Heidelberg University
Institute of Computer Science
Data Science Group

Master's Thesis

Exploring Table Representations for Question Answering

Name:	Johannes Gabriel Sindlinger
Matriculation Number:	3729339
Advisor:	Prof. Dr. Michael Gertz
Date of Submission:	April 14, 2025

I hereby certify that I have written this thesis myself and have not used any sources or aids other than those stated, and that I have labelled any work from other sources as such. I also confirm that the content and wording of the electronically submitted version of my thesis is completely identical to the printed version. I agree that this electronic version may be checked for plagiarism at the university using plagiarism software.

Johannes Gabriel Sindlinger

Date of Submission: April 14, 2025

Zusammenfassung

Aufgrund ihrer zweidimensionalen Struktur und oftmals sehr heterogenen Layouts stellen Tabellen eine erhebliche Herausforderung für Question Answering-Systeme dar – insbesondere dann, wenn sie nicht isoliert vorliegen, sondern in vollständige Dokumente bestehend aus textuellen und tabellarischen Inhalten eingebettet sind. Large Language Models gelten als vielversprechendes Mittel zur Beantwortung tabellenbezogener Fragen (Table QA). Ihre Anwendung auf derartige gemischte Dokumente erweist sich jedoch als schwierig, da sowohl strukturierte als auch unstrukturierte Inhalte berücksichtigt werden müssen. Diese Arbeit adressiert diese Herausforderungen und stellt ein System mit drei zentralen Beiträgen vor: (i) die Erstellung eines manuell annotierten Datensatzes aus Finanzdokumenten, (ii) die Entwicklung einer Retrieval-Augmented Generation-Pipeline, die gezielt die Herausforderungen von Large Language Models im Table QA adressiert, sowie (iii) die Einführung von TabTree, einer Serialisierungsmethode, die Tabellen als gerichtete Bäume auf Basis von Spalten- und Zeilenbezeichnungen modelliert und daraus vorlagenbasierte natürlichsprachliche Repräsentationen generiert.

Experimente zeigen, dass Large Language Models in der Lage sind, tabellenbasierte Fragen zuverlässig zu beantworten, wenn Tabellen in klassischen Formaten wie HTML, CSV, JSON oder Markdown serialisiert vorliegen. Während TabTree in der direkten Antwortgenerierung hinter diesen Formaten zurückbleibt, erzielt eine seiner Varianten – TabTree Base – die besten Ergebnisse im dokumentenbasierten Question-Answering-Szenario, in welchem relevante Tabelleninhalte zunächst identifiziert werden müssen. Insgesamt zeigen die Ergebnisse einen erheblichen Leistungsabfall bei allen Methoden im Übergang von Tabellen- zu Dokumentenebene, was die Bedeutung effizienter Retrieval-Methoden für zukünftige Forschung unterstreicht.

Abstract

Tables, with their two-dimensional structure and heterogeneous layouts, present significant challenges for question answering systems, especially when embedded in full documents alongside unstructured text. While Large Language Models show promise for Table Question Answering (Table QA), applying them to such mixed-format documents remains difficult due to the need to reason over both structured and unstructured content. To address these challenges, this thesis introduces a framework with three main contributions: (i) a manually annotated dataset of financial documents with table-related questions, (ii) a Retrieval-Augmented Generation pipeline tailored for table-aware processing, and (iii) TabTree, a table serialization method that models tables as directed trees based on column headers and row labels, generating template-based serializations.

Empirical results show that Large Language Models perform well, when tables are provided in advance, with standard serializations like HTML, CSV, JSON, and Markdown yielding the best performance. While the introduced TabTree serialization underperforms in this direct Table QA setting, one of its variants considerably improves retrieval effectiveness when relevant table content must first be identified within full documents. Overall, question answering performance decreases considerably across all serialization methods when moving from the direct-table setting to the document-level setting. These findings highlight retrieval as the key bottleneck in document-level Table QA and point towards potential future research opportunities.

