ST-Raptor: LLM-Powered Semi-Structured Table Question Answering

Zirui Tang
Shanghai Jiao Tong University
tangzirui@sjtu.edu.cn

Boyu Niu
Shanghai Jiao Tong University
nby2005@sjtu.edu.cn

Xuanhe Zhou*
Shanghai Jiao Tong University
zhouxh@cs.sjtu.edu.cn

Boxiu Li
Shanghai Jiao Tong University
lbxhaixing154@sjtu.edu.cn

Wei Zhou
Shanghai Jiao Tong University
weizhoudb@sjtu.edu.cn

Jiannan Wang
Simon Fraser University
jnwang@sfu.ca

Guoliang Li
Tsinghua University
liguoliang@tsinghua.edu.cn

Xinyi Zhang
Renmin University of China
xinyizhang.info@ruc.edu.cn

Fan Wu
Shanghai Jiao Tong University
fwu@cs.sjtu.edu.cn

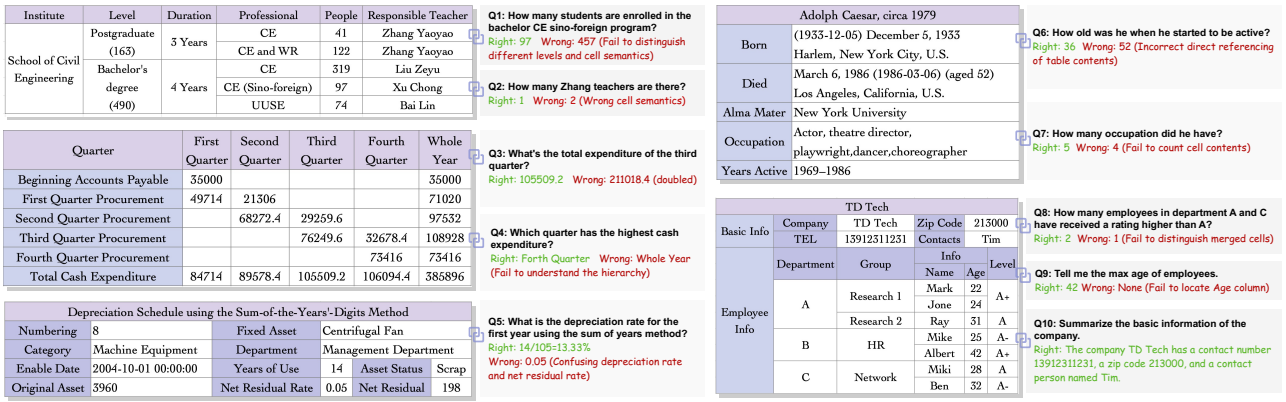

Figure 1: Example analytical questions over real-world semi-structured tables (e.g., Excel spreadsheets).

Abstract

Semi-structured tables, widely used in real-world applications (e.g., financial reports, medical records, transactional orders), often involve flexible and complex layouts (e.g., hierarchical headers and merged cells). These tables generally rely on human analysts to interpret table layouts and answer relevant natural language questions, which is costly and inefficient. To automate the procedure, existing methods face significant challenges. First, methods like NL2SQL require converting semi-structured tables into structured ones, which often causes substantial information loss. Second, methods like NL2Code and multi-modal LLM QA struggle to understand the complex layouts of semi-structured tables and cannot accurately answer corresponding questions.

To this end, we propose ST-Raptor, a tree-based framework for semi-structured table question answering (*semi-structured table QA*) using large language models. First, we introduce the Hierarchical Orthogonal Tree (HO-Tree), a structural model that captures complex semi-structured table layouts, along with an effective algorithm for constructing the tree by identifying headers, content values, and their implicit relationships. Second, we define a set of basic tree operations to guide LLMs in executing common QA tasks. Given a user question, ST-Raptor decomposes it into simpler sub-questions, generates corresponding tree operation pipelines, and conducts operation-table alignment for accurate pipeline execution. Third, we incorporate a two-stage verification mechanism: (1) forward validation checks the correctness of execution steps, while (2) backward validation evaluates answer reliability by reconstructing queries from predicted answers. To benchmark the performance, we present SSTQA, a dataset of 764 questions over 102 real-world semi-structured tables. Experiments show that ST-Raptor outperforms nine baselines by up to 20% in answer accuracy. The code is available at https://github.com/weAIDB/ST-Raptor.

1 Introduction

Semi-Structured Tables are a type of data structure that represents the flexible and complex layouts commonly found in real-world data across a variety of applications, such as Word Tables for financial reports [1], Excel spreadsheets for medical records [2], and PDF Tables for e-commerce transaction orders [3]. They often serve as the major type in these applications, e.g., accounting for up to 80% of patient records in Electronic Medical Record (EMR) systems [2].

For better understanding, in Figure 1, we showcase five example *semi-structured tables* with corresponding user questions from

*Xuanhe Zhou is the corresponding author.

diverse scenarios (e.g., human resource management, financial management, and personal information). Considering the bottom-right table (TD-Tech): (*i*) the top portion covers the company's fundamental details, while (*ii*) the lower portion contains basic information and performance ratings of employees per department. Different shades of blue in TD-Tech highlight the nested levels of the table, reflecting relationships like hierarchical headers (e.g., the "Basic Info" header linked to lower-level headers like "Company" and "TEL") and header-to-content (e.g., both "Department" and "Level" headers own the content values of "A"s). Even human analysts may need to carefully analyze the layout characters to fully understand such *semi-structured tables*.

This layout flexibility makes *semi-structured table QA* extremely challenging and distinguishes it from other common QA tasks on structured data (e.g., relational tables [12]) and unstructured data (e.g., textual documents or multimedia files [7]). In Figure 1, we showcase semi-structured table questions that commonly require the following analysis strategies: (*i*) Identify the headers based on the user question to locate the areas of relevant table cells. For instance, *Q*8 first identifies the "Level" header in the "Employee Info" sub-table, and then recognizes the "A+" cells. Meanwhile, it is essential to distinguish that the content value "A" under the "Department" header is different from "Level A" in the original question. (*ii*) Leverage the identified cells and the original question to analyze the surrounding table structures for capturing nested relationships and extracting additional information. For instance, for *Q*9, the merged cell "A+" applies to two employees to get the right answer "2". (*iii*) Explore all potentially relevant header and content cells required to form an accurate answer. For instance, for *Q*10, relevant information about the company is needed for summarization.

Existing works (including powerful LLMs like GPT4-o[29] and DeepSeek-R1 [13]) face significant limitations in *semi-structured table QA*. First, NL2SQL methods [8, 10, 16, 34, 35, 41] generate SQL statements executed on relational tables to get the final answer. However, it requires converting semi-structured tables into fully structured formats, causing substantial information loss. Second, methods like NL2Code [43] generate python code to operate on pandas dataframes. However, they struggle to understand many complex semi-structured tables and conduct precise information retrieval. A more promising approach involves converting tables into images for processing with Vision Language Models (VLM) [20, 47]. But it has three main limitations: (*i*) Table2image causes precise loss and often misleads to irrelevant table areas; (*ii*) It requires extensive fine-tuning on QA tasks and has poor generalization ability; (*iii*) It cannot work for relatively large tables those with over 100+ rows (see more analysis in Section 2.3).

There are several challenges in achieving automatic *semi-structured table QA*. First, understanding the structure of *semi-structured tables* involves two main problems: (*i*) how to distinguish header and content cells that could distribute in any areas of the table, which involves semantic understanding (e.g., in the bottom-right table of Figure 1, the "A" cells under "department" and "level" headers) and cannot be handled by rule-based matching approaches [9]; (*ii*) how to understand the nested or containment relationships across the header and content cells, where the same question to different cell relationships could lead to different answers. For instance, in the bottom-right table of Figure 1, when splitting the merged cell of department A, the answer to the question "what is the second department" would be department A instead of B (**C1**).

Second, due to the complexity of semi-structured table layouts, answering questions over such tables can be challenging, often requiring a variety of analytical "tricks". For instance, we may need to apply both left-to-right and top-to-down lookup strategies (e.g., identifying departments in the bottom-right table of Figure 1 by referencing the top-level title header, the left "Employee Info" header, and the nested "Department" header). Conversely, we may sometimes need to examine the content cells to identify the relevant headers (i.e., bottom-up lookup), which makes the *semi-structured table QA* procedure even more complicated (**C2**).

Third, an effective validation mechanism for *semi-structured table QA* remains absent, which is especially crucial for resolving issues such as hallucination in LLMs [10, 34]. In related tasks like NL2SQL, many methods do not validate the accuracy of generated answers and thus produce the final result in a single shot. Others merely check whether executing the SQL statements yields result tables sufficient to answer the question. However, in *semi-structured table QA*, the retrieved cells can (*i*) still involve complex semi-structured layouts (e.g., a single cell may contain multi-row text or even a nested sub-table) and (*ii*) be derived through multiple lookups, making it difficult for general LLMs to verify the accuracy of these retrieved semi-structured table cells (**C3**).

To address these challenges, we propose a novel *semi-structured table QA* framework (ST-Raptor). First, we introduce a graph model (HO-Tree) to represent semi-structured table layouts, with nodes denoting table headers / content values and edges capturing their hierarchical and containment relationships. This model also includes nine basic tree operations, covering most common QA tasks and addressing the structural complexity challenge (for **C1**). Second, we present an effective HO-Tree construction strategy: (1) a multi-modal LLM identifies semi-structured table headers, (2) heuristic rules separate the semi-structured table into basic table units based on the identified headers, and (3) a depth-first search (DFS) algorithm constructs the HO-Tree. This stage addresses the difficulty of accurately representing the complex implicit relations of *semi-structured tables*. Third, we propose a question decomposition method with two key techniques: (1) semantic alignment between the input question and the derived operation pipeline, and (2) a column-type-aware tagging approach that annotates discrete, continuous, and unstructured columns (e.g., listing [man, woman] for a *sex* column) to enhance data retrieval accuracy (for **C2**). Finally, we introduce a two-stage QA verification mechanism to ensure solution stability. The first stage checks constraints (e.g., non-empty, question-related) to validate the generated operations and their execution results. The second stage provides a confidence score by comparing the original questions with those derived from the final answers (for **C3**).

We summarize our contributions as follows:
(1) We present a novel framework that enables effective and robust *semi-structured table QA* using large language models.
(2) We provide a tree-based representation method for semi-structured tables (HO-Tree), and design basic tree operations in the model to support common QA tasks.

(3) We propose a DFS-based algorithm that combines VLM and heuristic rules to construct HO-Tree from semi-structured table.
(4) We design a question decomposition strategy that ensures semantic alignment with the generated operation pipelines, and introduce a column-type-aware tagging strategy to improve lookup accuracy on relatively large *semi-structured tables*.
(5) We propose a two-stage QA verification mechanism that conducts constraint examinations and compares the pipelines of the origin question and those derived from the final answer.
(6) We curate the *SSTQA* dataset, featuring 102 diverse *semi-structured tables* and 764 representative queries commonly found in real-world scenarios.
(7) We conduct thorough evaluations to verify `ST-Raptor` can effectively tackle the structural and semantic complexities of *semi-structured tables*, resulting in improved QA accuracy and reliability.

2 Preliminaries

In this section, we first explain the typical semi-structured table layouts, followed by the formalization of the *semi-structured table QA* task and a discussion of the limitations of existing approaches.

2.1 Semi-Structured Tables

Semi-structured tables can be far more complex than structured ones due to the combination of diverse table layouts. However, compared with other semi-structured data, they still adhere to stricter layout constraints and may suffer information loss when stored in formats like JSON (e.g., splitting merged cells). To preserve their structure, richer formats like HTML [31] are required. Furthermore, questions over such tables often demand precise layout-aware reasoning (e.g., interpreting layout constraints) and operator grounding (e.g., identifying the intersection of relevant rows and columns).

