ASTRA: Adaptive Semantic Tree Reasoning Architecture for Complex Table Question Answering

Anonymous ACL submission

Abstract

Table serialization remains a critical bottleneck for Large Language Models (LLMs) in complex table question answering, hindered by challenges such as structural neglect, representation gaps, and reasoning opacity. Existing serialization methods fail to capture explicit hierarchies and lack schema flexibility, while current tree-based approaches suffer from limited semantic adaptability. To address these limitations, we propose **ASTRA** (**A**daptive **S**emantic **T**ree **R**easoning **A**rchitecture) including two main modules, **AdaSTR** and **DuTR**. First, we introduce **AdaSTR**, which leverages the global semantic awareness of LLMs to reconstruct tables into Logical Semantic Trees. This serialization explicitly models hierarchical dependencies and employs an adaptive mechanism to optimize construction strategies based on table scale. Second, building on this structure, we present **DuTR**, a dual-mode reasoning framework that integrates tree-search-based textual navigation for linguistic alignment and symbolic code execution for precise verification. Experiments on complex table benchmarks demonstrate that our method achieves state-of-the-art (SOTA) performance.

1 Introduction

Large Language Models (LLMs) have achieved remarkable success across a wide spectrum of natural language tasks. As a fundamental format for data storage and presentation, tables—particularly complex tables characterized by hierarchical headers and merged cells—are ubiquitous in high-value domains. Consequently, the community has devoted significant efforts to enhance the tabular reasoning and question-answering capabilities of LLMs.

Recent studies (Fang et al., 2024; Sui et al., 2024b) have identified table serialization—the process of converting structured tables into sequential representations compatible with LLM inputs—as a critical bottleneck for Table Question Answering (TableQA). Specifically, we observe that current serialization methods face four major challenges when handling complex tables: (1) **Structural Neglect:** Complex tables often contain intricate layouts with hierarchical relationships and semantic dependencies embedded in their structure, which LLMs frequently overlook. (2) **Representation Gap:** The intrinsic representational mismatch between two-dimensional structured tables and the one-dimensional sequential nature of LLMs hinders the precise localization of fine-grained evidence, which is important for reasoning over complex tables. (3) **Reasoning Opacity:** Existing methods directly employ LLMs for numerical computation over tables, which function as a black box lacking interpretable execution traces, leading to unverified numerical hallucinations. (4) **Schema Inflexibility:** Complex tables exhibit significant structural diversity. Existing table parsing strategies that rely on rigid, rule-based approaches fail to adapt to irregular layouts, limiting generalization.

Despite the variety of existing approaches, none address these challenges simultaneously: First, graph-based approaches like GraphOTTER (Li et al., 2024) atomize tables into triples. While this strategy mitigates schema inflexibility, it still fails to explicitly represent the hierarchical structures and semantic information in complex tables. Conversely, methods such as RelationalCoder (Dong et al., 2025b) attempt to convert complex tables into relational formats. Although these approaches can elucidate semantic associations among headers via schemas and facilitate the integration of Text-to-SQL methodologies (Wang et al., 2024b), they frequently introduce data redundancy or sparsity when applied to asymmetric structures. For instance, a 'Laptop' sub-table may contain a 'GPU' attribute, whereas a 'Mouse' sub-table lacks 'GPU' but includes 'DPI'; forced integration into a unified schema induces data sparsity, while split storage introduces data redundancy. Furthermore, while cur-

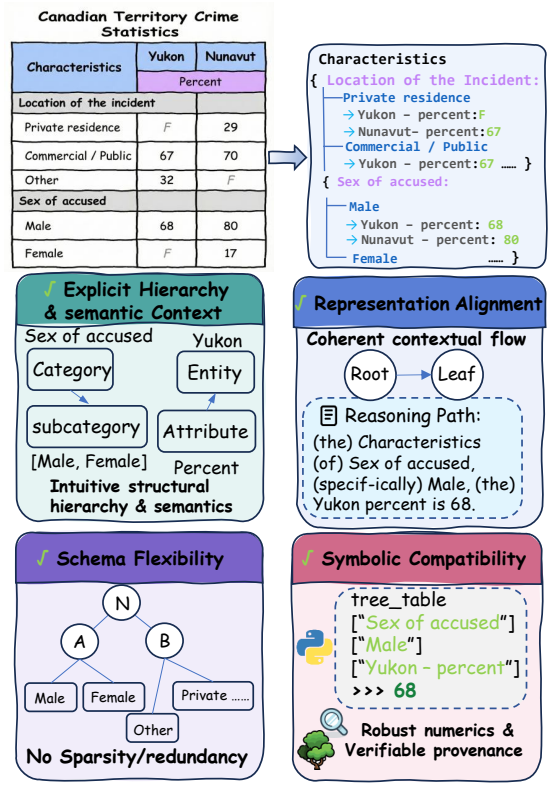

Figure 1: Key desiderata for robust table serialization and their instantiation in AdaSTR.

rent tree-based solutions like ST-Raptor (Tang et al., 2025a) demonstrate the utility of hierarchical representations, their reliance on rule-based construction renders them fragile (details are discussed in Appendix D). Crucially, they fail to discern between physical hierarchy and semantic associations.

To achieve a more effective serialization that enhances LLMs' reasoning capabilities over complex tables, we propose **Ada**ptive **S**emantic **T**ree **R**econstruction (**AdaSTR**), as illustrated in Table 1, which simultaneously addresses the challenges by satisfying the following requirements:

Explicit Hierarchy & Semantic Context: To resolve *Structural Neglect*, we make the serialization preserve the explicit hierarchical structure inherent in table headers (e.g., taxonomic levels like Category $\rightarrow$ Subcategory), while simultaneously revealing the relational implicit semantic dependencies within the data dimension (e.g., Entity $\leftrightarrow$ Attribute mappings). This dual focus enables a more precise representation of the complex table's global semantics.

Representation Alignment: To mitigate *Representation Gap*, we design the serialization to bridge the 2D-to-1D modality gap by expressing tabular content as coherent, natural-language-like sequences that align with the distribution of LLM's pre-training data (Wu et al., 2025). Such representation alignment not only enhances comprehension for LLMs but also facilitates the retrieval of critical content relevant to specific queries.

Method	Explicit Hierarchy	Semantic Context	Representation Alignment	Schema Flexibility	Symbolic Compatibility
Textual Serialization *(Html/MarkDown)*	▲	✗	✗	▲	✗
Triples *(GraphOTTER)*	✗	✗	✗	✓	▲
Relational *(TabFormer)*	▲	✓	▲	✗	✓
Physical Tree *(ST-Raptor)*	✓	✗	▲	✓	✓
Semantic Tree (Ours)	✓	✓	✓	✓	✓

Table 1: Comparison of serialization methods (✓ Support; ▲ Partial; ✗ No Support).

Symbolic Compatibility: To tackle *Reasoning Opacity*, we structure the serialization to support *code-based* symbolic reasoning. This not only mitigates LLMs' weaknesses in numerical computations (Liu et al., 2023b; Zhang et al., 2024a) but also ensures deterministic and verifiable data provenance, allowing users to validate exact data sources.

Schema Flexibility: To overcome *Schema Inflexibility*, we optimize the serialization to avoid introducing irrelevant redundancy that exacerbates the model's inference burden, ensuring robust adaptability across arbitrary table formats.

As illustrated in Figure 1, our serialized semantic tree captures **explicit hierarchy** and **semantic context** through node-to-node relationships. To effectively leverage this structure, we introduce **DuTR** (**Du**al-Mode **T**ree **R**easoning). By incorporating tree-search-based textual reasoning, DuTR efficiently navigates the tree structure to locate critical information, where the resulting paths **with aligned representations** serve as coherent text streams. Furthermore, the tree structure inherently offers **schema flexibility** and can be stored in Python dictionary format to facilitate interpretable **symbolic reasoning**. To balance efficiency and accuracy, we also introduce an adaptive mechanism that dynamically selects tree construction strategies based on **table scale** and **token density**.

We name our unified method **ASTRA** (**A**daptive **S**emantic **T**ree **R**easoning **A**rchitecture), which integrates the adaptive construction module (**AdaSTR**) with the reasoning engine (**DuTR**). Specifically, DuTR treats the tree simultaneously

as a navigable text corpus for semantic retrieval and a structured object for symbolic execution, effectively combining the semantic flexibility of linguistic retrieval with the computational precision of symbolic execution. Overall, our main contributions are as follows:

- We identify four critical challenges that constitute the central bottleneck in complex TableQA, and distill the corresponding desiderata for an effective serialization strategy to address them.
- We introduce AdaSTR, which exploits LLMs' global semantic awareness to transform a table into a semantic-tree representation. Building on this representation, we further propose the DuTR framework, which achieves high accuracy while providing strong interpretability for table QA.
- Driven by adaptive tree construction and hybrid reasoning, ASTRA achieves **SOTA performance** on multiple benchmarks, demonstrating superior generalization and robustness across varied and irregular table formats.

2 Related Work

2.1 Adapting Tabular Data for LLMs

Textual Table Serialization. Methods utilizing Markdown, HTML, or separators (Sui et al., 2024a) serialize tables into token sequences. However, the syntax of these formats deviates from natural linguistic flow, and it is extremely difficult to retrieve relevant information within such structures, hindering the model's ability to understand large tables.

Structure-Aware Model Adaptation. Beyond naive serialization, model-centric approaches explicitly encode structure via specialized embeddings (Korkmaz and Del Rio Chanona, 2024; He et al., 2025), dedicated table encoders (Li et al., 2025), or continued pre-training (Li et al., 2023; Su et al., 2024; Zhang et al., 2024b). However, these methods typically incur significant adaptation overhead when migrating to new backbones.

Intermediate Structural Representations. To enhance table understanding without training, researchers have investigated transforming tables into model-friendly formats (Tang et al., 2025b). Approaches range from relational format (Dong et al., 2025b), which often suffers from data sparsity on asymmetric layouts, to triple-based atomization (Li et al., 2024), which tends to obscure explicit semantic hierarchies. More recently, tree-based methods like ST-Raptor (Tang et al., 2025a) have been proposed; however, their reliance on brittle physical layout heuristics limits their robustness against diverse formatting variations.

2.2 Table Reasoning Paradigms

Table reasoning typically follows two paradigms (Liu et al., 2023b) : Textual reasoning (End-to-End) directly generates natural-language rationales and answers from the serialized table context, but may incur numerical errors (Wei et al., 2023); Symbolic Reasoning (program-aided) generates executable code (e.g., SQL, Python) for precision (Chen et al., 2023). However, it can be sensitive to complex table structures (Details are provided in Appendix E). Our Semantic Tree framework unifies these paradigms by serving as a navigable context for semantic retrieval and a structured object for code generation, effectively combining robustness with precision.

3 Method

The goal of TableQA is to synthesize an answer A for a natural language query Q, grounded in a source table T. While prior benchmarks and methods have largely focused on *flat* tables—typically featuring a single header row where each data cell maps uniquely to one header—we focus on the more challenging task of **Complex Table QA**.

3.1 Method Overview

To address the structural challenges inherent in complex table QA, we formulate the problem as a multi-stage reasoning process consisting of structural transformation and reasoning execution.

General Formulation. First, the raw table T is processed by a representation function τ to yield a model-readable structured view $\tilde{T} = \tau(T)$. We define τ as a generalized transformation function, which abstracts the complex layout into a machine-friendly format. Subsequently, a model $\mathcal{M}_\theta$ serves as the core reasoning engine within a workflow Φ. The model interacts with the transformed view $\tilde{T}$ and the query Q to derive the final answer:

$$A = \Phi_{\mathcal{M}}(\tilde{T}, Q) \tag{1}$$

Method Instantiation. Specifically, our method ASTRA comprises two core phases, mapping specifically to τ and Φ: (1) **AdaSTR ($\tau \rightarrow$ Table Serialization)**, which transforms a complex

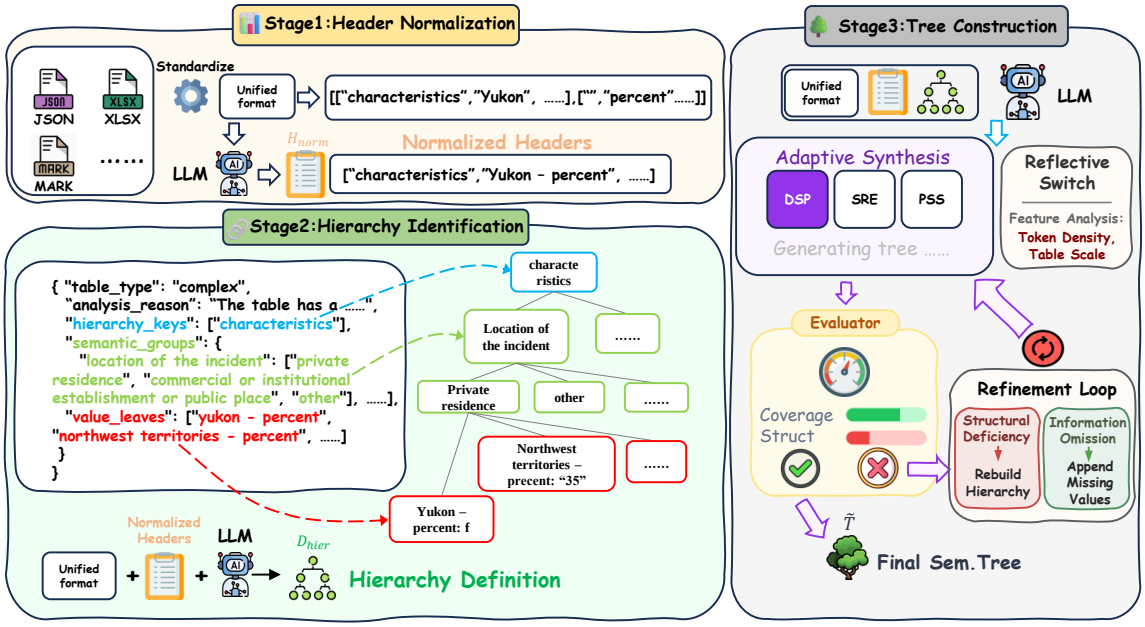

Figure 2: Overview of the Adaptive Semantic Tree Reconstruction process.

table T into a semantic tree $\tilde{T}$; and (2) **DuTR ($\Phi \rightarrow$ Table Reasoning)**, which integrates innovative tree-based textual reasoning and symbolic code execution, culminating in an answer selection module to determine the optimal final answer.

