Exploring Table Representations for Question Answering

Johannes Gabriel Sindlinger

Heidelberg University
Institute of Computer Science
Data Science Group
johannes.sindlinger@stud.uni-heidelberg.de

May 12, 2025

Outline

Outline

Table Question Answering

Table Question Answering (Table QA) QA in which the **supporting context** or **evidence** to answer questions is provided in **tabular form**

❓ **LLMs as solution?**

Using which table representation?

🚧 **Issue:** Most research focuses on **Vanilla Table QA**

- *But:* **Not realistic** in many scenarios, since **tables rarely occur in isolation**
- Focus on **Table QA on Full Documents**

Structural Misalignment of Tables

🚧 **Issue:** Tables are structured in a **2D manner**, while LLMs process data sequentially in a **1D manner**

- Need for **Table Serialization**
- *But:* Hard to **capture spatial relationships** in 1D

2D-Visual Representation

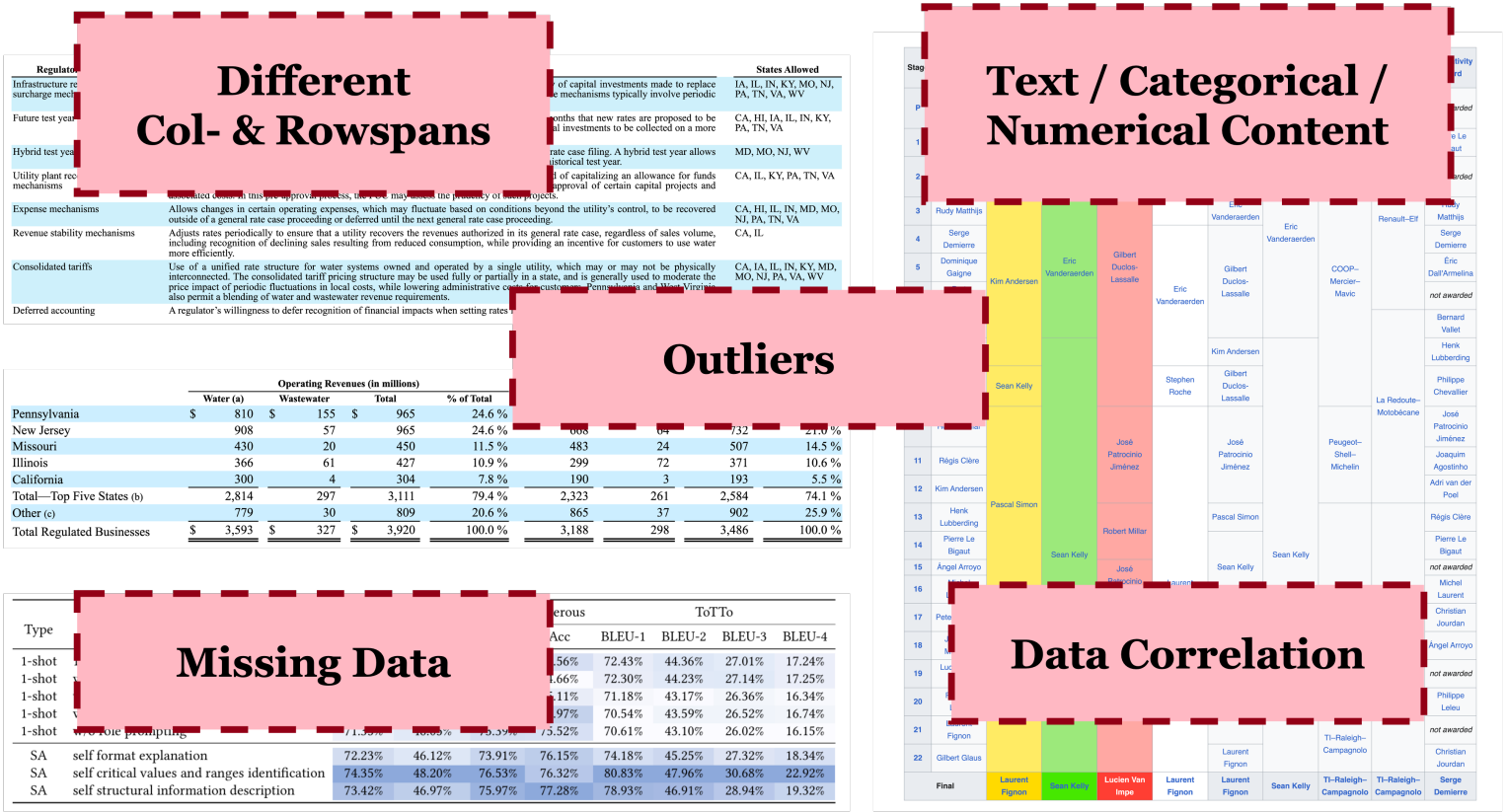
grades						
			2023		2024	
class	name	age	math	english	math	english
A	John	17	A	C	A	B
	Tiffany	16	B	B	C	B
B	Michael	17	D	D	D	D

1D-HTML Representation

```
<table><tr><th colspan="2">grades</th><th
colspan="2“>2023</th><th colspan="2">2024</th>
</tr><tr><th>class</th><th>name</th><th>age</t
h><th>math</th><th>english</th><th>math</th><t
h>english</th></tr><tr><td>A</td><td>John</td>
<td>17</td><td>A</td><td>C</td><td>A</td><td>B
</td></tr><tr><td>A</td><td>Tiffany</td><td>16
</td><td>B</td><td>B</td><td>C</td><td>B</td><
/tr><tr><td>B</td><td>Michael</td><td>17</td><
td>D</td><td>D</td><td>D</td><td>D</td></tr></
table>
```

Table Heterogeneity

🚧 **Issue:** Real-world tables vary significantly in **format, content, sparsity**, and **complexity**



Outline

Related Work

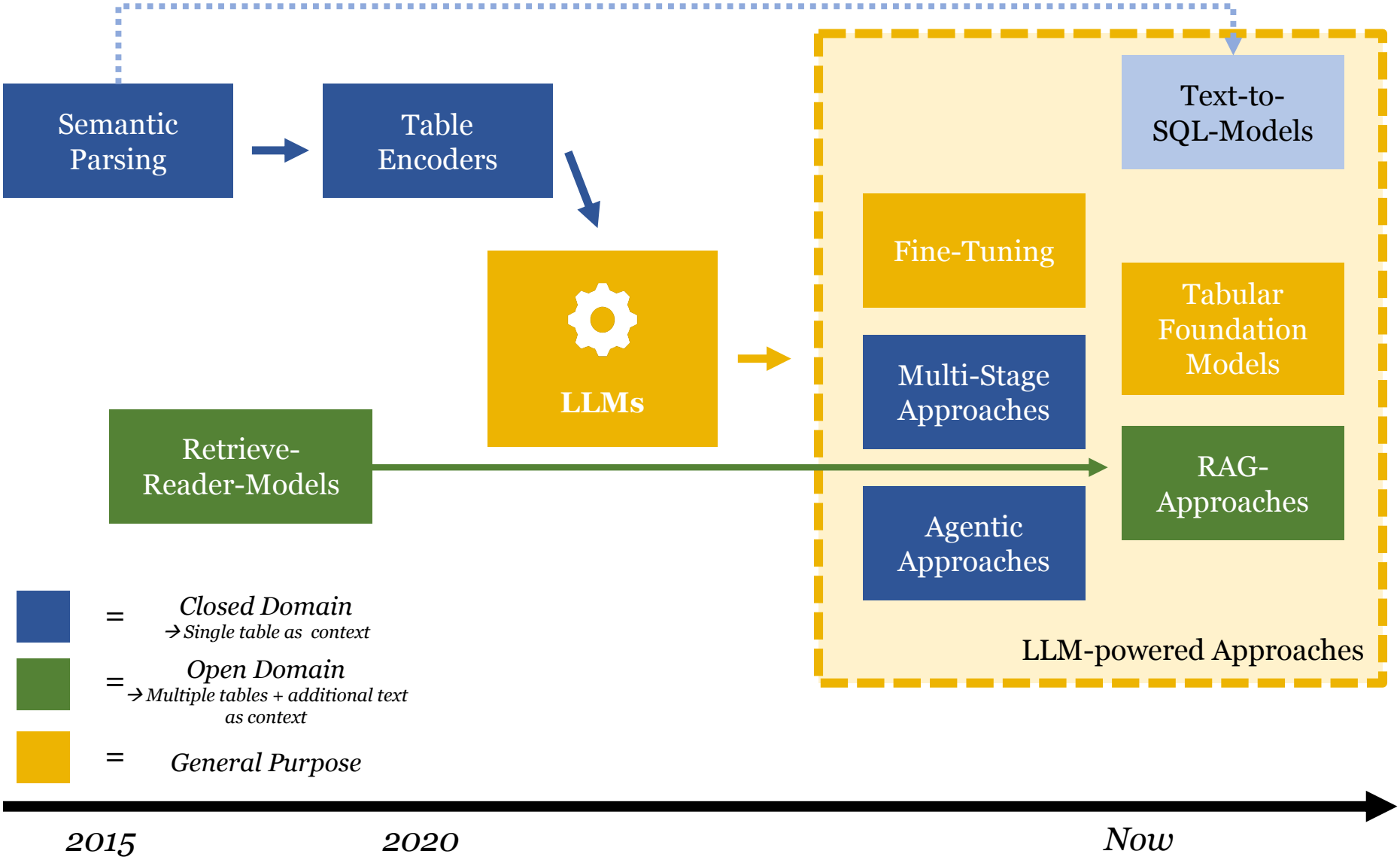


Table Serializations

Method	Example
HTML	`<table><thead><tr><th></th><th>name</th> <th>age</th></tr></thead><tbody><tr><th>0</th> ...`
JSON	`{"0": {"name": "helen", "age": "47"}}`

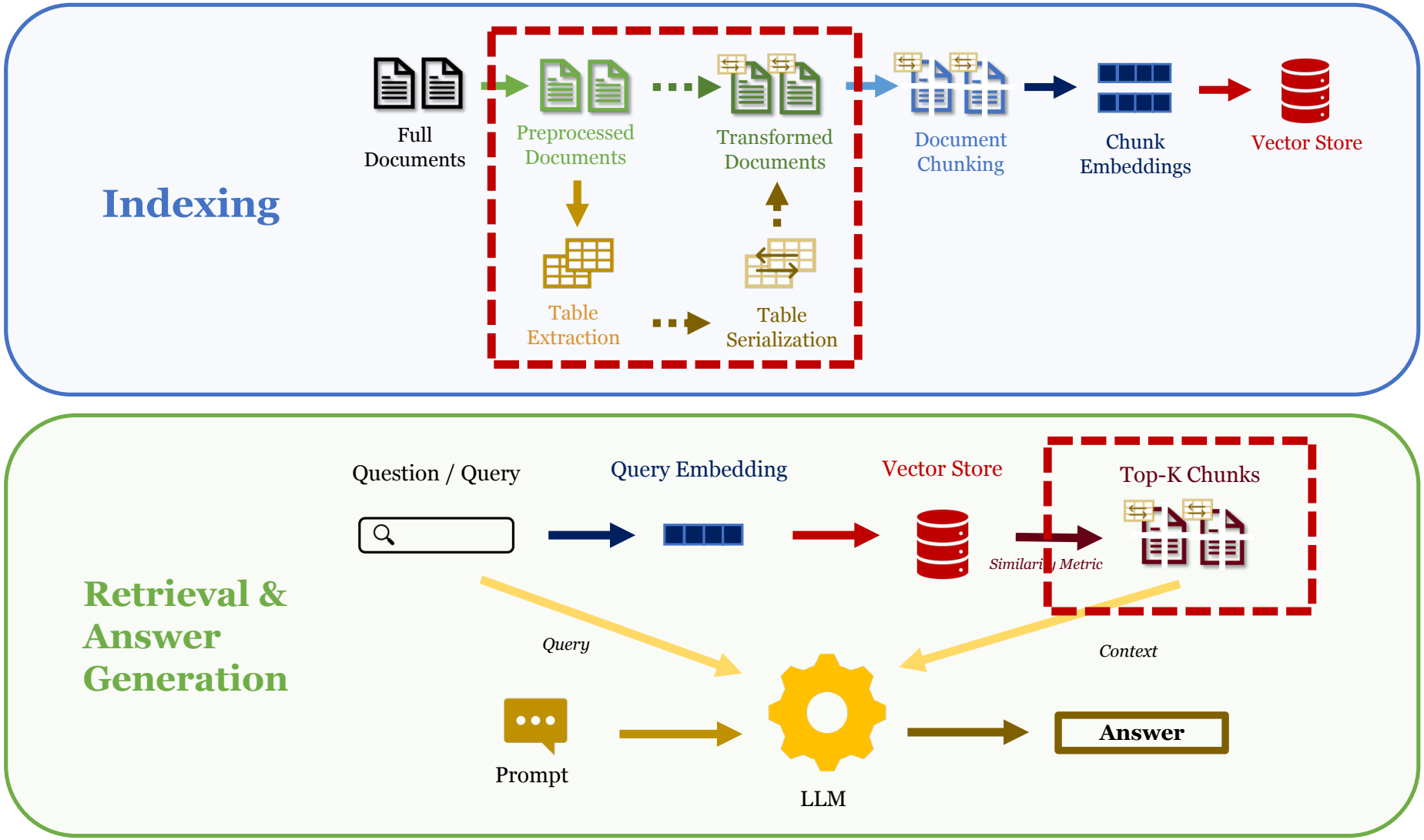
Markdown	`|  | name | age |`
	`|:--|:-----|----:|`
	`| 0 |helen | 47 |`

CSV	`, name, age` `0, helen, 47`
Sentence-based	`name is helen, age is 47`
Data Matrix	`[['','name','age'], [0, 'helen', 47]]`
LaTeX	`name & age \\ \hline helen & 47`
...	...

