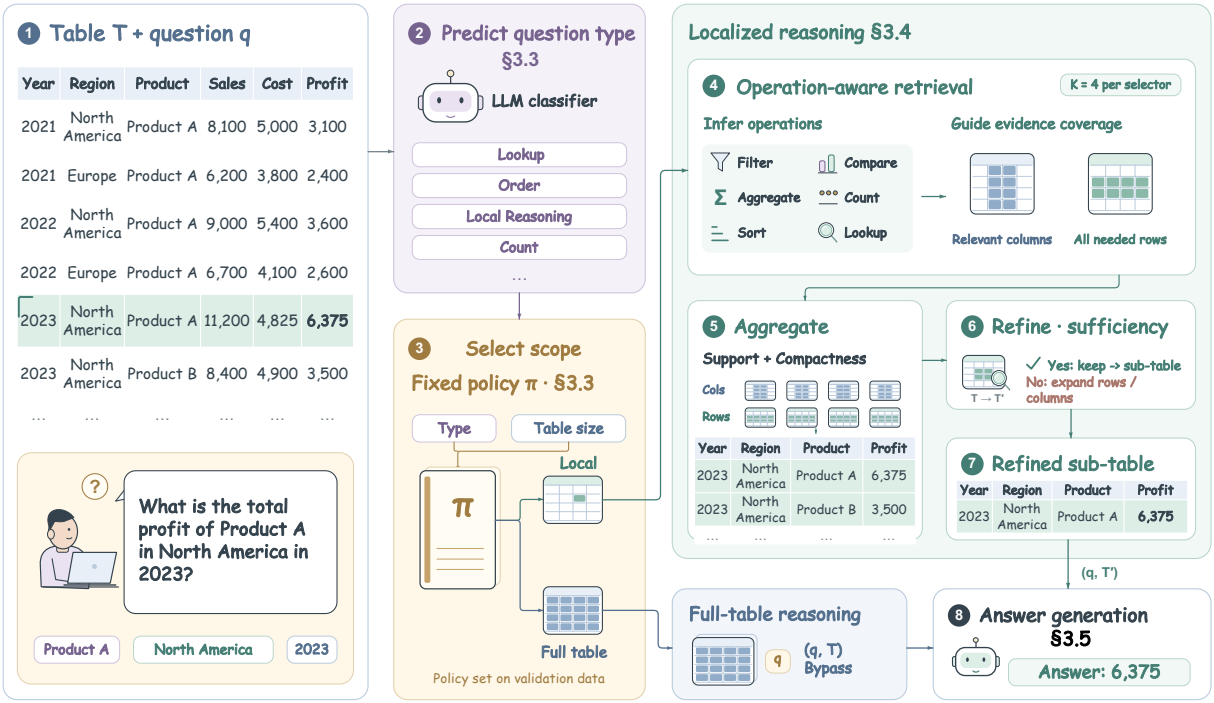


Figure 2: Overview of TABSCOPE. The framework selects localized or full-table reasoning for each question. The localized path constructs a refined sub-table, whereas the full-table path bypasses decomposition.

## 3 Methodology

### 3.1 Problem Formulation

Given a table $T$, a question $q$, and a reference answer $a$, the goal of TableQA is to predict an answer $\hat{a}$ that matches $a$. We represent the table as $T = (H, R)$, where $H = h_1, \ldots, h_m$ denotes the column headers and $R = r_1, \ldots, r_n$ denotes the table rows. For each question, a model can either reason over the full table $T$ or reduce it to a question-specific sub-table $T'$. We define the sub-table as $T' = T[R', C']$, where $R' \subseteq R$ and $C' \subseteq H$ are the selected rows and columns. Our goal is not to always minimize the table, but to select the appropriate table scope for each question.

### 3.2 Overview of TABSCOPE

We propose TABSCOPE, a question-adaptive framework that selects the appropriate table scope before answer generation. As illustrated in Figure 2, TABSCOPE consists of three components. (i) An LLM-based scope selector predicts the question type and uses a fixed policy to choose localized or full-table reasoning. (ii) When localization is selected, an operation-aware decomposer retrieves the required rows and columns and refines them into a question-specific sub-table. (iii) The answer model generates the answer from the selected input, either the refined sub-table or the original full table.

### 3.3 Question-Adaptive Scope Selection

TABSCOPE is designed to determine whether localization should be applied to each question. The scope selector consists of an LLM-based operation classifier $C_\theta$ and a validation-calibrated scope policy $\pi$. Given a question $q$ and table $T$, the classifier predicts an operation-defined question type $\hat{\tau}$. The policy then selects the reasoning mode according to the predicted type and table-size regime:

$$\begin{aligned} \hat{\tau} &= C_\theta(q, T), \\ z &= \pi\big(\hat{\tau}, s(T)\big), \\ z &\in \{\text{local}, \text{full}\}. \end{aligned} \tag{1}$$

Here, $s(T)$ denotes the table-size regime based on serialized token count. If $z = \text{local}$, TABSCOPE constructs a question-specific sub-table before answer generation. Otherwise, it bypasses decomposition and reasons directly over the original table.

We calibrate $\pi$ using validation accuracy. The default type-wise policy is derived from the validation of WikiTQ, while SLQA validation results adjust frequency counting for large tables (Appendix A). At inference time, the LLM predicts only $\hat{\tau}$, while the policy remains fixed and determines the final scope. The classification prompt is provided in Appendix B.4, and the operation definitions and calibration results are reported in Appendix A. We further compare this policy with direct LLM and learned routing in Appendix E.2.

### 3.4 Operation-Aware Table Decomposition

When the scope selector chooses localized reasoning, TABSCOPE constructs a question-specific sub-table through operation-aware decomposition. The goal is to retrieve the rows and columns needed for the question's reasoning process, rather than relying only on lexical overlap between the question and table content. The decomposer has three steps: operation-aware retrieval, evidence aggregation, and sub-table refinement. All the prompts used are shown in Appendix B.

**Operation-aware retrieval.** The decomposer first identifies the operation required by the question, such as lookup, filtering, comparison, or counting. This operation determines both the rows and columns to retrieve. For example, for the question *"How many Australian teams scored above 10?"*, the decomposer should retrieve rows whose country is Australia and whose score is greater than 10, together with columns such as country, score, and team identifier. Although the final answer is only a count, the sub-table must preserve the evidence needed to verify the filtering operation. This shows why decomposition should be guided by the operation rather than lexical overlap alone.

**Evidence aggregation.** A single LLM retrieval may be unstable because generation can vary across samples. It may miss necessary evidence or include weakly related rows and columns. To make retrieval more robust, we sample $K$ retrieval outputs for the same question, where $K = 4$ by default.