3.2 AdaSTR: Adaptive Semantic Tree Reconstruction

As illustrated in Figure 2, the AdaSTR framework is orchestrated through three interconnected modules designed to ensure structural robustness and semantic fidelity across heterogeneous tables.

3.2.1 Semantic Parsing and Schema Detection

Given a complex table T, we first standardize it into a unified format—a flat ordered list structure l, establishing a reference for the subsequent quality assessment. We then employ a **Header Identification & Normalization (HIN)** module. As shown in Figure 2 (stage 1), raw tables often contain sub-headers like `Percent` that are meaningless without their parent context; **HIN** resolves this by consolidating vertical dependencies into qualified keys (e.g., merging `Yukon` with `Percent` to form `Yukon-Percent`). This yields a normalized header list H_{norm}, which excavates the implicit semantic context, anchoring isolated attributes to their corresponding entities. The subsequent critical phase is **Hierarchy Identification (HID)**. As depicted in Stage 2 in Figure 2, the **HID** module harnesses the LLM to mine Hidden Semantic Groups from H_{norm}. Continuing the previous example, LLM observes that `Yukon-Percent` and `Northwest-Percent` share an identical structure (Region-Metric). Consequently, it abstracts them into a latent high-level concept (e.g., *Regional Statistics*), thereby organizing the extracted semantic units into an explicit hierarchy. Finally, the HID module synthesizes the identified hierarchical structure into a formal representation D_{hier}.

3.2.2 Adaptive Tree Synthesis Strategies

This step transforms the table T into a semantic tree $\tilde{T}$ given D_{hier}. Considering the heterogeneity of real-world tables—ranging from massive, sparse datasets to text-dense complex tables—a monolithic construction strategy cannot effectively balance efficiency and accuracy. To address this, we devise three adaptive Tree construction modes, with the selection strategy in Appendix F.

Direct Semantic Parsing (DSP Mode). This mode is optimized for standard complex tables with moderate structural complexity and manageable size. In this setting, the LLM functions as an end-to-end generator, directly outputting the complete semantic tree $\tilde{T}$ based on the parsed schema D_{hier}.

Symbolic Reference Encoding (SRE Mode). Tailored for verbose-content tables (e.g., financial reports) where generating full textual content is inefficient and risks diluting the model's structural focus, we introduce a Symbolic Placeholder Strategy. Inspired by address-based compressed encodings like SpreadsheetLLM (Dong et al., 2025c),

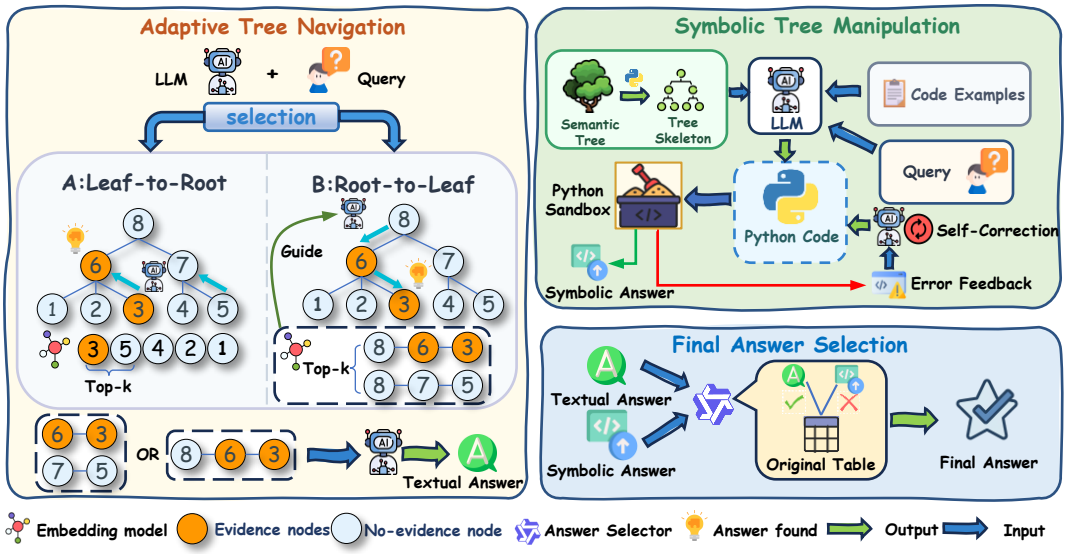

Figure 3: Overview of the Dual-Mode Tree Reasoning process.

we employ coordinate abstraction. Guided by the schema D_{hier} and a coordinate-tagged view T_{grid}, the LLM is instructed to generate *cell addresses* (e.g., A1 notation like `C7`) as placeholders within the tree nodes. This yields a compact tree skeleton. Subsequently, a script retrieves the original text via these coordinates to populate the tree $\tilde{T}$. This approach significantly reduces token overhead while preserving structural precision.

Programmatic Structure Synthesis (PSS Mode). Targeting large-scale tables with repetitive substructures, the LLM leverages D_{hier} and T_{coord} to synthesize a loop-based construction script. This script iterates over the coordinate space to instantiate the full tree $\tilde{T}$, ensuring scalability and efficiency.

3.2.3 Evaluator-Guided Refinement Loop

To mitigate errors, we employ an **Evaluator-Guided Refinement Loop**. The Evaluator validates $\tilde{T}$ via: (1) **Structural Integrity**, which checks path consistency against grid coordinates to prevent logical hallucinations; and (2) **Information Coverage**, which measures the percentage of mapped cells to verify no semantic information is omitted. If the composite score falls below a threshold, the LLM is instructed to iteratively revise $\tilde{T}$ based on feedback for up to maximum iterations. Implementation details are in Appendix F.

3.3 DuTR: Dual-Mode Tree Reasoning

As illustrated in Figure 3, we propose a dual-mode reasoning framework grounded in the semantic tree $\tilde{T}$. This structure serves as a foundation for both symbolic manipulation and context-aware retrieval.

3.3.1 Adaptive Tree Navigation

This module analyzes the query to select an optimal traversal strategy (*Leaf-to-Root* or *Root-to-Leaf*), filtering critical paths for the answer. A key advantage of the semantic tree is that it binds each value to its header-path semantics, transforming fragmented cells into context-rich nodes, enabling reliable semantic retrieval on complex tables.

Strategy A: Leaf-to-Root (Algorithm 2). As illustrated in Panel A of Figure 3, we first filter the top-k_1 leaf subset L_{rel} based on query–leaf semantic similarity. Then, it iteratively expands the context depth d: at each step, paths associated with L_{rel} are merged into the evidence set C. The LLM evaluates whether C is sufficient to derive the answer A; if not, the system increments d to include broader parent context until reaching D_{max}. This strategy excels in **aggregation queries**, such as conditional counting of dispersed data points.

Strategy B: Root-to-Leaf (Algorithm 3). As depicted in Panel B of Figure 3, the system retrieves the top-k_2 root-to-leaf paths P_{topk} via semantic similarity, which serve as global semantic guidance G. We then perform a top-down traversal using a stack S. At each step, the LLM selects a subset of promising child nodes N_{sel} to expand, conditioned on query Q and guidance G. Upon reaching leaf nodes, the path information is accumulated into the evidence set C. The LLM dynamically evaluates whether C is sufficient to derive the answer A; if affirmative, the traversal triggers early stopping,

effectively bypassing the remaining paths. This strategy is well-suited for lookup-style queries requiring precise, localized evidence.

3.3.2 Symbolic Tree Manipulation

Pure textual reasoning is prone to error in multi-step queries involving aggregation or complex logical predicates (Zhang et al., 2024a). Thus, as illustrated in Figure 3 (top-right), we incorporate symbolic reasoning to ensure computational precision. To accommodate the LLM's context window and focus on logical generation, we abstract the semantic tree into a **structural skeleton** devoid of verbose values. Furthermore, we construct targeted **code examples** (e.g., selection, aggregation, and comparison) for tree operations and provide a secure execution environment. If a runtime error occurs, the system enters a **Self-Correction Loop**, feeding error traces back to the LLM to trigger code regeneration. This approach unifies accuracy and stability in complex logical reasoning.

3.4 Final Answer Selection

As shown in Figure 3 (bottom-right), our method yields two candidates: a *Textual Answer* from tree navigation and a *Symbolic Answer* from code execution. To resolve conflicts, we employ a lightweight, open-source LLM as an Answer Selector. When answers differ, it evaluates both against the original table to determine the plausible one. We term this **Adaptive Selection**.

4 Experiment

4.1 Experimental Setup

Datasets. We evaluate on three complex table benchmarks: (1) **AIT-QA** (Katsis et al., 2022): A domain-specific dataset (airline industry) characterized by deeply nested headers. (2) **HiTab** (Cheng et al., 2022): A widely used benchmark containing hierarchical tables extracted from statistical reports, emphasizing numerical aggregation. (3) **SSTQA** (Tang et al., 2025a): A dataset introduced by the ST-Raptor framework, included to assess generalization on semi-structured tables.

Baselines. We compare our method against four categories of baselines: (1) **Foundation Models**: *GPT-4o* (OpenAI et al., 2024), *DeepSeek-V3* (DeepSeek-AI et al., 2025), and a recent reasoning-focused model *OpenAI o3* (OpenAI, 2025). All are prompted via standard textual serialization to benchmark their baseline reasoning performance. (2) **Prompting/Tool-augmented Methods**: including *EEDP* (Srivastava et al., 2025), which relies on structured prompting, and *E5* (Zhang et al., 2024c), which adopts a agentic workflow; (3) **Table-Specific Adapted Models**: *TableGPT2* (Su et al., 2024) and *TableLlama* (Zhang et al., 2024b), which improve tabular reasoning via additional table-centric pretraining or instruction tuning. (4) **Intermediate-Representation Methods**: *GraphOTTER* (Li et al., 2024), which represents tables as triples, and *ST-Raptor* (Tang et al., 2025a), a tree-based baseline.

Metrics. Rigid string-matching metrics (e.g. Exact Match) often fail to capture semantic equivalence in generative tasks. To ensure robust evaluation, we employ LLMs as evaluators. Following recent methodologies validating the reliability of LLM-as-a-Judge frameworks (Zheng et al., 2023; Liu et al., 2023c; Dubois et al., 2025), we utilize GPT-5 as a binary judge: given a prediction and the gold answer, it outputs `CORRECT`/`INCORRECT`. We parse the label to compute Accuracy (Acc).

Implementations. For fairness, we use **DeepSeek-V3-250324** as the backbone for all training-free methods, including our method (AdaSTR and DuTR), (*E5*, *EEDP*), and (*GraphOTTER*, *ST-Raptor*). Details are in Appendix A.

4.2 Main Results

Table 2 demonstrates **ASTRA**'s strong performance and trade-offs between the two reasoning modes (Zhang et al., 2024a). On SSTQA, rich in semantically intensive questions, Textual Reasoning (79.8%) outperforms Symbolic Reasoning (75.3%) by effectively capturing semantic nuances. Conversely, on HiTab, which features numerical aggregation, Symbolic Reasoning (89.3%) surpasses Textual Reasoning (82.2%), validating its computational accuracy. Notably, **ASTRA** (90.1% on HiTab) outperforms **o3** (85.3%), validating the necessity of structural guidance. In contrast, rule-based **ST-Raptor** shows limited generalization on AIT-QA (62.7%) and HiTab (49.0%), exposing its fragility across diverse table layouts.

4.3 Ablation Studies

We conducted ablation studies on **SSTQA** to examine the impact of distinct components on tree reconstruction and reasoning efficacy.

Model	AITQA	SSTQA	HiTab
(1) Foundation Models			
DeepSeek-V3	78.5	63.2	82.0
GPT-4o	80.6	66.4	78.6
o3	89.1	78.2	85.3
(2) Prompting/Tool-augmented Methods			
E5	87.1	70.2	85.1
EEDP	85.6	76.8	79.2
(3) Table-Specific Adapted Models			
TableLlama†	-	40.4	64.7
TableGPT2-72B†	-	-	75.6
(4) Intermediate-Representation Methods			
GraphOTTER	90.4	71.5	88.8
ST-Raptor	62.7	71.1	49.0
(5) ASTRA			
Textual Reasoning	86.1	79.8	82.2
Symbolic Reasoning	87.3	75.3	89.3
Adaptive Selection	**91.6**	**81.9**	**90.1**
Oracle	93.5	86.1	94.1

Table 2: Accuracy (%) on AIT-QA, SST-QA, and HiTab. **Bold** denotes best, underlined denotes second best. † indicates results cited from prior literature. *Oracle* represents the theoretical upper bound via ideal selection between Textual and Symbolic reasoning.

Tree Construction *(DeepSeek-V3)*	Coverage Rate		Struct Acc	
	Avg	Min	Avg	Min
AdaSTR	**0.929**	**0.738**	**0.792**	**0.654**
w/o Evaluator-Guided	0.745	0.473	0.698	0.143
w/o Synthesis Strategies	0.795	0.153	0.643	0.457

Table 3: Ablation of **AdaSTR**. We report Average (Avg) and Minimum (Min) Coverage Rate and Structural Accuracy to evaluate robustness across diverse tables.