Table: Table serialization methods used in the literature, adapted from Fang et al. (2024).

Outline

RAG Pipeline



Implementation Details

🧩 **Semantic Chunking** adopted from Kamradt (2024)

- **Goal**: Avoid splits within semantic coherent content
- **Compare embeddings** of 3-sentence-windows & split on high disparity points – treat **serialized tables** as **one sentence**

🔍 **Basic Dense Retrieval** approach

- without reranking & filtering
- Cosine similarity, Top-$k \in 1, 3, 5$, threshold $= 0.5$

🧠 **Zero-Shot Prompting** + **Chain-of-Thought** reasoning

- Incorporate 'Think step-by-step' into prompts

Outline

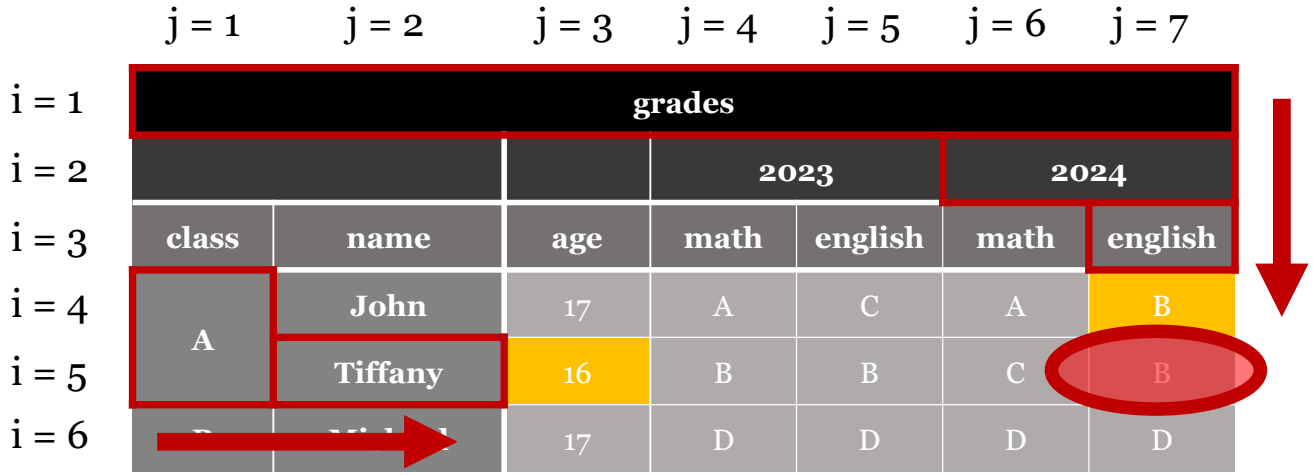
TabTree – Motivation

How do we humans assign cell contents within complex table structures?

$\Rightarrow$ i.e., tables with nested structures of multiple col- and rowspans

TabTree – Motivation

🔍 *Which grade did 'Tiffany' receive in her English classes in 2024?*



1. **Identify context** of table, i.e., row labels and column headers
2. Leverage **nested table structure** to navigate to 'value' cells **hierarchically** from top to bottom and left to right

TabTree – Overview

Input Table

grades		grades				
			2023		2024	
class	name	age	math	english	math	english
A	John	17	A	C	A	B
	Tiffany	16	B	B	C	B
B	Michael	17	D	D	D	D

TabTree Model

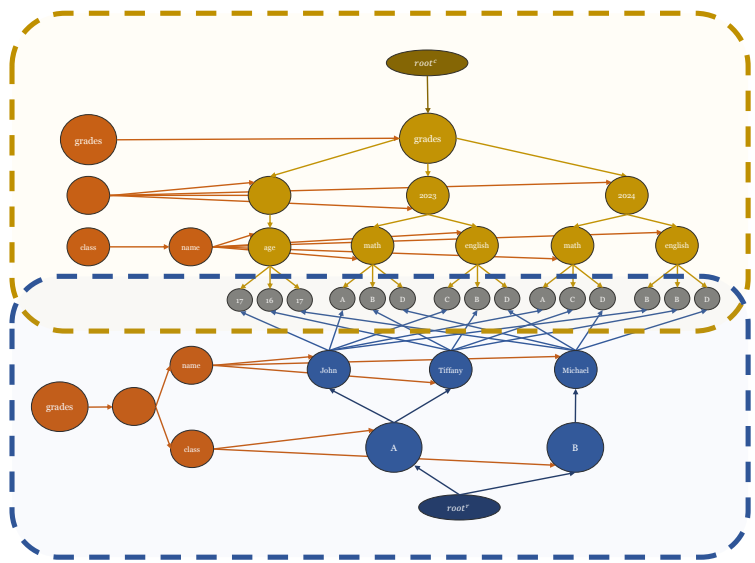


Table Serialization

```
grades:
    null:
        age:
            - A > John: 17
            - A > Tiffany: 16
            - B > Michael: 17
    2023:
        math:
            - A > John: A
            - A > Tiffany: B
            - B > Michael: D
        english:
            - A > John: C
            - A > Tiffany: B
            - B > Michael: D
    2024:
        math:
            ...
        english:
            ...
```

1. Create **two directed trees** based on **column headers** and **row labels** – leveraging their **hierarchical** structure
2. **Merge the two subtrees** at the value cell level
3. Use **pre-order DFS-traversal** to serialize the table

TabTree Serializations – Sample Serialization

```
grades:
  null:
    age:
      A > John: 17
      A > Tiffany: 16
      B > Michael: 17
  2023:
    math:
      A > John: A
      A > Tiffany: B
      B > Michael: D
    english:
      A > John: C
  2024:
    math:
      A > Tiffany: C
    english:
      A > John: B
```

Figure: Context String Approach: Base w/o Context-Intersection – Value String Approach: Base w/o Context-Intersection

TabTree Serializations – Sample Serialization

```
The table captures grades as its main column header.
  The column header grades has no siblings. The children of grades are null, 2023,
2024.
    The column header null has siblings 2023, 2024. The children of null are age.
      The values of the column header age are:
        The value of the column header age and the row label combination A, John is
17 (row index: 3, column index: 2).
        The value of the column header age and the row label combination A, Tiffany
is 16 (row index: 4, column index: 2).
        The value of the column header age and the row label combination B, Michael
is 17 (row index: 5, column index: 2).
    The column header 2023 has siblings null, 2024. The children of 2023 are math,
english.
      The column header math has siblings english. The values of math are:
        The value of the column header math and the row label combination A, John is
A (row index: 3, column index: 3).
        The value of the column header math and the row label combination A, Tiffany
is B (row index: 4, column index: 3).
        The value of the column header math and the row label combination B, Michael
is D (row index: 5, column index: 3).
```

Figure: Context String Approach: Text-Augmentation w/o Context-Intersection – Value String Approach: Text w/o Context-Intersection

TabTree Serializations – Sample Serialization

```
The table captures grades as its main column header.
  The column header grades has no siblings. The children of grades are null, 2023,
2024.
    The column header null has siblings 2023, 2024. The children of null are age.
      The column header age represents class and name. The values of the column
header age are:
        The value of the column header combination grades, null, class & name & age
and the row label combination grades & class & A, grades & name & John is 17 (row
index: 3, column index: 2).
        The value of the column header combination grades, null, class & name & age
and the row label combination grades & class & A, grades & name & Tiffany is 16 (row
index: 4, column index: 2).
        The value of the column header combination grades, null, class & name & age
and the row label combination grades & class & B, grades & name & Michael is 17 (row
index: 5, column index: 2).
    The column header 2023 has siblings null, 2024. The children of 2023 are math,
english.
      The column header math represents class and name. The column header math has
siblings english. The values of the column header math are:
        The value of the column header combination grades, 2023, class & name & math
```

Figure: Context String Approach: Text-Augmentation w/ Context-Intersection – Value String Approach: Text-Augmentation w/ Context-Intersection

Context Detection

🚧 **Issue:** TabTree requires **explicit row labels** and **column headers**

$$\Rightarrow \text{Often not available in raw tables}$$

✔️ **Solution** **Ask LLM!**

1. **Iteratively query LLM** whether a given row or column is a header or label
2. Start from the **top row** or **left column**, stop at the first **non-header row/non-label column**
3. **Evaluated 3 approaches** and selected the most effective one

Outline

Datasets

1. **WikiTableQuestions** (Pasupat et al., 2015) for Vanilla Table QA
   - 22,003 question-answer pairs linked to a table from a Wikipedia article
     - Sampling strategy: 4 samples of 50 questions each
   - *But:* Not well-suited for Table QA on Full Documents
2. **SEC-Filing Tables** for Retrieval & Table QA on Full Documents
   - 310 manually annotated questions from 2023 U.S. SEC Form 10-K filings for American Water Works[1] and Uber[2].

[1] `https://www.sec.gov/Archives/edgar/data/1410636/000141063624000050/awk-20231231.htm`, visited on 04/28/2025

[2] `https://www.sec.gov/Archives/edgar/data/1543151/000154315124000012/uber-20231231.htm`, visited on 04/28/2025

Evaluation Setup

- **Balanced mix** of large and small LLMs for text generation:
  - *GPT-4o-mini* (OpenAI, 2024a)
  - *Phi 4* (Abdin et al., 2024)
  - *GPT-4o* (OpenAI et al., 2024)
  - *Llama 3.3 70B Instruct* (Grattafiori et al., 2024)
- **Single, well-established embedding model** for retrieval:
  - *text-embedding-3-small* (OpenAI, 2024b)

Results – Vanilla Table QA

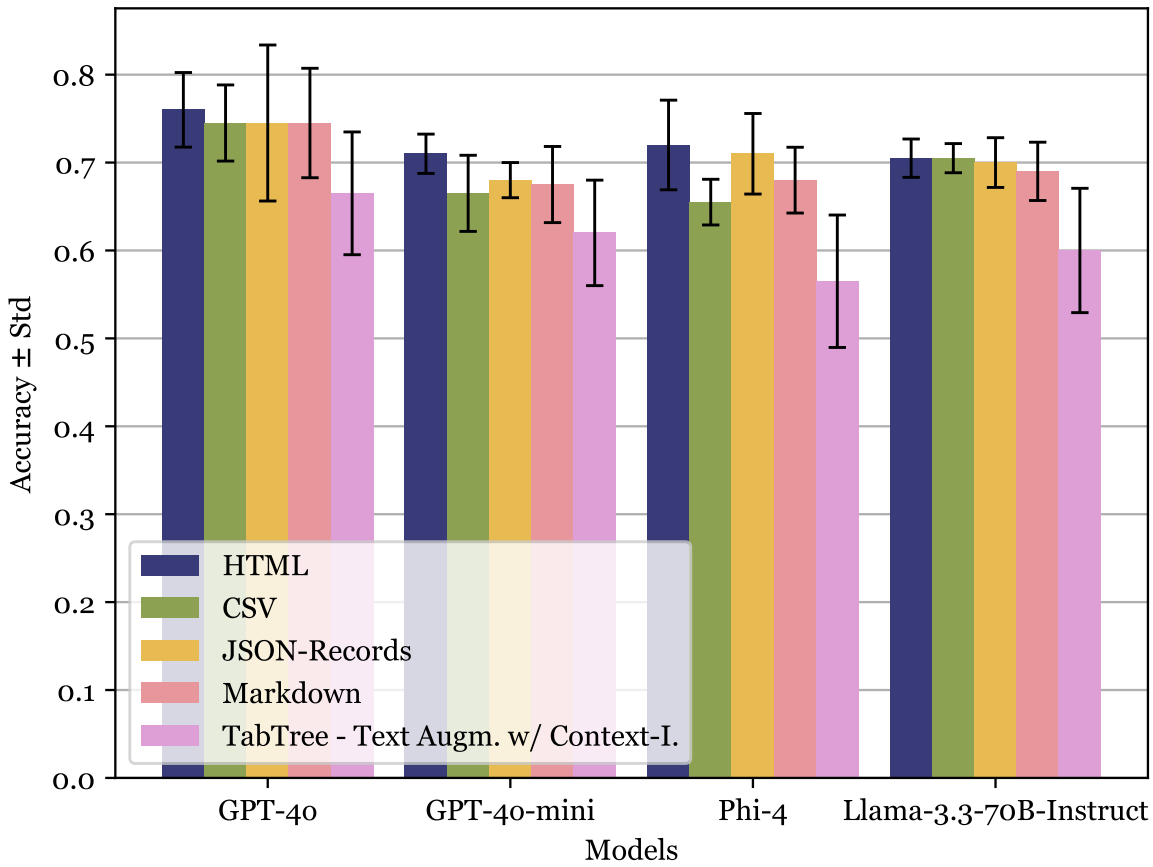
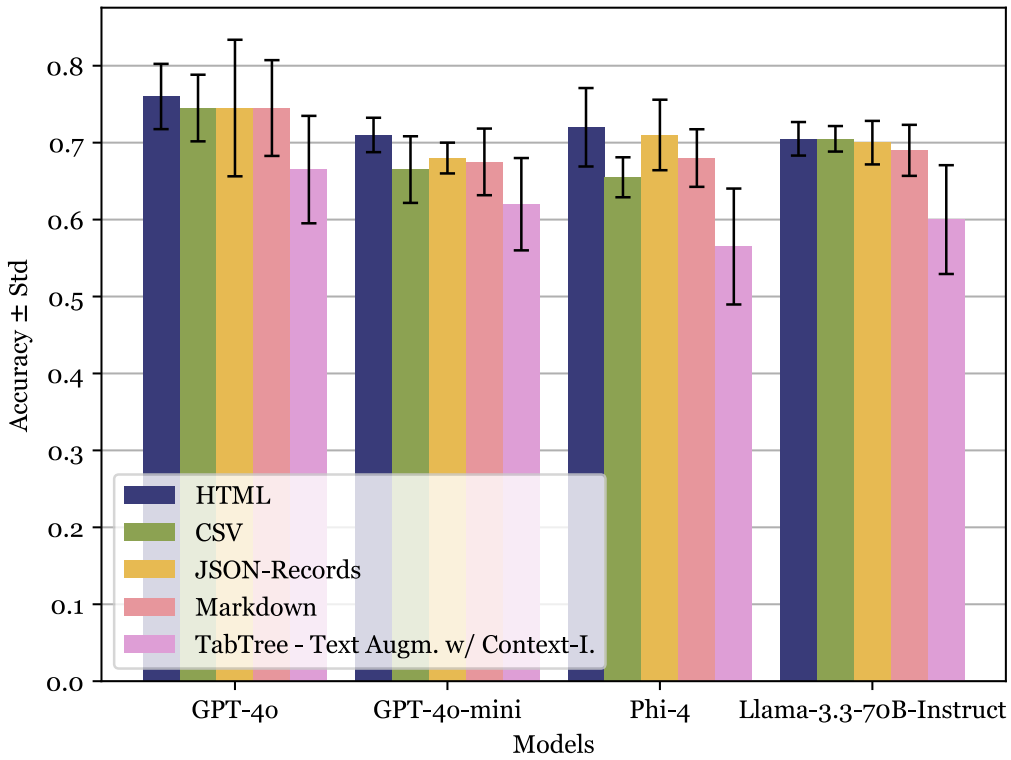