Core Elements. In a semi-structured table T, a *table cell* is the atomic unit of a semi-structured table (e.g., the intersection of one row and column). A *table header* consists of one or more rows or columns that label the table body. A *merged cell* spans multiple adjacent rows or columns to convey hierarchical information. A *subtable* is a semantically and structurally self-contained table embedded within a parent table with at least one level of nested headers, rows, and internal layout.

Table Layouts. The semi-structured table T (in standard form and ignoring issues likes data cleaning [6]) can be composed of four typical layouts (independent with each other):
(L.1) ***Header-Single-Value.*** The simplest layout pairs a header $H = h_1$ with a single content value $V = v_1$, arranged either vertically or horizontal: $T_t = \{\ H = h_1,\ V = v_1,\ H \rightarrow V\}$. *For instance, in Table 1, the atomic header "Name" is associated with the single value "Albert", demonstrating this minimal semi-structured table layout.*
(L.2) ***Header-Multiple-Values.*** A more prevalent structure in semi-structured table is an atomic or hierarchical header $H = h_1$ accompanied by a list of values $V = [\ v_1, v_2, \ldots, v_n\ ]$. We represent this as $T_t = \{\ H = h_1,\ V = [\ v_1, v_2, \ldots, v_n\ ], H \rightarrow V\}$. *For instance, Table 1 presents an example in which the atomic header "Name" corresponds to multiple values, specifically the list ["Albert", "Tim", "Jack"].*
(L.3) ***Orthogonal-Subtables.*** An orthogonal-subtable layout consists of two or more subtables, whose top-level headers appear at the same level, either horizontally or vertically. The subtables each contain the same number of rows, forming a parallel structure, while the content values of these subtables are weakly related or unrelated (e.g., personal information for employees in two separate departments). Suppose we have n such subtables $T_1, T_2, \ldots, T_n$. We represent their orthogonal combination as $T = \left\{\ H{=}[\ H_1, H_2, \ldots, H_n\ ],\ \ T{=}[\ T_1, T_2, \ldots, T_n\ ],\ \ H_i{\rightarrow}T_i, 1 \le i \le n\right\}$. *For instance, as demonstrated in Table 1, the three atomic-header-single-value tables are combined in Orthogonal-Table layout.*
(L.4) ***Header-Orthogonal-Subtables.***

Table 1: Semi-Structured Table Layouts – Cells marked in blue in examples represent table headers.

Layout	Formal Representation	Example
L1: Header-Single-value	$T_t = \{\ H = h_1,\ V = v_1,\ H \rightarrow V\}$.	Company | TD Tech
L2: Header-Multiple-Values	$T_t = \{\ H = h_1,\ V = [\ v_1, v_2, \ldots, v_n\ ],\ H \rightarrow V\}$.	Name | Mark | Jone | Ray
L3: Orthogonal Tables	$T = \left\{\ H = [\ H_1, H_2, \ldots, H_n\ ],\ T = [\ T_1, T_2, \ldots, T_n\ ],\ H_i \rightarrow T_i,\ 1 \le i \le n\right\}$.	Company | TD Tech Zip Code | 213000 Contacts | Tim
L4: Header-Orthogonal-Tables	$T_t = \left\{H = H^{\mathrm{p}},\ T = [\ T_1, T_2, \ldots, T_n\ ],\ H^{\mathrm{p}} \rightarrow T_i,\ 1 \le i \le n\right\}$	Info Name | Age Mark | 22

Header-Orthogonal-Subtables consists of an atomic or hierarchical header paired with one or more orthogonal-subtables (*(L.3)*), which means all orthogonal-subtables in *(L.4)* share the same header. Formally, let the grouped subtables be $T_1, T_2, \ldots, T_n$. We represent this layout as $T_t{=}\left\{H{=}H^{\mathrm{p}}, T{=}[\ T_1, T_2, \ldots, T_n\ ], H^{\mathrm{p}}{\rightarrow}T_i,\ 1 \le i \le n\right\}$. *For instance, in Table 1, the header "Info" groups two orthogonal subtables (i.e., "Name" and "Age" subtables) that each follows the Header-Multiple-Value layout.*

A subtable $T_{\mathrm{sub}} \subset T$ follows one or a combination of the above layouts. Note that: (1) we do not consider irregular layouts, such as content values presented without corresponding headers [4]; (2) unlike structured tables, common *semi-structured tables* (e.g., Word tables, transaction records) are relatively small (e.g., tables with over 100 rows are already considered large); (3) we assume the tables are error-free, and issues such as table cleaning (e.g., missing value imputation [5]) are beyond the scope of this paper.

Example 2.1. For the bottom-right semi-structured table in Figure 1, we can extract two orthogonal subtables sharing a common header "TD Tech" ($L.4 \rightarrow L.3$). Within the "Employee Info" subtable, we can extract four header-multiple-values subtables ($L.3{\rightarrow}[L2_1, \ldots, L2_4]$). In this way, we can use the formal expression $L.4{\rightarrow}L.3{\rightarrow}[L.4{\rightarrow}\{L.1_1, \ldots, L.1_4\}, L.4{\rightarrow}L.3{\rightarrow}[L.2[1], \ldots, L.2[4]]]$ to recursively traverse the entire structures. Furthermore, the one-to-many relationships between these subtables motivate us to adopt a tree-based strategy to represent *semi-structured tables* (see Section 4).

2.2 Semi-Structured Table QA

Given a semi-structured table T, the QA tasks aim to answer an input question Q expressed in natural language based on T [36].

Definition 1 (Semi-Structured Table QA). *Let T be a semi-structured table with a multi-layered organization, and let Q be a question that may reference one or multiple subtables in T.* Semi-Structured Table QA *is defined as a mapping* $(T, Q) \longmapsto A$, *where*

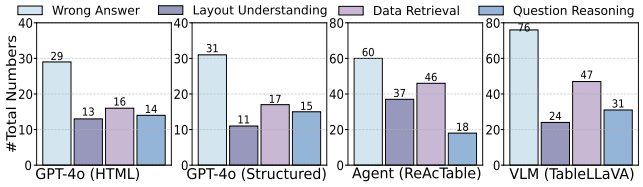

Figure 2: Error Distribution – GPT-4o is evaluated on both HTML and structured (JSON) formats; ReAcTable (NL2SQL) converts *semi-structured tables* into structured representations; while TableLLaVA utilizes LVM to process them as images.

the answer A is derived by identifying the subtables $\{T_{sub} \mid T_{sub} \subseteq T\}$ *relevant to Q.*

QA Tasks. Commonly, QA tasks in this problem often require understanding the layouts of target *semi-structured tables*. Here we showcase three typical QA tasks together with relevant layouts.
(1) ***Numerical Computation.*** The question $Q2$ in Figure **??** requests the age of the oldest employee (*Header-Multiple-Values in L.2*). Correctly answering it involves (i) locating the target column "Age" within the hierarchical header "Info", (ii) extracting all age values from the corresponding records, and (iii) computing the maximum value from the retrieved data. *Such QA requires accurate header identification and data retrieval, upon which we can easily apply basic computational functions to get accurate results.*
(2) ***Information Extraction.*** The question $Q1$ in Figure **??** relies on the "Employee Info" layout (*Header-Orthogonal-Tables in L.4*). To derive the correct answer "2", we must extract the records of employees meeting the rating condition and aggregate them. *Failure to properly interpret the semantic relationship between the merged cell "A+" and its associated employees may lead to a wrong answer.*
(3) ***Summarization.*** The question $Q3$ in Figure **??** requests a summary of the company's basic information (*Orthogonal-Tables in L.3*). Addressing this question involves two critical steps: (i) identifying and extracting the semantically relevant table segments corresponding to the question, and (ii) leveraging the reasoning capabilities of LLMs to generate a coherent summary from the retrieved data. *The key challenge lies in robustly interpreting complex table layouts to ensure precise alignment between the question intent and the extracted data to generate accurate summaries.*

2.3 Limitations of Existing Methods

In this section, we discuss the limitations and challenges of existing approaches that could potentially be adapted to *semi-structured table QA*. As shown in Table 2, these methods can be categorized based on their table representation strategies (e.g., table serialization [37], HTML/JSON [19]) and the question comprehension techniques (e.g.,[33]). Figure 2 illustrates the error distribution when applying typical methods to *semi-structured table QA*.

(Limitation 1) Poor Table Layout Understanding. The failure in layout understanding indicates that the model incorrectly captures structural evidence, primarily due to the limitations of semi-structured table representation. Among existing representation methods, structured table representation (i.e., converting *semi-structured tables* into fully structured formats) leads to major loss of layout information. Alternative approaches (e.g., HTML, JSON, Spreadsheet [37]) employ common serialization strategies that partially preserve structural information in textual form. However, due to the inherent one-dimensional nature of these formats, LLMs face significant challenges in effectively interpreting complex table layouts. In contrast, image-based methods exhibit the lowest layout understanding error among wrong answers (31.58% compared to 35.48% for GPT-4o with structured input, 44.82% for GPT-4o with HTML input, and 61.67% for NL2Code agent), but own relatively low overall accuracy caused by the following two limitations. This observation motivates us to *design a proper semi-structured table representation method that simultaneously stores the structural information and content of* semi-structured tables.

(Limitation 2) Inaccurate Table Data Retrieval. As shown in Figure 2, data retrieval errors constitute a substantial proportion of failures. Specifically, the four methods (i.e., GPT-4o with structured input, GPT-4o with HTML input, NL2Code agent, and vision-language model) exhibit retrieval error rates of 55.17%, 54.84%, 76.67%, and 61.84%, respectively, which are primarily due to their inability to identify question-relevant tabular data. Among these methods, GPT-4o achieves superior performance, attributable to its advanced contextual comprehension capabilities. In contrast, the agent-based approach, which operates on structured table representations using external tools, incurs significant information loss during structural transformation. We observe that (1) these methods lack robust data retrieval mechanisms, and (2) vanilla LLMs struggle to accurately locate target table content, likely due to inherent limitations in semantic understanding. This indicates that *significant improvements remain achievable in accurately locating and retrieving question-relevant content in* semi-structured tables.

(Limitation 3) Question Comprehension Errors. Question comprehension errors occur when a model misinterprets the semantics of a question and consequently retrieves incorrect answers from *semi-structured tables*, often due to inadequate integration of table layout and content. VLMs demonstrate the weakest performance, mainly due to their weak understanding of rich-text images. In contrast, both GPT-4o and agent-based methods demonstrate better performance, benefiting from the advanced reasoning capabilities of LLMs. However, their remaining errors are predominantly attributable to the failure to align the semantics of the input question with complex semi-structured table layouts, motivating us to *design tailored question decomposition and table semantic alignment techniques for semi-structured tables.*

3 ST-Raptor Overview

Architecture. Figure 3 shows the architecture of ST-Raptor, which consists of four main modules: ***(1) Table2Tree*** converts the given semi-structured table into a Hierarchical Orthogonal Tree (HO-Tree), which effectively represents the header / content relationships within the original table (see Section 4.2). Note that the *Table2Tree* module is utilized only once when handling multiple questions related to the same table. ***(2) Question2Pipeline*** transforms complex questions into simpler sub-questions, from which corresponding operation pipeline are generated for each sub-question (see Section 5.1). ***(3) AnswerGenerator*** then executes these operations to obtain intermediate results or produce the final answer (see Section 5.2). ***(4) AnswerVerifier*** adopts a two-stage validation strategy. In the forward stage, it checks whether the execution results

Table 2: Comparison of Relevant Methods for *Semi-Structured Table QA* – More stars indicates better performance.