Table of Contents

List of Figures

List of Tables

List of Algorithms

1. Introduction

Tables are like cobwebs, like the sieve of Danaides; beautifully reticulated, orderly to look upon, but which will hold no conclusion. Tables are abstractions, and the object a most concrete one, so difficult to read the essence of. There are innumerable circumstances; and one circumstance left out may be the vital one on which all turned.

Thomas Carlyle (1842)

1.1. Motivation

Tables are ubiquitous in various domains, serving as primary means of structuring and representing information. They are used in corporate settings for production, sales, and management data (Richardson, 2022), in scientific research for presenting experimental results, and as foundational elements in modern machine learning applications (Chui et al., 2018), among many other fields. The primary challenge for human users lies in extracting, processing, and interpreting the information contained within tables, a task that is often complex, as highlighted by Carlyle.

Recent advancements in artificial intelligence (AI), particularly Large Language Models (LLMs), have significantly enhanced natural language processing (NLP) capabilities (Minaee et al., 2024). LLMs are applied to multimodal domains, including images, video, and tables (S. Yin et al., 2024). Since tables mainly consist of textual content, researchers have begun leveraging LLMs for table-related tasks (Fang et al., 2024; Lu et al., 2024; Ruan et al., 2024).

However, tables exhibit a structural complexity distinct from conventional text. Unlike continuous natural language text, which follows a sequential structure, tables are two-dimensional. Thus, understanding tables requires recognizing relationships between rows and columns. This structural difference poses challenges when adapting LLMs for table-related tasks, as these models are trained on sequential text data and process information sequentially through self-attention mechanisms (Lu et al., 2024).

To accommodate LLM processing, tables are often serialized into a one-dimensional format, such as HTML (Fang et al., 2024). However, this transformation disrupts their inherent structure, stripping away spatial cues that are critical for accurate interpretation (Sui et al., 2024). As a result, comprehension becomes more challenging for both humans and LLMs. Figure 1.1 illustrates this issue by comparing a human-readable table with its machine-readable HTML representation. When answering a question such as *'What is the average age of the students?'*, extracting relevant information from an HTML-encoded table is significantly more complex than from its original two-dimensional layout.

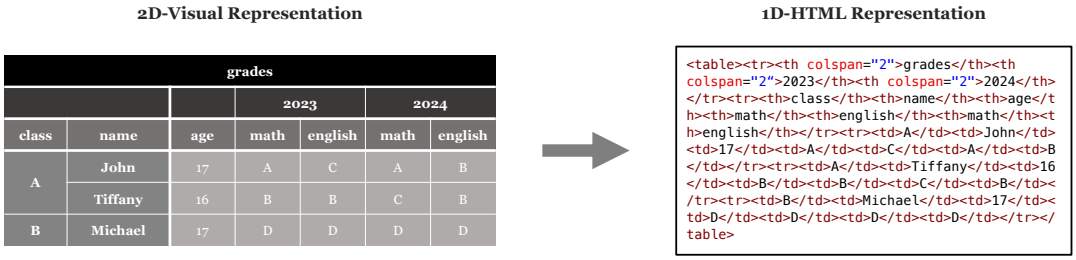


Figure 1.1.: Comparison of a human-readable, two-dimensional table representation and its corresponding machine-readable, one-dimensional HTML representation.

.

In addition to these architectural limitations, tables pose further challenges for automated systems due to their heterogeneity. Real-world tables vary widely in structure, data types, suffering from sparsity or imbalance, and often exhibit complex inter-row and inter-column relationships (S. Zhang et al., 2020; Fang et al., 2024). Figure 1.2 showcases this structural diversity through four examples.

Given these complexities, traditional machine learning methods such as decision trees and gradient boosting machines have outperformed neural approaches in various tabular tasks (Shwartz-Ziv et al., 2021). Similarly, in Table Question Answering (Table QA), non-neural methods, especially semantic parsing approaches that translate natural language queries into executable logical forms, have remained competitive (Fang et al., 2024). However, with LLMs increasing reasoning capabilities (W. Chen, 2023) and the introduction of advanced prompting strategies such as those proposed by Sui et al. (2024), new opportunities emerge for leveraging LLMs in table-related tasks. Despite these advancements, research on table processing in the context of document integration, where tables coexist with unstructured text, remains limited.