Figure: Vanilla Table QA Results (Exec. Accuracy ± Std.) on different LLMs for baselines and best-performing TabTree approach

Results – Vanilla Table QA

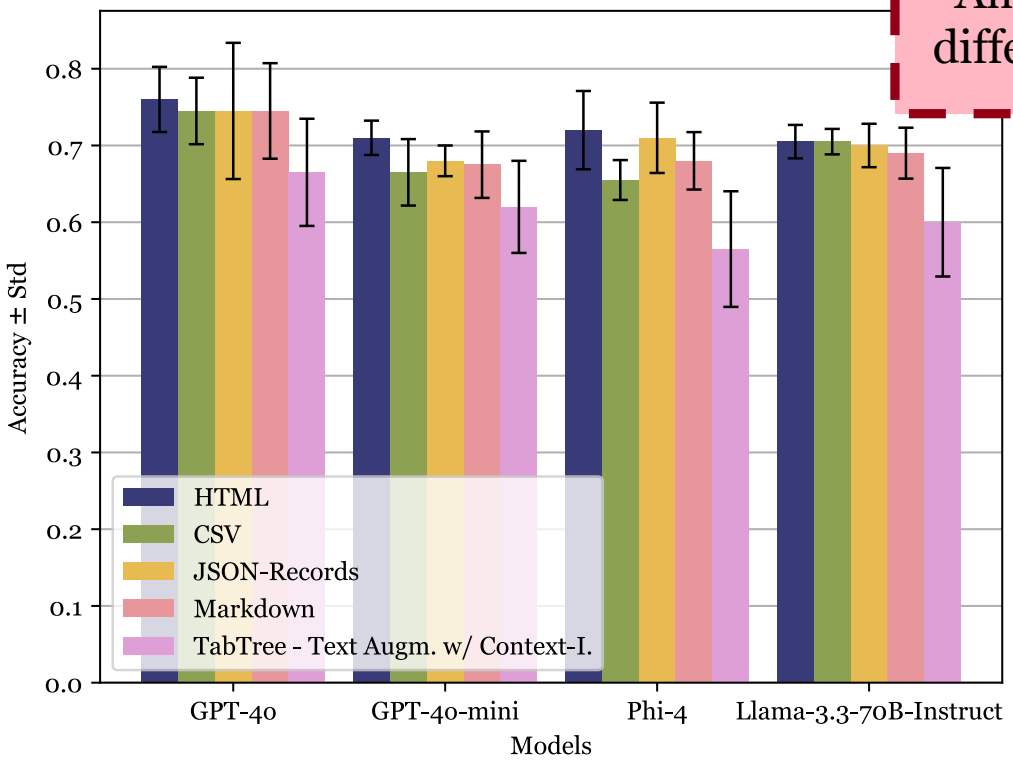
TabTree variants **underperform** relative to baselines



Results – Vanilla Table QA

TabTree variants **underperform** relative to baselines

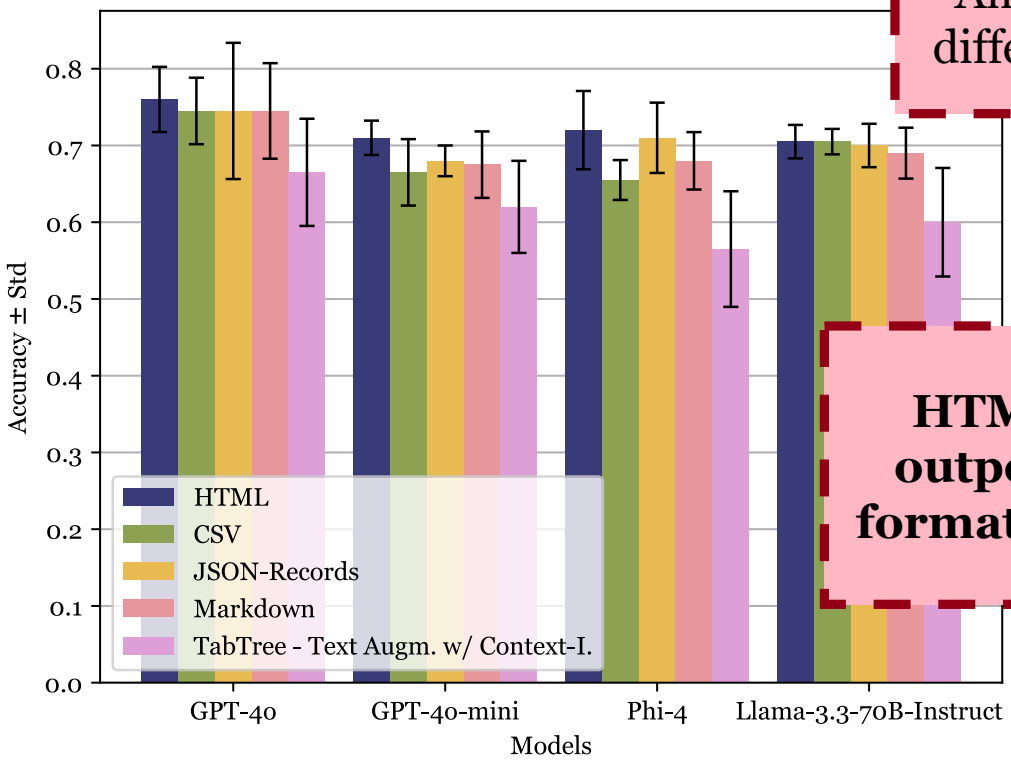
Among **baselines**, differences are minor



Results – Vanilla Table QA

TabTree variants **underperform** relative to baselines

Among **baselines**, differences are minor



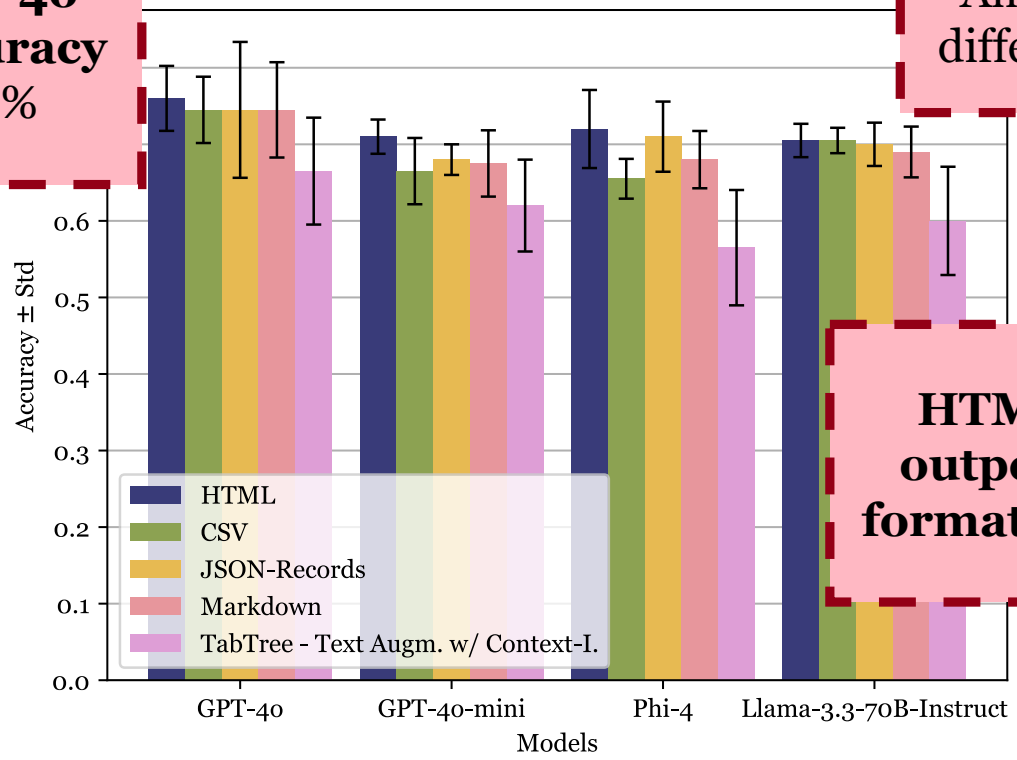
HTML consistently **outperforms other formats** across all LLMs

Results – Vanilla Table QA

TabTree variants **underperform** relative to baselines

Across LLMs, **GPT-4o** yields **highest accuracy** → Best score: 76 %

Among **baselines**, differences are minor



HTML consistently **outperforms other formats** across all LLMs

Results – Vanilla Table QA

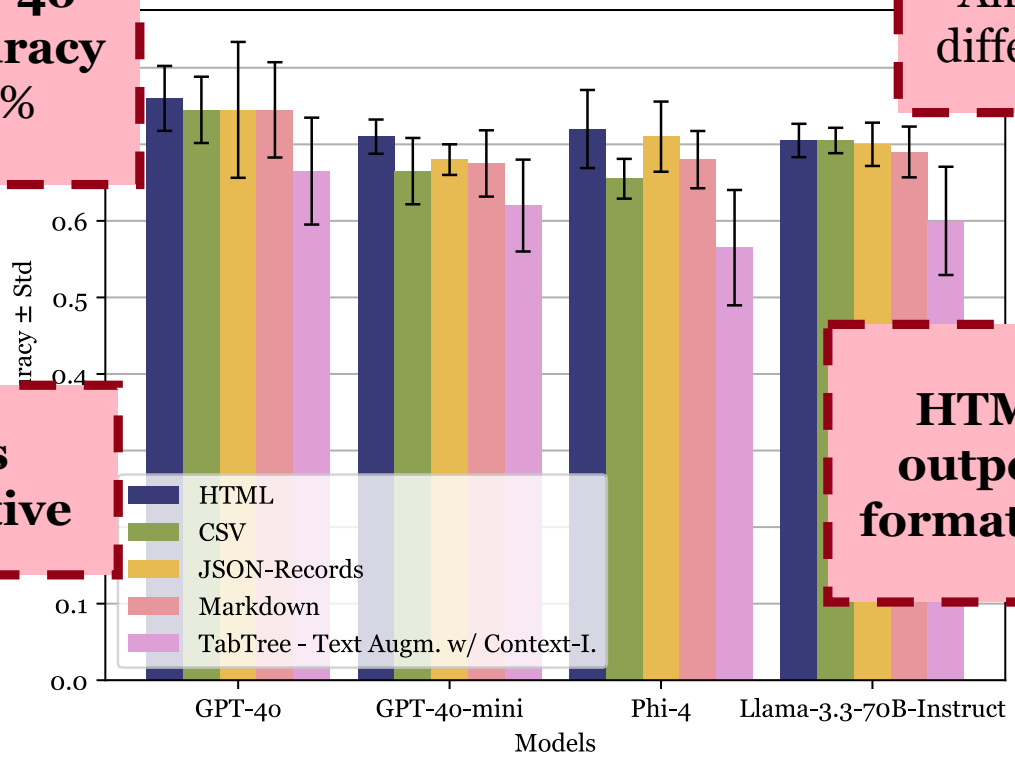
TabTree variants **underperform** relative to baselines