Representation	Structure Information	Method	Structure Understanding	Data Retrieval	Supported Table Scale	Answer Accuracy	Main Challenge
HTML/JSON/Spreadsheet	✓	NL2SQL	-	-	-	-	Fail to operate the table.
HTML	✓	NL2Code	-	-	-	-	Fail to operate the table.
HTML/JSON	✓	LLM	★	★	★★	★	Fail to understand table structure.
JSON/Spreadsheet	✓	NL2Code	★	★★	★★	★★	Fail to understand table structure.
Structured	✗	NL2SQL / NL2Code	-	★	★★★	★	Structural information loss.
Structured	✗	LLM	-	★	★★	★	Structural information loss.
Structured	✗	Agent	-	★★	★★	★	Structural information loss.
Image	✓	Multimodal LLM	★★	★	★	★★	Fail to process big tables.
HO-Tree	✓	ST-Raptor	★★★	★★★	★★★	★★★	Modeling and operation.

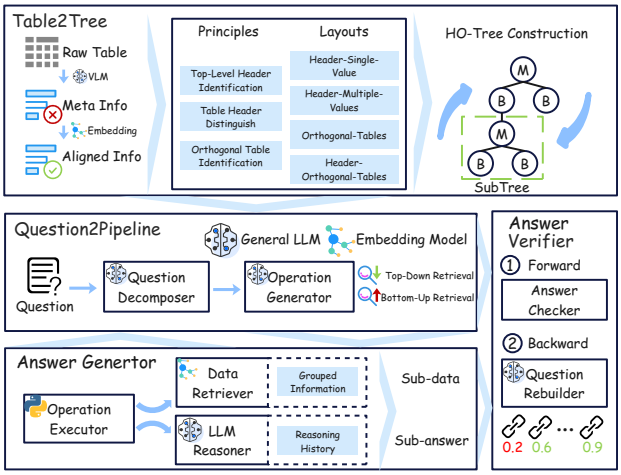

Figure 3: The ST-Raptor Architecture.

are non-empty and consistent with the question; otherwise, the operation is regenerated or terminated early. In the backward stage, similar questions are generated from the output, and their similarity to the original question is used to score answer's reliability (see Section 6).

System Workflow. When a new semi-structured table and its associated questions arrive, *Table2Tree* first preprocesses the table into an HO-Tree and serializes the object into a local file. The *Question2Pipeline* then decomposes each question into subquestions and iteratively interacts with the *AnswerGenerator* to generate answers for each subquestion. The answer to the final subquestion serves as the question answer. The *AnswerVerifier* is involved throughout each step of operation execution, identifying and discarding incorrect intermediate results, based on which ST-Raptor iterates until generating correct answer.

4 Tree Model for Semi-Structured Table Representation

As discussed in Section 2.1, *semi-structured tables* consist of complex combinations of basic table layouts (e.g., hierarchical headers with nested subtables). Accurately performing question answering over such tables remains challenging even for advanced LLMs like GPT-4o (see Section 2.3). Thus, in this section, we study how to effectively represent the layout relationships within *semi-structured tables* to enable accurate question answering.

4.1 Hierarchical Orthogonal Tree

Following the recursive definition in Section 2.1, a semi-structured table composed of layouts $L.1$–$L.4$ (Table 1) can be decomposed into two parts: (1) *Metadata*, which provides high-level semantic abstraction (e.g., headers), and (2) *Data*, which contains the actual content values (e.g., the header "sex" associated with ['female', 'male', 'female']). Both metadata and data components may exhibit hierarchical and orthogonal structures. Given this inherent one-to-many organization, we model them using trees: one for metadata and one for data. In each, nodes store metadata or content values, while edges encode containment or orthogonal relationships.

Definition 2 (Hierarchical Orthogonal Tree (*HO-Tree*)). *Given a semi-structured table T, we model T into HO-Tree that links the metadata with corresponding content values by pointing the leaf node in the $MTree$ to each level of $BTree$, representing the association between metadata and the corresponding table column:*
(1) **Meta Tree (*MTree*).** *It represents the structural information and content of the table's metadata. Each path from the root to a leaf corresponds to an abstract description of a specific column. Nodes in $MTree$ are denoted as $MNode$;*
(2) **Body Tree (*BTree*).** *It represents the structural information and content of the table body. Each node corresponds to a single cell value, each path from the root to a leaf represents a row, and each level of the tree corresponds to a column—aligned with a path in $MTree$. Nodes in $BTree$ are denoted as $BNode$.*
A single HO-Tree is represented as $T = \{MTree = M, BTree = B, M \rightarrow B\}$. Within a semi-structured table, a collection of such HO-Trees may exist, each comprising an $MTree$ and a $BTree$. Moreover, hierarchical containment may exist among these trees, where one HO-Tree can serve as the value of a node within another $BTree$.

Example 4.1. The right part of Figure 4 illustrates an example of a HO-Tree, where "TD Tech" serves as the top-level cell. This cell is stored in a single-node *MTree*, which points to the remaining structure stored in a *HO-Tree*. The subsequent *MTree* contains two *MNode* instances (i.e., "Basic Info" and "Employee Info"), each referencing a distinct *BNode*, with each *BNode* storing a sub *HO-Tree*.

4.2 HO-Tree Construction

Based on the definition, constructing an HO-Tree from a given semi-structured table requires precisely identifying (1) meta-information (table headers), (2) content values, as well as (3) the relationships between subtables. Two main challenges remain. First, headers and content cells can appear in arbitrary positions, making it difficult to

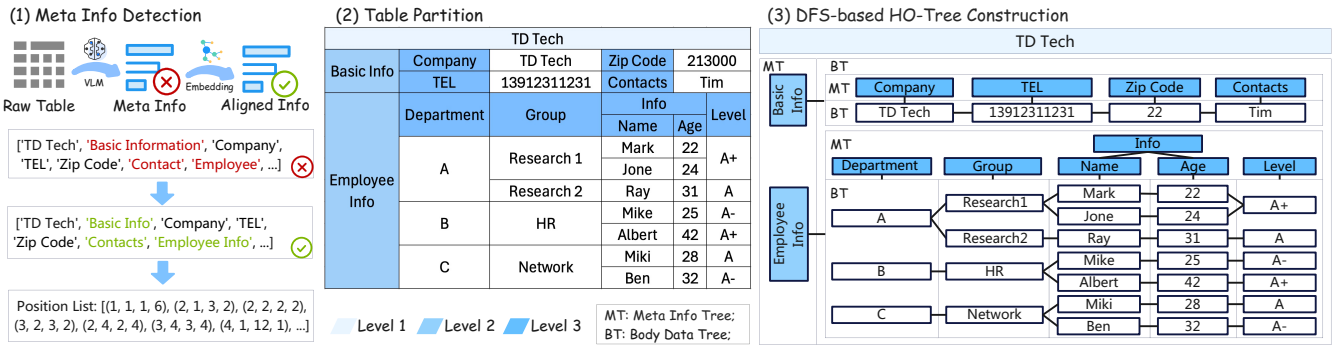

Figure 4: An example of constructing a three-level nested Hierarchical Orthogonal Tree (*HO-Tree*) – Different shades of blue highlight the table's nested levels. For example, in the bottom-right table, the metadata, marked in dark blue, is constructed top-down into a tree of depth two, while the unshaded data section structured into a left-to-right tree of depth five.

Table 3: Table of Atomic Operations.

Operation	Formal Representation	Description
Children	$CHL(V)$	Get children based on the given value.
Father	$FAT(V)$	Get Father based on the given value.
Value	$EXT(V_1, V_2)$	Get the cross of two values.
Condition	$Cond(D, Func)$	Filter values based on the function.
Calculation	$Math(D, Func)$	Calculate result based on the function.
Compare	$Cmp(D_1, D_2, Func)$	Compare values based on the function.
Execute	$Foreach(D, Func)$	Apply function to the given values.
Align	$Align(P, HO\text{-}Tree)$	Operation-Table alignment.
Reason	$Rea(Q, D)$	Reasoning based on LLMs.

distinguish between them. Second, understanding the nested or containment relationships among these cells is non-trivial, especially given the flexible and irregular structure of *semi-structured tables*. Thus, next we introduce the detailed steps in HO-Tree construction.

Algorithm 1: HO-Tree Construction (HOTC)

Input: A Semi-Structured Table T
Output: Extracted HO-Tree $HOTree$

$MetaInfo \leftarrow MetaInfoDetect(T)$;
$T_{list} \leftarrow TablePart(T, MetaInfo)$;
$HOTree_{list} \leftarrow []$;
for T_{sub} *in* T_{list} **do**
  switch $type(T_{sub})$ **do**
    case L_1, L_2, L_4 **do**
      $HOTree_{list}.push\_back(ConsTree(T_{sub}))$;
    case L_3 **do**
      $HOTree_{list}.push\_back(HOTC(T_{sub}))$;
  end
end
return $ConsTree(HOTree_{list})$;

4.2.1 Meta Information Detection

Unlike structured tables with fixed schemas, *semi-structured tables* often exhibit complex and irregular nesting, posing challenges for meta-information extraction. Rule-based approaches fail to capture such nuances, especially when similar structural patterns represent different semantics. Serializing tables (e.g., as images or structured formats) allows prompting models for metadata extraction or format generation. However, LLMs trained on 1D text struggle with 2D tables [23], often exhibiting issues like hallucination and the "Lost in the Middle" effect [27], undermining consistency and reliability.

To overcome these limitations, we adopt a hybrid method that combines rule-based matching with LLM-based reasoning. As shown in Figure 4, given a semi-structured table in Excel format, we first convert it to HTML, render it using a headless browser, and capture a high-resolution screenshot as input image for the VLM. Then, we prompt the VLM to output all possible keys that would be present in a JSON-formatted representation of the table as the candidate meta-information cell values. Subsequently, we calculate similarity scores between candidates and all table cells using embedding-based similarity metrics. Cells exceeding a predefined threshold are identified as meta-information, and their positions guide the subsequent table partitioning process.

4.2.2 Table Partition Principles

We introduce three principles to guide the HO-Tree construction process by interpreting different layouts in *semi-structured tables*:

(Principle 1) Top-Level Header Identification. If a merged cell spans an entire row or column, it is treated as a header in a Header-Orthogonal-Tables layout ($L.4$), and adjacent cells (below or to the right) are interpreted as a subtable.

(Principle 2) Header-Content Differentiation. When both top-aligned and left-aligned headers are present, the one with more cells is selected to construct the $MTree$, while the other is integrated into the $BTree$.

(Principle 3) Orthogonal Table Identification. When Orthogonal-Tables ($L.3$) are detected, we segment them and process each subtable recursively and sequentially.

As illustrated in Figure 4, applying these principles enables table partitioning based on meta-information locations and the recursive construction of HO-Trees for *semi-structured tables*.

4.2.3 DFS-based Tree Model Construction

An HO-Tree is then constructed for each subtable according to its identified layout type: for layouts $L.1$ and $L.2$, the HO-Tree is built directly; otherwise, a recursive DFS is applied. For example, in Figure 4, the vertically structured *semi-structured tables* demonstrate a top-down alignment of metadata and content.

Based on the detected metadata, the table is partitioned into a list of subtables following predefined principles. An HO-Tree

is then constructed for each subtable according to its identified layout type: for layouts $L.1$ and $L.2$, the HO-Tree is built directly; otherwise, a recursive DFS is applied. Take the vertically structured *semi-structured tables* where metadata and content are aligned top-down in Figure 4 as an example: (1) In the $MTree$, each root-to-leaf path represents a column, and the tree grows vertically to reflect vertical relationships among metadata. (2) In the $BTree$, each path represents a row, with horizontally aligned attributes, so the tree grows horizontally. Each leaf node in the $MTree$ points to a corresponding level in the $BTree$, linking metadata to the associated content column.