Configuration	Acc (%)	Δ
Adaptive Tree Navigation	**79.84**	-
w/o Embedding Model	71.34	-8.50
w/o Dynamic Switching	-	-
Force Root-to-Leaf	77.23	-2.61
Force Leaf-to-Root	75.65	-4.19
Symbolic Tree Manipulation	**75.26**	-
w/o Code Examples	70.42	-4.84
w/o Structural Skeleton	71.73	-3.53
w/o Self-Correction Loop	73.69	-1.57
Impact of Data Representation		
Textual Serialization (Direct Prompting)	63.20	-
Semantic Tree (Direct Prompting)	**70.55**	+7.35

Table 4: Ablation analysis of **DuTR (Deepseek-V3)**. We report Accuracy (%) and performance change (Δ) vs. full configuration per mode. Bottom evaluates representational benefits of the tree structure.

Analysis of AdaSTR. Table 3 presents the ablation results for the tree construction phase. The full **AdaSTR** pipeline achieves the highest performance across all metrics, demonstrating the synergy between the Evaluator-Guided Loop and the synthesis strategies. **(1) Impact of Evaluator-Guided Loop:** Removing the evaluator reduces Average Coverage ($0.929 \rightarrow 0.745$), proving the loop effectively rectifies generation errors. **(2) Impact of Synthesis Strategies:** Disabling the adaptive synthesis strategies (i.e., defaulting to static DSP) leads to a substantial drop in the *Minimum* Coverage Rate (0.153). This underscores that SRE and PSS are vital for handling large-scale complex tables where basic prompting fails.

Remarks on Metric Absolute Values. Results for Structural Accuracy and Coverage are partially influenced by artifacts in dataset annotation and the inherent effects of information filtering, rather than solely model generation errors. Detailed analysis on evaluation scores is provided in Appendix F.

Analysis of DuTR. Table 4 isolates the contributions of specific sub-modules:

(1) Textual Reasoning Ablation: Removing the path guide drops performance to 71.34% (Δ −8.50%), confirming that semantic navigation provides essential global guidance. Furthermore, our dynamic variant (79.84%) outperforms fixed strategies (max 77.23%), validating the necessity of query-adaptive traversal.

(2) Symbolic Reasoning Ablation: Omitting few-shot examples triggers the sharpest decline (70.42%), followed by skeleton removal (71.73%), highlighting format guidance and context filtering as critical drivers for effective program generation.

(3) Intrinsic Advantage of Data Representation. To isolate representation gains, we compared static **Semantic Tree** inputs against **Textual Serialization**. Table 11 (bottom) confirms that even without advanced reasoning, the Static Tree (70.55%) outperforms the Raw Table baseline (63.20%), validating the intrinsic benefit of hierarchical serialization.

4.4 Efficiency Analysis

Table 5 decomposes latency into offline Tree Construction and online QA Inference. Our method ensures superior online efficiency and faster construction than ST-Raptor via PSS mode, avoiding costly element-wise recursion. While GraphOTTER builds faster, its high inference latency proves inefficient for multi-turn sessions. Amortized analysis ($T_{\text{avg}} = T_{\text{tree}}/N + T_{\text{qa}}$) confirms our method

Method	AIT-QA		HiTab		SST-QA	
	Tree	QA	Tree	QA	Tree	QA
ST-Raptor	55.73	31.18	139.61	26.61	233.19	41.36
GraphOTTER	-	19.54	-	22.29	-	21.83
Ours	**29.18**	**7.80**	**43.42**	**6.19**	**93.71**	**11.62**

Table 5: Comparison of time efficiency (in seconds) across different datasets. We record the latency for both the Tree Construction phase (Tree) and the Question Answering phase (QA). "-" denotes the method does not involve a tree construction phase.

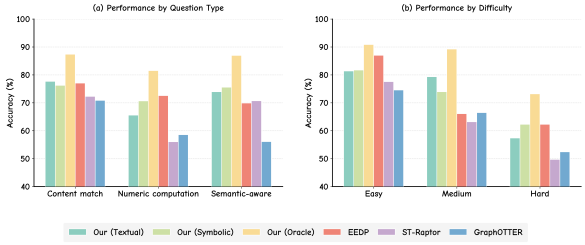

Figure 4: Performance breakdown by question type and difficulty for DuTR and baselines.

surpasses GraphOTTER when query count $N \geq 3$ (AIT-QA/HiTab) or $N \geq 10$ (SSTQA), offering an optimal accuracy-efficiency trade-off.

4.5 Diagnostic Analysis

We conduct a fine-grained analysis on SSTQA. (1) **Complementarity of Reasoning Modes.** Figure 4 (a) confirms the synergy of Dual-Mode design. *Textual Reasoning* excels in **Semantic** tasks, while *Symbolic Reasoning* dominates **Numeric Computation**. Conversely, GraphOTTER struggles in semantic categories due to structural and semantic loss, a limitation resolved by our Semantic Tree. (2) **Resilience to Query Complexity.** Figure 4 (b) highlights robustness against increasing difficulty. While all models degrade on *Hard* queries, our method maintains stability via synergistic textual-symbolic reasoning.

4.6 Case Study

To intuitively demonstrate how ASTRA addresses the serialization bottlenecks, we present a case on **Structural Enumeration** in Figure 5. The query requires enumerating all children under the parent category "Fixed Expenses". GraphOTTER fail due to the **Representation Gap**; its triple-based format isolates cells as discrete entities, making it unable to capture the structural dependency of aggregation rows. ST-Raptor exhibits **Structural Neglect**, erroneously regarding the parent header "Fixed Expenses" as an atomic data point. EEDP suffers from **Reasoning Opacity**; by relying on purely textual

Case Study: Enumeration on Complex Hierarchy

Query: *"How many items are listed in Fixed Expenses?"*
Gold Answer: 9

Table Snippet (Semantic Tree Representation):

```
Manufacturing Overhead Budget
├ Variable Overhead // Sibling Branch
|  ├ Indirect Labor ..............  ...
|  └ ...  (4 items omitted) ...
└ Fixed Expenses // Target Branch
├ Quarter 1
|  ├ Repair ......................  2,500
|  ├ ...  (8 standard items) ...
|  └ Cash Output .................  16,635
└ ...  (Quarter 2-4 omitted) ...
```

Baselines (Failure Analysis)

- **GraphOTTER (Output: 5):** *Representation Gap.* Retrieves only standard items (e.g., "Repair") but misses structurally bound rows (e.g., "Total", "Less").
- **ST-Raptor (Output: 1):** *Structural Neglect.* Overlooks the implicit parent-child hierarchy, treating the header "Fixed Expenses" as a single data point rather than a container.
- **EEDP (Output: 5):** *Reasoning Opacity.* Relies on textual commonsense to identify "expenses" during CoT generation, ignoring structural aggregations (e.g., "Total") that don't look like standard items.

ASTRA (Ours)

- **Logic (Symbolic Tree Manipulation):**

```
target = tree['Manuf.  Budget']['Fixed
Expenses']['Q1']
Answer = len(target.keys())
```

- **Result: Correct (9).** The dictionary structure strictly enforces parent-child containment, ensuring all irregular items are counted.

Figure 5: A representative case study involving sibling branches and irregular children. **ASTRA** utilizes precise dictionary traversal to correctly enumerate all items.

inference, it introduces *semantic bias*—prioritizing the model's parametric internal knowledge over the table's explicit structural constraints—thereby hallucinating filters that exclude valid but irregular items. In contrast, ASTRA leverages its Semantic Tree representation to preserve explicit parent-child relationships. By navigating the tree structure, it successfully retrieves the complete subtree, demonstrating the critical need for **Explicit Hierarchy** and **Semantic Context** when handling complex nested headers.

5 Conclusion

We present ASTRA, a training-free method for semantic tree reconstruction that introduces a Text-Symbolic Reasoning paradigm to enhance tabular QA. By restoring explicit hierarchy, our approach achieves SOTA performance, outperforming advanced reasoning models like OpenAI o3. Notably, we find that the Semantic Tree representation alone—even without specialized reasoning mechanisms—yields superior performance compared to textual serializations. These findings offer a critical insight: an explicit hierarchy enriched with semantic context is critical for unlocking the full reasoning potential of LLMs.

Limitations

While ASTRA achieves state-of-the-art performance on complex table benchmarks, there remain avenues for future exploration tied to the inherent challenges of the task setting. First, regarding efficiency trade-offs: Our Semantic Tree construction is specifically optimized to disentangle the intricate dependencies in complex, hierarchical tables. For extremely simple flat tables where explicit hierarchy is absent, this reconstruction process may incur a computational overhead compared to direct textual serialization. Second, ASTRA primarily relies on textual and structural parsing. However, real-world complex tables often utilize visual cues (e.g., background colors, bold fonts) to convey implicit semantic constraints. Incorporating multi-modal vision encoders to capture these stylistic features remains a promising direction.

Ethics Statement

In this paper, we introduce ASTRA, a framework for complex table question answering. Our experimental evaluation is conducted using publicly available and widely recognized benchmarks, including AIT-QA, HiTab, and SSTQA. These datasets are constructed from open-domain sources (e.g., Wikipedia, public financial reports) and, to the best of our knowledge, do not contain any personally identifiable information or offensive content. Furthermore, all Large Language Models employed in this work were utilized in strict adherence to their respective usage policies and safety guidelines. We believe that our work does not pose any significant ethical concerns.

References

Mubashara Akhtar, Abhilash Shankarampeta, Vivek Gupta, Arpit Patil, Oana Cocarascu, and Elena Simperl. 2023. Exploring the numerical reasoning capabilities of language models: A comprehensive analysis on tabular data. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 15391–15405, Singapore. Association for Computational Linguistics.

Lang Cao and Hanbing Liu. 2025. Tablemaster: A recipe to advance table understanding with language models. *Preprint*, arXiv:2501.19378.

Wenhu Chen. 2023. Large language models are few(1)-shot table reasoners. *Preprint*, arXiv:2210.06710.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Preprint*, arXiv:2211.12588.

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. *Preprint*, arXiv:2108.06712.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, and 181 others. 2025. Deepseek-v3 technical report. *Preprint*, arXiv:2412.19437.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms. *Preprint*, arXiv:2305.14314.

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*, SIGIR '25, page 1382–1391, New York, NY, USA. Association for Computing Machinery.

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.

Haoyu Dong, Jianbo Zhao, Yuzhang Tian, Junyu Xiong, Shiyu Xia, Mengyu Zhou, Yun Lin, José Cambronero, Yeye He, Shi Han, and Dongmei Zhang. 2025c. Spreadsheetllm: Encoding spreadsheets for large language models. *Preprint*, arXiv:2407.09025.

Yann Dubois, Balázs Galambosi, Percy Liang, and Tatsunori B. Hashimoto. 2025. Length-controlled alpacaeval: A simple way to debias automatic evaluators. *Preprint*, arXiv:2404.04475.

Xi Fang, Weijie Xu, Fiona Anting Tan, Jiani Zhang, Ziqing Hu, Yanjun Qi, Scott Nickleach, Diego Socolinsky, Srinivasan Sengamedu, and Christos Faloutsos. 2024. Large language models(llms) on tabular data: Prediction, generation, and understanding – a survey. *Preprint*, arXiv:2402.17944.

Xinyi He, Yihao Liu, Mengyu Zhou, Yeye He, Haoyu Dong, Shi Han, Zejian Yuan, and Dongmei Zhang. 2025. Tablelora: Low-rank adaptation on table structure understanding for large language models. *Preprint*, arXiv:2503.04396.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. 2024. Ruler: What's the real context size of your long-context language models? *Preprint*, arXiv:2404.06654.

Yannis Katsis, Saneem Chemmengath, Vishwajeet Kumar, Samarth Bharadwaj, Mustafa Canim, Michael Glass, Alfio Gliozzo, Feifei Pan, Jaydeep Sen, Karthik Sankaranarayanan, and Soumen Chakrabarti. 2022. AIT-QA: Question answering dataset over complex tables in the airline industry. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Industry Track*, pages 305–314, Hybrid: Seattle, Washington + Online. Association for Computational Linguistics.

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 28270–28296, Suzhou, China. Association for Computational Linguistics.

Buse Sibel Korkmaz and Antonio Del Rio Chanona. 2024. Integrating table representations into large language models for improved scholarly document comprehension. In *Proceedings of the Fourth Workshop on Scholarly Document Processing (SDP 2024)*, pages 293–306, Bangkok, Thailand. Association for Computational Linguistics.

Liyao Li, Chao Ye, Wentao Ye, Yifei Sun, Zhe Jiang, Haobo Wang, Jiaming Tian, Yiming Zhang, Ningtao Wang, Xing Fu, Gang Chen, and Junbo Zhao. 2025. Table as a modality for large language models. *Preprint*, arXiv:2512.00947.

Peng Li, Yeye He, Dror Yashar, Weiwei Cui, Song Ge, Haidong Zhang, Danielle Rifinski Fainman, Dongmei Zhang, and Surajit Chaudhuri. 2023. Table-gpt: Table-tuned gpt for diverse table tasks. *Preprint*, arXiv:2310.09263.

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. *Preprint*, arXiv:2412.01230.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2023a. Lost in the middle: How language models use long contexts. *Preprint*, arXiv:2307.03172.

Tianyang Liu, Fei Wang, and Muhao Chen. 2023b. Rethinking tabular data understanding with large language models. *Preprint*, arXiv:2312.16702.

Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. 2023c. G-eval: NLG evaluation using gpt-4 with better human alignment. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 2511–2522, Singapore. Association for Computational Linguistics.

Xinyuan Lu, Liangming Pan, Yubo Ma, Preslav Nakov, and Min-Yen Kan. 2025. Tart: An open-source tool-augmented framework for explainable table-based reasoning. *Preprint*, arXiv:2409.11724.

Qingyang Mao, Qi Liu, Zhi Li, Mingyue Cheng, Zheng Zhang, and Rui Li. 2025. Potable: Towards systematic thinking via stage-oriented plan-then-execute reasoning on tables. *Preprint*, arXiv:2412.04272.

Md Mahadi Hasan Nahid and Davood Rafiei. 2025. Normtab: Improving symbolic reasoning in llms through tabular data normalization. *Preprint*, arXiv:2406.17961.