Across LLMs, **GPT-4o** yields **highest accuracy** → Best score: 76 %

Among **baselines**, differences are minor

Smaller LLMs perform **competitive**

HTML consistently **outperforms other formats** across all LLMs



Results – Retrieval

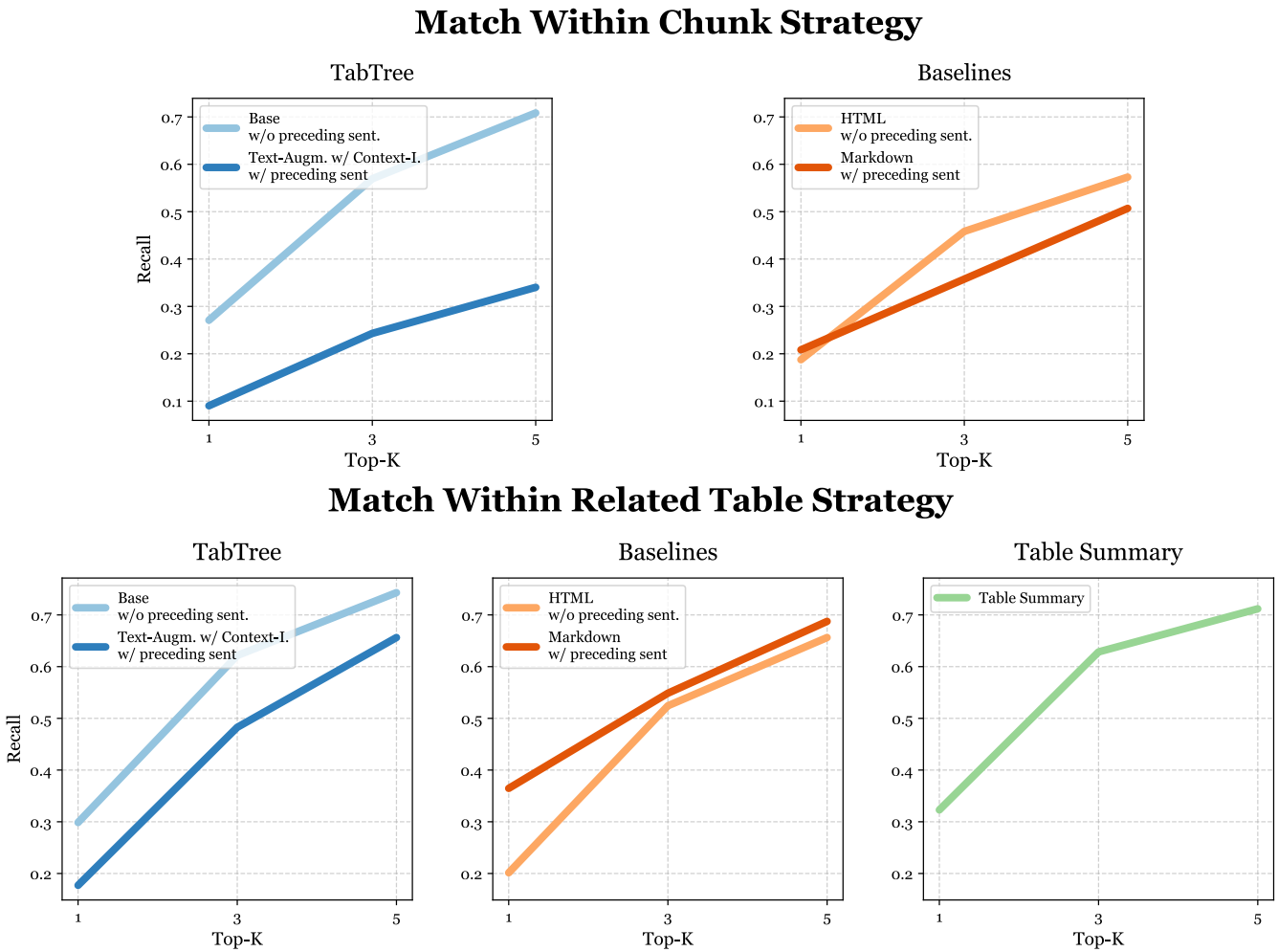
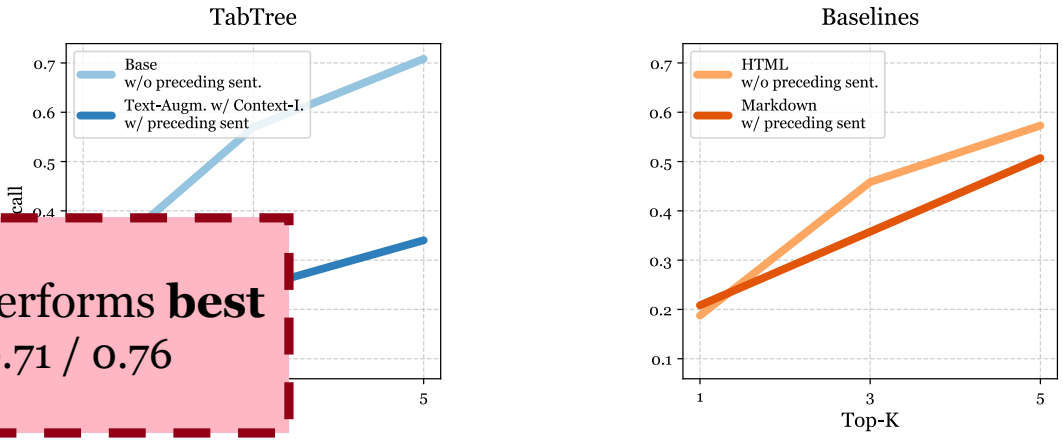


Figure: Retrieval Results (Recall@k) for representative TabTree approaches, baselines, and Table Summary approach

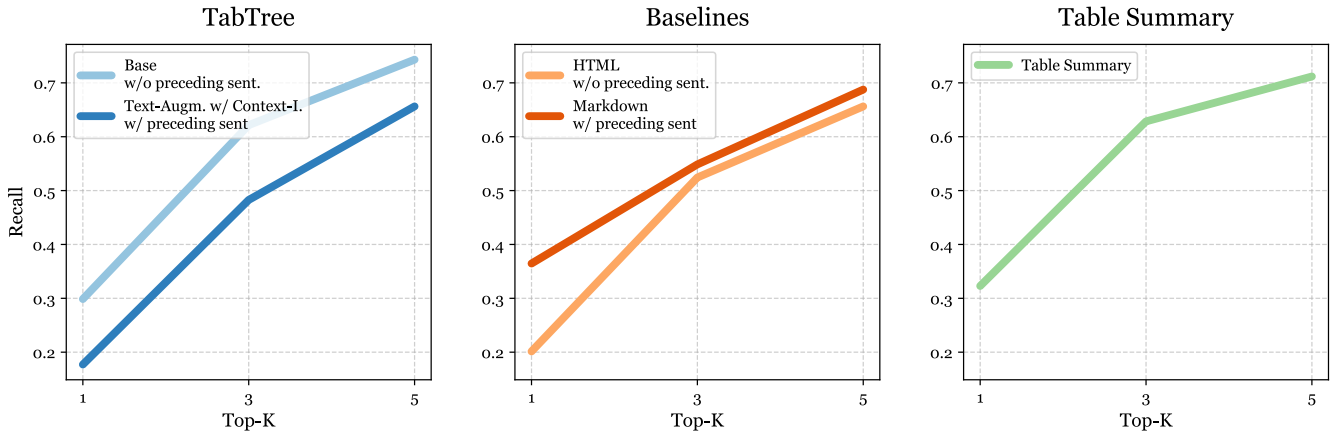
Results – Retrieval

TabTree Base performs **best**
→ Recall@5: 0.71 / 0.76

Match Within Chunk Strategy

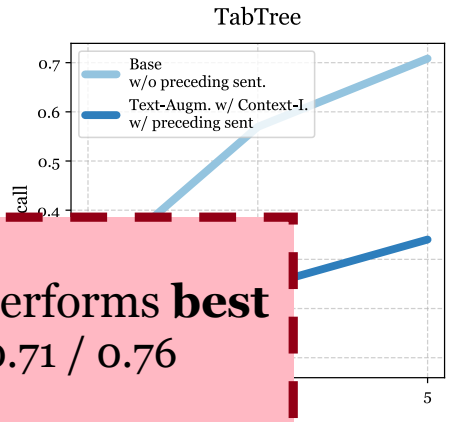


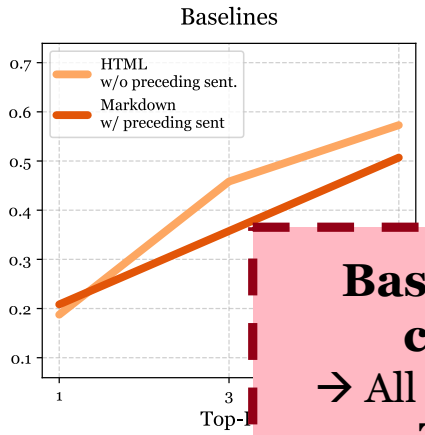
Match Within Related Table Strategy



Results – Retrieval

Match Within Chunk Strategy

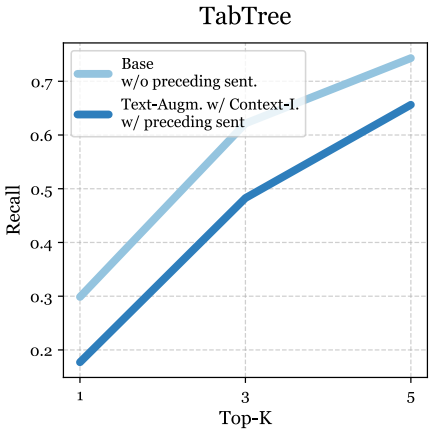


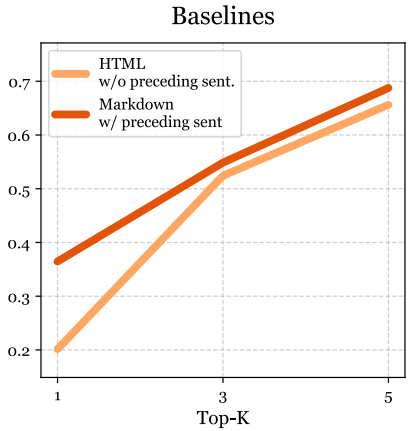


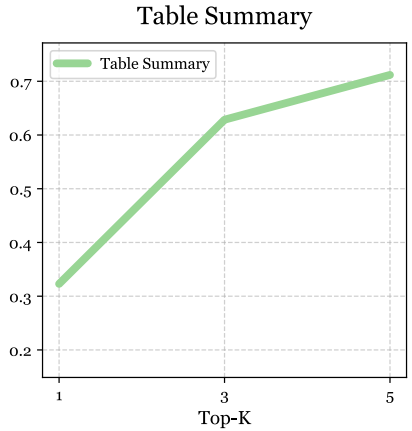
TabTree Base performs **best**
→ Recall@5: 0.71 / 0.76

Baselines perform **comparably**
→ All slightly worse than TabTree Base

Match Within Related Table Strategy

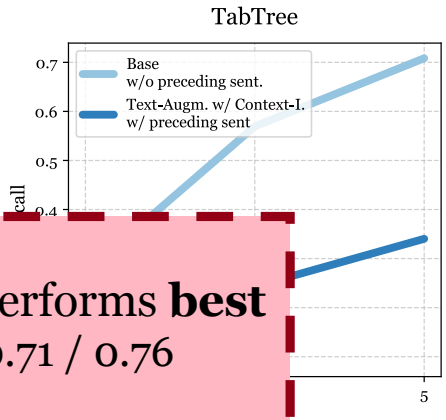


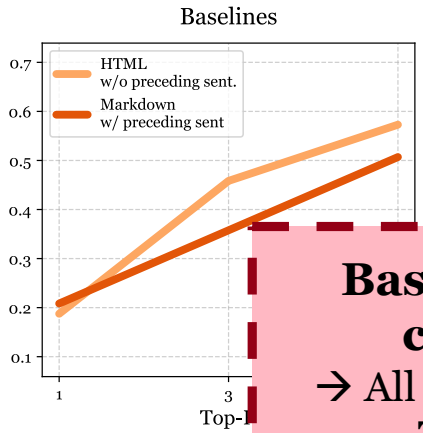




Results – Retrieval

Match Within Chunk Strategy

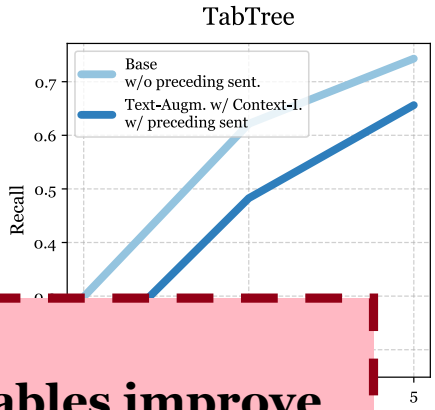


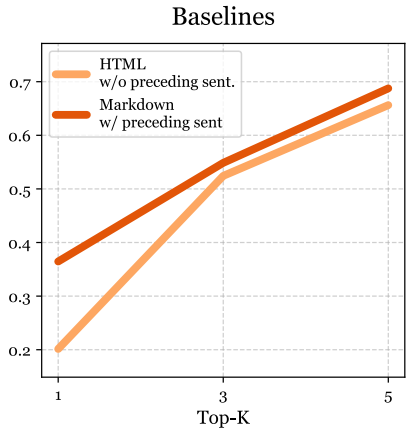


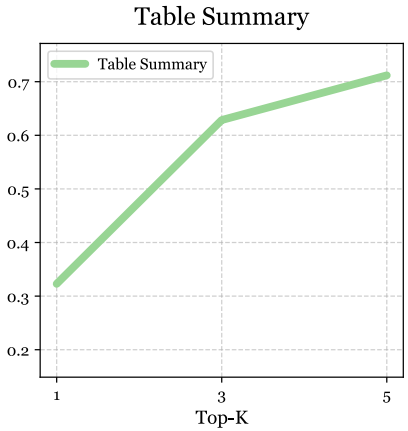
TabTree Base performs **best**
→ Recall@5: 0.71 / 0.76