During DFS backtracking, we model Orthogonal-Tables and Header-Orthogonal-Tables layouts (i.e., $L.3$ and $L.4$). In such cases, a node in the $BTree$ may recursively contain another HO-Tree as its value.

Algorithm 1 outlines the HO-Tree construction process. VLMs identify table headers via the $MetaInfoDetect$ function (Section 4.2.1). Using the extracted metadata and predefined principles, the table is partitioned into subtables through the $TablePart$ function (Section 4.2.2). To address structural complexity, each subtable is transformed into an HO-Tree via the $ConsTree$ function (Section 4.2.3) and recursively merged using depth-first search to reconstruct the full structure of the original semi-structured table.

Example 4.2. Figure 4 illustrates the construction of a three-level nested HO-Tree. The "Basic Info" subtable is initially misidentified by the VLM as "Basic Information" and corrected by alignment. Metadata is used to identify the $L.4 \rightarrow L.3$ layout, guiding the partition of the "Basic Info" subtable into $MNode$ and a sub-$HO\text{-}Tree$ (constructing $L.4$). Finally, the leaf nodes of the $MTree$ in subtree link to corresponding levels of the $BTree$ (constructing $L.3$).

5 Pipeline-based Question Answering

With the HO-Tree model in place, it poses three key challenges in question answering over *semi-structured tables*. First, such questions often require multi-hop reasoning rather than one-shot retrieval, and unlike NL2SQL tasks, lack a standardized operation set for pipeline construction. Second, answering necessitates hybrid traversal strategies (e.g., left-to-right, top-down, bottom-up) to locate relevant content. Third, the large number of cells in some *semi-structured tables* complicates the precise identification of question-relevant information.

To address these challenges, we propose a pipeline-based QA strategy centered on a set of tree-specific operations that cover most QA scenarios: (1) a question decomposition method for complex multi-hop queries, (2) an operation generation mechanism with parameter-content alignment, and (3) a column-type-aware tagging mechanism that annotates columns based on data characteristics, enabling efficient and accurate retrieval from large, complex tables.

5.1 Basic Operations over HO-Tree

We design a suite of atomic operations to enable structured and interpretable QA over the $HO\text{-}Tree$. These operations support both precise tree traversal and auxiliary tasks correspond to common semi-structured table sub-tasks. The operations fall into four categories: *(1) Data Retrieval Operations*, which retrieve relevant values from the tree; *(2) Data Manipulation Operations*, which process or transform retrieved data; *(3) Alignment Operations*, which align operation parameters with table content; *(4) Semantic Reasoning Operations*, which invoke LLMs for contextual inference.

For any user question, we first decompose it into one or more sub-questions, each resolved through a sequence of operations to retrieve relevant information or derive the answer. Ideally, the generated operation pipeline includes: (1) retrieval to collect non-redundant data, (2) manipulation to structure the data into a model comprehensible form, (3) alignment to ensure question-content alignment, and (4) reasoning to produce the final answer. In worst-case scenarios, the model may retrieve nearly the entire table and rely heavily on reasoning, highlighting the need for fine-grained, modular operation design.

Data Retrieval Operation. These operations are responsible for extracting relevant table content from the $HO\text{-}Tree$ based on arguments derived from the user question.

- ***Children Retrieval*** $(CHL(V))$ This operation retrieves all successor nodes of any node whose value matches V. If multiple such nodes exist, each set of successors is returned separately. Use the $HO\text{-}Tree$ in Figure 4 as an example, $CHL(Basic\ Info)$ returns the $HO\text{-}Tree$ containing the company information.
- ***Father Retrieval*** $(FAT(V))$ This operation is used to obtain the set of ancestor nodes of a given tree node. For instance, in Figure 4, $FAT(Department)$ would return the $HO\text{-}Tree$ containing the employee information.
- ***Value Retrieval*** $(EXT(V_1, V_2))$ This operation is used to find nodes in a certain layer of $BTree$ pointed to by a $MTree$ leaf node, while giving them a common ancestor $BNode$ as a filtering criterion. Specifically, one of V_1 and V_2 needs to be a $MNode$ value, and the other needs to be a $BNode$ value. Assume that V_1 is the $MNode$ value and V_2 is the $BNode$ value, the operation returns a set of node values that satisfy the following: (1) The $BNode$ is in the $MTree$ column V_1 (2) $BNode$ with value V_2 is an ancestor of the $BNode$ in $MTree$ column.

Data Manipulation Operation is used to perform specific operations on the data, including filtering based on conditions, performing calculations, and making comparisons.

- ***Condition*** $(Cond(D, Func))$ filters a data set D using a predicate function $Func$ and returns the filtered values or a new $HO\text{-}Tree$. For instance, $Condition(EXT(Lily, Grade), def(x) : return\ x < 60)$ retrieves Lily's grades that are less than 60.
- ***Calculation*** $(Math(D, Func))$ applies a numerical computation function $Func$ over a data set D. For instance, $Math(CHL(Price), def(x) : return\ sum(x))$ returns the sum of the column "Price".
- ***Compare*** $(Cmp(D_1, D_2, Func))$ compares two data sets D_1 and D_2 using a function $Func$, and returns the boolean result. For instance, $Compare(EXT(Lily, Grade), EXT(Cindy, Grade), def(x1, x2) : return\ x1 > x2$ returns the truth value of the statement "Lily's grade is greater than Cindy's".
- ***Execute*** $(Foreach(D, Func))$ applies a function $Func$ to each element in data set D and returns the resulting set. For instance, $Foreach(EXT(Lily, Grade), def(x)\ return\ x-10)$ retrieves all Lily's grades and returns the values after minus ten.

Alignment Operation aims to ensure consistency between operation parameters and the content of the $HO\text{-}Tree$.

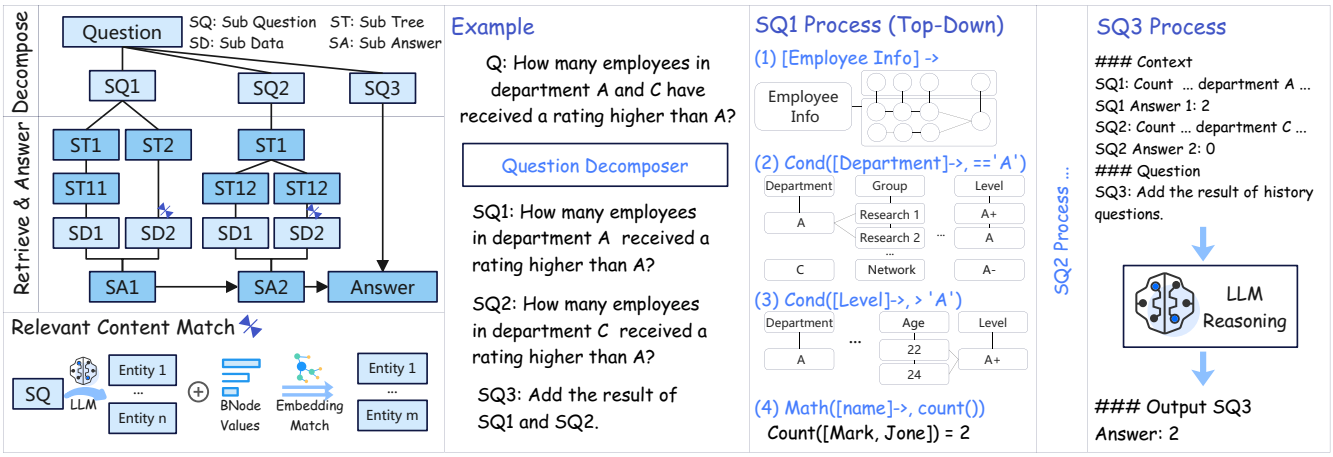

Figure 5: Question Decomposition and Pipeline Generation.

• ***Align*** *(Align(P, HO-Tree))* This operation aligns the parameters P in a given operation with the nodes in the *HO-Tree* using an embedding-based similarity model [39]. The process involves computing embeddings for the parameters and table content:

$$E_a = Embed(a_1, a_2, \ldots, a_n) \tag{1}$$

$$E_c = Embed(c_1, c_2, \ldots, c_m) \tag{2}$$

$$SimMatrix = CosSim(E_a, E_c) \tag{3}$$

where a_i represents the i-th parameter and c_j denotes the content of the j-th node in the tree, with m representing the total number of nodes. Cosine similarity (*CosSim*) is used to identify the most semantically aligned table node for each parameter, which is then used for downstream operations.

Example 5.1. For the operation $CHL(Identification)$, if the meta-information of the table is ['ID', 'Name', 'Age'], the alignment operation $Align(Identification, HOTree)$ attempts to output "ID" to align the operation parameter to the table content.

Semantic Reasoning Operation leverages LLMs for high-level reasoning over retrieved data.

• ***Reason*** *(Rea(Q, D))* This operation takes the original question Q and the data D obtained from prior operations, and uses an LLM to generate the final answer. In cases where the question is decomposed into multiple sub-questions, semantic reasoning can also be used to aggregate intermediate answers into a final response.

5.2 Question Decomposition and Pipeline Generation

Figure 5 illustrates the overall process of question decomposition, operation generation, and step-by-step data retrieval via both top-down and bottom-up strategies.

Step 1: Question Decomposition. As shown in Figure 5, upon receiving a user question, we first utilize the semantic understanding capabilities of an LLM to decompose complex multi-hop questions into multiple simpler, single-step sub-questions. Specifically, we prompt the LLM with the input question, sampled table content as semantic supplements, and example decomposition cases that are dynamically retrieved based on question similarity to generate sub-questions. These sub-questions are often interdependent: some can be directly resolved using the *HO-Tree*, while others rely on intermediate results produced by earlier sub-questions. For example, a question such as "What is the total salary for 2021 and 2022?" can be decomposed into three sub-questions: (1) retrieve the 2021 salary, (2) retrieve the 2022 salary, and (3) compute the sum of the two. For table operation generation, we prompt the LLMs with a predefined set of operations and illustrative examples.

Step 2: Relevant Data Retrieval. Each sub-question is then answered independently through a combination of top-down and bottom-up retrieval. `ST-Raptor` always starts with top-down retrieval and switches to bottom-up retrieval when the former fails. The left section of Figure 5 shows this iterative process. For each sub-question, the LLM first generates an operation statement based on the sub-question content and the meta-information extracted from the *HO-Tree*. Executing this operation yields intermediate results (i.e., sub-trees), which are then used to inform subsequent operation generation.

We employ the `Align` operation for two purposes: (1) Operation-Table Alignment, applied upon each operation's generation to align sub-questions with the corresponding table content, and (2) Relevant Content Match, invoked when the number of data nodes is large, to extract key entities from the question and constrain the search space within the HO-Tree.

Step 3: Answer Generation Once the relevant sub-data is retrieved, each sub-question is resolved via the `Reason` operation, and all sub-answers are aggregated to produce the final answer.

5.3 Relatively Large Table QA Enhancement

The abundance of available *BNodes* poses challenges in selecting accurate operation parameters. To address this, we introduce a data grouping strategy that leverages inherent structural and semantic features, combining top-down grouping based on data characteristics with parallel grouping via tree structure traversal.