OpenAI. 2025. Openai o3 and o4-mini system card. Technical report, OpenAI.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, and 262 others. 2024. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

Pragya Srivastava, Manuj Malik, Vivek Gupta, Tanuja Ganu, and Dan Roth. 2025. Evaluating llms' mathematical reasoning in financial document question answering. *Preprint*, arXiv:2402.11194.

Aofeng Su, Aowen Wang, Chao Ye, Chen Zhou, Ga Zhang, Gang Chen, Guangcheng Zhu, Haobo Wang, Haokai Xu, Hao Chen, Haoze Li, Haoxuan Lan, Jiaming Tian, Jing Yuan, Junbo Zhao, Junlin Zhou, Kaizhe Shou, Liangyu Zha, Lin Long, and 14 others. 2024. Tablegpt2: A large multimodal model with tabular data integration. *Preprint*, arXiv:2411.02059.

Yuan Sui, Mengyu Zhou, Mingjie Zhou, Shi Han, and Dongmei Zhang. 2024a. Table meets llm: Can large language models understand structured table data? a benchmark and empirical study. *Preprint*, arXiv:2305.13062.

Yuan Sui, Jiaru Zou, Mengyu Zhou, Xinyi He, Lun Du, Shi Han, and Dongmei Zhang. 2024b. TAP4LLM: Table provider on sampling, augmenting, and packing semi-structured data for large language model reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 10306–10323, Miami, Florida, USA. Association for Computational Linguistics.

Zirui Tang, Boyu Niu, Xuanhe Zhou, Boxiu Li, Wei Zhou, Jiannan Wang, Guoliang Li, Xinyi Zhang, and Fan Wu. 2025a. St-raptor: Llm-powered semi-structured table question answering. *Preprint*, arXiv:2508.18190.

Zirui Tang, Weizheng Wang, Zihang Zhou, Yang Jiao, Bangrui Xu, Boyu Niu, Dayou Zhou, Xuanhe Zhou, Guoliang Li, Yeye He, Wei Zhou, Yitong Song, Cheng Tan, Xue Yang, Chunwei Liu, Bin Wang, Conghui He, Xiaoyang Wang, and Fan Wu. 2025b. Llm/agent-as-data-analyst: A survey. *Preprint*, arXiv:2509.23988.

Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. 2024a. Multilingual e5 text embeddings: A technical report. *Preprint*, arXiv:2402.05672.

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

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models. *Preprint*, arXiv:2201.11903.

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

Zhen Yang, Ziwei Du, Minghan Zhang, Wei Du, Jie Chen, Fulan Qian, and Shu Zhao. 2025. Causality meets the table: Debiasing LLMs for faithful tableQA via front-door intervention. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Siyue Zhang, Anh Tuan Luu, and Chen Zhao. 2024a. Syntqa: Synergistic table-based question answering via mixture of text-to-sql and e2e tqa. *Preprint*, arXiv:2409.16682.

Tianshu Zhang, Xiang Yue, Yifei Li, and Huan Sun. 2024b. Tablellama: Towards open large generalist models for tables. *Preprint*, arXiv:2311.09206.

Xuanliang Zhang, Dingzirui Wang, Keyan Xu, Qingfu Zhu, and Wanxiang Che. 2025. Rot: Enhancing table reasoning with iterative row-wise traversals. *Preprint*, arXiv:2505.15110.

Yunjia Zhang, Jordan Henkel, Avrilia Floratou, Joyce Cahoon, Shaleen Deep, and Jignesh M. Patel. 2023. Reactable: Enhancing react for table question answering. *Preprint*, arXiv:2310.00815.

Zhehao Zhang, Yan Gao, and Jian-Guang Lou. 2024c. e^5: 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.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Preprint*, arXiv:2306.05685.

A Implementation Details

All experiments were conducted on Ubuntu 20.04.1 LTS servers equipped with two NVIDIA A100 GPUs. We accessed large-scale open-source and proprietary models via respective APIs. For smaller models, specifically Qwen3-8B, Qwen2.5-7B, InternVL2.5-26B, and Llama-3.1-8B, we deployed them locally on the servers.

A.1 Implementation of Baselines

All baseline methods were implemented using the official codebases provided by the original authors, adhering to the default hyperparameter settings unless otherwise noted. Specific adaptations are detailed below:

(1) **EEDP** (Srivastava et al., 2025): Due to the absence of official source code, we reproduced the method relying strictly on the prompts documented in the original paper.

(2) **Foundation Model**: We utilize standard **textual serialization** (e.g., HTML or Markdown) to represent tabular data, evaluating the models' intrinsic reasoning capabilities without structural augmentation. The detailed prompt templates used for these baselines are provided in Listing 4.

A.2 Implementation of Our method

Algorithm 1 Adaptive Tree Navigation

Require: Tree Structure $\mathcal{T}$, Question Q
Ensure: Evidence Context $\mathcal{C}$, Answer A

```
Preprocess T (normalize values)
d ← DETERMINEDIR(T, Q)
if d = "Root2Leaf" then
    (C, A) ← ROOT2LEAF(T, Q)
else
    (C, A) ← LEAF2ROOT(T, Q)
end if
return (C, A)
```

AdaSTR. We utilized *DeepSeek-V3* for the tree construction phase. To ensure deterministic outputs, the temperature was set to 0. For the validation loop, we set the acceptance thresholds for Information Coverage and Structural Integrity to 80% and 70%, respectively, with a maximum of 3 reconstruction attempts. If none of the attempts satisfies both thresholds within three trials, we select the candidate from the attempted reconstructions that achieves the highest average score across the two metrics. Upon completion, the constructed

trees were cached in JSON format to facilitate efficient retrieval in downstream tasks. Specific details regarding the adaptive strategy selection policy are provided in Appendix F.

DuTR. For the question-answering phase within our proposed framework (Algorithm 1), we employed *DeepSeek-V3* with the temperature set to 0.3. In the textual reasoning stage (detailed in Algorithm 3 and Algorithm 2), *Multilingual-E5-Large* (Wang et al., 2024a) served as the embedding model to support path retrieval and leaf ranking. Regarding retrieval hyperparameters, we set $k_2 = 5$ for the Root2Leaf strategy, while adopting a larger $k_1 = 50$ for Leaf2Root to ensure comprehensive coverage of candidate leaf nodes.

To determine the final answer from the candidate outputs (Textual vs. Symbolic), we employed a locally deployed lightweight model, **Qwen3-8B**, as the answer selector. This approach is highly efficient, introducing negligible computational overhead; statistical analysis shows an average execution time of only **0.42s** for AIT-QA, **0.51s** for HiTab, and **0.63s** for SSTQA per query. The prompt used for this adaptive selection is as follows:

```
You are a rigorous data verification
assistant specializing in complex
JSON tree-structured table data.
Your task is to determine which
candidate answer is correct based on
the provided table facts.
## Table:
{table}
## Question:
{question}
## Candidate Answers:
Answer A: {answerA}
Answer B: {answerB}
Analysis Steps:
1. Locate relevant data within the
table.
2. Compare the consistency of both
answers against the table data.
Output Requirements:
Do not output any explanations,
punctuation, or analysis processes.
Strictly output ONLY a single
character: "A" or "B".
The Correct Answer:
```

We also conduct preliminary explorations on improving the selector; details are provided in Appendix G.

A.3 Implementation of Evaluation Metrics

Following recent best practices validating the reliability of automated evaluation (Zheng et al., 2023; Liu et al., 2023c), we adopted an **LLM-as-a-Judge** paradigm to assess answer accuracy, utilizing *GPT-5* with a temperature of 0.3.To explicitly validate the reliability of this automated metric, we conducted a **manual verification** on a subset of 200 queries randomly sampled from the **SSTQA dataset**. Since our framework generates two candidate answers per query (derived from *Textual* and *Symbolic* reasoning modes, respectively), this validation set comprised a total of 400 generated answers. We manually reviewed these instances to establish ground-truth correctness. As shown in our analysis, the automated judge demonstrated an exceptionally high alignment with human decision-making, achieving an agreement rate of **98.25%** $(393/400)$. Notably, the discrepancies were minimal and balanced, comprising only 3 false positives (judged correct but actually incorrect) and 4 false negatives (judged incorrect but actually correct), indicating no significant bias in the evaluator. Given this near-perfect consistency, we proceeded with the automated evaluation for the full dataset. The exact prompt used for judgment is:

```
Judge Prompt:
Question: {$question}
Correct Answer: {$ground_truth}
Predicted Answer: {$prediction}
Please judge whether the predicted
answer is correct. If the answer
involves numerical values, slight
numerical errors are allowed.
Please answer only "Correct" or
"Incorrect".
```

B Baseline Descriptions

In this section, we provide detailed descriptions of the baseline methods used in our experiments:

B.1 Prompting and Tool-augmented Methods

- **E5** (Zhang et al., 2024c) is a zero-shot, code-augmented framework specifically designed for hierarchical table analysis. It operates via a five-step pipeline: *Explain* the table structure, *Extract* information via code generation, *Execute* the code to ensure accuracy, *Exhibit* the results, and *Extrapolate* to derive the final answer. It effectively addresses the challenges of complex table structures and implicit semantics without requiring fine-tuning.

- **EEDP** (Srivastava et al., 2025) is a Chain-of-Thought (CoT) prompting strategy tailored for reasoning over semi-structured financial documents. It guides the LLM through four logical steps: *Elicit* domain knowledge, *Extract* relevant evidence (rows/numbers), *Decompose* the problem into atomic calculations, and *Predict* the final answer.

B.2 Table-Specific Adapted Models

- **TableLlama** (Zhang et al., 2024b) is an open-source LLM fine-tuned on TableInstruct, a large-scale dataset comprising diverse table tasks. It treats tables as serialized text sequences and utilizes LongLoRA to efficiently handle the long context windows typically required for processing large tables.

- **TableGPT2** (Su et al., 2024) is a 72B-parameter multimodal model that treats tables as a distinct modality. It integrates a dedicated table encoder with the LLM decoder and is pre-trained on massive table-text pairs, allowing it to capture schema-level structural information and perform robust numerical reasoning.

B.3 Intermediate-Representation Methods

- **GraphOTTER** (Li et al., 2024) addresses complex layouts by converting tables into an undirected graph representation, where cells are nodes connected by spatial edges. It employs an LLM-driven agent to iteratively traverse this graph and perform reasoning, effectively handling merged cells and headers.

- **ST-Raptor** (Tang et al., 2025a) models semi-structured tables as Hierarchical Orthogonal Trees (HO-Trees) to explicitly capture nesting relationships. It resolves queries by decomposing them into atomic tree operations (e.g., retrieval, aggregation) which are executed deterministically against the structured tree representation.

C Dataset Details

We evaluate on three public table QA benchmarks (Table 6). Below we summarize their characteristics.

HiTab (test split). HiTab is an open-domain benchmark focusing on *hierarchical tables*, where multi-level row/column headers and implicit structure make numerical reasoning and indexing challenging. The tables are collected from real statistical reports and Wikipedia, and the QA pairs are constructed by rewriting analyst-authored descriptive sentences into questions, which helps preserve real-world analytical intent. The released annotations include fine-grained entity/quantity alignments; each QA instance additionally records its aggregation operator and may provide an explicit answer formula. In our setting, we report results on the HiTab **test set** (538 tables / 1,584 QA pairs as in Table 6).

Dataset	# Tables	# QAs	Domain
HiTab	538	1,584	Open domain
AIT-QA	80	367	Airline industry
SSTQA	102	764	Open domain

Table 6: **Dataset Statistics.** Details of the test sets used in our experiments. SSTQA specifically targets real-world semi-structured tables.

AIT-QA. AIT-QA is a domain-specific benchmark built from tables in public U.S. SEC filings of major airline companies (fiscal years 2017–2019). Compared with Wikipedia-style flat tables, AIT-QA tables often have more complex layouts, including hierarchical headers and domain-specific terminology. The dataset also provides metadata/annotations indicating whether a question requires hierarchical header understanding, involves domain terminology, or is a paraphrase. In our experiments we follow the common evaluation subset reported in Table 6 (80 tables / 367 QA pairs).

SSTQA. SSTQA targets *real-world semi-structured tables* (e.g., spreadsheet-like layouts) that contain irregular structures such as merged cells, multi-row/column headers, nested regions (subtables), and flexible layouts. It contains 764 questions over 102 curated tables, selected from a larger pool of real-world tables to ensure diverse structures and broad scenario coverage (e.g., administration, finance, HR, schedules, etc.). This benchmark is designed to stress layout-aware reasoning beyond the assumptions of fully structured (relational) tables.