We aggregate row retrievals and column retrievals separately. Each retrieval output is treated as a candidate $c$, which contains either a selected row set or a selected column set. Each candidate is assigned a reliability score $s(c)$. When generation confidence is available for the LLM, $s(c)$ is computed from the average token log probability of the candidate. Otherwise, we set $s(c) = 0$, reducing $w(g)$ to frequency-based voting. Candidates that select the same normalized row set or column set are merged into a candidate group $g$.

For each group, we compute a support weight:

$$w(g) = \sum_{c \in g} \exp(s(c)). \tag{2}$$

Here, $w(g)$ measures how strongly the retrieval samples support group $g$. The term $\exp(s(c))$ converts the candidate score into a positive voting weight, so a group receives higher support when the same evidence set appears multiple times or when its candidates have higher reliability scores.

After aggregation, each distinct row set returned by the retrieval samples forms a row group, and each distinct column set forms a column group. Groups returned more frequently or with higher confidence receive larger weights. For example, if three out of four samples retrieve the same rows, a candidate sub-table containing these rows should receive stronger support.

To construct candidate sub-tables, we first rank the row groups and column groups by their support weights. We progressively union the top-ranked groups to form candidate row sets $R'$ and candidate column sets $C'$, removing duplicate candidates. We then enumerate the resulting row–column combinations and score each candidate sub-table $T[R', C']$ using the support and compactness criteria defined below. For a candidate row set $R'$, we compute:

$$\mathrm{RowScore}(R') = \frac{\sum_{g \in G_r; g \subseteq R'} w(g)}{\sum_{g \in G_r} w(g)}. \tag{3}$$

Here, $G_r$ is the set of distinct row groups, $g$ is one retrieved row set, and $w(g)$ is its weight. Thus, $\mathrm{RowScore}(R')$ is the proportion of total row-group weight covered by $R'$. The column score $\mathrm{ColScore}(C')$ is computed in the same way over the column groups. We combine the two scores as:

$$\mathrm{Support}(R', C') = \sqrt{\mathrm{RowScore}(R') \cdot \mathrm{ColScore}(C')}. \tag{4}$$

The support score is high only when the candidate sub-table covers both the rows and columns repeatedly selected across retrieval samples.

We then balance this support with sub-table size:

$$S(R', C') = \frac{\mathrm{Support}(R', C')}{\rho(R', C')^{\alpha}}, \rho(R', C') = \frac{|R'| \cdot |C'|}{|R| \cdot |H|}. \tag{5}$$

Here, $\rho(R', C')$ is the proportion of table cells retained by the candidate, and coefficient $\alpha \in [0.1, 0.3]$ controls the penalty on large sub-tables. We select the row–column combination with the highest $S(R', C')$ as the final decomposed table $T' = T[R', C']$.

**Sub-table refinement.** After evidence aggregation, TABSCOPE performs one refinement round by default to verify whether the sub-table $T'$ is sufficient for answering the question $q$. Given $q$, the full table $T$, and $T'$, the verifier determines whether additional rows, columns, or both are needed. If necessary, the missing evidence is added to $T'$, which is then passed to the answer model $M$.

| Dataset | Table Statistics | | | # QA Pairs | | |
|---|---|---|---|---|---|---|
| | Cols. | Rows | Tokens | Train | Valid. | Test |
| WikiTQ | 6.4 | 25.4 | 662.6 | 11,321 | 2,831 | 4,344 |
| SLQA | 11.0 | 733.8 | 9,786.2 | 1,324 | 239 | 1,110 |
| WTQ-SubTab | 1.8 | 6.5 | 104.8 | – | – | 4,344 |

Table 1: Statistics of the evaluation resources, averaged over questions. WikiTQ mainly contains small and medium tables, whereas all SLQA tables exceed 4,096 serialized tokens. WTQ-SubTab contains silver sub-tables for the 4,344 WikiTQ test questions.

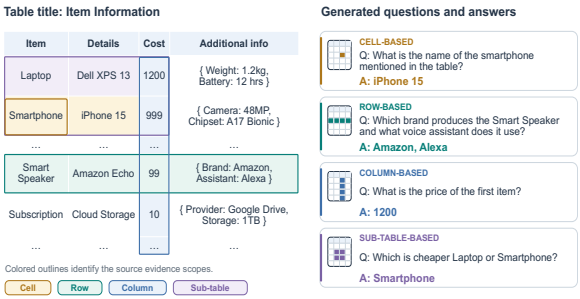


Figure 3: Generated QA pair examples.

### 3.5 Answer Generation

After scope selection, each question follows a single reasoning path. The QA model $M$ answers from the $(q, T')$ pair when localized reasoning is selected, and from $(q, T)$ pair otherwise.

## 4 Experiments

We evaluate TabScope to answer three questions. **RQ1:** How effectively does operation-aware decomposition identify compact and sufficient evidence? **RQ2:** Can localization mitigate performance degradation on long tables? **RQ3:** Can question-adaptive scope selection outperform fixed full-table or localized reasoning?

### 4.1 Experimental Setup

**Base Models.** We use two base LLMs from different families: LLaMA-3.3-70B (Dubey et al., 2024), and GPT-5-mini (OpenAI, 2026). They cover both open-source and closed-source settings, as well as large and relatively smaller LLMs. Detailed settings are provided in Appendix C.

**Benchmarks and Evaluation Resources.** We evaluate TabScope on three benchmarks. Table 1 shows the statistics for these benchmarks.

**WikiTableQuestions (WikiTQ).** WikiTableQuestions (Pasupat and Liang, 2015) is a widely used TableQA benchmark with diverse question types and tables spanning a broad range of sizes, including tables whose sizes exceed 4,096 tokens. We use WikiTQ for standard TableQA evaluation, table-size analysis, and type-wise scope analysis.

**SLQA.** Existing TableQA benchmarks contain relatively few long tables (token size exceeds 4,096 tokens), making it difficult to systematically evaluate long-table reasoning. We therefore construct SLQA, a dedicated long-table TableQA benchmark. We extract tables from Spider (Yu et al., 2018) whose table sizes exceed 4,096 tokens, but replace the original NL-to-SQL questions with newly generated TableQA questions. We use GPT-4o mini (OpenAI, 2024) to generate question–answer pairs from four evidence scopes: a cell, a row, a column, or a sub-table. The model first identifies an answer supported by the specified evidence and then formulates the corresponding question. All generated pairs are manually reviewed with fewer than 30% requiring correction. The generation prompts and detailed analysis are provided in Appendix D.1. Figure 3 shows examples of generated QA pairs.

**WTQ-SubTab.** Final-answer accuracy alone does not reveal whether a decomposition method retrieves the correct intermediate rows and columns. We therefore construct WTQ-SubTab, a silver-reference evaluation set derived from WikiTQ. Each instance consists of a question, its original table, and a silver sub-table containing the rows and columns needed to answer or verify the question.