1. Characteristic-based Grouping. The top-down grouping process is performed during HO-Tree construction, clustering data based on column characteristics. As shown in Figure 6, column values are classified by data type (e.g., Numeric, Datetime, Unstructured String) and then further categorized as follows: ***Discrete*** — columns with a limited set of values (e.g., grades ['A', 'B', 'C', 'D'] or binary options [Yes, No]); ***Continuous*** — columns with numeric values spanning a range (e.g., height or temperature); and

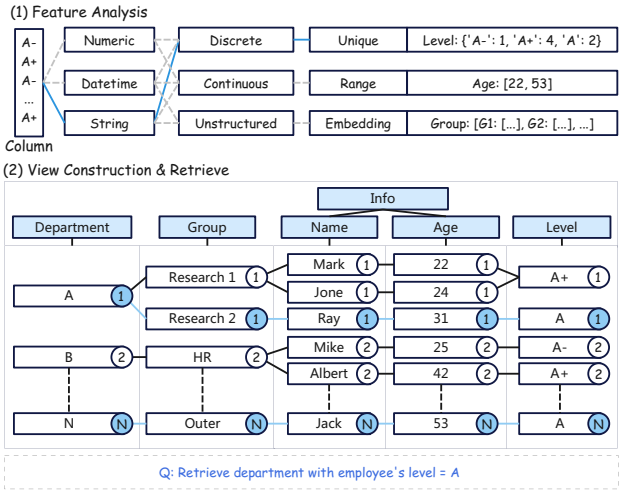

Figure 6: Characteristic-based Grouping Mechanism

Unstructured — columns with free-form text (e.g., comments or descriptions). A rule-based classifier performs this categorization to reduce complexity in tables with numerous similar cells. With clearly defined rules, it achieves near-perfect accuracy.

Each category is then grouped using tailored strategies: (1) ***Discrete*** values are clustered by exact matches (e.g., students with Grade A); (2) ***Continuous*** values are sorted and partitioned into fixed intervals to preserve numeric order (e.g., stress levels by range); and (3) ***Unstructured*** values are grouped via embedding-based clustering to capture semantic similarity.

As shown in Figure 6, a column with values like Grade (A+, A, A-, ...) is classified as ***Discrete***, allowing identical values to be grouped. Queries such as "fetch all students with Grade A" can then be resolved efficiently within the corresponding group.

2. Group-based Data Retrieval. The parallel-direction grouping leverages structural relationships within the *HO-Tree*. After locating the target node(s) based on content, the tree is traversed to retrieve related information by: (1) searching *upwards* for ancestor nodes and (2) searching *downwards* for descendant nodes.

This bi-directional traversal allows for the reconstruction of a minimal sub-tree that contextualizes the target node, revealing the complete information associated with the same entity (e.g., retrieving all attributes of a specific student or transaction).

6 Two-Stage QA Verification

Robust validation mechanisms are essential for the reliability of *semi-structured table QA*, particularly given that existing solutions (e.g., NL2SQL [24, 41]) often lack comprehensive answer verification. The main challenge arises from the fact that, different from structured table QA, the retrieved result cells in *semi-structured table QA* often exhibit complex layouts and may be derived through multi-step lookup processes, which is tricky for general LLMs to verify. To address this problem, we propose a two-stage verification framework that integrates both forward and backward validation to enhance the robustness and trustworthiness of the final answers.

Forward Verification. The first stage focuses on validating the correctness of intermediate operations and execution results during the QA process. Specifically, at each step of generating and executing operations, we verify whether the parameters produced by LLM are consistent with the actual contents of the table (e.g., when asking for company information and the given data only contain employee information, we stop the process). This involves directly matching generated parameters against table cells to ensure semantic and syntactic alignment.

After executing each operation over the HO-Tree, the general LLM evaluates whether the resulting data sufficiently answers the corresponding sub-question. If the result is found to be inadequate, a new operation statement is generated to continue the reasoning process. Notably, real-world queries often lack sufficient information to be answered directly from the table, which may cause the model to hallucinate incorrect answers. To mitigate this issue, the forward verification mechanism allows the model to halt the pipeline and return an "unanswerable" signal when it detects that the answer cannot be reliably inferred. Furthermore, when operation statements are found to misalign with the table content, the system triggers a regeneration process to refine and correct the statements, thereby improving both the data retrieval accuracy and efficiency.

Backward Verification. In the second stage, we evaluate the correctness of the generated operation pipeline by verifying it against alternative reasoning paths. Notably, different questions over the same semi-structured table may yield identical answers via distinct yet similar pipelines (e.g., "Who is the highest-paid employee?" vs. "Who leads the technical department?"). Leveraging this property, we perform backward verification by generating alternative questions with the same answer, deriving corresponding pipelines, and measuring their similarity to the original. The average similarity serves as an implicit indicator of pipeline and answer reliability. Specifically, we employ few-shot learning, retrieving question-similar examples to guide the generation of alternative questions to solve the problem that questions with the same answers may show tremendous difference in their operation pipeline.

7 Experiments

7.1 Experiment Setup

LLMs. We use InterVL2.5 26B [11] as the vision language model, Deepseek-V3 [14] as the general LLM, and Multilingual-E5-Large [39] as the semantic embedding model.

Evaluated Methods. The evaluated methods include: (1) NL2SQL. OpenSearch-SQL [41] employs a dynamic few-shot learning strategy (Query-CoT-SQL) and introduces an SQL-Like intermediate language to optimize reasoning chains. (2) Foundation Model. GPT-4o [30], the cutting-edge LLM developed by OpenAI, and DeepSeekV3 [14], a strong Mixture-of-Experts LLM developed by DeepSeek-AI is evaluated. (3) Fine-tuning based Methods. TableLLaMA [44] is a fine-tuned version of LLaMA-2 7B tailored for tabular data processing across eight table-specific tasks. The model handles various table types, including Wikipedia tables and spreadsheets. TableLLM [45] is a 13B-parameter LLM designed for tabular data manipulation tasks, which is fine-tuned on a diverse mix of table-centric datasets in processing document tables and spreadsheets, making it well-suited for real-world office scenarios. (4) Agent based Methods. ReAcTable [46] is an agent-driven approach that integrates reasoning and action-based decision-making for

Table 4: Characteristics of SSTQA Benchmark.

Dataset	Nesting Depth	Merge Ratio	Cell Count	Avg. length of Contents
WikiTQ	1.2970	0.0091	178.3564	1.9568
TEMPTABQA	2.0000	0.1780	44.8350	3.6696
INFOTABS	2.0000	0.0548	23.6683	2.0769
SSTQA	2.5196	0.0544	147.4608	2.7287

table question answering. It iteratively generates operations, updates the table, and constructs a reasoning chain as a proxy for intermediate thought processes through prompting LLMs and in-context learning. TAT-LLM [48] extracts relevant segments from the context, generates logical rules or equations, and then applies these rules or executes the equations to derive the final answer through LLM prompting. (5) VLM based Methods. TableLLaVA [47] extends the training of LLaVA-7B/13B on 150K table recognition samples, allowing the model to align table structures and elements with textual modality. We choose the 7B version in our experiment. mPLUG-DocOwl1.5 [20] is a fine-tuned VLM with 8B parameters. It incorporates a spatial-aware vision-to-text module designed to represent high-resolution, text-rich images while preserving structural information and reducing the length of visual features.

Input Formats. NL2SQL and agent-based methods take structured tables as input, typically stored in databases or CSV files. Fine-tuning-based approaches operate on structured tables in Markdown format, while VLM-based methods accept table images as input. Foundation models generally utilize the HTML representation of tables. `ST-Raptor` currently supports Excel as the input format and is compatible with all lossless table representations like HTML.

Benchmarks. We evaluate nine baselines and `ST-Raptor` on three benchmarks: (*i*) WikiTQ, featuring Wikipedia tables with complex natural language questions, (*ii*) TempTabQA, targeting on temporal question answering over semi-structured tables, and (*iii*) SSTQA, our proposed dataset detailed in Section 7.2. Since the table formats in other datasets do not fully adhere to the definition of semi-structured tables, we select a subset of tables that meet the criteria and denote them as WikiTQ-ST and TempTabQA-ST, respectively.

7.2 SSTQA Benchmark

Existing semi-structured table datasets face two key limitations: (1) they consist of small, structurally simple tables that fail to evaluate a model's capacity to comprehend complex *semi-structured tables*; and (2) their queries are misaligned with practical applications, limiting real-world utility. For example, although WikiTQ [31] includes a large number of semi-structured tables from Wikipedia, these tables typically exhibit simple layouts with merged cells and are converted into structured formats as part of the benchmark preprocessing. Meanwhile, TempTabQA [17] consists solely of shallowly nested tables with fewer than five columns, lacking the structural complexity commonly observed in real-world datasets.

To fill the gap, we introduce SSTQA, a dataset specifically designed to evaluate a model's ability to conduct **S**emi-**S**tructured **T**able **Q**uestion **A**nswering task in real-world scenarios. As shown in Table 4, WikiTQ has the highest cell count but the shallowest nesting depth. In contrast, SSTQA exhibits the deepest nesting and the relatively large table size among the existing semi-structured table benchmarks.

Table 5: Overall Performance Comparison.

Methods	WikiTQ-ST	TempTabQA-ST	SSTQA	
	Acc	Acc	ROUGE-L	Acc
OpenSearch-SQL [41]	38.89	4.76	23.87	24.00
TableLLaMA [44]	35.01	32.70	26.71	40.39
TableLLM [45]	62.40	9.13	2.93	7.84
ReAcTable [46]	68.00	35.88	7.49	37.24
TAT-LLM [48]	23.21	61.86	19.26	39.78
TableLLaVA [47]	20.41	6.91	5.92	9.52
mplug-DocOwl1.5 [20]	39.8	39.80	28.43	29.65
GPT-4o [30]	60.71	74.83	43.86	66.45
DeepSeekV3 [14]	69.64	63.81	46.17	63.22
ST-Raptor (Ours)	**71.17**	**77.59**	**52.19**	**72.39**

Data Collection. The 102 tables in SSTQA are carefully curated from over 2031 real-world tables coverage across 19 representative real scenarios (e.g., administrative and financial management) by considering tables featuring semi-structured formats, such as nested cells, multi-row/column headers, irregular layouts, which ensures the representativeness both in structure and information.

For question-answer pair generation, we employ a two-stage approach. First, we augment the question set by extracting information from tables as answers, then generating corresponding questions to enhance QA pair alignment. Second, we sample question templates and prompt a LLM to generate open-ended question-answer pairs based on the table and template. To ensure data quality, we implement a two-step verification process. Initially, a LLM validates the alignment between tables, queries, and answers. This is followed by manual inspection to verify answer correctness by 11 professional annotators, which results in a high-quality dataset of 764 meticulously curated table-based QA pairs.

Table Complexity. We categorize table difficulty based on a weighted combination of three key features: (i) nesting depth (0.5), (ii) structural irregularity, including the number of header rows and column spans (0.3), and (iii) average cell content length (0.2). After z-score normalization and feature aggregation, SSTQA tables are grouped into 59 simple, 33 medium, and 10 hard instances.

Table + QA Complexity. Table-question difficulty varies with the combination of table structure and query complexity. Accordingly, we categorize Table+QA tasks into three levels: (i) simple, where answers can be directly retrieved from the table; (ii) medium, requiring logical inference or conditional operations; and (iii) hard, where answers are not explicitly present and demand semantic reasoning. We obtain 299 simple, 284 medium, and 178 hard cases. The corresponding experimental results are presented in section 7.3.

Evaluation Metrics. We adopt two primary evaluation metrics: Answer Accuracy (Acc), following prior work [23, 44], and ROUGE-L to accommodate summarization-style questions in SSTQA. To address the limitations of exact string matching, we further employ general-purpose LLMs to compare model predictions and ground-truth answers, enabling a more nuanced evaluation.