Algorithm 2 Leaf-to-Root Reasoning

Require: Tree Structure $\mathcal{T}$, Question Q
Ensure: Evidence $\mathcal{C}$, Answer A

$\mathcal{L} \leftarrow \text{GETALLLEAVES}(\mathcal{T})$
if using embedding **then**
  $\mathcal{L} \leftarrow \text{EMBSORT}(\mathcal{L}, Q)$ {Rank leaf nodes by relevance}
end if
$\mathcal{L}_{rel} \leftarrow \text{LLM}_{\text{filter}}(\mathcal{L}, Q)$ {Filter irrelevant leaves}
$k \leftarrow 0, K_{max} \leftarrow 5$
while $k \leq K_{max}$ **do**
  $\mathcal{C}_k \leftarrow \emptyset$
  for $(p_{leaf}, v) \in \mathcal{L}_{rel}$ **do**
    $p_{sub} \leftarrow p_{leaf}[: |p_{leaf}| - k]$ {Prune path upward by depth k}
    $v_{sub} \leftarrow \text{GETDATA}(\mathcal{T}, p_{sub})$
    $\mathcal{C}_k \leftarrow \mathcal{C}_k \cup \{(p_{sub}, v_{sub})\}$
  end for
  $\mathcal{C} \leftarrow \text{MERGE}(\mathcal{C}_k)$ {Merge overlapping paths}
  $(ready, A) \leftarrow \text{LLM}_{\text{check}}(Q, \mathcal{C})$
  if $ready$ **then**
    return $(\mathcal{C}, A)$
  end if
  $k \leftarrow k + 1$ {Increase depth to expand context}
end while
$A \leftarrow \text{LLM}_{\text{gen}}(Q, \mathcal{C})$ {Fallback generation}
return $(\mathcal{C}, A)$

Algorithm 3 Root-to-Leaf Reasoning

Require: Tree Structure $\mathcal{T}$, Question Q
Ensure: Evidence $\mathcal{C}$, Answer A

$S \leftarrow [(\text{root}, \mathcal{T}, [\text{root}])]$ {Init Stack}
$\mathcal{C} \leftarrow \emptyset$ {Accumulated Evidence Context}
$\mathcal{G} \leftarrow \emptyset$ {Global Semantic Guidance}
if using embedding **then**
  $\mathcal{P}_{topk} \leftarrow \text{EMBRANK}(\mathcal{T}, Q)$ {Retrieve top-k relevant paths}
  $\mathcal{G} \leftarrow \text{EXTRACTROOTS}(\mathcal{P}_{topk})$
end if
while $S \neq \emptyset$ **do**
  $(n, v, p_{curr}) \leftarrow \text{POP}(S)$
  if $\text{ISLEAF}(n)$ **then**
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{(p_{curr}, v)\}$
    $(ready, A_{temp}) \leftarrow \text{LLM}_{\text{check}}(Q, \mathcal{C})$
    if $ready$ **then**
      return $(\mathcal{C}, A_{temp})$
    end if
  else
    $\mathcal{N}_{child} \leftarrow \text{GETCHILDREN}(v)$
    $\mathcal{N}_{sel} \leftarrow \text{LLM}_{\text{select}}(n, \mathcal{N}_{child}, Q, \mathcal{G})$
    for $c \in \mathcal{N}_{sel}$ **do**
      $\text{PUSH}(S, (c, \mathcal{N}_{child}[c], p_{curr} \cup \{c\}))$
    end for
  end if
end while
$A \leftarrow \text{LLM}_{\text{gen}}(\mathcal{C}, Q)$
return $(\mathcal{C}, A)$

D Detailed Comparison between ST-Raptor and AdaSTR

This appendix elucidates the key distinctions between ST-Raptor (Tang et al., 2025a) and our proposed AdaSTR framework through a concrete case study. ST-Raptor employs a rule-based, open-loop approach to construct hierarchical trees (HO-Trees) primarily reliant on physical layout heuristics, such as cell merging annotations and positional alignment. While effective for simple layouts, this method introduces vulnerabilities in complex tables where semantic dependencies may not align with a predefined rule, leading to local bias and error propagation from initial detections.

In contrast, AdaSTR reframes table parsing as a closed-loop "Semantic Reconstruction" task, leveraging LLMs' global context-awareness to infer latent logical subordination directly from content. This results in a more robust semantic tree where nodes represent subjects and attributes, edges denote logical relationships (e.g., subordination, qualifications), and leaves hold data values. The adaptive mechanism in AdaSTR further ensures scalability by dynamically selecting construction strategies based on table characteristics, with feedback loops for quality assurance. To illustrate, we use a real-world financial table: an "Operating Expenses Analysis" breakdown comparing unit costs year-over-year (2019 vs. 2018), including per ASM changes and percent changes. The table features merged headers (e.g., "Year ended December 31," spanning 2019 and 2018 columns) and irregular alignments, making it representative of complex, semi-structured data in finance domains.

Table description. The input is an HTML-rendered table (visualized in Figure 6). Key structural elements include: A top-level header "Year ended December 31," merged across two sub-columns (2019 and 2018). Separate columns for

"Per ASM Change" and "Percent Change." Rows for categories like "Salaries, wages, and benefits," with values in cents (¢) or plain numbers, and parentheses indicating decreases.

ST-Raptor: geometry-driven HO-Tree leads to semantic detachment. Listing 1 shows the output of ST-Raptor. We highlight several concrete failure modes: (i) **Unanchored header node:** a placeholder node (`"None"`) is created for {2019, 2018, change} without binding them to the body cells, indicating the header hierarchy is not grounded to the data region. (ii) **Positional keys instead of semantic attributes:** for most rows, columns are indexed by `"0","1","2","3"`, which removes column semantics and prevents deterministic querying. (iii) **Value-as-key corruption:** for *Depreciation and amortization*, numerical values (`0.78, 0.75, 0.03, 4.0`) are mistakenly promoted to keys with empty values, suggesting that local alignment cues override global header constraints once header-cell mapping fails. (iv) **Schema inconsistency:** the *Total* row uses `value1..value4`, a schema incompatible with other rows, reflecting the lack of a global consistency check. These errors are consistent with ST-Raptor's open-loop, rule-based construction that assumes physical layout reliably reflects logical hierarchy; once header detection is imperfect, the error cascades irreversibly.

AdaSTR: semantic reconstruction leads to coherent structural alignment. Listing 2 shows AdaSTR's reconstructed semantic tree. AdaSTR represents each row subject (*Category*) as a node, and attaches fully-qualified attributes derived from the header path, e.g., `Year ended December 31, - 2019` and `Year ended December 31, - 2018`. Crucially, repeated headers are disambiguated by parent context (`Per ASM - change` vs. `Percent - change`). All categories share a consistent attribute set, making the representation directly consumable by symbolic querying and program synthesis.

Takeaway. This case illustrates that AdaSTR better satisfies the four desiderata for complex table serialization: (1) **Explicit hierarchy & semantic context** via header-path attributes; (2) **Representation Alignment** by enabling natural-language-like statements such as "(For) Fuel and oil, Year ended December 31 2019 (is) 2.76; ..."; (3) **Schema flexibility** since attributes are attached per subject node without forcing a rigid relational schema; (4) **Symbolic compatibility** because the tree can be deterministically converted to key-value records or executable queries.

E Comprehensive Taxonomy of Table Reasoning Methods

Table question answering (TableQA) in the LLM era can be understood as a pipeline of two design choices: (i) how the table is *represented* to the model, (ii) how the model *reasons* over that representation. Following recent TableQA literature (Liu et al., 2023b; Zhang et al., 2024a), we summarize two major paradigms—**textual (end-to-end) reasoning** and **symbolic (program-aided) reasoning**—and highlight **hybrid** variants that combine the strengths of both.

Notation. We denote a table as T with cells $T[r, c]$, headers $\mathcal{H}$ (flat or multi-level), and a natural language question q. A TableQA system outputs an answer $\hat{a}$ and optionally a rationale $\hat{y}$ or an executable program $\hat{p}$ such that $\hat{a} = \texttt{Exec}(\hat{p}, T)$.

1) Textual reasoning (End-to-End approaches). Textual reasoning treats TableQA as conditional generation:

$$(\hat{y}, \hat{a}) = \texttt{LLM}\big(q \,\|\, \texttt{Serialize}(T)\big), \quad (2)$$

where $\texttt{Serialize}(\cdot)$ linearizes the table into a token sequence (e.g., Markdown/CSV/TSV, row-wise templates, or hierarchical header strings). The LLM produces a natural-language reasoning trace $\hat{y}$ (optional) and the final answer $\hat{a}$ directly. In practice, this paradigm is often implemented via direct prompting, chain-of-thought (CoT), self-consistency, or decomposition prompts (Wei et al., 2023; Chen, 2023). Recent work emphasizes that careful *table provisioning*—selecting relevant rows/columns, augmenting metadata, and packing contexts—can substantially improve end-to-end reasoning accuracy under context limits (Sui et al., 2024b). More recently, *Row-of-Thought* (RoT) prompting structures the generation as iterative *row-wise traversals* with intermediate reflection/refinement, effectively scaling reasoning length while keeping the model grounded in the table and reducing table-content hallucinations in a training-free manner (Zhang et al., 2025).

Strengths. Textual reasoning is semantically flexible: it can handle vague questions, implicit references, and multi-hop explanations without requiring a fixed operator set. It is also easy to deploy

Operating Expenses Analysis

Unit cost breakdown (Year over Year)

Source: Internal Report

Category	Year ended December 31,		Per ASM	Percent
	2019	2018	Change	Change
Salaries, wages, and benefits	5.27¢	4.79¢	**0.48¢**	10.0 %
Fuel and oil	2.76	2.89	**(0.13)**	(4.5)
Maintenance materials and repairs	0.78	0.69	**0.09**	13.0
Landing fees and airport rentals	0.87	0.83	**0.04**	4.8
Depreciation and amortization	0.78	0.75	**0.03**	4.0
Other operating expenses	1.92	1.79	**0.13**	7.3
Total	**12.38¢**	**11.74¢**	**0.64¢**	**5.5 %**

* Note: Values in parentheses indicate a decrease or negative value.

Figure 6: Visualization of the Operating Expenses Analysis table.

(no executor/tooling) and can be robust to minor schema variations because the model reasons over natural language rather than strict schemas.

Failure modes. Despite strong semantic understanding, end-to-end generation may suffer from:

- **Numerical and arithmetic errors**: LLMs may perform approximate calculation or lose track of intermediate values, especially for multi-step aggregation and comparison (Wei et al., 2023; Akhtar et al., 2023).
- **Faithfulness issues**: rationales may not correspond to actual table evidence (hallucinated), and answers can be sensitive to serialization order or irrelevant rows (Yang et al., 2025).
- **Context pressure**: Large or wide tables require truncation or retrieval to fit context windows (Wang et al., 2024b). However, effective retrieval is inherently difficult over discrete table units (e.g., isolated cells or triples).
- **Complex table structure**: multi-level headers, merged cells, and irregular layouts can be hard to linearize without losing semantics (Cheng et al., 2022).

2) Symbolic reasoning (Program-aided approaches). Symbolic reasoning explicitly produces an executable program (e.g., SQL, pandas/Python) whose execution yields the final answer:

$$\begin{aligned} \hat{p} &= \texttt{LLM}\big(q \,\|\, \texttt{Schema}(T)\big) \\ \hat{a} &= \texttt{Exec}(\hat{p}, T) \end{aligned} \tag{3}$$

This paradigm includes classical semantic parsing (text-to-SQL) and modern LLM tool-use variants where the model generates code and calls an executor iteratively (Chen et al., 2023). Agentic variants extend this into multi-step tool interaction: the model alternates between reasoning, selecting an action (Zhang et al., 2023; Lu et al., 2025). PoTable (Mao et al., 2025) further adopts a stage-oriented *plan-then-execute* pipeline, where the LLM plans and generates Python operations within analyst-inspired stages and iteratively corrects errors using real-time execution feedback. Recent approaches also evolve intermediate tabular states in the reasoning chain (e.g., Chain-of-Table) so that each step materializes partial results as a new table, reducing long-horizon reasoning burden (Wang et al., 2024b).

Strengths. Symbolic methods typically offer precision by delegating arithmetic and set operations to deterministic executors, while simultaneously ensuring verifiability through auditable program traces that allow for intermediate validation.

Failure modes. However, symbolic reasoning can be brittle:

- **Semantic inflexibility**: fixed operator libraries may not capture nuanced question intent, and LLM-generated programs may be logically incorrect or mishandle data; small mismatches in column naming or value formats/units can derail parsing and execution (Cao and Liu, 2025; Nahid and Rafiei, 2025).
- **Sensitivity to complex tables**: irregular lay-

Listing 1: ST-Raptor output (Baseline). Note the semantic loss in headers ("None") and structural corruption in the "Depreciation" row (lines 28-33).

```
{
    "table": {
        "None": {
        "2019": "", "2018": "", "change": ""
        },
        "Salaries, wages, and benefits": {
            "0": "5.27¢", "1": "4.79¢",
            "2": "0.48¢", "3": "10.0 %"
        },
        "Fuel and oil": {
            "0": "2.76", "1": "2.89",
            "2": "(0.13)", "3": "(4.5)"
        },
        "Maintenance materials and repairs": {
            "0": "0.78", "1": "0.69",
            "2": "0.09", "3": "13.0"
        },
        "Landing fees and airport rentals": {
            "0": "0.87", "1": "0.83",
            "2": "0.04", "3": "4.8"
        },
        "Depreciation and amortization": {
            // CRITICAL FAILURE HERE
            "0.78": "", "0.75": "",
            "0.03": "", "4.0": ""
        },
        "Other operating expenses": {
            "0": "1.92", "1": "1.79",
            "2": "0.13", "3": "7.3"
        },
        "Total": {
            "value1": "12.38¢", "value2": "11.74¢",
            "value3": "0.64¢", "value4": "5.5 %"
        }
    }
}
```

Listing 2: AdaSTR output (Ours). Note the fully preserved semantic paths and correct value alignment across all rows.

```
{
  "tree_table": {
    "Salaries, wages, and benefits": {
      "Year ended... - 2019": "5.27¢",
      "Year ended... - 2018": "4.79¢",
      "Per ASM - change": "0.48¢",
      "Percent - change": "10.0 %"
    },
    "Fuel and oil": {
      "Year ended... - 2019": "2.76",
      "Year ended... - 2018": "2.89",
      "Per ASM - change": "(0.13)",
      "Percent - change": "(4.5)"
    },
    "Maintenance materials...": {
      "Year ended... - 2019": "0.78",
      "Year ended... - 2018": "0.69",
      "Per ASM - change": "0.09",
      "Percent - change": "13.0"
    },
    "Landing fees...": {
      "Year ended... - 2019": "0.87",
      "Year ended... - 2018": "0.83",
      "Per ASM - change": "0.04",
      "Percent - change": "4.8"
    },
    "Depreciation and amortization": {
      // CORRECT ALIGNMENT
      "Year ended... - 2019": "0.78",
      "Year ended... - 2018": "0.75",
      "Per ASM - change": "0.03",
      "Percent - change": "4.0"
    },
    "Other operating expenses": {
      "Year ended... - 2019": "1.92",
      "Year ended... - 2018": "1.79",
      "Per ASM - change": "0.13",
      "Percent - change": "7.3"
    },
    "Total": {
      "Year ended... - 2019": "12.38¢",
      "Year ended... - 2018": "11.74¢",
      "Per ASM - change": "0.64¢",
      "Percent - change": "5.5 %"
    }
  }
}
```

outs, hierarchical headers, and implicit relationships can make schema extraction ambiguous; symbolic parsers may fail without careful canonicalization (Cao and Liu, 2025).