We compare three construction strategies using 200 randomly sampled WikiTQ QA pairs with manually annotated gold sub-tables. Schema-based construction first asks an LLM to identify question-relevant schema elements and values, which are then mapped back to the original table. Direct generation instead asks the LLM to predict the target sub-table directly. The third strategy applies a verifier to the directly generated sub-table and repairs missing, invalid, or redundant selections. Direct generation with refinement achieves the highest agreement with the human annotations and is therefore used to construct WTQ-SubTab. Detailed results are reported in Appendix D.2.

Given an instance $(q, T, a)$, an LLM selects the minimal supporting rows and columns. A verifier then detects invalid, incomplete, or unnecessarily large selections and requests a repair when needed. The resulting reference sub-table is defined as $T^* = T[R^*, C^*]$, where $R^*$ and $C^*$ denote the verified rows and columns.

| Category | Method | WikiTQ | | | SLQA | | |
|---|---|---|---|---|---|---|---|
| | | *GPT* | *LLaMA* | **Avg.** | *GPT* | *LLaMA* | **Avg.** |
| Decomposition | TableRAG (Chen et al., 2024) | 69.5 | 56.8 | 63.2 | 47.4 | 49.7 | 48.6 |
| | TabSQLify (Nahid and Rafiei, 2024) | 73.5 | 72.3 | 72.9 | 59.3 | 34.0 | 46.7 |
| | Table-Critic (Yu et al., 2025) | – | 70.1 | – | – | 52.3 | – |
| | DATER (Ye et al., 2023) | 76.3 | 70.5 | 73.4 | 56.0 | 54.2 | 55.1 |
| | Chain-of-Table (Wang et al., 2024) | 75.8 | 71.8 | 73.8 | 63.1 | 57.2 | 60.2 |
| | TABSCOPE (decomposition-only) | <u>82.4</u> | 76.8 | 79.6 | <u>72.3</u> | 61.9 | 67.1 |
| Full-table | RoT (Zhang et al., 2025) | 81.7 | <u>78.7</u> | 80.2 | 70.3 | 54.1 | 62.2 |
| | CoT (Wei et al., 2022) | 82.1 | 78.5 | <u>80.3</u> | 72.1 | <u>62.5</u> | <u>67.3</u> |
| Adaptive | TABSCOPE | **83.3** | **81.1** | **82.2** | **73.4** | **62.8** | **68.1** |

Table 2: Exact-match accuracy (%) on WikiTQ and SLQA. The decomposition-only variant applies operation-aware decomposition to every question, whereas the TABSCOPE adaptively selects between localized and full-table reasoning. *GPT* and *LLaMA* denote GPT-5-mini and LLaMA-3.3-70B, respectively. *Avg.* denotes the average across the two answer models and is omitted when one result is unavailable. The best and second-best results per column are shown in bold and underlined, respectively. Rows highlighted in green denote variants of our method.

WTQ-SUBTAB is used only to evaluate decomposition quality through row-, column-, and cell-level matching. Its silver annotations are not available to TABSCOPE during inference. Construction prompts are provided in Appendix D.3.1.

**Baselines.** We compare TABSCOPE with two groups of baselines.

*Decomposition-based methods* reduce or transform the table before answering. DATER (Ye et al., 2023) decomposes questions and tables to select relevant rows and columns. Chain-of-Table (Wang et al., 2024) iteratively applies table operations and uses the final transformed table for answer generation. TabSQLify (Nahid and Rafiei, 2024) generates SQL queries to construct question-relevant sub-tables. TableRAG (Chen et al., 2024) retrieves relevant schema and cells for large-table reasoning, while Table-Critic (Yu et al., 2025) iteratively critiques and refines a candidate sub-table before answer generation. For a controlled comparison, we use each decomposition method only to produce its final sub-table. All resulting sub-tables are passed to the same answer model using an identical chain-of-thought prompt and answer normalization.

*Full-table reasoning methods* retain the original table during answer generation. Full-table CoT (Wei et al., 2022) serializes the complete table and reasons step by step over it. RoT (Zhang et al., 2025) performs row-wise reasoning before producing the final answer. These baselines represent two fixed scope strategies: always localizing or always retaining the full table.

**Evaluation Metrics.** For WikiTQ and SLQA, we report answer exact-match accuracy using the official WikiTableQuestions evaluator (Pasupat and Liang, 2015). For WTQ-SUBTAB, we independently compare the predicted rows, columns, and cells with their silver references. We report exact match, which requires the predicted set to match the reference set completely, and F1 score, which measures partial overlap between the two sets.

### 4.2 Main Results

Table 2 compares TABSCOPE with fixed full-table and decomposition-based methods on WikiTQ and SLQA, using GPT-5-mini and LLaMA-3.3-70B as answer models.

**RQ1: Operation-aware decomposition.** The decomposition-only variant consistently outperforms existing decomposition-based methods. On WikiTQ, it improves over the strongest baseline by 6.1 and 4.5 points with GPT-5-mini and LLaMA-3.3-70B, respectively. The gains increase to 9.2 and 4.7 points on SLQA. The larger improvement on SLQA suggests that operation-aware decomposition becomes particularly useful as tables grow longer and contain more irrelevant rows and columns.

To determine whether these gains arise from better sub-table selection, we directly evaluate decomposition quality on WTQ-SUBTAB. Predicted rows, columns, and cells are compared with the silver references in WTQ-SUBTAB using F1 and exact match. F1 measures partial agreement with the reference, whereas exact match requires the complete predicted set to be correct.

| Method | F1 | | | Exact Match | | |
|---|---|---|---|---|---|---|
| | Cell | Row | Column | Cell | Row | Column |
| TableRAG (Chen et al., 2024) | 26.56 | 44.15 | 52.30 | 3.20 | 13.92 | 16.98 |
| TabSQLify (Nahid and Rafiei, 2024) | 40.87 | 56.22 | 75.67 | 9.23 | 26.95 | 39.69 |
| Chain-of-Table (Wang et al., 2024) | 52.91 | 76.86 | 67.17 | 25.39 | 56.22 | 38.20 |
| DATER (Ye et al., 2023) | <u>68.00</u> | <u>78.46</u> | <u>85.93</u> | <u>36.63</u> | <u>61.50</u> | <u>59.34</u> |
| TABSCOPE (decomposition-only) | **70.70** | **82.78** | **86.95** | **41.03** | **70.74** | **60.54** |

Table 3: Sub-table decomposition quality on WTQ-SUBTAB. We report F1 and exact match (%) at the cell, row, and column levels. Methods are ordered from lower to higher average performance across the six metrics. The best and second-best results are shown in bold and underlined, respectively.

| Question Type | WikiTQ Δ | SLQA Δ | Preferred Scope |
|---|---|---|---|
| Lookup | +1.17 | +1.39 | Localized |
| Order/Superlative | +1.01 | +0.53 | Localized |
| Local Reasoning | +3.33 | – | Localized |
| Count-Diff | +2.13 | – | Localized |
| Count-General | -2.40 | -2.34 | Full table |
| Count-Frequency | -2.68 | +2.22 | Size-dependent |
| Compare | -1.24 | – | Full table |

Table 4: Type-wise localization gain in percentage points. Δ is computed as localized reasoning minus full-table reasoning. Positive values favor localization, while negative values favor full-table reasoning. WikiTQ is dominated by small and medium tables, whereas SLQA contains only large tables exceeding 4,096 tokens. "–" indicates insufficient examples. Definitions of the question types and the complete type-wise results are provided in Appendix A.