7.3 Overall Performance Comparison

We conduct a comprehensive evaluation of `ST-Raptor` against nine state-of-the-art table question answering (QA) methods, spanning five technical paradigms, i.e., NL2SQL, fine-tuning based methods, agent based methods, VLM based methods, and foundation LLMs. The general accuracy of these methods across three *semi-structured table QA* benchmarks is shown in Table 5. Then, we further classify

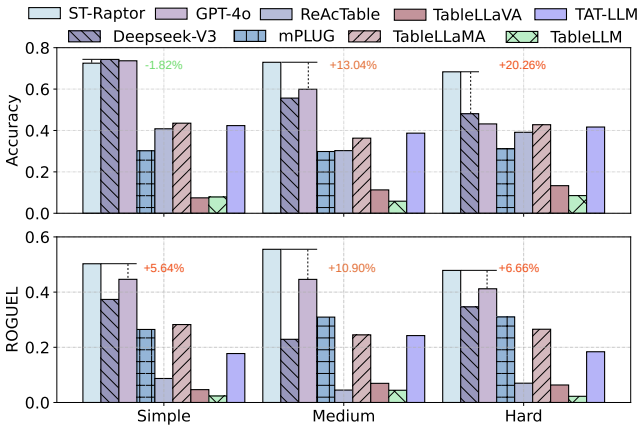

Figure 7: Evaluation Results under Different Table Difficulty.

the difficulty of *semi-structured tables* into Simple, Medium, and Hard tiers, and visualize the accuracy variation upon different table difficulties, which is shown in Figure 7.

s **Overall Accuracy.** Experiments show that ST-Raptor consistently outperforms all nine baselines across the evaluated WikiTQ-ST, TempTabQA-ST and SSTQA benchmarks. Table 5 shows ST-Raptor achieves the highest accuracy, exceeding the second-best method by 10.23% on SSTQA benchmark. This consistent outperformance can be attributed to three folds.

First, ST-Raptor leverages the HO-Tree to represent *semi-structured tables*, enabling explicit structural modeling while decoupling layout understanding from question answering. This design allows the general-purpose LLM to operate without directly parsing complex table layouts. Instead, given JSON-formatted header information, ST-Raptor generates atomic operations and executes them on the HO-Tree for relevant data retrieval. In contrast, most methods, excluding vision-language models (VLMs) which can directly perceive layout information from table images, struggle to capture two-dimensional semantics using linear text representations.

Second, ST-Raptor incorporates a novel question decomposition mechanism that breaks complex queries into simpler sub-questions, followed by precise operation-table alignment. This improves both operation generation accuracy and execution reliability, thereby improving overall question answering performance.

Third, ST-Raptor dynamically combines top-down and bottom-up retrieval strategies based on question characteristics, enabling robust handling of diverse *semi-structured table QA* scenarios. When the top-down retrieval fails or a question lacks explicit header references, bottom-up retrieval is employed. This flexible approach allows the system to effectively navigate complex table structures, outperforming other methods in challenging scenarios.

Additionally, we observe performance increase (around 3%) even on the simple tables of WikiTQ-ST and TempTabQA-ST. For WikiTQ, where the majority of tables are fully structured, ST-Raptor outperforms the second-best model by around 2%. This modest improvement reflects ST-Raptor's specialization for semi-structured tables with complex nested hierarchies. In contrast, on TempTabQA, where all tables are semi-structured but exhibit only shallow nesting and small sizes, ST-Raptor achieves around 3% improvement over the second-best approach. While our model could effectively models such structures, the overall retrieval pipeline remains relatively simple, limiting the performance gap over LLMs like Deepseek-V3 which can directly interpreting relevant information.

Meanwhile, the experimental results highlight significant performance variations among the methods. NL2SQL-based approaches perform the worst on the SSTQA and TempTabQA dataset but achieve better results on WikiTQ, as the SQL generation paradigm is ill-suited for non-relational data, making it ineffective for *semi-structured tables*. TableLLM ranks second-lowest on SSTQA due to two main limitations: (1) its training is restricted to structured datasets, reducing its generalizability to semi-structured formats, and (2) it struggles with large-scale tables and complex layouts due to limited context length and one-dimensional semantic reasoning. Its improved performance on WikiTQ can be attributed to task-specific fine-tuning on this dataset. Agent-based methods perform better on SSTQA due to their integration of external tools, but they fail to operate directly on semi-structured tables. The transformation of semi-structured data into structured formats results in the loss of critical layout information, reducing their effectiveness. TAT-LLM exhibits unexpectedly strong performance on the TempTabQA dataset. We attribute this to its fine-tuning on a large volume of financial data, which shares similarities with the temporal question types prevalent in TempTabQA, thereby contributing to its effectiveness. Vision-language models excel at layout recognition through visual encoding but underperform in text-dense scenarios due to limited textual comprehension, especially in tables requiring strong semantic understanding. Foundation models, while not explicitly designed for table-related tasks, achieve the second-best performance on SSTQA dataset. This is attributed to their robust contextual reasoning and semantic interpretation abilities, enabling accurate answer inference especially when tabular layout understanding is secondary to interpreting textual content.

Accuracy under Different Table Difficulties. We categorize tables in SSTQA benchmark into three levels of difficulty (i.e., Simple, Medium, Hard) by layout complexity and content length. Figure 7 presents the comparative performance evaluation across these difficulty levels. Three key observations emerge from our analysis.

First, both ST-Raptor and foundation models exhibit progressively decreasing performance as table difficulty increases, underscoring the inherent challenges posed by complex layouts and large-scale *semi-structured tables*. In contrast, other methods demonstrate only marginal performance variation across difficulty levels. We posit that these models primarily address questions with less table structure comprehension. Consequently, performance differences among them are largely driven by architectural variations rather than structural modeling capabilities.

Second, although ST-Raptor shows a modest performance decline on hard-level tables, it consistently outperforms all methods by a substantial margin (e.g., exceeding the second-best model by over 20% on the SSTQA dataset). The reasons are three-fold: (1) the hierarchical HO-Tree representation, which facilitates efficient processing of large tables; (2) the question decomposition mechanism, which simplifies complex queries into tractable sub-questions; and (3) the operation-table alignment strategy, which ensures accurate and context-aware data retrieval.

Third, performance differences across models are less pronounced. This can be attributed to three key factors: (1) the smaller table

Table 6: Analysis of Table + QA Difficulty on SSTQA.

Methods (Acc)	Simple	Medium	Hard
DeepseekV3	92.94%	61.83%	47.19%
GPT-4o	86.64%	59.75%	43.26%
ST-Raptor	**93.97%**	**62.66%**	**58.43%**

sizes, which reduce structural complexity; (2) the predominance of semantically driven questions that require less explicit layout reasoning; and (3) the dataset's focus on temporal question answering within a single scenario, which limits question diversity and diminishes the impact of advanced structural modeling.

Analysis on Table + QA Difficulty. We categorize the Table+QA tasks into three difficulty levels. As shown in Table 6, ST-Raptor outperforms all baselines across these levels. While both foundation models and ST-Raptor perform well on simple cases, accuracy drops as difficulty increases. Notably, ST-Raptor demonstrates superior performance on hard cases, attributed to its HO-Tree-based representation and operation-pipeline-driven QA strategy.

We categorize Table+QA tasks into three difficulty levels (Table 6). ST-Raptor consistently outperforms all baselines across these levels. While both foundation models and ST-Raptor perform well on simple cases, accuracy drops as difficulty increases. Notably, ST-Raptor excels on hard cases, benefiting from its HO-Tree representation and operation-pipeline-driven QA strategy.

7.4 Fine-grained Analysis

In this section, we discuss the quality of meta-information detection, analyze question answering latency, and examine the impact of pipeline errors.

Quality of Meta Information Detection. We evaluate the Table2Tree module's accuracy in converting semi-structured table into HO-Tree. Experiment results show that the untuned VLM achieves 93.14% on SSTQA, 94.32% on TempTabQA and 92.31% on WikiTQ, which is sufficiently high for accurate HO-Tree construction.

Analysis of Backward Verification.

To assess the potential negative impact of generating suboptimal question alternatives on question answering accuracy, we quantify the number of such bad alternatives and their corresponding answers on the SSTQA dataset. Experimental results show a false negative rate of 4.78% under the few-shot learning setting, indicating that misjudgments in backward verification have minimal impact on table QA performance.

Latency Analysis. The runtime of ST-Raptor is primarily influenced by the cost of accessing the LLM, largely due to network latency. As ST-Raptor performs question answering via pipeline-based operation generation, the runtime per query is inherently unstable. ST-Raptor requires around 30 seconds per question (ignoring bias caused by factors like network communications), with 2.89 pipeline operations on average. This is substantially faster than the agent-based method, which incurs higher latency due to a greater number of operations with more API calls, and slightly slower than the fine-tuning approach, which benefits from local deployment and direct reasoning.

Effects of Pipeline Mistakes. Pipeline Mistakes Analysis. Mistakes in the ST-Raptor pipeline primarily arise from two sources. First, mistakes in meta-information detection by the VLM can lead to incorrect HO-Tree representations, resulting in more complex data retrieval paths (e.g., locating related data dispersed across different subtrees) and reduced overall efficiency (e.g., from 10 to 40 seconds). Second, semantic misinterpretations by the LLM, such incorrectly splitting a combined address-phone entry into separate fields, can trigger unnecessary lookups and potentially yield incorrect answers, leading to additional verification and iteration, and thereby diminishing efficiency.

Table 7: Ablation Study on ST-Raptor Modules

Model	SSTQA	
	Acc	ROUGE-L
Full Model (DeepseekV3)	72.39%	52.19%
GPT-4o	62.12%	43.86%
DeepseekV3	62.26%	46.17%
w/o Table2Tree	57.24%(-15.15%)	41.55%(-10.64%)
w/o Question Decomposition	68.06%(-4.33%)	48.09%(-4.10%)
w/o Operation-Table Alignment	71.07%(-1.32%)	50.86%(-1.33%)
w/o Data Manipulation Operation	65.09%(-7.30%)	47.13%(-5.06%)
w/o Answer Verifier	66.10%(-6.29%)	47.46%(-4.73%)

7.5 Ablation Study on ST-Raptor Modules

In this section, we perform an ablation study on ST-Raptor from five perspectives. Results are reported in Table 7.

Without Table2Tree. We disable the Table2Tree module and instead apply ST-Raptor directly to raw *semi-structured tables*, which evaluates the importance of explicit table layout modeling. The removal leads to the most significant degradation (an absolute accuracy drop of 15.15%), demonstrating the critical role of HO-Tree-based structural representation in handling complex *semi-structured tables*. This also highlights that foundation models alone struggle to capture intricate layout semantics without explicit structural guidance.

Without Question Decomposition. We remove the question decomposition module, requiring ST-Raptor to process complex queries in a single step. This results in a 4.33% accuracy drop, confirming the necessity of decomposition for effective multi-hop reasoning. Without decomposition, the ST-Raptor fails to isolate intermediate steps, leading to compounding errors in reasoning chains.

Without Operation-Table Alignment. We omit the operation-table alignment mechanism to test whether the LLM in ST-Raptor can inherently align operations with table content. A 1.32% performance decline is observed, indicating that while LLMs possess semantic reasoning ability, explicit alignment could still improve execution precision. This suggests that structural grounding remains beneficial even for advanced models with strong language understanding capabilities.

Without Data Manipulation Operations. We restrict ST-Raptor to data retrieval, alignment, and reasoning operations, disabling data manipulation functions. This leads to a 7.30% accuracy drop, underscoring the frequent necessity of manipulation operations and validating the completeness of our atomic operation set. Many questions inherently require operations such as filtering and calculation, which cannot be bypassed through reasoning alone.

Without Answer Verifier. To evaluate the impact of self-verification, we remove the answer verifier module. Accuracy drops by 6.29%, suggesting that the verifier plays a vital role in detecting and correcting execution errors, thereby enhancing output reliability. This

Table 8: Case Study on SSTQA Dataset.