Baselines perform **comparably**
→ All slightly worse than TabTree Base

Match Within Related Table Strategy



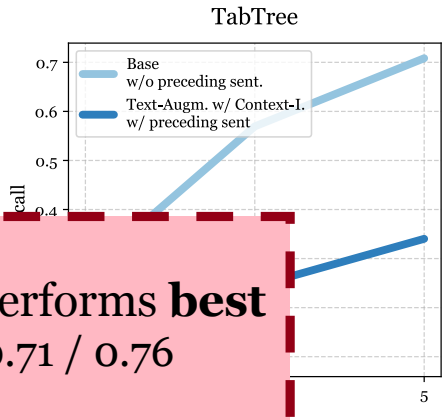


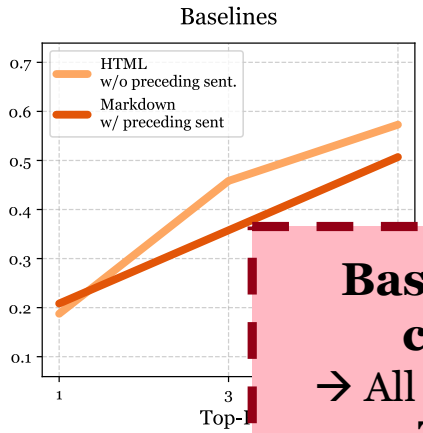


Related tables improve results in most cases

Results – Retrieval

Match Within Chunk Strategy

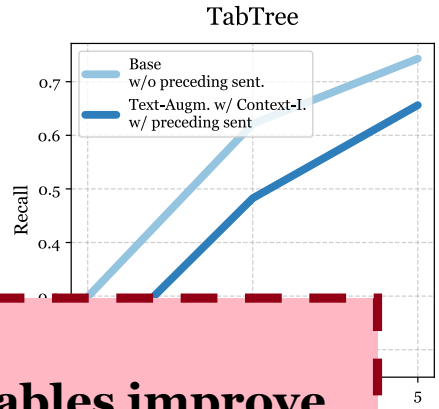


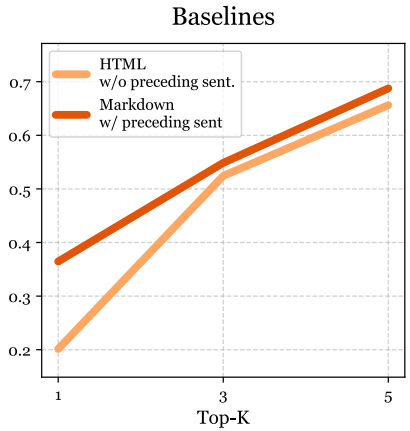


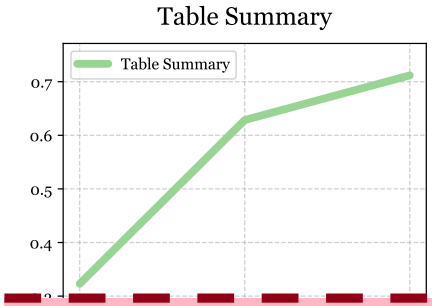
TabTree Base performs **best**
→ Recall@5: 0.71 / 0.76

Baselines perform **comparably**
→ All slightly worse than TabTree Base

Match Within Related Table Strategy







Related tables improve results in most cases

Table Summaries perform **comparably** to baselines
→ *But:* Costly LLM summary calls

Results – Table QA on Full Documents

Retrieval Serialization	Related Table Serialization	GPT-4o-mini		Phi-4	
		Exec. Acc.	Wind. Exceed	Exec. Acc.	Wind. Exceed
HTML	HTML	0.46	0	0.43	44
	–	0.45	0	0.40	0
Markdown	HTML	0.45	0	0.44	77
	–	0.39	0	0.35	0
TabTree - Base	HTML	**0.55**	1	**0.51**	35
	TabTree - Text	**0.55**	1	0.50	168
	–	**0.53**	0	**0.48**	0
Table Summary	HTML	0.39	0	0.35	47

Table: Full Document Results (Exec. Accuracy & Context Window Overflows) for representative table serialization approaches ($k = 3$)

Results – Table QA on Full Documents

Accuracy in full document setting **remains modest (best : 55 %)**
→ *TabTree Base > HTML / Markdown > Table Summary*

Retrieval Serialization	Related Table Serialization	GPT-4o-mini		Phi-4	
		Exec. Acc.	Wind. Exceed	Exec. Acc.	Wind. Exceed
HTML	HTML	0.46	0	0.43	44
	–	0.45	0	0.40	0
Markdown	HTML	0.45	0	0.44	77
	–	0.39	0	0.35	0
TabTree - Base	HTML	**0.55**	1	**0.51**	35
	TabTree - Text	**0.55**	1	0.50	168
	–	**0.53**	0	**0.48**	0
Table Summary	HTML	0.39	0	0.35	47

Results – Table QA on Full Documents

Accuracy in full document setting **remains modest (best : 55 %)**
→ *TabTree Base > HTML / Markdown > Table Summary*

Full document results **mirror** retrieval results
→ **Retrieval = Bottleneck**

Retrieval Serialization	Related Table Serialization	GPT-4o-mini		Phi-4	
		Exec. Acc.	Wind. Exceed	Exec. Acc.	Wind. Exceed
HTML	HTML	0.46	0	0.43	44
	–	0.45	0	0.40	0
Markdown	HTML	0.45	0	0.44	77
	–	0.39	0	0.35	0
TabTree - Base	HTML	**0.55**	1	**0.51**	35
	TabTree - Text	**0.55**	1	0.50	168
	–	**0.53**	0	**0.48**	0
Table Summary	HTML	0.39	0	0.35	47

Results – Table QA on Full Documents

Accuracy in full document setting **remains modest (best : 55 %)**
→ *TabTree Base > HTML / Markdown > Table Summary*

Full document results **mirror** retrieval results
→ **Retrieval = Bottleneck**

Retrieval Serialization	Related Table Serialization	GPT-4o-mini Exec. Acc.	GPT-4o-mini Wind. Exceed	Phi-4 Exec. Acc.	Phi-4 Wind. Exceed
HTML	HTML	0.46	0	0.43	44
	–	0.45	0	0.40	0
Markdown	HTML	0.45	0	0.44	77
	–	0.39	0	0.35	0
TabTree - Base	HTML	**0.55**	1	**0.51**	35
	TabTree - Text	**0.55**	1	0.50	168
	–	**0.53**	0	**0.48**	0
Table Summary	HTML	0.39	0	0.35	47

Including related tables leads to **small**, but **consistent gains**
→ *But*: Costly

Results – Table QA on Full Documents

Accuracy in full document setting **remains modest (best : 55 %)**
→ *TabTree Base > HTML / Markdown > Table Summary*

Full document results **mirror** retrieval results
→ **Retrieval = Bottleneck**

Retrieval Serialization	Related Table Serialization	GPT-4o-mini Exec. Acc.	GPT-4o-mini Wind. Exceed	Phi-4 Exec. Acc.	Phi-4 Wind. Exceed
HTML	HTML	0.46	0	0.43	44
	–	0.45	0	0.40	0
Markdown	HTML	0.45	0	0.44	77
	–	0.39	0	0.35	0
TabTree - Base	HTML	**0.55**	1	**0.51**	35
	TabTree - Text	**0.55**	1	0.50	168
	–	**0.53**	0	**0.48**	0
Table Summary	HTML	0.39	0	0.35	47

Including related tables leads to **small**, but **consistent gains**
→ *But*: Costly

GPT-4o-mini outperforms Phi-4
→ Smaller context window of Phi-4 yields more errors

Outline

Key Findings

⚙️ **LLMs perform well** in Vanilla Table QA (up to 76%), within **RAG-setting performance drops** to max. 55%

📝 **Markup-based** table serializations – expecially **HTML** – work best for **Vanilla Table QA**

📝 **Condensed table serialization** – as **TabTree Base** – work best for **retrieval** and **full-document QA**

🔑 **Retrieval** is the **key bottleneck** in full-document setting

- Once relevant chunks are retrieved, LLMs answer reliably

Questions

Thanks for your attention!

Questions?

References I


Abdin, Marah et al. (Dec. 2024). *Phi-4 Technical Report*. arXiv:2412.08905 [cs]. DOI: 10.48550/arXiv.2412.08905. URL: http://arxiv.org/abs/2412.08905 (visited on 03/07/2025).


Fang, Xi et al. (June 2024). *Large Language Models(LLMs) on Tabular Data: Prediction, Generation, and Understanding – A Survey*. arXiv:2402.17944 [cs]. DOI: 10.48550/arXiv.2402.17944. URL: http://arxiv.org/abs/2402.17944 (visited on 03/19/2025).


Grattafiori, Aaron et al. (Nov. 2024). *The Llama 3 Herd of Models*. arXiv:2407.21783 [cs]. DOI: 10.48550/arXiv.2407.21783. URL: http://arxiv.org/abs/2407.21783 (visited on 03/07/2025).


Kamradt, Greg (Dec. 2024). *5 Levels Of Text Splitting*. en. URL: https://github.com/FullStackRetrieval-com/RetrievalTutorials/blob/main/tutorials/LevelsOfTextSplitting/5_Levels_Of_Text_Splitting.ipynb (visited on 03/06/2025).


OpenAI (July 2024a). *GPT-4o mini: advancing cost-efficient intelligence*. en-US. URL: https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/ (visited on 03/07/2025).

References II


OpenAI (Mar. 2024b). *New embedding models and API updates*. en-US. URL: https://openai.com/index/new-embedding-models-and-api-updates/ (visited on 03/07/2025).


OpenAI et al. (Oct. 2024). *GPT-4o System Card*. arXiv:2410.21276 [cs]. DOI: 10.48550/arXiv.2410.21276. URL: http://arxiv.org/abs/2410.21276 (visited on 03/07/2025).


Pasupat, Panupong et al. (Aug. 2015). *Compositional Semantic Parsing on Semi-Structured Tables*. arXiv:1508.00305 [cs]. DOI: 10.48550/arXiv.1508.00305. URL: http://arxiv.org/abs/1508.00305 (visited on 02/19/2025).

Outline

Construction of the Column Header Tree I

1. **Root Node**: Start with a single root v_{root^c} and the colour $\phi_c(v_{\text{root}^c}) = \texttt{yellow}$.
2. **Column Header Rows**: For each column header row $r_i \in h_t, i = 1, \ldots, i^*$ do the following:
   1. Create child nodes $v_{\text{new}} = \tilde{\gamma}_{ij}$ for all column header cells within the current row ($v_{\text{new}} \in r_i \setminus l_t$) containing distinct values ($\tilde{s}^c_{\text{prev}}(i,j) = 0$) and colour them `yellow`.
   2. Connect each cell v_{new} to its corresponding parent node from the preceding column header, or v_{root^c} for the first row.
   3. Add reference nodes v_{ref} for cells of the intersection of the current column header row and any row label of the same row ($r_i \cap l_t$) and colour them `orange`.
   4. Connect reference nodes v_{ref} iteratively from left to right until j^*. Additionally, add edges from the most right reference node to all column header cells of the same row.

Construction of the Column Header Tree II

3. **Leaf Nodes**: Add all value cells of the table, i.e., $\tilde{\gamma}_{ij} \in \{r_{i^*+1}, \ldots, r_n\} \times \{c_{j^*+1}, \ldots c_m\}$, as leaf nodes:
   1. Create leaf nodes $v_{\text{new}} = \tilde{\gamma}_{ij}$ and colour them `gray`.
   2. Connect v_{new} to its parent column header node.