As shown in Table 3, TABSCOPE achieves the best result across all six metrics. Compared with DATER, the strongest baseline, it improves cell, row, and column F1 by 2.70, 4.32, and 1.02 points, respectively. Under the stricter exact-match criterion, the corresponding gains are 4.40, 9.24, and 1.20 points. The largest improvement occurs in row exact match, indicating that TABSCOPE more often preserves the complete set of rows required by a question rather than only a partially relevant subset. This improvement is consistent with our operation-aware retrieval, which selects evidence based on the required operations rather than semantic relevance alone. Evidence aggregation and sub-table refinement further improve the completeness and reliability of the selected rows and columns. The gains in cell F1 and cell exact match further confirm the improved quality of the selected sub-tables. Together with the QA gains in Table 2, these results suggest that better evidence selection contributes to more accurate answer generation.

**RQ2: Localization on long tables.** As shown in Table 2, the decomposition-only variant achieves an average accuracy of 67.1% on SLQA, outperforming the strongest decomposition baseline by 6.9 points and nearly matching full-table CoT at 67.3%. This is notable because SLQA contains only tables exceeding 4,096 tokens. The result shows that a large portion of the original table can be removed while retaining nearly the same overall QA accuracy, suggesting that localization can mitigate the distracting effect of long-table context.

However, the similar average performance of localized and full-table reasoning hides substantial variation across question types. Table 4 shows that localization is more effective when the required rows and columns are concentrated within a small table region. This includes entity lookup, ranking or extrema, reasoning over nearby records, and differences between two filtered counts. In these cases, removing unrelated content makes the required relationships easier to identify.

Full-table reasoning remains stronger for questions requiring broader coverage, such as counting all matching rows or comparing multiple entities or groups. Frequency counting further illustrates the interaction between question type and table size. It favors full-table reasoning on WikiTQ but shifts toward localization on SLQA, suggesting that preserving the complete table is useful at moderate sizes, while excessive context becomes increasingly harmful on longer tables. Localization can therefore reduce long-table degradation, but its effectiveness depends on both the reasoning requirement and the amount of table content. Detailed type definitions and complete results are provided in Appendix A.

**RQ3: Question-adaptive scope selection.** The complete TABSCOPE augments the decomposition-only variant with scope selection, which determines

whether each question should be answered from the localized sub-table or the original full table. As shown in Table 2, this complete framework achieves the highest accuracy across both datasets and answer models.

Compared with always applying operation-aware decomposition, evidence scoping improves accuracy by 0.9 and 4.3 points on WikiTQ with GPT-5-mini and LLaMA-3.3-70B, respectively. The corresponding gains on SLQA are 1.1 and 0.9 points. These improvements show that even a strong decomposition method should not be applied uniformly, since localization may remove rows or columns that remain useful for questions requiring broader table coverage.

The complete framework also outperforms the strongest full-table baseline in every setting. On WikiTQ, it improves accuracy by 1.2 points with GPT-5-mini and 2.4 points with LLaMA-3.3-70B. On SLQA, the gains are 1.3 and 0.3 points, respectively. Although full-table reasoning retains all table content, it also introduces more irrelevant rows and columns for questions that depend on a restricted table region.

Although TABSCOPE improves evidence selection, localization may omit rows or columns needed for operations requiring complete evidence, such as counting and comparison. Full-table reasoning avoids this risk but may introduce irrelevant content that complicates answering. The results also show that the preferred scope varies across question types. For frequency counting, the preferred scope differs between WikiTQ and SLQA, suggesting that table size may affect this choice. By selecting between localized and full-table reasoning, TABSCOPE avoids applying either strategy uniformly and achieves better overall accuracy. Appendix E reports oracle diagnostics and comparisons with direct LLM routing and a learned router.

**Ablation results.** Table 5 shows that all components contribute to the performance of TABSCOPE. Removing evidence aggregation causes the smallest decrease, reducing accuracy by 1.3 points on WikiTQ and 0.7 points on SLQA, while removing sub-table refinement leads to slightly larger drops of 1.7 and 1.5 points. These results suggest that aggregation mainly improves the stability of row and column selection, whereas refinement helps recover missing content from initially incomplete sub-tables. Removing operation-aware retrieval has the largest effect among the decomposition components, decreasing accuracy by 3.1 points on WikiTQ and 2.0 points on SLQA. This confirms that semantic relevance alone may fail to preserve the rows and columns required for operations such as counting, comparison, ranking, and aggregation.

| Variant | WikiTQ | SLQA |
|---|---|---|
| TABSCOPE | **81.1** | **62.8** |
| w/o evidence aggregation | 79.8 (−1.3) | 62.1 (−0.7) |
| w/o sub-table refinement | 79.4 (−1.7) | 61.3 (−1.5) |
| w/o operation-aware retrieval | 78.0 (−3.1) | 60.8 (−2.0) |
| w/o scope selection | 76.8 (−4.3) | 61.9 (−0.9) |

Table 5: Ablation results using LLaMA-3.3-70B, measured by exact-match accuracy (%). Values in parentheses denote the decrease relative to the TABSCOPE.

Removing scope selection yields the decomposition-only variant, which applies localization to every question. This reduces accuracy by 4.3 points on WikiTQ but only 0.9 points on SLQA. The larger drop on WikiTQ is consistent with the type-wise results, where some questions benefit from retaining the full table. In contrast, the small decrease on SLQA shows that operation-aware decomposition remains effective for most long-table questions by removing substantial irrelevant content while preserving the information needed for answering. Nevertheless, the complete framework still performs best, indicating that adaptive scope selection complements decomposition by retaining the full table when broader coverage is required.