Table Id	Table Difficulty	Layout Representation	Question	TableLLaMA	ReAcTable	mPLUG-DocOwl1.5	GPT-4o	ST-Raptor
5	Simple	$L.4 \rightarrow L.3 \rightarrow [L.2_1, \ldots, L.2_{12}]$	Summarize the reimbursement activities of Tian Xiaohong.	✗	✗	✗	✗	✗
19	Simple	$L.4 \rightarrow L.3 \rightarrow [L.2_1, \ldots, L.2_3]$	What are the components of employee compensation?	✗	✗	✓	✓	✓
15	Simple	$L.4 \rightarrow L.3 \rightarrow [L.2_1, L.2_2, L.2_3]$	What documents must a Continuity and Availability Planner submit to the IT Service Management Committee?	✗	✓	✓	✓	✓
20	Simple	$L.4 \rightarrow L.3 \rightarrow [\{L.4 \rightarrow L.3 \rightarrow [L.2_1, \ldots, L.2_6]\}_1, \{L.4 \rightarrow [L.2_1, \ldots, L.2_6]\}_2]$	What are the categories of variable manufacturing overhead?	✓	✗	✗	✓	✓
4	Medium	$L.3 \rightarrow [\{L.4 \rightarrow [L.2_1, L.2_2, L.2_3]\}_1, \ldots, \{L.4 \rightarrow [L.2_1, L.2_2, L.2_3]_6\}]$	How many items are there in drawing technology?	✗	✗	✗	✗	✓
95	Medium	$L.4 \rightarrow L.3 \rightarrow [L.2_1, \ldots, L.2_8]$	Which employees in the table have 18 years of service?	✗	✗	✗	✗	✓
100	Medium	$L.4 \rightarrow \{L.4 \rightarrow L.3 \rightarrow [L.1_1, L.1_2, \ldots, L.1_{34}]\}$	What is the net cash flow generated from investing activities?	✓	✓	✓	✗	✓
87	Medium	$L.4 \rightarrow L.3 \rightarrow [L.2_1, \ldots, L.2_{10}]$	What are the evaluation criteria for work attitude?	✗	✗	✗	✓	✓
1	Hard	$L.4 \rightarrow [L.1, \{L.4 \rightarrow \{L.3 \rightarrow [L.1_1, L.1_2]\}\}_1, \{L.4 \rightarrow \{L.3 \rightarrow [L.2_1, \ldots, L.2_4]\}\}_2, \ldots]$	How many secondary indicators are included under the performance metric's efficiency indicators?	✗	✗	✗	✗	✓
10	Hard	$L.3 \rightarrow [\{L.4 \rightarrow [L.2_1, L.2_2]\}_1, \ldots, \{L.4 \rightarrow [L.2_1, L.2_2]_4\}]$	How many phases are included in the Change Phase Code Table?	✗	✗	✗	✓	✓
91	Hard	$L.4 \rightarrow L.3 \rightarrow [L.2_1, \ldots, L.2_8]$	What are the beginning and ending balances of total assets?	✗	✗	✗	✓	✓
30	Hard	$L.3 \rightarrow [L.1, L.2_1, \ldots, L.2_6]$	How many categories are there for service sub items with service number 'XX-R-I-4'?	✗	✗	✗	✗	✓

module is especially useful when wrong intermediate results or final answer are generated during multi-step execution.

Collectively, these results demonstrate that each module in ST-Raptor addresses distinct yet complementary challenges in *semi-structured table QA*, and their synergistic integration is vital for effectively tackling the layout-intensive questions in SSTQA.

7.6 Case Study on SSTQA Dataset

For each table difficulty level in the SSTQA dataset, we select four representative question-answering cases. Table 8 presents the abstract semi-structured table layout representations, the corresponding questions, and the results from five selected methods. The layout representations follow the table definitions introduced in Section 3, where $L.1$ denotes a Header-Single-Value structure, $L.2$ denotes Header-Multiple-Value, $L.3$ denotes Orthogonal Tables, and $L.4$ denotes Header-Orthogonal-Tables. A rightward arrow indicates the construction of a layout. For example, $L.4 \rightarrow L.2$ indicates that headers are added to the $L.2$ layout.

Two key observations emerge from the Table 8: (1) In terms of structural complexity, tables with complex layouts (e.g., Tables 20, 4, 1, 10) often lead to errors for most methods except ST-Raptor. (2) Regarding questions, those requiring math operations (e.g., Tables 4, 1, 30) demand a deeper understanding of table structure, where ST-Raptor consistently excels other methods.

Regarding limitations, ST-Raptor may occasionally irregular layout patterns, such as erroneously treating horizontally merged content cells as headers, which can negatively affect the QA accuracy. Besides, both ST-Raptor and the baselines face challenges in decomposing questions involving complex pipelines (e.g., resembling multi-level nested SQL queries), requiring techniques such as specialized LLM fine-tuning. Nonetheless, such cases are infrequent in semi-structured table QA tasks.

8 Related Works

8.1 Structured Table QA.

Mainstream approaches can be categorized into NL2SQL, NL2Code and vision-language model based methods. NL2SQL [16, 24, 41] focuses on translating natural language queries into structured SQL commands by leveraging techniques such as (1) schema linking, which aligns user intents with database schema to resolve ambiguities, and (2) content retrieval, which dynamically extracts relevant information from the database to refine query generation. VLM based methods [20, 47] transform the table into image for analysis and question answering.

8.2 Semi-Structured Table QA.

Semi-structured tables bring a large challenge for table understanding and render traditional Text2SQL strategy ineffective. To address the issue, numerous excellent research efforts have been carried out [21, 26, 32, 38]. For instance, Wang et al. proposed an end-to-end system [38] that uses semi-structured tables as knowledge sources by first finding the most similar tables and then selecting the most relevant table cells to derive the answer. Additionally, Liu et al. proposed the GrabTab method [26] featuring a Component Deliberator that efficiently leverages multiple table components without requiring complex post-processing, for addressing the challenge of recognizing complex and irregular table structures. Inspired by NL2SQL techniques, Lu et al. [28] propose to convert natural language queries into NoSQL ones, but only produce intermediate results, lacking the end-to-end semi-structured table QA capability like ST-Raptor. Moreover, Gupta et al. built a TEMPTABQA dataset [18] from 1,208 Wikipedia Infobox tables to evaluate the temporal reasoning capabilities, and found that even top-performing LLMs fall behind human performance by over 13.5 F1 points. Some works conduct information extraction over semi-structured data. TWIX [25] assumes that many semi-structured data is generate from one similar layout template and proposes a method that first

reconstruct the template than extract the content. However, it lacks support for merged cells and cannot be readily transformed into HO-Trees within our problem scope. Another approach is convert the semi-structured data into structured formats for downstream analysis [6]. However, this conversion process can introduce information loss and reduce answer accuracy. Overall, these findings highlight that despite significant advancements, substantial challenges remain in effectively addressing semi-structured table QA.

8.3 In-Context Table QA.

Although table-based reasoning has shown remarkable progress with the emergence of LLMs [15], table QA solutions encounter significant performance degradation when confronted with the rich and diverse evidence present in tables. To enable LLMs to sufficiently understand both the tables and question information, decomposition plays a pivotal role in table QA [22, 24, 40, 42, 46]. Ye et al. [42] introduces a method that effectively leverages LLMs to decompose large tables into relevant sub-tables and complex questions into simpler sub-questions through in-context prompting. In addition to in-context learning from data sources, the ReAcTable framework [46] proposed by Zhang et al. aims to improve complex table QA performance by incorporating execution feedback. This feedback mechanism, rooted in the ReAct framework, enables the system to dynamically adjust its operations based on the results of previous actions and addresses challenges such as interpreting complex data semantics, handling generated errors, and performing intricate data transformations.

9 Conclusion

In this paper, we introduced `ST-Raptor`, a tree-based framework aimed at addressing the critical challenges of automating question answering over semi-structured tables. Central to our approach is the Hierarchical Orthogonal Tree (HO-Tree), a formal representation capable of capturing complex table layouts, including hierarchical headers, merged cells, and implicit relationships. We designed a set of basic tree operations over HO-Trees to enable LLMs to perform layout-aware tasks. Given a user question, `ST-Raptor` decomposes it into simpler subquestions, constructs corresponding tree-operation pipelines, and executes them to retrieve relevant information or derive the final answer. To ensure both execution correctness and answer reliability, we proposed a two-stage verification mechanism combining forward constraint checking and backward answer validation. Additionally, we constructed the SSTQA benchmark, consisting of 764 questions over 102 real-world semi-structured tables. Experimental results demonstrate that `ST-Raptor` outperforms all baselines by up to 20% in answer accuracy.

References

[1] [n.d.]. https://www.frontiersin.org/research-topics/21489/knowledge-discovery-from-unstructured-data-in-finance

[2] [n.d.]. https://enterprises.upmc.com/resources/insights/health-cares-unstructured-data-challenge/

[3] [n.d.]. https://pages.cs.wisc.edu/~jbeckham/TR/cnet.pdf

[4] Serge Abiteboul. 1997. Querying semi-structured data. In *Database Theory — ICDT '97*, Foto Afrati and Phokion Kolaitis (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 1–18.

[5] Majed Alwateer, El-Sayed Atlam, Mahmoud Mohammed Abd El-Raouf, Osama A. Ghoneim, and Ibrahim Gad. 2024. Missing Data Imputation: A Comprehensive Review. *Journal of Computer and Communications* 12, 11 (2024), 53–75. https://doi.org/10.4236/jcc.2024.1211004

[6] Simran Arora, Brandon Yang, Sabri Eyuboglu, Avanika Narayan, Andrew Hojel, Immanuel Trummer, and Christopher Ré. 2025. Language Models Enable Simple Systems for Generating Structured Views of Heterogeneous Data Lakes. arXiv:2304.09433 [cs.CL] https://arxiv.org/abs/2304.09433

[7] Camille Barboule, Benjamin Piwowarski, and Yoan Chabot. 2025. Survey on Question Answering over Visually Rich Documents: Methods, Challenges, and Trends. arXiv:2501.02235 [cs.CL] https://arxiv.org/abs/2501.02235

[8] Łukasz Borchmann and Marek Wydmuch. 2025. Query and Conquer: Execution-Guided SQL Generation. arXiv:2503.24364 [cs.CL] https://arxiv.org/abs/2503.24364

[9] Douglas Burdick, Marina Danilevsky, Alexandre V. Evfimievski, Yannis Katsis, and Nancy Wang. 2020. Table Extraction and Understanding for Scientific and Enterprise Applications. *Proc. VLDB Endow.* 13, 12 (2020), 3433–3436. https://doi.org/10.14778/3415478.3415563

[10] Kaiwen Chen, Yueting Chen, Nick Koudas, and Xiaohui Yu. 2025. Reliable Text-to-SQL with Adaptive Abstention. *Proceedings of the ACM on Management of Data* 3, 1 (Feb. 2025), 69:1–69:30. https://doi.org/10.1145/3709719

[11] Zhe Chen, Weiyun Wang, Yue Cao, Yangzhou Liu, Zhangwei Gao, Erfei Cui, Jinguo Zhu, Shenglong Ye, Hao Tian, Zhaoyang Liu, Lixin Gu, Xuehui Wang, Qingyun Li, Yimin Ren, Zixuan Chen, Jiapeng Luo, Jiahao Wang, Tan Jiang, Bo Wang, Conghui He, Botian Shi, Xingcheng Zhang, Han Lv, Yi Wang, Wenqi Shao, Pei Chu, Zhongying Tu, Tong He, Zhiyong Wu, Huipeng Deng, Jiaye Ge, Kai Chen, Kaipeng Zhang, Limin Wang, Min Dou, Lewei Lu, Xizhou Zhu, Tong Lu, Dahua Lin, Yu Qiao, Jifeng Dai, and Wenhai Wang. 2025. Expanding Performance Boundaries of Open-Source Multimodal Models with Model, Data, and Test-Time Scaling. arXiv:2412.05271 [cs.CV] https://arxiv.org/abs/2412.05271