3) Hybrid reasoning (Textual ⊕ Symbolic). Hybrid systems integrate the semantic flexibility of textual reasoning with the precision of symbolic execution, typically employing paradigms such as **adaptive routing** for dynamic selection (Liu et al., 2023b; Zhang et al., 2024a), **interleaved modularity** for step-wise refinement (Khoja et al., 2025) to mitigate hallucinations. However, these methodologies are predominantly restricted to flat tables with canonical structures; they struggle to generalize to complex tables, where irregular schemas and nested headers obstruct the standardized data manipulation required for symbolic engines. In this work, we bridge this gap by introducing **AdaSTR**, which serializes complex tables into Semantic Trees. By normalizing irregular layouts into a structure with explicit hierarchy and unified semantics, AdaSTR renders complex data compatible with symbolic execution, thereby successfully extending the novel high-precision hybrid paradigm (**DuTR**) to the domain of complex TableQA.

F Metrics and Mode Selection for Adaptive Tree Synthesis

F.1 Adaptive Mode Selection Policy

Directly encoding large tables for LLM processing is often infeasible due to context-length constraints and the sharply increasing computational cost with table size (e.g., even a medium-sized table can exceed tens of thousands of tokens). We therefore adopt an automatic mode-selection policy that chooses between DSP/SRE/PSS according to

a context budget and table statistics.

Token Budget and Lightweight Statistics. Instead of utilizing the entire physical context window L_{max}, we define an *effective reasoning budget* B to ensure robust attention performance. Let $L_{\text{safe}} = \alpha \cdot L_{\text{max}}$ be the safe context limit (Liu et al., 2023a; Hsieh et al., 2024), and L_{sys} be the system prompt cost. The available budget for table serialization is:

$$B = L_{\text{safe}} - L_{\text{sys}} - \mathcal{E} \tag{4}$$

where $\mathcal{E}$ estimates the reserved tokens for the synthesized tree structure. We estimate the table footprint using a lightweight serialization $\tau(T)$. Specifically, $\tau(T)$ transforms the table into a compact nested list format, which eliminates redundant separators to serve as a minimal-token baseline. We compute the following statistics:

$$S = \text{Tok}(\tau(T)) \tag{5}$$

$$\bar{s} = \frac{1}{|C|} \sum_{c \in C} \text{Tok}(c) \tag{6}$$

$$r_{\text{long}} = \frac{|\{c \in C : \text{Tok}(c) > \gamma\}|}{|C|} \tag{7}$$

$$n = |\text{rows}| \cdot |\text{cols}| \tag{8}$$

where S represents the estimated total token consumption of the table. To capture content density and scale, we define $\bar{s}$ as the average token length per cell, and r_{long} as the long-cell ratio (the proportion of cells exceeding a length threshold γ). The variable n serves as the table scale indicator, representing the total number of cells. Here, C denotes the set of non-empty cells, and $\text{Tok}(\cdot)$ is measured using the target model tokenizer.

Decision Rule. The selection logic prioritizes fitting the context first, then addressing verbose (text-dense) overflow, and finally handling massive-scale tables.

1. **DSP (Default)**: If the estimated token footprint fits within the context budget, we use direct generation for maximum semantic fidelity.

$$S \leq B \tag{9}$$

2. **SRE (Density-First)**: If the table exceeds the budget but the scale is not massive (cell count is manageable), we attribute the overflow primarily to verbose content (e.g., high long-cell ratio) and switch to Symbolic Reference Encoding for compression via address placeholders.

$$\begin{aligned} S > B \ \wedge \ n \leq n_{high} \\ \wedge \ (\bar{s} > \mu \ \vee \ r_{\text{long}} > \eta) \end{aligned} \tag{10}$$

3. **PSS (Scale-First)**: If the table exceeds the budget and the number of cells is massive, we switch to Programmatic Structure Synthesis. PSS is most effective for hyperscale tables because loop-based code expands large structures more reliably than token-by-token enumeration.

$$S > B \ \wedge \ n > n_{high} \tag{11}$$

If $S > B$ but neither SRE nor PSS conditions are strictly met (a rare edge case), we default to SRE as the safer fallback to reduce context length while preserving structure.

Default Hyperparameters. Unless otherwise stated, we use $\alpha = 0.6$, $\mu = 80$ tokens, $\eta = 0.3$, and $n_{high} = 1 \times 10^3$. We did not perform extensive hyperparameter tuning; thresholds were chosen once as reasonable defaults. In our ablation studies (Section 4.3), enabling this policy improves evaluator scores (IC/SI) and, critically, reduces DSP-only failures caused by output truncation on oversized tables, thereby improving robustness without.

F.2 Evaluation Metrics for Semantic Tree Quality

To rigorously assess the quality of the constructed semantic trees and guide the **Reflective Switching** mechanism, we define two quantitative metrics: **Information Coverage** and **Structural Integrity**. Although we do not provide a theoretical proof, our empirical observations (as illustrated in Figure 7) demonstrate that these metrics serve as reliable proxies for downstream Question Answering (QA) performance.

1. Information Coverage. This metric measures the completeness of the information transfer from the tabular structure to the tree structure. It is calculated as the ratio of original table cells whose content is successfully represented in the generated tree nodes:

$$\text{Coverage} = \frac{|C_{\text{mapped}}|}{|C_{\text{total}}|} \tag{12}$$

where C_{mapped} denotes the set of cells found in the tree and C_{total} is the total number of non-empty cells in the original table.

2. Structural Integrity. This metric evaluates the correctness of the hierarchical relationships in the generated tree. We employ a bottom-up path verification strategy. For every data leaf node (value) in the tree, we trace the path back to the `ROOT`. The validity of a path is determined as follows:

- **Initialization**: Start from the Leaf node. Let the Leaf be the distinct anchor.
- **Traversal**: Move upwards to the parent node.
- **Verification Logic**: Check the spatial alignment in the original table:
  1. If the current node is in the **same row or same column** as the Leaf node, the relationship is valid; continue to the parent.
  2. If not, check if the current node is in the **same row or same column** as its immediate child node (the node just visited). If yes, the relationship is valid (transitive alignment); continue.
  3. If neither condition is met, the path is deemed **Structurally Broken**, and verification terminates.
- **Success**: If the traversal reaches the `ROOT` without error, the path is valid.

The *Structural Integrity* score is the percentage of valid paths out of all leaf-to-root paths.

Discussion on Evaluation Metrics. (1) **Merged Cell Representation:** Many table datasets lack explicit annotations for merged cell spans. For example, a header row ['Gender', ''] implies that the second cell inherits the label "Gender," yet the raw data often represents this as a null value. Consequently, during structural evaluation, items (e.g., "Male") nested under these implicit headers fail to align with their correct parent nodes, resulting in unwarranted penalties to the structural accuracy score. (2) **Information Redundancy & Normalization:** The AdaSTR method is designed to intelligently filter redundant information and normalize headers (e.g., factoring units out of cell values: 10 kg $\rightarrow$ Key: Weight (kg), Value: 10). While this restructuring offers superior semantic precision, it can trigger false negatives in our information coverage evaluation scripts, which may fail to recognize the data in its transformed state.

Correlation with Downstream Accuracy. To validate the proposed metrics, we analyzed their relationship with downstream task performance. As illustrated in Figure 7, we conducted correlation experiments on the SSTQA dataset. We sampled 200 instances and processed them exclusively using the SRE tree construction strategy. This specific strategy was selected to ensure that the metric scores were broadly distributed across the entire spectrum. For the downstream evaluation, we employed our proposed DuTR method to perform the question answering tasks. Our analysis reveals a significant positive correlation between the evaluation metrics and QA accuracy. This trend suggests that higher metric scores correspond to improved QA performance. Given that downstream accuracy serves as an intuitive proxy for tree quality, these results confirm the effectiveness of our evaluation framework. Notably, fluctuations in accuracy observed at higher metric scores are expected, as QA performance depends not only on tree structure but also on external factors such as question difficulty. Crucially, this empirical distribution guides the threshold configuration for our Evaluator-Guided Refinement Loop. We observe that when the **Coverage Rate** exceeds 0.8 (the 0.8–1.0 bin), the average QA accuracy remains robust at 84.00%. Similarly, **Structure Accuracy** demonstrates a consistent performance plateau once it surpasses 0.6, with the 0.8–1.0 bin achieving 86.82%. Based on these findings, we set the acceptance thresholds of quality assessment.

Metric-Guided Self-Correction. We implement a feedback loop that triggers specific refinement actions when the generated tree fails to meet predefined quality thresholds. The correction strategy is adaptive to the type of deficiency detected: (1) **Structural Deficiency:** If the *Structural Integrity* score is low, it indicates fundamental flaws in the logical hierarchy (e.g., incorrect parent-child relationships). In this case, we instruct the system to revert to the **Hierarchy Identification** phase to reconstruct the semantic backbone from scratch. (2) **Information Omission:** Conversely, if the structure is valid but *Information Coverage* is insufficient, it implies that the hierarchy is correct but specific data points are missing. Here, we retain the current tree and prompt the LLM to supple-

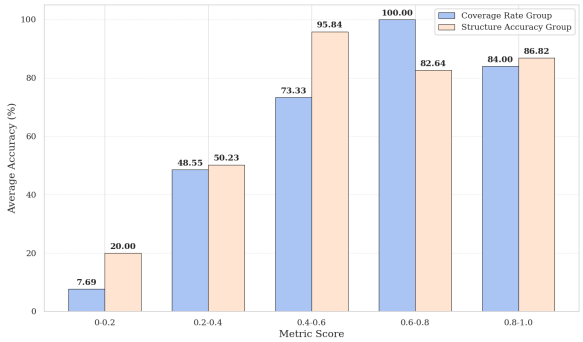

Figure 7: Relationship between evaluation metric scores (information coverage and structure integrity) and downstream QA accuracy across score bins.

ment missing information based on the raw table, while strictly maintaining the backbone hierarchy established in the Hierarchy Identification phase.

G Optimization Strategies for the Selector Module

The Selector module plays a critical role in determining the final output when the upstream generators produce conflicting answers. Thus, we conducted an in-depth analysis on a subset of 163 samples where the candidate answers were inconsistent. We investigate three distinct optimization strategies: Chain-of-Thought (CoT) reasoning, Supervised Fine-Tuning (SFT), and Logit-based Debiasing (Debias). To ensure deterministic reproducibility, we configured the Qwen3-8B selector with greedy decoding settings (temperature=0, do_sample=False) and disabled its reasoning mode. Furthermore, for the SFT and Debiasing strategies, we directly utilized the model's output logits to determine the selected answer.

Baseline Performance. Our default implementation utilizes a direct Zero-shot Answer extraction prompt shown in Appendix A. On the conflicting subset, the baseline achieves an accuracy of 70.55% (115/163), significantly outperforming random selection. The average inference latency is minimal (∼0.5s), making it highly efficient.

Strategy I: Chain-of-Thought (CoT). We first attempted to enhance the reasoning capability of the Selector by incorporating Chain-of-Thought (CoT) prompts. The accuracy improved marginally to 72.39% (118/163), a gain of only 3 correct samples. However, the inclusion of reasoning steps caused a drastic spike in latency, increasing from 0.5s to ∼13s on average. This suggests that for the specific task of binary/multiple choice selection, the computational overhead of CoT yields diminishing returns. The small-scale model may struggle to effectively leverage the generated reasoning path to correct subtle errors.

Strategy II: Supervised Fine-Tuning (SFT). To better align the model with the selection task, we constructed a specialized SFT dataset derived from the HiTab training set. **Data Construction:** We generated 1,000 training samples by pairing the ground truth with a "distractor" answer (a plausible but incorrect answer generated by the LLM). To prevent the model from memorizing answer positions, we applied data augmentation by swapping the order of correct and incorrect answers with a probability of 0.7, resulting in a total of 2,700 samples. **Training Setup:** We fine-tuned the Qwen3-8B model using QLoRA (4-bit quantization) to ensure efficiency (Dettmers et al., 2023). Key hyperparameters included a LoRA rank of $r = 16$ targeting all linear layers, a learning rate of 2.0×10^{-5} with a cosine scheduler (0.1 warmup ratio), and an effective batch size of 8 trained for 3 epochs using BF16 precision. **Result & Analysis:** After fine-tuning, the accuracy rose significantly to **76.68%** (125/163). The substantial improvement indicates that the Selector benefits more from learning the specific error patterns and task format through parameter updates than from zero-shot reasoning.

Strategy III: Logit-based Debiasing (Debias). Recent studies (Zheng et al., 2023) show that LLMs exhibit Positional Bias, often preferring the first option (Option A) in multiple-choice tasks. We implemented a parallel decoding strategy to mitigate this without training. **Method:** We feed the input twice in parallel with the candidate answers swapped (Order A, B and Order B, A). Instead of relying on generated text, we sum the distinct token logits for each candidate across both permutations to determine the winner. **Result:** This method achieved an accuracy of 73.61% (120/163). While slightly less effective than SFT, this approach requires no training and maintains low latency (∼0.6s via parallel batching), serving as a cost-effective alternative.

Summary. Table 7 summarizes the trade-offs. We conclude that SFT offers the highest performance ceiling, while the Baseline or Logit-based methods provide the best efficiency.

Method	Accuracy	Latency	Training
Baseline (Direct)	70.55%	∼0.5s	No
Strategy I (CoT)	72.39%	∼13.0s	No
Strategy II (SFT)	**76.68%**	∼0.5s	Yes
Strategy III (Debias)	73.61%	∼0.6s	No

Table 7: Performance comparison of optimization strategies on the conflicting subset ($N = 163$).