## 5 Conclusion

We presented TABSCOPE, a framework that treats table scope as a question-dependent decision rather than a fixed preprocessing choice. By combining question-adaptive scope selection with operation-aware decomposition, TABSCOPE can reduce distracting table content when localization is helpful while preserving the full table when broader coverage is required. Experiments on WikiTQ and SLQA show that this design improves answer accuracy across both moderate- and large-scale tables, while the results on WTQ-SUBTAB confirm that the decomposed tables more closely match the rows, columns, and cells needed for answering. These findings suggest that effective TableQA depends not only on how a table is processed, but also on deciding how much of the table should be retained for each question.

## 6 Limitations

Our evaluation focuses on English TableQA benchmarks and two representative LLMs, with WikiTQ and SLQA covering different table scales. Extending the evaluation to additional domains, languages, and model families would provide a broader empirical assessment. Building on our comparisons of type-based, direct LLM, and learned routing, future work could further investigate adaptive scope selection across these broader settings.

## References


Nikhil Abhyankar, Vivek Gupta, Dan Roth, and Chandan K. Reddy. 2025. H-STAR: LLM-driven hybrid SQL-text adaptive reasoning on tables. In *NAACL*, pages 8841–8863.

Si-An Chen, Lesly Miculicich, Julian M Eisenschlos, Zifeng Wang, Zilong Wang, Yanfei Chen, Yasuhisa Fujii, Hsuan-Tien Lin, Chen-Yu Lee, and Tomas Pfister. 2024. TableRAG: Million-token table understanding with language models. In *NeurIPS*, pages 74899–74921.

Wenhu Chen. 2023. Large language models are few(1)-shot table reasoners. In *EACL*, pages 1090–1100.

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

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 *ACL*, pages 1094–1110.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The Llama 3 herd of models. *CoRR*, abs/2407.21783.

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

Md Mahadi Hasan Nahid and Davood Rafiei. 2024. Tabsqlify: Enhancing reasoning capabilities of llms through table decomposition. In *NAACL*, pages 5725–5737.

Giang Nguyen, Ivan Brugere, Shubham Sharma, Sanjay Kariyappa, Anh Totti Nguyen, and Freddy Lécué. 2025. Interpretable llm-based table question answering. *Trans. Mach. Learn. Res.*, 2025.

OpenAI. 2024. GPT-4o mini. https://platform.openai.com/docs/models/#gpt-4o-mini.

OpenAI. 2026. Openai GPT-5 system card. *CoRR*, abs/2601.03267.

Feifei Pan, Mustafa Canim, Michael R. Glass, Alfio Gliozzo, and James A. Hendler. 2022. End-to-end table question answering via retrieval-augmented generation. *CoRR*, abs/2203.16714.

Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables. In *ACL/IJCNLP*, pages 1470–1480.

Yuxiang Wang, Junhao Gan, Shengxiang Gao, Shenghao Ye, Zhengyi Yang, and Jianzhong Qi. 2026. Beyond linearization: Attributed table graphs for table reasoning. *CoRR*.

Yuxiang Wang, Junhao Gan, and Jianzhong Qi. 2025a. Tabsd: Large free-form table question answering with sql-based table decomposition. *CoRR*.

Yuxiang Wang, Jianzhong Qi, and Junhao Gan. 2025b. Accurate and regret-aware numerical problem solver for tabular question answering. In *AAAI*, pages 12775–12783.

Zilong Wang, Hao Zhang, Chun-Liang Li, Julian Martin Eisenschlos, Vincent Perot, Zifeng Wang, Lesly Miculicich, Yasuhisa Fujii, Jingbo Shang, Chen-Yu Lee, et al. 2024. Chain-of-Table: Evolving tables in the reasoning chain for table understanding. In *ICLR*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*, pages 24824–24837.

Xianjie Wu, Jian Yang, Linzheng Chai, Ge Zhang, Jiaheng Liu, Xeron Du, Di Liang, Daixin Shu, Xianfu Cheng, Tianzhen Sun, Tongliang Li, Zhoujun Li, and Guanglin Niu. 2025. Tablebench: A comprehensive and complex benchmark for table question answering. In *AAAI*, pages 25497–25506.

Shenghao Ye, Yu Guo, Dong Jin, Yuxiang Wang, Yikai Shen, Yunpeng Hou, Shuangwu Chen, Jian Yang, and Xiaofeng Jiang. 2026. When tableqa meets noise: A dual denoising framework for complex questions and large-scale tables. In *ACL*, pages 24022–24045.

Yunhu Ye, Binyuan Hui, Min Yang, Binhua Li, Fei Huang, and Yongbin Li. 2023. Large language models are versatile decomposers: Decompose evidence and questions for table-based reasoning. In *SIGIR*, pages 174–184.

Peiying Yu, Guoxin Chen, and Jingjing Wang. 2025. Table-Critic: A multi-agent framework for collaborative criticism and refinement in table reasoning. In *ACL*, pages 17432–17451.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir R. Radev. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *EMNLP*, pages 3911–3921.

Xuanliang Zhang, Dingzirui Wang, Keyan Xu, Qingfu Zhu, and Wanxiang Che. 2025. RoT: Enhancing table reasoning with iterative row-wise traversals. In *EMNLP*, pages 559–579.

Yunjia Zhang, Jordan Henkel, Avrilia Floratou, Joyce Cahoon, Shaleen Deep, and Jignesh M. Patel. 2024. Reactable: Enhancing react for table question answering. *Proc. VLDB Endow.*, 17(8):1981–1994.

Jiaru Zou, Dongqi Fu, Sirui Chen, Xinrui He, Zihao Li, Yada Zhu, Jiawei Han, and Jingrui He. 2025. GTR: graph-table-rag for cross-table question answering. *CoRR*, abs/2504.01346.

| Question Type | Definition |
|---|---|
| Lookup | Retrieve an attribute of a specified entity or record. |
| Order/Superlative | Identify an item through ranking, maximum, minimum, or relative order. |
| Local Reasoning | Reason over a small, restricted region of the table, such as related or neighboring records. |
| Count-Diff | Count two filtered subsets and compare or compute the difference between them. |
| Count-General | Count all records that satisfy one or more conditions. |
| Count-Frequency | Determine how often a particular value or category occurs. |
| Compare | Compare values associated with multiple entities or groups. |

Table 6: Definitions of the question types.

# A Appendix

## A Question Type-wise Scope Selection Analysis

We derive the scope-selection policy through an offline analysis on the validation set of WikiTQ. Table 6 summarizes the question types definitions.

Table 7 compares localized reasoning with full-table CoT for each question type. The *Diff.* column is computed as localized reasoning minus full-table CoT, so a positive value indicates an advantage for localization. Localized reasoning performs better for lookup, order/superlative, local reasoning, and count-difference questions, whose evidence is typically concentrated within a restricted table region. Full-table reasoning is stronger for general counting, frequency counting, and comparison questions, which often require evidence distributed across a broader portion of the table.