[12] E. F. Codd. 1970. A relational model of data for large shared data banks. *Commun. ACM* 13, 6 (June 1970), 377–387. https://doi.org/10.1145/362384.362685

[13] DeepSeek-AI. 2025. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. arXiv:2501.12948 [cs.CL] https://arxiv.org/abs/2501.12948

[14] DeepSeek-AI, Aixin Liu, Bei Feng, et al. 2025. DeepSeek-V3 Technical Report. arXiv:2412.19437 [cs.CL] https://arxiv.org/abs/2412.19437

[15] Naihao Deng, Zhenjie Sun, Ruiqi He, Aman Sikka, Yulong Chen, Lin Ma, Yue Zhang, and Rada Mihalcea. 2024. Tables as Texts or Images: Evaluating the Table Reasoning Ability of LLMs and MLLMs. In *Findings of the Association for Computational Linguistics: ACL 2024*. Association for Computational Linguistics, Bangkok, Thailand, 407–426. https://doi.org/10.18653/v1/2024.findings-acl.23

[16] Yingqi Gao, Yifu Liu, Xiaoxia Li, Xiaorong Shi, Yin Zhu, Yiming Wang, Shiqi Li, Wei Li, Yuntao Hong, Zhiling Luo, Jinyang Gao, Liyu Mou, and Yu Li. 2025. A Preview of XiYan-SQL: A Multi-Generator Ensemble Framework for Text-to-SQL. arXiv:2411.08599 [cs.AI] https://arxiv.org/abs/2411.08599

[17] Vivek Gupta, Pranshu Kandoi, Mahek Vora, Shuo Zhang, Yujie He, Ridho Reinanda, and Vivek Srikumar. 2023. TempTabQA: Temporal Question Answering for Semi-Structured Tables. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, Singapore, 2431–2453. https://doi.org/10.18653/v1/2023.emnlp-main.149

[18] Vivek Gupta, Pranshu Kandoi, Mahek Vora, Shuo Zhang, Yujie He, Ridho Reinanda, and Vivek Srikumar. 2023. TempTabQA: Temporal Question Answering for Semi-Structured Tables. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, Singapore, 2431–2453. https://doi.org/10.18653/v1/2023.emnlp-main.149

[19] Jonathan Herzig, Paweł Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Martin Eisenschlos. 2020. TAPAS: Weakly Supervised Table Parsing via Pre-training. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*. Association for Computational Linguistics, 4320–4333. https://doi.org/10.18653/v1/2020.acl-main.398

[20] Anwen Hu, Haiyang Xu, Jiabo Ye, Ming Yan, Liang Zhang, Bo Zhang, Chen Li, Ji Zhang, Qin Jin, Fei Huang, and Jingren Zhou. 2024. mPLUG-DocOwl 1.5: Unified Structure Learning for OCR-free Document Understanding. arXiv:2403.12895 [cs.CV] https://arxiv.org/abs/2403.12895

[21] Sujay Kumar Jauhar, Peter Turney, and Eduard Hovy. 2016. Tables as Semi-structured Knowledge for Question Answering. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Katrin Erk and Noah A. Smith (Eds.). Association for Computational Linguistics, Berlin, Germany, 474–483. https://doi.org/10.18653/v1/P16-1045

[22] Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. 2023. Decomposed Prompting: A Modular Approach for Solving Complex Tasks. arXiv:2210.02406 [cs.CL] https://arxiv.org/abs/2210.02406

[23] 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. arXiv:2310.09263 [cs.CL] https://arxiv.org/abs/2310.09263

[24] Zhishuai Li, Xiang Wang, Jingjing Zhao, Sun Yang, Guoqing Du, Xiaoru Hu, Bin Zhang, Yuxiao Ye, Ziyue Li, Rui Zhao, and Hangyu Mao. 2024. PET-SQL: A Prompt-Enhanced Two-Round Refinement of Text-to-SQL with Cross-consistency. arXiv:2403.09732 [cs.CL] https://arxiv.org/abs/2403.09732

[25] Yiming Lin, Mawil Hasan, Rohan Kosalge, Alvin Cheung, and Aditya G. Parameswaran. 2025. TWIX: Automatically Reconstructing Structured Data from Templatized Documents. arXiv:2501.06659 [cs.DB] https://arxiv.org/abs/2501.06659

[26] Hao Liu, Xin Li, Mingming Gong, Bing Liu, Yunfei Wu, Deqiang Jiang, Yinsong Liu, and Xing Sun. 2024. Grab What You Need: Rethinking Complex Table Structure Recognition with Flexible Components Deliberation. *Proceedings of the AAAI Conference on Artificial Intelligence* 38, 4 (Mar. 2024), 3603–3611. https://doi.org/10.1609/aaai.v38i4.28149

[27] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2023. Lost in the Middle: How Language Models Use Long Contexts. arXiv:2307.03172 [cs.CL] https://arxiv.org/abs/2307.03172

[28] Jinwei Lu, Yuanfeng Song, Zhiqian Qin, Haodi Zhang, Chen Zhang, and Raymond Chi-Wing Wong. 2025. Bridging the Gap: Enabling Natural Language Queries for NoSQL Databases through Text-to-NoSQL Translation. arXiv:2502.11201 [cs.DB] https://arxiv.org/abs/2502.11201

[29] OpenAI. 2024. GPT-4o System Card. arXiv:2410.21276 [cs.CL] https://arxiv.org/abs/2410.21276

[30] OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, et al. 2024. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL] https://arxiv.org/abs/2303.08774

[31] Panupong Pasupat and Percy Liang. 2015. Compositional Semantic Parsing on Semi-Structured Tables. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, Chengqing Zong and Michael Strube (Eds.). Association for Computational Linguistics, Beijing, China, 1470–1480. https://doi.org/10.3115/v1/P15-1142

[32] Panupong Pasupat and Percy Liang. 2015. Compositional Semantic Parsing on Semi-Structured Tables. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, Chengqing Zong and Michael Strube (Eds.). Association for Computational Linguistics, Beijing, China, 1470–1480. https://doi.org/10.3115/v1/P15-1142

[33] Bowen Qin, Binyuan Hui, Lihan Wang, Min Yang, Jinyang Li, Binhua Li, Ruiying Geng, Rongyu Cao, Jian Sun, Luo Si, Fei Huang, and Yongbin Li. 2022. A Survey on Text-to-SQL Parsing: Concepts, Methods, and Future Directions. *arXiv preprint arXiv:2208.13629* (2022). https://arxiv.org/abs/2208.13629

[34] Ge Qu, Jinyang Li, Bowen Li, Bowen Qin, Nan Huo, Chenhao Ma, and Reynold Cheng. 2024. Before Generation, Align it! A Novel and Effective Strategy for Mitigating Hallucinations in Text-to-SQL Generation. In *Findings of the Association for Computational Linguistics: ACL 2024*. Association for Computational Linguistics, Bangkok, Thailand, 5456–5471. https://doi.org/10.18653/v1/2024.findings-acl.324

[35] Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding from Language Models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Online and Punta Cana, Dominican Republic, 9895–9901. https://doi.org/10.18653/v1/2021.emnlp-main.779

[36] Tal Schuster, Adam Lelkes, Haitian Sun, Jai Gupta, Jonathan Berant, William Cohen, and Donald Metzler. 2024. SEMQA: Semi-Extractive Multi-Source Question Answering. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*. Association for Computational Linguistics, Mexico City, Mexico, 1363–1381. https://doi.org/10.18653/v1/2024.naacl-long.74

[37] Yuan Sui, Mengyu Zhou, Mingjie Zhou, Shi Han, and Dongmei Zhang. 2024. Table Meets LLM: Can Large Language Models Understand Structured Table Data? A Benchmark and Empirical Study. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining (WSDM 2024)*. Association for Computing Machinery, Mérida, Mexico, 123–134. https://doi.org/10.1145/3616855.3635752

[38] Hao Wang, Xiaodong Zhang, Shuming Ma, Xu Sun, Houfeng Wang, and Mengxiang Wang. 2018. A Neural Question Answering Model Based on Semi-Structured

Tables. In *Proceedings of the 27th International Conference on Computational Linguistics*, Emily M. Bender, Leon Derczynski, and Pierre Isabelle (Eds.). Association for Computational Linguistics, Santa Fe, New Mexico, USA, 1941–1951. https://aclanthology.org/C18-1165/

[39] Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. 2024. Multilingual E5 Text Embeddings: A Technical Report. arXiv:2402.05672 [cs.CL] https://arxiv.org/abs/2402.05672

[40] Zhongyuan Wang, Richong Zhang, Zhijie Nie, and Jaein Kim. 2024. Tool-Assisted Agent on SQL Inspection and Refinement in Real-World Scenarios. arXiv:2408.16991 [cs.CL] https://arxiv.org/abs/2408.16991

[41] Xiangjin Xie, Guangwei Xu, Lingyan Zhao, and Ruijie Guo. 2025. OpenSearch-SQL: Enhancing Text-to-SQL with Dynamic Few-shot and Consistency Alignment. arXiv:2502.14913 [cs.CL] https://arxiv.org/abs/2502.14913

[42] Yunhu Ye, Binyuan Hui, Min Yang, Binhua Li, Fei Huang, and Yongbin Li. 2023. Large Language Models are Versatile Decomposers: Decompose Evidence and Questions for Table-based Reasoning. arXiv:2301.13808 [cs.CL] https://arxiv.org/abs/2301.13808

[43] Daoguang Zan, Bei Chen, Fengji Zhang, Dianjie Lu, Bingchao Wu, Bei Guan, Yongji Wang, and Jian-Guang Lou. 2023. Large Language Models Meet NL2Code: A Survey. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Toronto, Canada, 7443–7464. https://doi.org/10.18653/v1/2023.acl-long.411

[44] Tianshu Zhang, Xiang Yue, Yifei Li, and Huan Sun. 2024. TableLlama: Towards Open Large Generalist Models for Tables. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*. Association for Computational Linguistics, Mexico City, Mexico, 6024–6044. https://doi.org/10.18653/v1/2024.naacl-long.335

[45] Xiaokang Zhang, Sijia Luo, Bohan Zhang, Zeyao Ma, Jing Zhang, Yang Li, Guanlin Li, Zijun Yao, Kangli Xu, Jinchang Zhou, Daniel Zhang-Li, Jifan Yu, Shu Zhao, Juanzi Li, and Jie Tang. 2025. TableLLM: Enabling Tabular Data Manipulation by LLMs in Real Office Usage Scenarios. arXiv:2403.19318 [cs.CL] https://arxiv.org/abs/2403.19318

[46] Yunjia Zhang, Jordan Henkel, Avrilia Floratou, Joyce Cahoon, Shaleen Deep, and Jignesh M. Patel. 2023. ReAcTable: Enhancing ReAct for Table Question Answering. arXiv:2310.00815 [cs.DB] https://arxiv.org/abs/2310.00815

[47] Mingyu Zheng, Xinwei Feng, Qingyi Si, Qiaoqiao She, Zheng Lin, Wenbin Jiang, and Weiping Wang. 2024. Multimodal Table Understanding. arXiv:2406.08100 [cs.CL] https://arxiv.org/abs/2406.08100

[48] Fengbin Zhu, Ziyang Liu, Fuli Feng, Chao Wang, Moxin Li, and Tat-Seng Chua. 2024. TAT-LLM: A Specialized Language Model for Discrete Reasoning over Tabular and Textual Data. arXiv:2401.13223 [cs.CL] https://arxiv.org/abs/2401.13223