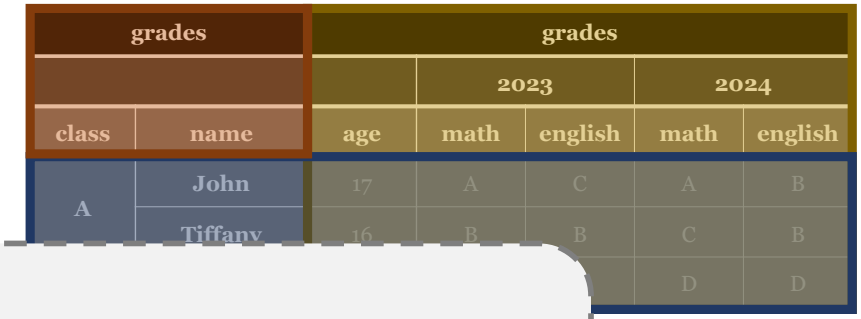
Construction of the Row Label Tree I

1. **Root Node**: Start with a single root v_{root^r} and the colour $\phi_r(v_{\text{root}^r}) = \texttt{blue}$.
2. **Row Label Columns**: For each row label column $c_j \in h_t, j = 1, \ldots, j^*$ do the following:
   1. Create child nodes $v_{\text{new}} = \tilde{\gamma}_{ij}$ for all row label cells within the current column ($v_{\text{new}} \in c_j \setminus h_t$) containing distinct values ($\tilde{s}^r_{\text{prev}}(i,j) = 0$) and colour them `blue`.
   2. Connect each cell v_{new} to its corresponding parent node from the preceding column header, or v_{root^r} for the first column.
   3. Add reference nodes v_{ref} for cells of the intersection of the current row label column and any column header of the same column ($c_j \cap h_t$) and colour them `orange`.
   4. Connect reference nodes v_{ref} iteratively from top to bottom until i^*. Additionally, add edges from the most bottom reference node to all column header cells of the same column.

Construction of the Row Label Tree II

3. **Leaf Nodes**: Add all value cells of the table, i.e., $\tilde{\gamma}_{ij} \in \{r_{i^*+1}, \ldots, r_n\} \times \{c_{j^*+1}, \ldots c_m\}$, as leaf nodes:
   1. Create leaf nodes $v_{\text{new}} = \tilde{\gamma}_{ij}$ and colour them `gray`.
   2. Connect v_{new} to its parent row label node.

Exemplary Derivation of the Column Header Tree



grades		grades				
			2023		2024	
class	name	age	math	english	math	english
A	John	17	A	C	A	B
	Tiffany	16	B	B	C	B
					D	D

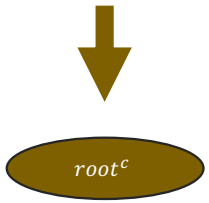
Apply necessary table modifications

→ *Split cells spanning across row labels & column headers*

→ *Remove spans from value level cells*

Exemplary Derivation of the Column Header Tree

grades		grades				
			2023		2024	
class	name	age	math	english	math	english
A	John	17	A	C	A	B
	Tiffany	16	B	B	C	B
B	Michael	17	D	D	D	D



Start with $root^c$

→ Colour: Yellow

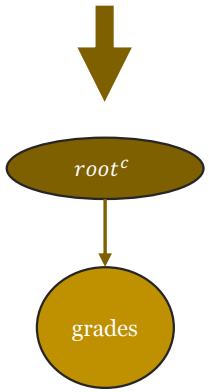
Exemplary Derivation of the Column Header Tree

grades		grades				
			2023		2024	
class	name	age	math	english	math	english
A	John	17	A	C	A	B
	Tiffany	16	B	B	C	B
	ael	17	D	D	D	D

For each column header row:

Add column header node for each distinct cell

→ Colour: Yellow



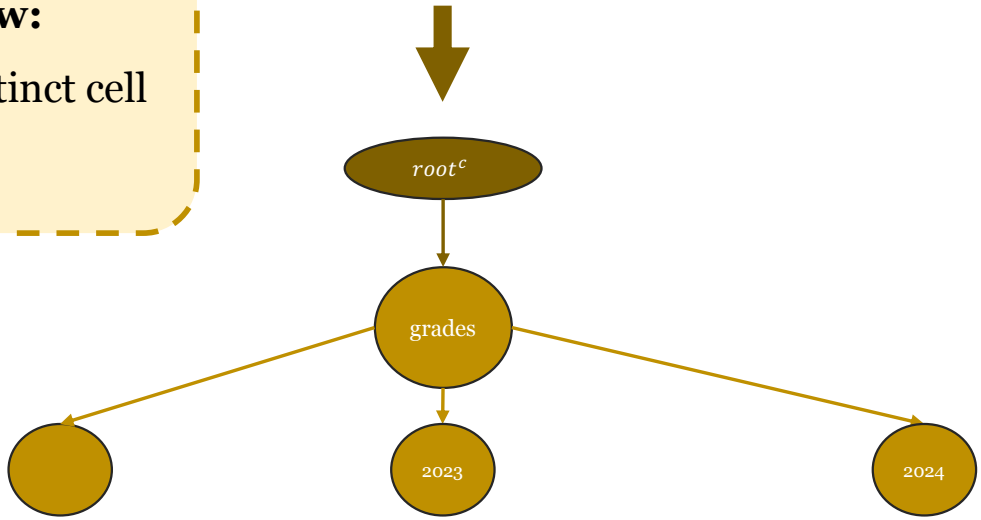
Exemplary Derivation of the Column Header Tree

grades		grades				
			2023		2024	
class	name	age	math	english	math	english
A	John	17	A	C	A	B
	Tiffany	16	B	B	C	B
	ael	17	D	D	D	D

For each column header row:

Add column header node for each distinct cell

→ Colour: Yellow



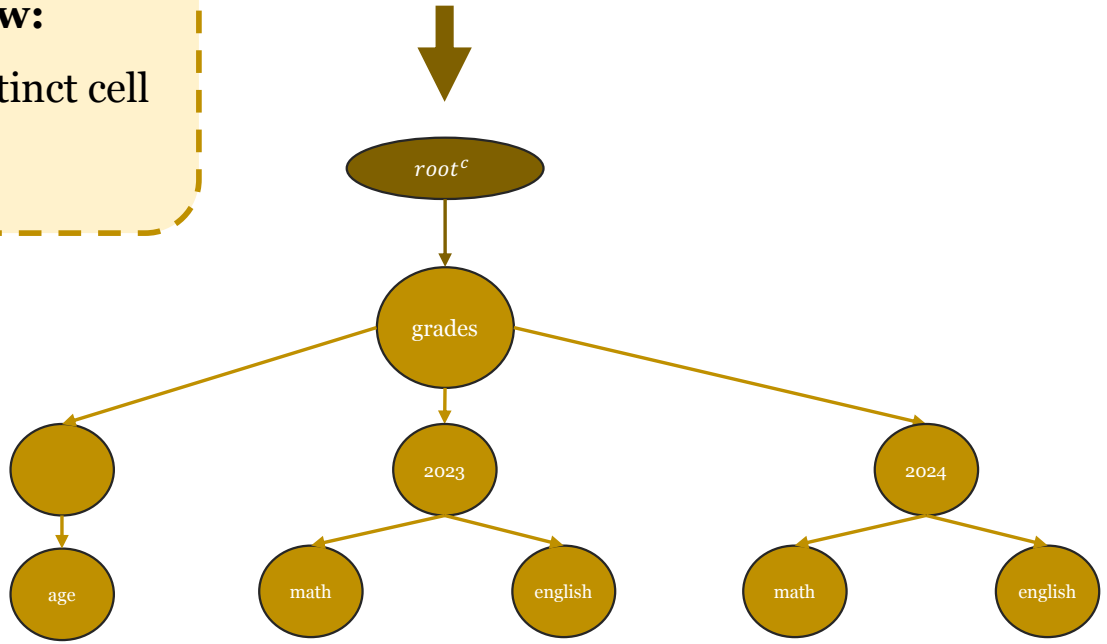
Exemplary Derivation of the Column Header Tree

grades		grades				
			2023		2024	
class	name	age	math	english	math	english
A	John	17	A	C	A	B
	Tiffany	16	B	B	C	B
	ael	17	D	D	D	D

For each column header row:

Add column header node for each distinct cell

→ Colour: Yellow



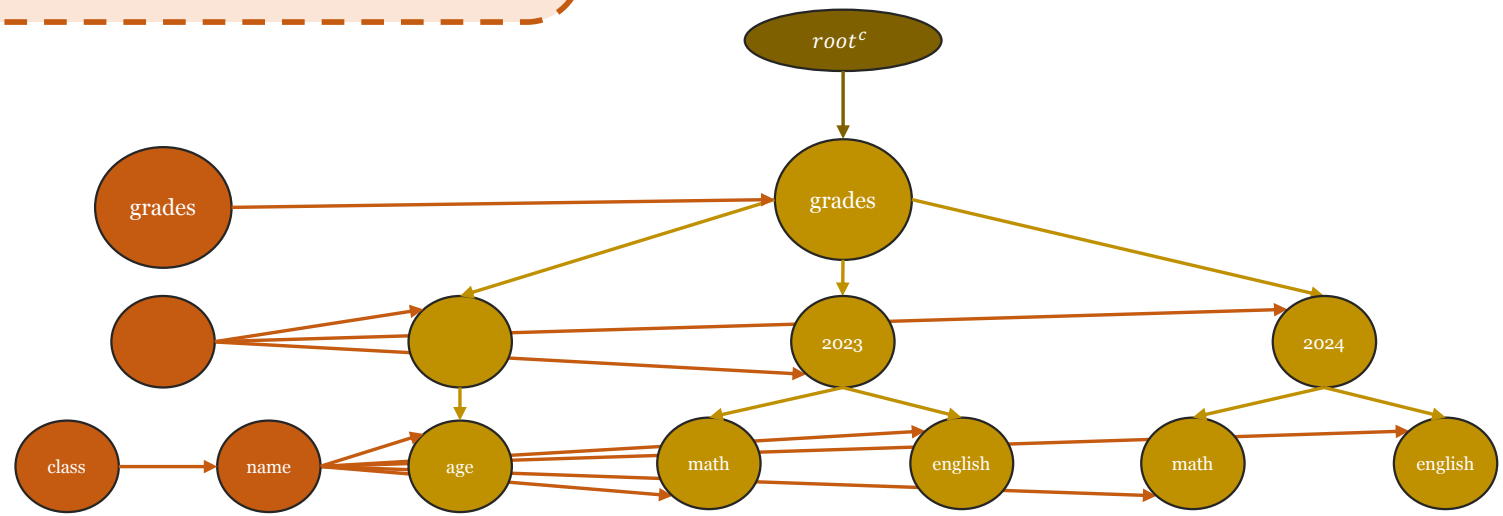
Exemplary Derivation of the Column Header Tree

grades		grades				
			2023		2024	
class	name	age	math	english	math	english
	John	17	A	C	A	B
	ny	16	B	B	C	B
	ael	17	D	D	D	D

For each row:

Add missing context intersection cells for potential data augmentation

→ Colour: Orange

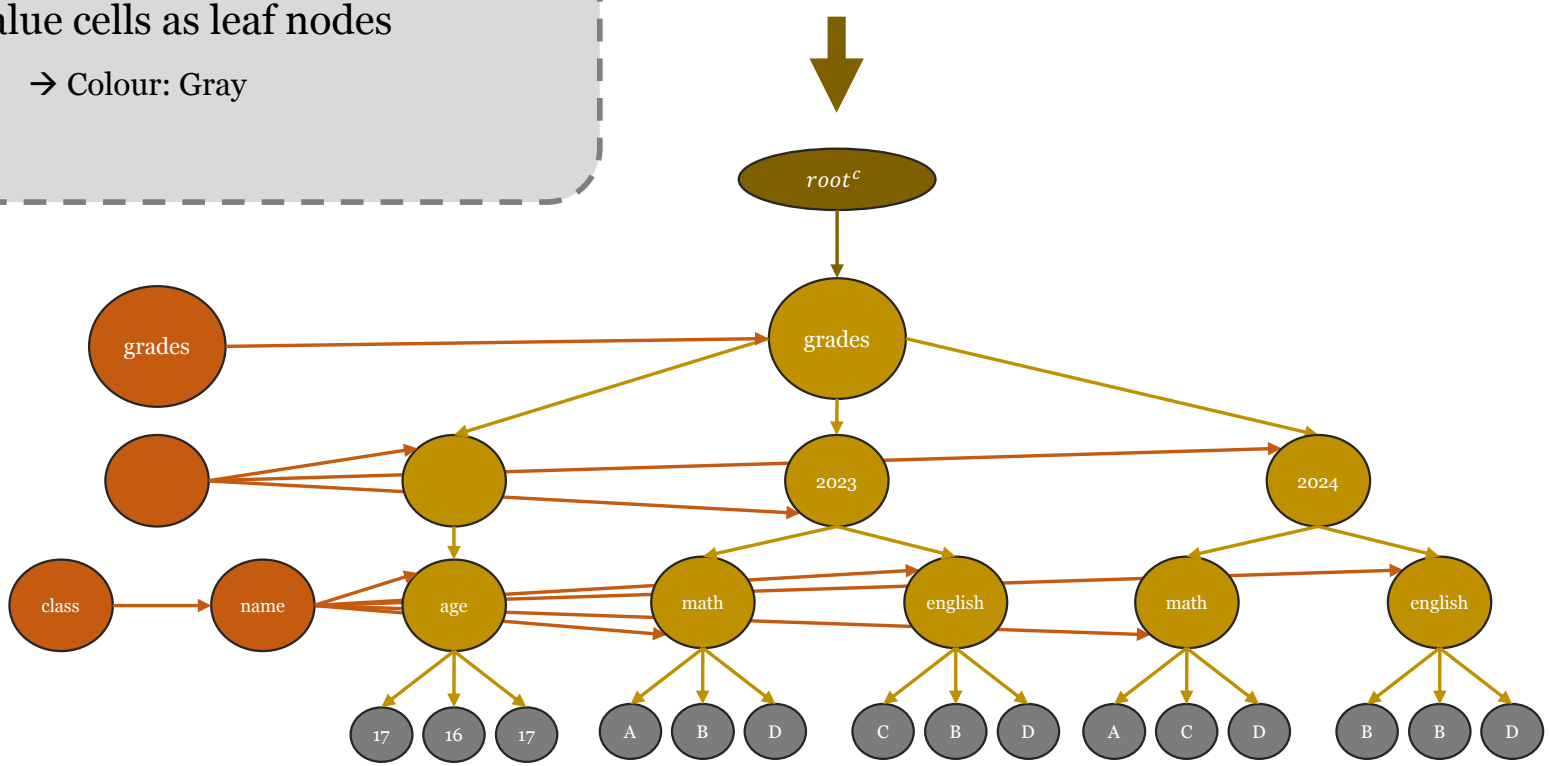


Exemplary Derivation of the Column Header Tree

grades		grades				
			2023		2024	
class	name	age	math	english	math	english
A	John	17	A	C	A	B
	ny	16	B	B	C	B
	l	17	D	D	D	D