We further examine whether these preferences change with table size. Table 8 reports the same comparison on the large-table validation subset. Only question types with sufficient examples are reported. The overall pattern remains stable, except for frequency counting, where localized reasoning becomes more effective on large tables.

Based on these results, we define the scope-selection policy in Table 9. The default policy follows the overall validation-set preference for each question type. For large tables, we adopt the large-table preference when sufficient validation examples are available. Otherwise, we retain the default choice.

| Question Type | Localized | Full-table CoT | Diff. |
|---|---|---|---|
| Lookup | **86.51%** | 85.34% | +1.17 |
| Order/Superlative | **85.01%** | 84.00% | +1.01 |
| Local Reasoning | **69.90%** | 66.67% | +3.33 |
| Count-Diff | **76.60%** | 74.47% | +2.13 |
| Count-General | 79.78% | **82.18%** | -2.40 |
| Count-Frequency | 82.14% | **84.82%** | -2.68 |
| Compare | 67.90% | **69.14%** | -1.24 |

Table 7: Type-wise accuracy comparison between localized reasoning and full-table CoT on the validation set of WikiTQ. *Diff.* denotes localized reasoning minus full-table CoT. The better result for each type is highlighted in bold.

| Question Type | Localized | Full-table CoT | Diff. |
|---|---|---|---|
| Lookup | **70.68%** | 69.29% | +1.39 |
| Order/Superlative | **77.66%** | 77.13% | +0.53 |
| Count-General | 76.83% | **79.17%** | -2.34 |
| Count-Frequency | **84.44%** | 82.22% | +2.22 |

Table 8: Type-wise accuracy comparison on the large-table validation subset of SLQA. Question types with insufficient examples are omitted. *Diff.* denotes localized reasoning minus full-table CoT.

## B Prompt Templates

This appendix presents the prompt templates used by TABSCOPE. We omit benchmark-specific demonstrations and implementation metadata for clarity, while retaining the instructions, input fields, and output constraints used in our experiments.

### B.1 Operation-Aware Retrieval

We use separate prompts for row and column retrieval. Both prompts instruct the model to identify the operation required by the question and retrieve sufficient evidence for performing and verifying that operation.

#### B.1.1 Column Retrieval

**Column Retrieval Prompt**

```
Instruction:
Select the columns needed to answer the
    question using the
f_col() function.

Return exactly one function call:
f_col([column_1, column_2, ...])

Requirements:
- Copy column names exactly from the table
    schema.
- First identify the table operation required
     by the question,
  such as lookup, filtering, comparison,
    ranking, neighborhood
```

| Question Type | Default Scope | Large-table Scope |
|---|---|---|
| Lookup | Localized | Localized |
| Order/Superlative | Localized | Localized |
| Local Reasoning | Localized | Localized |
| Count-Diff | Localized | Localized |
| Count-General | Full table | Full table |
| Count-Frequency | Full table | Localized |
| Compare | Full table | Full table |

Table 9: Scope-selection policy derived from Tables 7 and 8. When the large-table subset does not contain sufficient examples for a question type, the default validation-set preference is retained.

```
  retrieval, counting, or aggregation.
- Retain the columns needed to perform and
    verify the operation,
  rather than selecting columns only by
    lexical overlap.
- Preserve identifier, condition, comparison,
    ordering, counting,
  and answer-bearing columns whenever needed.
- Do not output explanations, code, or SQL.

Table:
{LINEARIZED_TABLE}

Optional caption:
{TABLE_CAPTION}

Question:
{QUESTION}

Output:
```

#### B.1.2 Row Retrieval

**Row Retrieval Prompt**

```
Instruction:
Select the rows needed to answer the question
    using the
f_row() function.

Return exactly one function call:
f_row([row 1, row 3, ...])

If all rows are required, return:
f_row([*])

Requirements:
- Use the row numbers shown in the input
    table.
- First identify the table operation required
    by the question,
  such as lookup, filtering, comparison,
    ranking, neighborhood
  retrieval, counting, or aggregation.
- Retrieve enough rows to perform and verify
    the operation,
  rather than selecting only a likely answer
    row.
- Preserve comparison sets, anchor rows,
    neighboring rows, and
  aggregation inputs whenever needed.
- Do not output explanations, code, or SQL.

Table:
{LINEARIZED_TABLE}

Optional caption:
{TABLE_CAPTION}

Question:
{QUESTION}

Output:
```

### B.2 Sub-table Refinement

TABSCOPE performs one refinement round to check whether the selected sub-table contains sufficient information for answering the question. The verifier may retain the current selection or add missing rows and columns. The refined evidence is returned as a structured specification and used to construct the final sub-table.

**Sub-table Refinement Prompt**

```
You are refining an evidence sub-table for
    table question
answering.

Given the question, the current sub-table,
    and the available
information from the original table,
    determine whether the
sub-table contains sufficient evidence to
    answer the question.

Return only valid JSON:
{
  "action": "keep" | "expand",
  "final_rows": [row_id, ...],
  "final_columns": ["column_name", ...],
  "reason": "brief explanation"
}

Keep the current rows and columns when the
    sub-table is
sufficient. Otherwise, add the missing rows,
    columns, or both.
Use only row IDs and column names provided in
     the table context.
Do not generate new table values or use
    external knowledge.

Question:
{QUESTION}

Current sub-table:
{CURRENT_SUBTABLE_WITH_ROW_IDS}

Original table context:
{TABLE_CONTEXT}
```

### B.3 Answer Generation

After scope selection, each question follows only the selected reasoning path. The placeholder `{TABLE}` is instantiated with the refined sub-table when localized reasoning is selected, and with the original table otherwise.

**System Prompt**

```
You are a precise table question answering
    assistant.
Follow the required output format exactly.
```

**User Prompt**

```
Read the following table and answer the
    question using only the
information provided in the table.

Table:
{TABLE}

Question:
{QUESTION}

Reason step by step and return exactly two
    sections:

<think>
Your reasoning
</think>

<answer>
Your final answer
</answer>

Do not include any text outside these two
    sections.
```

### B.4 LLM-Based Question-Type Selection Prompt

The scope selector does not directly ask the LLM to choose between localized and full-table reasoning. Instead, the LLM assigns the input question to one predefined question type. The fixed policy described in Appendix A then maps the predicted type and table-size regime to the final reasoning scope.