H Error Analysis

To understand the limitations of our Dual-Mode Tree Reasoning framework, we conducted a failure analysis on a stratified random sample of $N = 60$ error cases drawn from the SSTQA dataset. We sampled equally from three distinct failure populations ($n = 20$ each): (1) cases where both Symbolic and Textual modes failed, (2) cases where Textual succeeded but Symbolic failed, and (3) cases where Symbolic succeeded but Textual failed.

H.1 Error Taxonomy and Distribution

We categorized the errors into four primary types based on the root cause analysis (Table 8).

As illustrated in Figure 8, the distribution of error types varies significantly across the failure populations:

- **Mode-Specific Weaknesses:** There is a clear decoupling of errors. *Structural Misalignment* is the dominant failure mode for Symbolic Reasoning (12 cases), confirming that code generators struggle with rigid schema assumptions. Conversely, *Arithmetic Hallucination* is heavily skewed towards Textual Reasoning (11 cases), highlighting the LLM's inability to perform precise calculations without external tools.

- **The Consensus Insight:** The *Annotation Error* category shows a unique pattern: it is overwhelmingly composed of "Both Wrong" cases (12 out of 15). This indicates that when our Textual and Symbolic modules reach a consensus that disagrees with the gold label, the discrepancy is usually due to dataset flaws (e.g., wrong ground truth) rather than model error.

H.2 Analysis of Representative Failures

1. Structural Misalignment (36.7%). While this error affects both paradigms, it is predominantly a Symbolic failure (12 cases vs. 3 Textual). The Symbolic module struggles because code generators often hallucinate a flattened schema for deeply nested trees (e.g., missing a parent key like `Loan Bank` and iterating directly over children). *Textual reasoning is more robust* to structure because it reads the serialized tree linearly, though it occasionally fails (3 cases) when long context windows obscure the indentation levels of remote leaf nodes. This contrast highlights that while LLMs understand serialized semantics (Textual) well, mapping them to rigid executable paths (Symbolic) remains fragile without agentic inspection.

2. Annotation Errors (26.7%). This category represents a mode-agnostic failure, heavily concentrated in the "Both Wrong" intersection (12 cases). Unexpectedly, a quarter of failures were actually correct model predictions penalized by flawed datasets. For instance, given the query *"variance rate"*, both models correctly extracted the percentage (-1.4), but the gold label (-2800) referred to the absolute difference. The fact that *both* reasoning modes—despite their orthogonal mechanisms—arrived at the same answer contradicting the label, suggests that system consensus is often a stronger signal of truth than the noisy ground truth labels.

3. Arithmetic & Logic Hallucination (20.0%). This error manifests in distinct forms across the two modes. For **Textual Mode** (11 cases), it is a *Calculation Failure*: the model retrieves correct numbers but drifts during long-chain floating-point aggregation (e.g., $26.8 + \dots$) due to the lack of a calculator. However, **Symbolic Mode** is not immune (4 cases); it suffers from *Formula Logic Hallucination*. Although the Python execution is flawless, the model may generate the wrong operator (e.g., `sum` instead of `mean`) or fail to normalize units (e.g., treating "1,000" as a string), leading to numerically wrong answers. Thus, Symbolic reasoning solves calculation but introduces logic derivation risks.

4. Retrieval Bias (16.7%). This error reflects a "granularity mismatch" where models select high-level summary nodes instead of specific leaves. It is more prevalent in Textual Mode (4 cases) due to "semantic short-circuiting"—the model stops reading at the first keyword match (e.g., "Total Expenditure") ignoring temporal constraints. **Symbolic Mode** also encounters this (3 cases), but for a different reason: *Naming Ambiguity*. If a parent node and a child node share similar key names, the code

Error Type	Definition	Count	%
Structural Misalignment	The generated code assumes a rigid schema (e.g., specific depth, key-value placement) that does not match the heterogeneous tree structure, leading to execution failures.	22	36.7%
Annotation Error	The model output is factually correct given the table content, but the gold label is incorrect due to annotation errors, typos, or calculation mistakes in the dataset.	16	26.7%
Arithmetic Hallucination	Textual reasoning fails at precise operations: arithmetic aggregation, numerical comparison (especially negative numbers/decimals).	12	20.0%
Retrieval Bias	The model exhibits *semantic short-circuiting*, selecting high-level summary nodes (e.g., "Total") that match keywords but lack the required categorical granularity, failing to drill down to specific leaf details.	10	16.7%

Table 8: Distribution of error types across $N = 60$ sampled failure cases.

generator may ambiguously reference the parent object instead of drilling down, failing to satisfy the specific granularity required by the question.

I Additional Experimental Analysis

I.1 Comparison with Relational Transformation Methods

As illustrated in Table 9, to ensure a fair comparison with **RelationalCoder** and **TabFormer**, both of which utilize **GPT-3.5** as the reasoning backbone and report results using *Exact Match (EM)* on the HiTab dataset, we align our experimental setup with these established protocols by employing **GPT-3.5** as the downstream *QA reasoner*. Crucially, our primary objective is to investigate which intermediate representation format—our Logical Semantic Tree versus baseline relational or linear formats—more effectively enhances the reasoning capabilities of LLMs. By standardizing the downstream reasoning engine to GPT-3.5, we isolate the data representation as the key variable. Thus, while we utilize DeepSeek-V3 for the upstream serialization (AdaSTR), the performance difference observed under the same reasoner directly highlights the superior information accessibility and structural clarity provided by our tree representation.

Experimental results demonstrate that our **DuTR (Symbolic reasoning)** achieves an EM score of 74.1% on HiTab, significantly outperforming the strongest relational baseline, RelationalCoder (66.7%). This performance disparity highlights a fundamental divergence in data representation. RelationalCoder and TabFormer attempt to flatten or decompose nested complex tables into standardized relational tables—a transformation aligned with SQL logic. However, this process disrupts the original *Physical Locality* and *Hierarchical Context*. Consequently, the model incurs a higher cognitive load during reasoning, as it must implicitly execute complex "Join" operations to reconstruct cross-level relationships.

Structure Representation	Reasoning Method	EM (%)
Baselines		
TabFormer(Dong et al., 2025a)	Chain-of-Table(Wang et al., 2024b)	57.4
	E5(Zhang et al., 2024c)	56.9
	ReactTable(Zhang et al., 2023)	54.3
RelationalCoder(Dong et al., 2025b)	Chain-of-Table(Wang et al., 2024b)	66.7
AdaSTR (Ours)	DuTR(textual reasoning)	64.1
	DuTR(symbolic reasoning)	**74.1**
	DuTR(adaptive selection)	68.9
	DuTR(Oracle)	80.4

Table 9: Supplementary comparison with relational transformation methods. We evaluate different structure representations coupled with various reasoning methods.

In contrast, our **Semantic Tree** preserves the topological integrity of the data. Parent and child nodes are directly connected, allowing the LLM to perform aggregation via recursive operations on local substructures without the need for cross-table retrieval. This *Structure-Logic Isomorphism* is pivotal to the observed performance gains. On the computation-intensive HiTab dataset, our Symbolic Tree Manipulation yields optimal performance. Notably, our method maintains a substantial lead even when deployed on the relatively weaker GPT-3.5 backbone. This suggests that the performance gains are derived primarily from superior data structure design rather than solely from the intrinsic capabilities of a stronger model.

Analysis of Adaptive Mode Performance: The slightly lower performance of the Adaptive mode compared to the Symbolic mode can be attributed to the rigidity of the Exact Match (EM) metric.

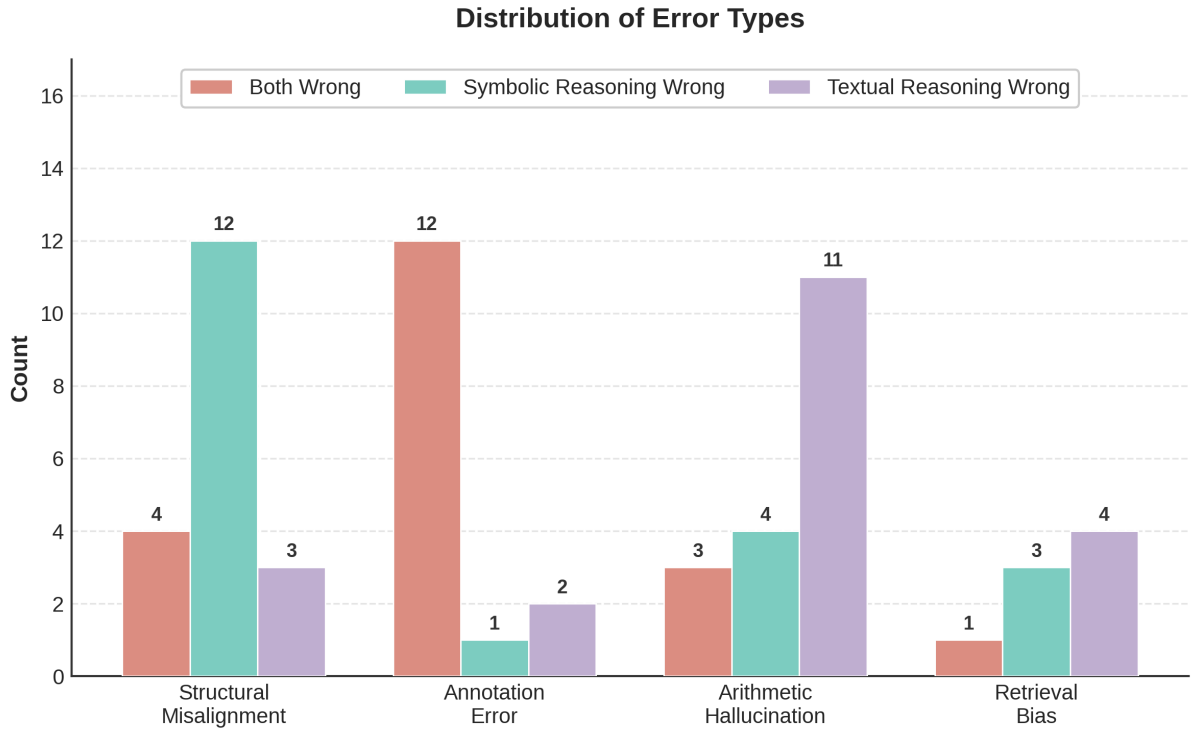

Figure 8: Statistics of error categories by reasoning failure mode.

EM demands character-level correspondence between the output and the reference answer. Since we did not perform granular optimization to align the textual output with specific EM formatting constraints, the Adaptive selector occasionally chooses responses that are semantically correct but syntactically mismatched (i.e., failing the EM check). This phenomenon introduces false negatives, thereby penalizing the overall score of the Adaptive strategy.

I.2 Generalization on Open-Source Models

As shown in Table 10, the Semantic Tree representation generalizes well across different open-source backbones and inference settings. Compared with directly prompting models with the original raw tables, converting the input into a Tree Table consistently improves performance for every model tested (e.g., +3.14 for Qwen2.5-7B, +1.44 for Llama3.1-8B, and up to +9.95 for Qwen3-8B in thinking mode). This suggests that providing an explicitly structured, hierarchy-aware layout reduces surface-form ambiguity and better aligns with models' internal reasoning patterns. We further observe that **symbolic-only** inputs can be brittle for some open-source models, most notably Llama3.1-8B (10.47), indicating that purely symbolic signals without sufficient linguistic grounding may be difficult to interpret reliably.

Method	Qwen2.5-7B	Llama3.1-8B	Qwen3-8B	
			no thinking	thinking
Direct Prompt (Raw Table)	46.60	35.99	52.75	54.97
Direct Prompt (Semantic Tree)	49.74	37.43	<u>60.86</u>	64.92
ASTRA (Textual Reasoning)	<u>50.14</u>	<u>40.45</u>	60.60	<u>65.45</u>
ASTRA (Symbolic Reasoning)	36.39	10.47	58.64	62.70
ASTRA (Adaptive Selection)	**53.01**	**41.49**	**65.31**	**70.02**
ASTRA (Oracle)	60.64	44.76	75.26	78.14

Table 10: Performance of Open-Source Models on the SSTQA Dataset. The best results are highlighted in **bold**, and the second-best results are <u>underlined</u>. Note that the "Direct Prompt" method follows the same workflow as the Foundation model approach used in the main experiments.

The comparison between Qwen3-8B with and without thinking shows that stronger inference-time reasoning amplifies the benefit of structured representations: thinking mode improves Tree Table prompting from 60.86 to 64.92 (+4.06), reinforcing that our Semantic Tree guidance remains effective even when the model's reasoning strategy changes.

To further investigate module contributions on smaller parameters, we conducted a detailed ablation study on Qwen3-8B (Table 11). A striking contrast with the larger DeepSeek-V3 model emerges in symbolic reasoning: while DeepSeek-V3 showed resilience to removing the **Self-Correction Loop** (dropping only 1.57%), Qwen3-8B suffered a severe degradation (-12.67%). This

Configuration	Acc (%)	Δ
Adaptive Tree Navigation	**60.60**	-
w/o Embedding Model	56.68	-3.92
w/o Dynamic Switching	-	-
Force Root-to-Leaf	55.76	-4.84
Force Leaf-to-Root	58.64	-1.96
Symbolic Tree Manipulation	**58.64**	-
w/o Code Examples	50.89	-7.75
w/o Structural Skeleton	53.12	-5.52
w/o Self-Correction Loop	45.97	-12.67
Impact of Data Representation		
Raw Table (Direct Prompting)	52.75	-
Semantic Tree (Direct Prompting)	**60.86**	+8.11

Table 11: Ablation analysis of **DuTR (Qwen3-8B)**. We report the Accuracy (%) and the performance change (Δ) compared to the full configuration within each mode.

indicates that while frontier models possess robust intrinsic code generation capabilities, smaller models rely critically on iterative execution feedback to rectify syntax or logic errors. Additionally, the **Static Tree** representation alone yielded a **+4.45%** gain over raw tables, confirming that the cognitive scaffold provided by our tree structure remains beneficial independent of model scale.

Overall, these results demonstrate that the proposed representation is not tailored to any single proprietary model, but instead serves as a robust interface for open-source LLMs with varying parameter scales and reasoning capabilities.