Add value cells as leaf nodes

→ Colour: Gray



TabTree Serialization I

Core Idea **Mitigate the limitations** of the two-dimensional structure of tables by **effectively leveraging the tree structure** in the TabTree model

Generation Process

1. **Initialize Serialization String**: Initialize empty serialization string $p_t = ""$
2. **Primary Subtree Selection:** Select a direction colour $CLR \in \{\texttt{yellow}, \texttt{blue}\}$ and a corresponding primary subtree T_1 and secondary subtree T_2.
3. **Context String Generation:** Perform a DFS traversal of the primary subtree T_1, starting at its root. For each non-leaf node in T_1 do the following:

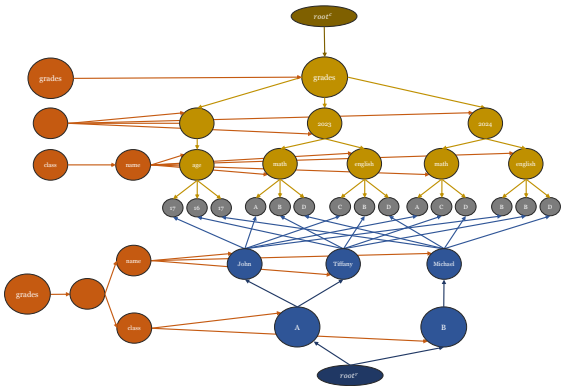
TabTree Serialization II

1. Generate a context string $p_{\text{context}}(v)$ capturing the context of v within T_1 and append it to p_t.
2. Optionally, include context-intersection connections (edges to nodes coloured `orange`)

4. **Value String Generation:** Once the DFS traversal of the primary tree arrives at a value cell / value node with colour `gray`, generate a value string $p_{\text{value}}(v)$ by the following:
   1. Traverse the reversed secondary subtree $\overleftarrow{T_2}$ from v towards its root.
   2. Aggregate information from the nodes encountered during this traversal and append the final value string $p_{\text{value}}(v)$ to the output string p_t.
   3. Optionally, include context-intersection connections (edges to nodes coloured `orange`).

TabTree Serialization – Modular System

TabTree Model



Modular System

- **Context String Approach**
  - Base
  - Text-Augmentation
  - Empty
- **Value String Approach**
  - Base
  - Text
  - Text-Augmentation
- **Primary Subtree Selection**
  - Column Header 1st
  - Row Label 1st
  - Concatenation
  - Heuristic

DFS-based Serialized String

```
grades:
    age:
        - A > John: 17
        - A > Tiffany: 16
        - B > Michael: 17
    2023:
        math:
            - A > John: A
            - A > Tiffany: B
            - B > Michael: D
        english:
            - A > John: C
            - A > Tiffany: B
            - B > Michael: D
    2024:
        math:
            ...
        english:
            ...
```

TabTree – Primary Subtree Selection

- **Column Header Tree as Primary Subtree:** Always select Column Header Tree
- **Row Label Tree as Primary Subtree:** Always select Row Label Tree
- **Concatenated Representation:** Generate string representations using both the Column Header Tree and the Row Label Tree as primary subtrees
- **Heuristic-Based Selection:** Select tree with deeper nested structure (i.e., more column headers or more row labels)

TabTree Serialization – String Generation Combinations

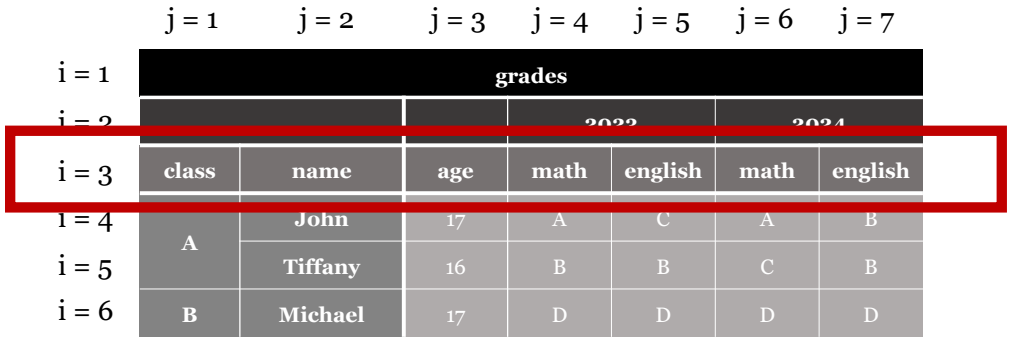
Name	Context String Approach	Incl. Context-Intersection?	Value String Approach	Incl. Context-Intersection?
Base	Base	❌	Base	❌
Text	Text-Augmentation	❌	Text	❌
Text w/ Context-Intersection	Text-Augmentation	✅	Text	✅
Text-Augm. w/ Context-I.	Text-Augmentation	✅	Text-Augmentation	✅
Context-Empty	Empty	❌	Text-Augmentation	✅

Table Context Detection

⚙️ Approach:

1. Start with an **empty sequence of column headers / row labels**
2. **Iteratively query LLM** whether a row or column is a header or label.
3. Start from the **top row** or **left column**.
4. Stop at the first **non-header row** or **non-label column**.

Table Context Detection



Full Table

```
You are tasked with analyzing table
data to determine whether the provided
row should be classified as a column
header or not.

Instructions:
…

Row to analyze:
['class', 'name', 'age', 'math',
'english', 'math', 'english']

The full table is the following:
<table>
    <thead>
        <tr>
            <th
colspan="7">grades</th>
        </tr>
        ...
</table>
```

1-Range

```
You are tasked with analyzing table
data to determine whether the provided
row should be classified as a column
header or not.

Instructions:
…

Row to analyze:
['class', 'name', 'age', 'math',
'english', 'math', 'english']

The previous row is the following:
['','','','2023','2023','2024','2024']

The next row is the following:
['A','John','17','age','A','C','A','B']
```

2-Range

```
You are tasked with analyzing table data
to determine whether the provided row
should be classified as a column header or
not.

Instructions:
…

Row to analyze:
['class', 'name', 'age', 'math',
'english', 'math', 'english']

The previous row is the following:
['','','','2023','2023','2024','2024']

The second previous row is the following:
['grades','grades', ..., 'grades']

The next row is the following:
['A','John','17','A','C','A','B']

The second next row is the following:
['A','Tiffany','16','B','B','C','B']
```

Outline

WikiTableQuestions – Question Categories

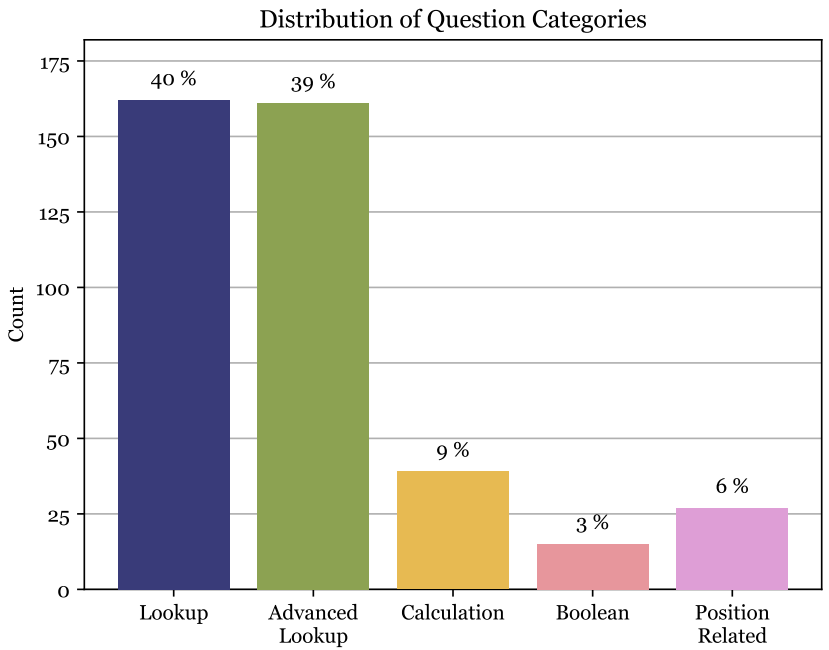
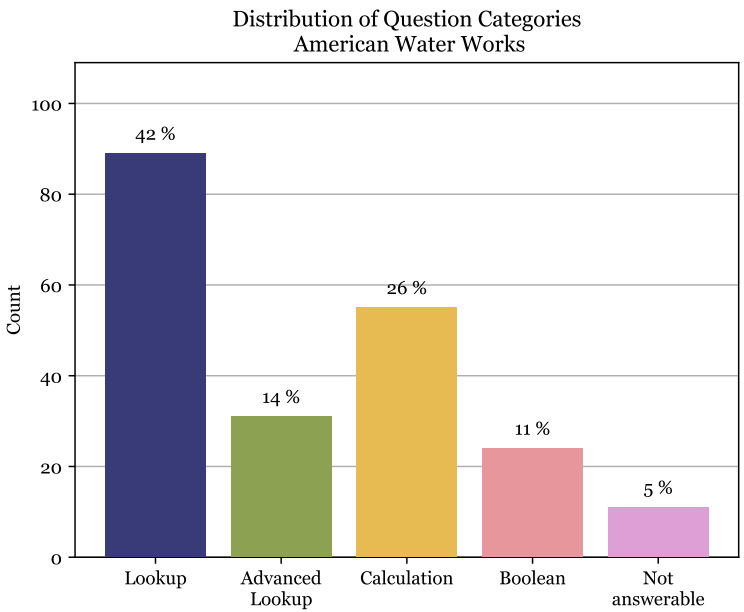
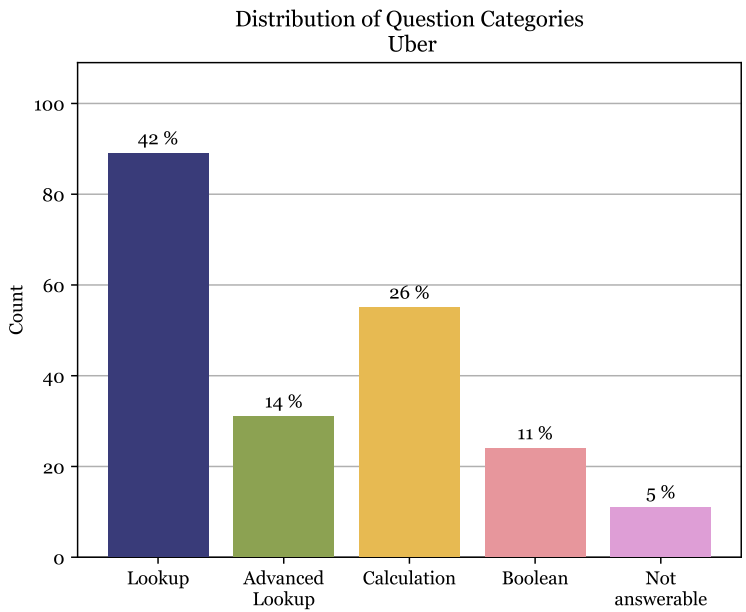


Figure: Question Category Distribution of the 200 evaluated samples from the WikiTableQuestions dataset

SEC Filing Tables – Question Categories



(a) American Water Works



(b) Uber

Figure: Question category distribution for the SEC-Filing Tables dataset

Outline

Context Detection Results

`Ground Truth Header Indices`	`[0,1]`	`[0]`	`[0,1]`
`Predicted Header Indices`	`[0]`	`[0,1,2]`	`[0,1]`
Accuracy Full	0	0	1
Accuracy Fine-grained	0.5	0.33	1
F1 Fine-grained	0.67	0.5	1

(a) Evaluation Metrics Example

Approach	**Column Headers**			**Row Labels**		
	Acc. Full	Acc. Fine-grained	F1 Fine-grained	Acc. Full	Acc. Fine-grained	F1 Fine-grained
Full Table	0.76	0.86	0.83	0.76	**0.66**	**0.63**
1-Range	0.76	0.86	0.83	0.76	0.51	0.47
2-Range	**0.81**	**0.87**	**0.85**	**0.81**	0.63	0.62

(b) Context Detection Results

Table: Evaluation results for Context Detection on $n = 21$ manually annotated samples created for tables from the WikiTableQuestions and SEC-Filings datasets.