**Question-Type Selection Prompt**

```
Classify the table question according to the
    reasoning required
to answer it.

Choose exactly one question type:

- "lookup": retrieve a value associated with
    a specified entity
  or record.
- "order_superlative": identify an item
    through ranking,
  maximum, minimum, or relative order.
- "local_reasoning": reason over a small set
    of related or
  neighboring records.
- "count_diff": count two filtered groups and
     compare or compute
  the difference between them.
- "count_general": count all records
    satisfying one or more
  conditions.
- "count_frequency": determine how often a
    value or category
  occurs.
- "compare": compare values associated with
    multiple entities
  or groups.

Select the type according to the operation
    and table coverage
required by the question. Do not directly
    predict whether the
system should use a localized sub-table or
    the full table.

Question:
{QUESTION}

Table columns:
{TABLE_COLUMNS}

Table size:
{NUMBER_OF_ROWS} rows, {NUMBER_OF_COLUMNS}
    columns,
{NUMBER_OF_TOKENS} serialized tokens

Table preview:
{TABLE_PREVIEW}

Return only valid JSON:
{
  "question_type": "lookup" | "
    order_superlative" |
      "local_reasoning" | "count_diff" |
      "count_general" | "count_frequency" | "
    compare"
}
```

## C Detailed Experimental Settings

**Base LLMs and serving.** For open-source models, we use a local inference backend based on `swift.llm` with HuggingFace model weights. For GPT-5-mini, we use the OpenAI Chat Completions API. QA decoding uses a maximum generation budget of $4096$ new tokens and model default input budget. Retrieval sampling uses a temperature of $0.5$, while all other components use temperature $0$.

# D Dataset Construction Details

## D.1 Construction of SLQA

**Source tables.** We construct SLQA from existing large tables in Spider (Yu et al., 2018). Specifically, we retain tables whose serialized representations exceed 4,096 tokens under our preprocessing. We do not reuse the original Spider questions because they are designed for NL-to-SQL evaluation and closely follow executable logical forms. Instead, we generate new questions intended for direct question answering over serialized tables.

**Self-adaptive QA generation.** Our preliminary study found that assigning a predetermined answer to the model and asking it to generate a corresponding question often produced unnatural or ambiguous question–answer pairs. The acceptance rate of this procedure was below 50% in our manual evaluation. We therefore use a self-adaptive generation procedure. Rather than forcing a predetermined answer, we provide an evidence scope and allow the model to first identify an answer supported by that evidence and then generate the corresponding question. We use four evidence scopes: a single cell, a selected row, a selected column, and a selected sub-table. These scopes encourage questions requiring different amounts and configurations of table evidence.

**Reasoning coverage.** We retain both SQL-executable and non-SQL-executable questions. SQL-executable questions cover structured operations such as filtering, comparison, counting, and aggregation. Non-SQL-executable questions support more flexible reasoning over semi-structured table content and prevent the benchmark from being limited to rigid SQL logical forms.

**Quality control.** Every generated pair is manually reviewed for answerability, table grounding, and answer correctness. Valid pairs are accepted directly. Questions with minor language or ambiguity issues are revised, and incorrect answers are corrected when the intended question remains valid. Pairs that cannot be reliably repaired are discarded. More than 70% of the generated pairs are accepted without modification, while fewer than 30% require manual correction.

### D.1.1 QA-Pair Generation Prompts

We use four prompt variants corresponding to the four evidence scopes. For cell-based generation, the model selects an answer cell directly. For the other variants, the selected row, column, or sub-table defines the evidence from which the question and answer are generated.

**Cell-Based QA Generation**

```
Use the given table as the only source of
    evidence.

Table header:
{TABLE_HEADER}

Table content:
{TABLE_CONTENT}

Randomly select one cell from the table as
    the answer and
generate a question whose answer is that cell.


Generate 10 diverse question--answer pairs.

Return one pair per line using exactly the
    following format:
Q: question; A: answer

Keep each answer concise and return only the
    answer value.
Do not provide explanations or any text
    outside the
question--answer pairs.
```

**Row-Based QA Generation**

```
Use the given table as the only source of
    evidence.

Table header:
{TABLE_HEADER}

Table content:
{TABLE_CONTENT}

Selected row:
{SELECTED_ROW}

Generate 4 diverse question--answer pairs
    using information
supported by the selected row. For each pair,
     first identify
a valid answer and then formulate a question
    whose answer can
be derived from the table.

Return one pair per line using exactly the
    following format:
Q: question; A: answer

The questions must be understandable without
    referring to
"the selected row" or "the given data." Keep
    each answer
concise and return only the answer value. Do
    not provide
explanations.
```

**Column-Based QA Generation**

```
Use the given table as the only source of
    evidence.

Table header:
{TABLE_HEADER}

Table content:
{TABLE_CONTENT}

Selected column:
{SELECTED_COLUMN}

Generate 4 diverse question--answer pairs
    using information
supported by the selected column. For each
    pair, first identify
a valid answer and then formulate a question
    whose answer can
be derived from the table.

Return one pair per line using exactly the
    following format:
Q: question; A: answer

The questions must be understandable without
    referring to
"the selected column" or "the given data."
    Keep each answer
concise and return only the answer value. Do
    not provide
explanations.
```

**Sub-Table-Based QA Generation**

```
Use the given table as the only source of
    evidence.

Table header:
{TABLE_HEADER}

Table content:
{TABLE_CONTENT}

Selected sub-table:
{SELECTED_SUBTABLE}

Generate 4 diverse question--answer pairs
    using information
supported by the selected sub-table. For each
     pair, first
identify a valid answer and then formulate a
    question whose
answer can be derived from the table.

Return one pair per line using exactly the
    following format:
Q: question; A: answer

The questions must be understandable without
    referring to
"the selected sub-table" or "the given data."
     Keep each answer
concise and return only the answer value. Do
    not provide
explanations.
```

### D.2 Selection of the Silver Construction Strategy

We randomly sample 200 WikiTableQuestions QA pairs and manually annotate the gold rows and columns required for each question. We then compare three silver sub-table construction strategies against these human annotations.

Schema-based construction first generates question-related schema elements and values and then retrieves the corresponding rows and columns from the original table. Direct generation predicts the target sub-table directly from the question, reference answer, and original table. Direct generation with refinement further uses a verifier to identify and repair missing, invalid, or redundant selections.

Table 10 shows that direct generation substantially improves over schema-based construction. Verifier-guided refinement further increases all six metrics, raising cell F1 from $71.58\%$ to $74.67\%$ and column F1 from $84.55\%$ to $90.15\%$. We therefore use direct generation with refinement to construct WTQ-SUBTAB.

### D.3 Construction of WTQ-SUBTAB

Most TableQA datasets provide only question–table–answer triples and do not annotate the rows and columns needed to support each answer. We construct WTQ-SUBTAB to enable direct evaluation of intermediate sub-table decomposition.

**Direct evidence construction.** For each `WikiTQ` instance $(q, T, a)$, an LLM predicts a structured evidence specification $E = (\tau, p, R^*, C^*)$. Here, $\tau$ denotes the inferred reasoning type, $p$ specifies the required evidence scope, and $R^*$ and $C^*$ denote the selected rows and columns. The reasoning types cover lookup, filtering, comparison, superlative and ordinal selection, neighborhood reasoning, counting, aggregation, and Boolean reasoning.