J Case Study

In Section 4.6, we demonstrated how ASTRA addresses *Structural Neglect* and *Representation Gap* through a representative case on hierarchical enumeration (**Explicit Hierarchy**). To provide a comprehensive evaluation, we present two additional case studies in this appendix, focusing on **Representation Alignment** (handling null/missing values) and **Symbolic Compatibility** (performing long-horizon arithmetic). These cases, summarized in Table 12 and Table 13, further validate the necessity of our Text-Symbolic paradigm.

J.1 Addressing Linguistic Misalignment in Retrieval

Queries involving negative constraints or null values (e.g., *"empty summary"* in Table 12) expose the failure of current serializations to achieve Representation Alignment. Existing methods like GraphOTTER convert tables into triples or vectors that fail to preserve the natural language context of "missingness," causing the retriever to miss empty cells entirely (Representation Gap). Similarly, EEDP's linear serialization lacks distinct markers for null values, leading to hallucination during row scanning. In contrast, our framework achieves Symbolic Compatibility by synthesizing programmatic constraints (`if not record['Summary']`). By shifting from ambiguous linguistic matching to precise symbolic logic, we accurately identify all 26 records, validating the necessity of supporting verifiable code-based reasoning over purely textual serialization.

J.2 Eliminating Arithmetic Hallucination (Re: Symbolic Compatibility)

Long-horizon numerical reasoning (Table 13) critically tests Symbolic Compatibility. While baselines may retrieve the correct data, they fail to satisfy the requirement for verifiable computation. As observed with GraphOTTER and EEDP, feeding serialized floating-point numbers directly into an LLM triggers arithmetic hallucination ($48,154 \neq 51,596$), as the text-based modality is ill-suited for high-precision calculation.Our approach fulfills the symbolic requirement by offloading computation to a Python environment. The generated code naturally handles data type filtering and performs operations with machine precision, demonstrating that an ideal serialization must support the seamless transition from textual representation to executable logic.

Question	**How many expense records have an empty summary?**
Gold Answer	**26**
GraphOTTER	**Output: 4** **Evidence:** The dense retriever only located 4 isolated empty cells out of 26: `[(5, 6, ''), (13, 6, ''), (16, 6, ''), (19, 6, '')]` *Failure Analysis:* The vector-based retriever struggled to match the query "empty summary" with actual empty strings, resulting in extremely low recall.
ST-Raptor	**Output: 46** **Process:** Decomposed the query and generated the operation chain: `[MATH] + [Summary] + [CNT]`. *Failure Analysis:* While it correctly identified the column and the counting task, the execution module failed to apply the "is empty" filter, effectively counting *all* 46 rows in the table instead of the specific subset.
EEDP	**Output: 29** **Process:** Attempted a manual row-by-row inspection in the CoT: *"The following rows have null... 004, 012, 015, 017..."* *Failure Analysis:* The model hallucinated the existence of empty values in non-empty rows during the text generation process, leading to an arithmetic error (counting 29 instead of 26).
Ours	**Output: 26** **Logic:** synthesized a programmatic constraint to filter data accurately: `if not record['Summary'].strip(): count += 1` *Result:* The symbolic execution over the structured tree yielded the exact count (26) without hallucination.

Table 12: **Case Study on Null Value Aggregation.** Comparison of failure modes: GraphOTTER suffers from *retrieval miss* on empty values (poor *Representation Alignment*); ST-Raptor correctly plans but fails in *execution* (returning total count); EEDP suffers from *hallucination* during manual enumeration. Our method ensures correctness via symbolic code generation.

Question	**What is the average standard error of product sales from month 1 to month 18?**
Gold Answer	**3685.4505**
GraphOTTER	**Output: 3439.61** **Evidence:** The model correctly identified all 14 relevant cells (Months 5–18) and recognized the operation as "Average". *Failure Analysis:* **Arithmetic Hallucination**. Despite correct retrieval, the LLM failed to sum the sequence of 14 floating-point numbers accurately during text generation, resulting in an incorrect sum ($48,154$ instead of $\approx 51,596$) and final average.
ST-Raptor	**Output: 4287.34** **Process:** Decomposed into `[COND] (Month 1-18) -> [MATH] + [AVR]`. *Failure Analysis:* **Execution Error**. While the retrieval and plan were semantically correct, the mathematical primitive likely mishandled the 'None' values present in Months 1–4 or failed to compute the weighted mean of the float values correctly.
EEDP	**Output: 3399.74** **Process:** The CoT explicitly listed the 14 values to sum: *"3355.12 + 3220.81 + ..."* *Failure Analysis:* **Calculation Error**. Similar to GraphOTTER, the Chain-of-Thought process suffered from calculation drift. The model calculated the sum as $47,596.35$, significantly deviating from the true sum, leading to an underestimated average.
Ours	**Output: 3685.4505** **Logic:** Generated a symbolic program to iterate and aggregate valid data: `if error is not None: total += error; count += 1` *Result:* By offloading the computation to a Python execution environment, our method achieved exact numerical precision, correctly handling null filtering and floating-point arithmetic.

Table 13: **Case Study on Arithmetic Aggregation.** Comparison of failure modes on long-sequence numerical queries. Baselines correctly retrieve data but suffer from *arithmetic hallucination* or *primitive execution errors*, proving the inadequacy of textual processing. Our method ensures verifiable precision via code execution, fulfilling **Symbolic Compatibility**.

K Prompt

Listing 1: AdaSTR: Header Normalization

```
You are a data analysis expert specializing in parsing complex table structures.

Your task is to analyze the first few rows of the provided table and determine a
    unique normalized header for each column. You must strictly follow these rules:

1. **Strict Preservation of Original Wording**: You must preserve the original text
    from the table cells. It is forbidden to create new names or summaries.
   (For example: if a cell contains "Total Students", the header must include
    "Total Students", not "Total Students Metrics".)
2. **Hierarchical Combination**: If header information is distributed across
    multiple header rows, combine them using the format: [Lower-Level Header] -
    [Upper-Level Header].
   The part before " - " must be the most specific level.
3. **Strict Column Count Matching**: The number of strings in the output array must
    exactly match the number of columns in the data rows.

The table is provided in JSON array form.

[Input Table]:
{TABLE_AS_JSON_STRING}

Your output must be a single, valid JSON string array representing the normalized
    headers. Do not provide any explanation.
```

Listing 2: AdaSTR: Hierarchy Identification

```
You are a data architecture expert specializing in analyzing the logical structure
    of tables.

You will be given a table and its normalized headers. Your task is to identify
    which columns belong to "Hierarchy Keys" and which belong to "Value Leaves".

You must reason about the table step by step and then produce a single JSON object
    as the final answer.

## Step 1: Inspect column roles

For each column, think about:
- Is it mostly numeric (measures), mostly text (categories), or mixed?
- Roughly how many distinct values does it have compared to the number of rows?
- Do the values look like IDs / timestamps / codes (often unique), or like
    categories (heavily reused)?

Use these heuristics:
- Columns that are mostly numeric or percentages are usually **value_leaves**.
- Columns whose values repeat a lot (far fewer unique values than rows) are strong
    candidates for **hierarchy_keys**.
- A column whose values are almost all unique (ID-like or timestamp-like) is a good
    **primary hierarchy_key** for simple flat tables.

## Step 2: Detect header-based hierarchies (multi-row headers)

If the original table had multiple header rows (already compressed into the
    normalized headers you see), then:
- When a normalized header clearly encodes multiple levels (e.g. "Sales - 2023 -
    Q1"), its column is usually a **value_leaf** grouped under a higher-level
    semantic group.
- Semantic group names must come directly from the original headers; do not invent
    new group names.

## Step 3: Detect row-based block structures (very important for complex tables)

Even if there is only one obvious text column, the table may still be **complex**
    if it uses row blocks.
```

```
Treat the table as **complex** (not simple) when you observe patterns like:

1. **Block headers + detail rows in the same column**:
   - Some rows in a text column look like high-level categories and have many empty
    cells or missing numeric values in other columns.
   - The rows immediately following such a row contain repeated labels such as
    "total", "weekday", "weekend", etc. with numeric values filled.
   - The same small set of labels ("total", "weekday", "weekend", ...) appears
    multiple times in separated blocks.

2. **Strong repetition of small categorical sets**:
   - A column alternates between "group header" values and a small fixed set of
    "detail" values in a patterned way.
   - This usually indicates a hierarchy like: Behaviour Group > Behaviour Detail.

In such cases:
- You should set "table_type" to "complex", even if there is only one text column.
- The **hierarchy_keys** should include that text column.

## Step 4: Decide simple vs complex

Use these refined rules:

- Classify as **simple** only if ALL of the following hold:
  1. There is no obvious multi-row header structure.
  2. There is no clear row-based block pattern as described in Step 3.
  3. Each row is largely independent.
  4. Typically there is at most one main categorical column and the others are
    mostly numeric measures.

- Classify as **complex** if ANY of the following holds:
  1. There are multiple categorical columns that naturally form a chain.
  2. Some categorical columns have heavily repeated values and clearly act as
    grouping keys.
  3. There are apparent aggregation / subtotal / summary rows.
  4. There is a row-based block structure as described in Step 3.

## Output Requirements

Your output must be a single, valid JSON object, and must strictly follow the
    structure below:

{
  "table_type": "simple" or "complex",
  "analysis_reason": "A brief explanation of the reasoning behind your judgment",
  "hierarchy_keys": ["header1", "header2", ...],
  "value_leaves": ["header3", "header4", ...],
  "semantic_groups": {
    "OriginalGroupName1": ["header_a", "header_b"]
  }
}

Additional constraints:
- Every header in "hierarchy_keys" and "value_leaves" must come from the normalized
    header list.
- The union of "hierarchy_keys" and "value_leaves" must exactly cover all
    normalized headers.
- If "table_type" is "simple", "hierarchy_keys" should usually contain only **one**
    element.

[Input Table]:
{TABLE_AS_JSON_STRING}

[Normalized Headers]:
{NORMALIZED_HEADERS_FROM_STEP_1}
```

Listing 3: AdaSTR: Tree Construction

```
You are a highly specialized data transformation engine responsible for converting
    tabular data into a nested JSON tree that preserves all semantic information.

You will be given the original table, its normalized headers, and the hierarchy
    definition. You must construct the final JSON tree based on these inputs,
    strictly following the rules below.

Before constructing the tree, you MUST:
- Read the "table_type" and "analysis_reason" from the hierarchy definition.
- Decide whether to use a flat structure (for simple tables) or a nested structure
    (for complex tables).
- When the analysis_reason mentions row-based or block hierarchy, you must
    reconstruct that hierarchy using the row patterns as well as the hierarchy_keys.

1. Semantic Hierarchy Keys
- For each hierarchy_key column, the corresponding key in the JSON must follow the
    format "[Header Name] - [Cell Value]".
- You must use " - " (space, hyphen, space) as the separator.
- Example: if the header is "Grade" and the cell value is "1", then the JSON key
    should be "Grade - 1".

2. Strict Preservation of Leaf Node Names
- The keys of value_leaves in the final JSON must exactly match the normalized
    headers.

3. General Structure Generation Rules
- Traverse the data row by row, skipping the header row.
- For each data row, use the hierarchy_keys (in order) to create or move into the
    corresponding nested structure.
- At the deepest level for a given row, create an object that stores all
    value_leaves for that row.

4. Handling of Simple Tables (table_type = "simple")
For simple tables, you should keep the structure shallow and flat:
- Typically there is only one hierarchy_key.
- For each data row:
  - Construct a single key by applying the "[Header Name] - [Cell Value]" rule to
    that hierarchy_key column.
  - Store an object under that key with all value_leaves for that row.

5. Handling of Complex Tables (table_type = "complex")

For complex tables, you MUST allow multi-level nesting, even if there is only one
    hierarchy_key column.

5.1 Column-based hierarchies
- When there are multiple hierarchy_keys across different columns, you should:
  - For each row, create or move into a nested path defined by those columns in
    order (e.g. Region > City > Store).

5.2 Row-based block hierarchies (very important)
When the analysis_reason indicates that the hierarchy is primarily row-based or
    block-structured:
- Treat rows whose hierarchy_key cell is non-empty but whose value_leaves are all
    empty or missing as PARENT or GROUP rows (block headers).
- The data rows that follow a parent row are CHILD rows of that parent.
- In the JSON tree:
  - Create a top-level key for each parent row using the hierarchy key rule.
  - Under that key, create nested keys for each child row.

6. No Data Omission
- You must not omit any cell content from the input table.
- Every non-header row must contribute at least one object into the JSON tree.

7. Prohibited Behaviors
- Do NOT invent new textual labels that do not appear in the original table or
    headers.
- Do NOT rename, summarize, or translate headers or cell values.
```

```
[Input Table]:
{TABLE_AS_JSON_STRING}

[Normalized Headers]:
{NORMALIZED_HEADERS_FROM_STEP_1}

[Hierarchy Definition]:
{HIERARCHY_DEFINITION_FROM_STEP_2}

Your output must be a single, valid JSON object that fully represents the tree
    structure of the entire table.
Ensure that all data types (numbers, strings) are correctly preserved.
Do not output any other text or explanation.
```

Listing 4: Prompt Template for Foundation Models

```
You are an expert table analyst. Your task is to answer the question based on the
    provided table.

# Instructions:

1. Please enclose your final answer in square brackets, e.g., [Answer].
2. **Pay strict attention to the required answer type.**
   - If the question starts with **"Which"** or **"What"** (e.g., "Which item?",
    "What category?"), your answer must be a *name or description* (e.g., "Sell
    Product A"), **NOT** the associated numerical value.
   - If the question starts with **"How much"**, **"What is the value"**, or **"How
    many"**, the answer must be a *numerical value* (e.g., "120000").
3. **ONLY when** the answer is numerical (as per rule 2), you must check for any
    associated "units" (e.g., "ten thousand", "million") and provide the final
    converted numerical result.

## Table:
{table_str}

## Question:
{question}

## Final Answer:
```