Retrieval Metrics

- **Binary Retrieval Setting**: Ground truth tables retrieved **at least once** among top k chunks – or **not at all**.
- **Recall@k = Hit Rate**: Measures if a relevant chunk appears in top-k retrieved results.

$$R@k = \frac{\#\text{ of queries with relevant chunk in top } k}{\text{Total } \#\text{ of queries}}$$

- **MRR@k**: Mean Reciprocal Rank of the first relevant chunk within top-k.

$$\text{MRR@k} = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{\text{rank}_i} \cdot \mathbb{I}(\text{rank}_i \leq k)$$

- Other metrics (e.g., Precision, mAP, nDCG) not applicable in this setting

Retrieval Metrics in Non-Binary Setting

- **Precision@k**: Measures the proportion of retrieved chunks in the top-k that are relevant.

$$\text{Precision@k} = \frac{\#\text{ of relevant chunks in top } k}{k}$$

- **Recall@k**: Measures the **proportion of all relevant chunks** that are retrieved in the top-k.

$$\text{Recall@k} = \frac{\#\text{ of relevant chunks in top } k}{\text{Total }\#\text{ of relevant chunks}}$$

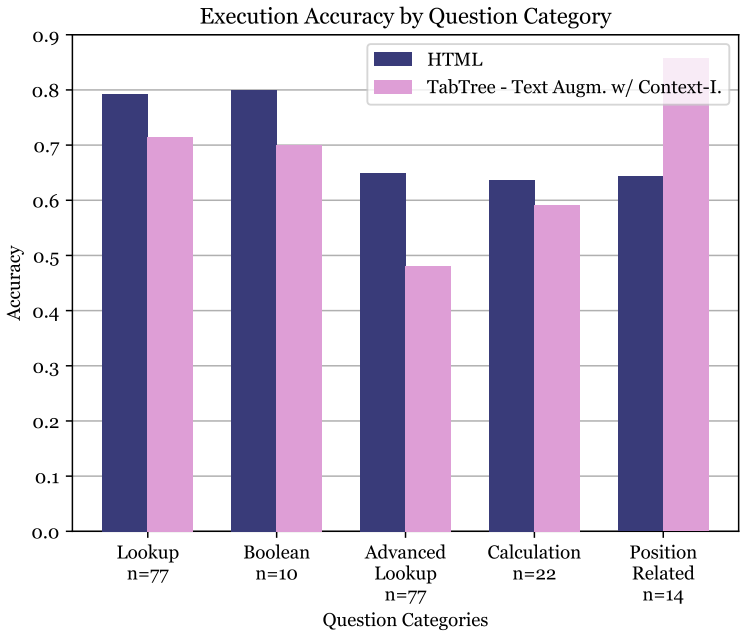
Table QA Metrics

- **Execution Accuracy**: Compares predicted and ground truth answers after normalization (e.g., case folding, numerical formatting) to account for formatting variations.
- **Macro F1-score**: Average of precision/recall based on word overlap, post normalization.

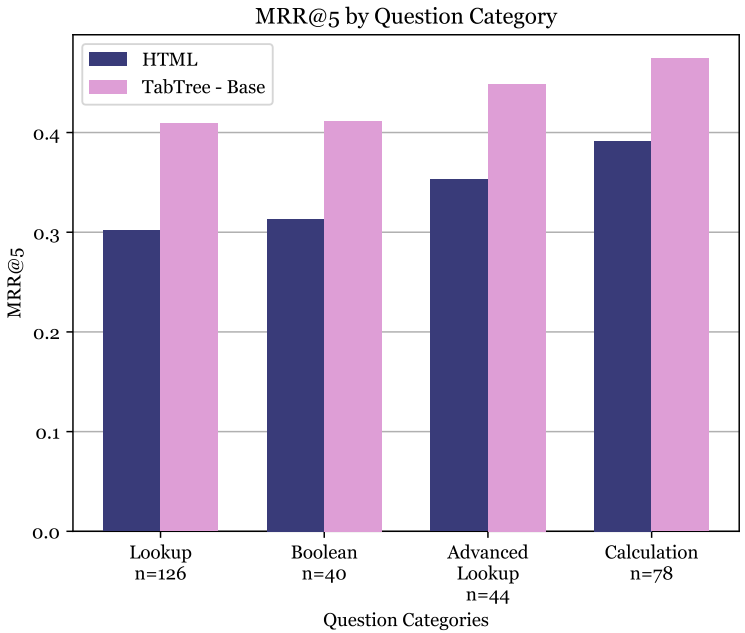
Results – Retrieval Methodology

- **Evaluation Strategy**: *Match within Chunk* vs. *Match within Related Table*
  - Semantic coherent chunks **might be split** due to **embedding size limits**
  - *Solution*: Store **"related tables"** as **metadata**
- **Inclusion of Preceding Sentences** (optional): Prepend sentence immediately preceding each table to serialized table
  - *Example:* "Presented in the table below is a breakout of..."
  - *Goal:* Enhance contextual coherence of embeddings

Question Category Analysis



(a) Vanilla Table QA



(b) Retrieval

Figure: Results by question complexity categories for two representative approaches (HTML and TabTree)

Question Category Analysis

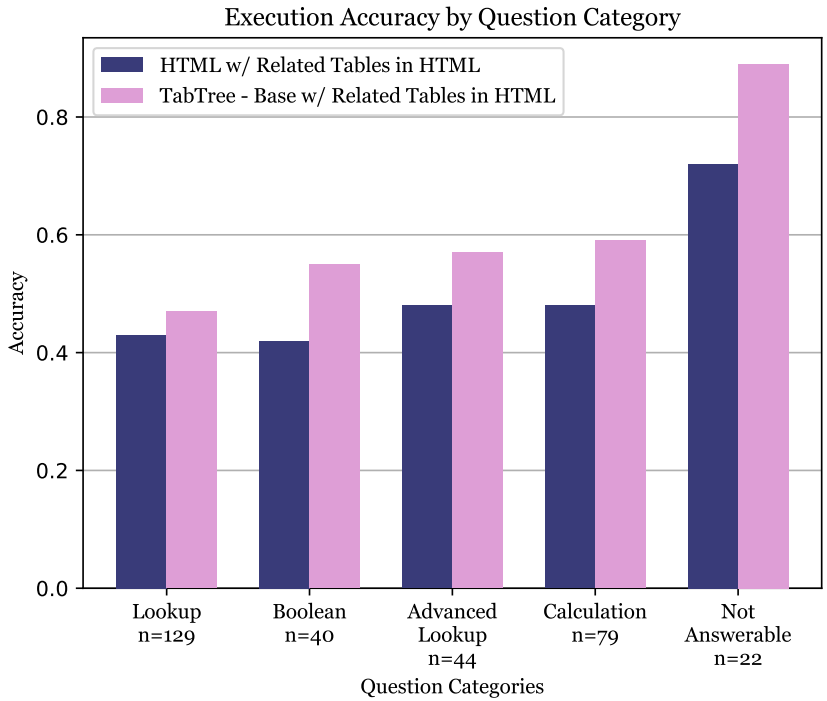
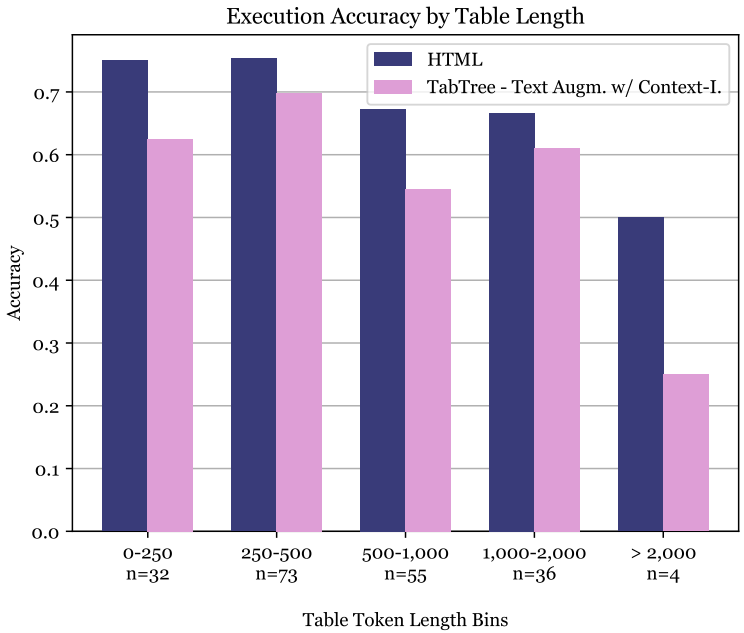
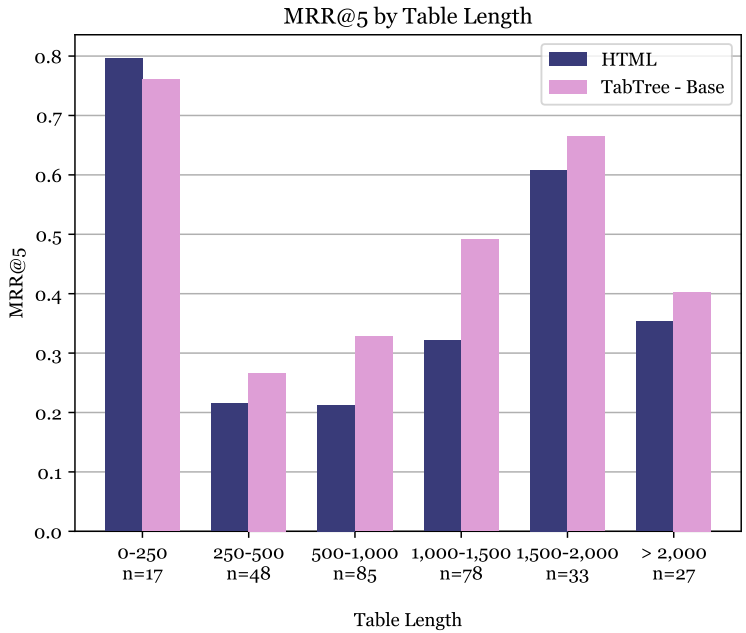


Figure: Table QA on Full Document results by question complexity categories for two representative approaches (HTML and TabTree)

Table Token Length Analysis



(a) Vanilla Table QA



(b) Retrieval

Figure: Results by bins of token lengths of corresponding tables for two representative approaches (HTML and TabTree)

Table Token Length Analysis

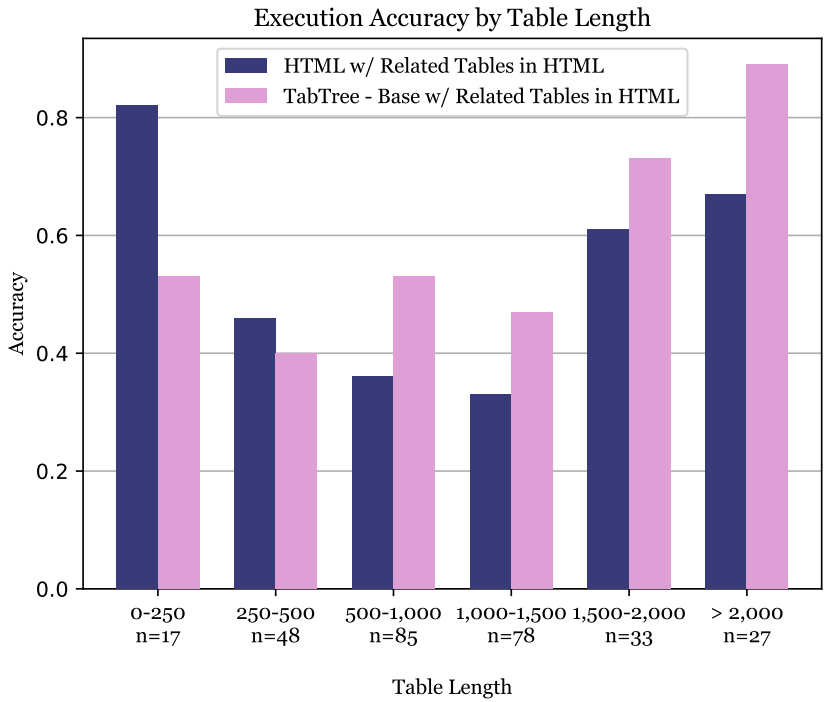


Figure: Table QA on Full Document results by bins of token lengths of corresponding tables for two representative approaches of baselines (HTML and TabTree)

Outline

Table Serializations

📈 Key Observations from Literature:

- **Markup-based formats** such as HTML, Markdown, CSV, and JSON yield the **best performance – sentence-based** approaches perform mostly **worse** (Fang et al., 2024).
  - Likely due to **high pre-training exposure** of markdown formats
  - *But:* Sentence-based formats usually rely **only on simple attribute-value patterns** (e.g., '<attribute> is <value>').
- **Limited research** on how serialization choices impact **retrieval effectiveness**

Limitations & Future Work

- **Limited Retrieval Scope:** Focus solely on dense retrieval
  - 💡 Hybrid approaches with reranking, query rewriting, or retrieval routing
- **Single Embedding Model:**
  - 💡 Additional embedding models
- **Long Content Handling:** Naive handling of content exceeding input limits of embedding models and LLMs
  - 💡 Multi-stage & table structure-aware content selection before answer generation
- **Solely RAG Architecture:**
  - 💡 Multi-route architectures, fine-tuning, or tabular foundation models