The evidence policy specifies whether the reference should contain only the answer-bearing evidence, the full comparison domain, all instances contributing to an aggregation, or the required neighboring context. The silver sub-table is then extracted from the original table as $T^* = T[R^*, C^*]$. The model outputs row identifiers and column names rather than regenerating table contents, ensuring that all selected evidence is grounded in the original table.

| Construction Method | Cell F1 | Cell P. | Cell R. | Cell EM | Row F1 | Col. F1 |
|---|---|---|---|---|---|---|
| Schema-based | 60.33 | 66.87 | 71.71 | 27.33 | 74.34 | 80.06 |
| Direct Generation | <u>71.58</u> | <u>80.59</u> | <u>75.25</u> | <u>42.67</u> | <u>80.38</u> | <u>84.55</u> |
| Direct Generation + Refinement | **74.67** | **85.58** | **77.48** | **44.67** | **82.90** | **90.15** |

Table 10: Comparison of silver sub-table construction strategies on 200 randomly sampled WikiTableQuestions QA pairs with manually annotated gold sub-tables. Results are reported as percentages. The best and second-best results are shown in bold and underlined, respectively.

**Verifier-guided repair.** The initial evidence specification may contain invalid indices, omit necessary evidence, or retain unnecessary table content. A verifier therefore checks whether the selected rows and columns are valid, sufficient, and compact. It also examines whether the selected evidence matches the inferred reasoning type and evidence policy.

When an issue is detected, the verifier feedback is passed to a repair prompt. The repair step may add missing rows or columns, remove redundant evidence, or revise the reasoning type and evidence policy. Only verified constructions are retained as silver references.

**Evaluation usage.** WTQ-SUBTAB is used to evaluate row selection, column selection, and cell-level evidence coverage. The reference answers and silver sub-tables are used only during offline construction and evaluation. They are not provided to TABSCOPE during inference.

### D.3.1 Silver Sub-Table Construction Prompts

We construct silver reference sub-tables using direct evidence selection followed by verifier-guided repair. Because this procedure is used only for offline evaluation-set construction, the prompts may access the reference answers. Implementation-specific fields and engineering details are omitted for clarity.

**Direct Silver Sub-Table Construction**

```
Given the question, reference answer, and
    full table, select the
smallest sub-table containing sufficient
    evidence to answer or
verify the question.

Return only valid JSON:
{
  "reasoning_type": "lookup" | "filter" | "
    comparison" |
      "superlative" | "ordinal" | "neighbor"
    | "count" |
      "aggregation" | "boolean" | "other",
  "evidence_policy": "answer_only" | "
    comparison_domain" |
      "all_relevant" | "neighbor_context",
  "rows": [row_id, ...],
  "columns": ["column_name", ...],
  "reason": "brief explanation"
}

Use only valid row IDs and exact column names
    from the table.
Retain all evidence required by the reasoning
    operation while
avoiding unnecessary rows and columns.

Question:
{QUESTION}

Reference answer:
{GOLD_ANSWERS}

Full table:
{TABLE_MARKDOWN}

Return only the JSON object.
```

**Silver Sub-Table Repair**

```
The previously selected sub-table was found
    to contain missing,
invalid, or unnecessary evidence.

Using the verifier feedback, return the
    smallest corrected
sub-table that is sufficient to answer or
    verify the question.

Return only valid JSON:
{
  "reasoning_type": "lookup" | "filter" | "
    comparison" |
      "superlative" | "ordinal" | "neighbor"
    | "count" |
      "aggregation" | "boolean" | "other",
  "evidence_policy": "answer_only" | "
    comparison_domain" |
      "all_relevant" | "neighbor_context",
  "rows": [row_id, ...],
  "columns": ["column_name", ...],
  "reason": "brief explanation"
}

Use only valid row IDs and exact column names
    from the table.
```

| Dataset | $N$ | Full | Local | Oracle | Gap |
|---|---|---|---|---|---|
| WikiTQ | 4,344 | 82.1 | 82.4 | 87.8 | 5.4 |
| SLQA | 1,110 | 72.1 | 72.3 | 75.3 | 3.0 |

Table 11: Full, Local, and Oracle are EM (%); Gap is oracle EM minus the better fixed-branch EM, in percentage points. Oracle scores are upper bounds for the scope selection.

```
Add missing evidence and remove unnecessary
    evidence when
needed.

Question:
{QUESTION}

Reference answer:
{GOLD_ANSWERS}

Previous prediction:
{PREVIOUS_SPEC}

Verifier feedback:
{VERIFIER_FEEDBACK}

Full table:
{TABLE_MARKDOWN}

Return only the corrected JSON object.
```

## E Scope Selection Analysis

### E.1 Oracle Scope Selection

The oracle results in Table 11 indicate clear room for improvement in scope selection. On WikiTQ, oracle selection improves EM from 82.4% to 87.8%, giving a 5.4 point gain over the better fixed branch. On SLQA, the corresponding gain is 3.0 points, from 72.3% to 75.3%. Since Full and Local achieve similar performance on both datasets, the results suggest that the main benefit comes from selecting the appropriate scope for each instance rather than relying on a single fixed scope.

### E.2 Comparison of Scope Selectors

**Setting.** We use GPT-5-mini as the base model, a QA output budget of 4,096 tokens, and $K = 4$ retrieval candidates with refinement. All selectors reuse the same full-table and local answers.

**Routing alternatives.** The fixed type-based selector uses the default policy in Table 9. Direct LLM routing selects between full-table and localized reasoning using the question and table information. The learned router uses logistic regression to predict how likely each scope is to produce a correct answer from question and table features.

| Selector | EM | R-Acc |
|---|---|---|
| Always full | 82.1 | 42.9 |
| Always local | 82.4 | 57.1 |
| Type-based (fixed) | **83.3** | **60.0** |
| Direct LLM | 80.0 | 50.5 |
| Learned router | 80.5 | 53.3 |
| Oracle | 87.8 | 100.0 |

Table 12: Supplementary routing comparison on WikiTQ test questions. EM and routing accuracy (R-Acc) are percentages. R-Acc is evaluated on the questions where branch correctness differs.

It is trained on 500 WikiTQ training questions and fixed before test evaluation, selecting the scope with the higher predicted probability. Always-full and always-local use the same scope for every question. The oracle is correct whenever either scope produces a correct answer.

**Results.** The fixed type-based policy attains the highest observed EM among the evaluated routers, exceeding direct LLM routing and the learned gate by 3.3 and 2.8 percentage points, respectively.