In real-world applications, tables are rarely standalone artifacts. Whether it's a business analyst extracting financial insights from an earnings report, a lawyer identifying clauses

	Operating Revenues (in millions)				Number of Customers (in thousands)			
	Water (a)	Wastewater	Total	% of Total	Water	Wastewater	Total	% of Total
Pennsylvania	$ 810	$ 155	$ 965	24.6 %	683	98	781	22.4 %
New Jersey	908	57	965	24.6 %	668	64	732	21.0 %
Missouri	430	20	450	11.5 %	483	24	507	14.5 %
Illinois	366	61	427	10.9 %	299	72	371	10.6 %
California	300	4	304	7.8 %	190	3	193	5.5 %
Total—Top Five States (b)	2,814	297	3,111	79.4 %	2,323	261	2,584	74.1 %
Other (c)	779	30	809	20.6 %	865	37	902	25.9 %
Total Regulated Businesses	$ 3,593	$ 327	$ 3,920	100.0 %	3,188	298	3,486	100.0 %

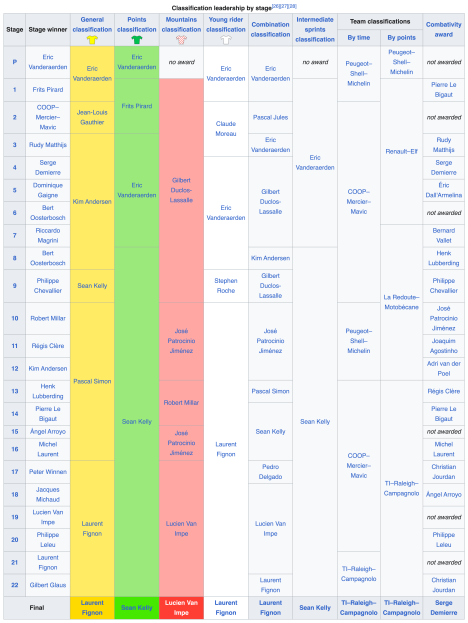
(a) U.S. SEC-filing data[1]

Type	Choice	TabFact	HybridQA	SQA	Feverous	ToTTo			
		Acc	Acc	Acc	Acc	BLEU-1	BLEU-2	BLEU-3	BLEU-4
1-shot	1-shot	72.04%	46.07%	73.81%	75.56%	72.43%	44.36%	27.01%	17.24%
1-shot	w/o table size	71.33%	45.52%	72.91%	74.66%	72.30%	44.23%	27.14%	17.25%
1-shot	w/o partition mark	71.25%	45.48%	73.09%	75.11%	71.18%	43.17%	26.36%	16.34%
1-shot	w/o format explanation	70.87%	45.39%	71.69%	75.97%	70.54%	43.59%	26.52%	16.74%
1-shot	w/o role prompting	71.35%	46.05%	73.39%	75.52%	70.61%	43.10%	26.02%	16.15%
SA	self format explanation	72.23%	46.12%	73.91%	76.15%	74.18%	45.25%	27.32%	18.34%
SA	self critical values and ranges identification	74.35%	48.20%	76.53%	76.32%	80.83%	47.96%	30.68%	22.92%
SA	self structural information description	73.42%	46.97%	75.97%	77.28%	78.93%	46.91%	28.94%	19.32%

(b) Scientific Paper[2]

Regulatory Practices	Description	States Allowed
Infrastructure replacement surcharge mechanisms	Allows rates to change periodically, outside a general rate case proceeding, to reflect recovery of capital investments made to replace infrastructure necessary to sustain safe and reliable services for the Company's customers. These mechanisms typically involve periodic filings and reviews to ensure transparency.	IA, IL, IN, KY, MO, NJ, PA, TN, VA, WV
Future test year	A "test year" is a period used for setting rates, and a future test year describes the first 12 months that new rates are proposed to be effective. The use of a future test year allows current or projected revenues, expenses and capital investments to be collected on a more timely basis.	CA, HI, IA, IL, IN, KY, PA, TN, VA
Hybrid test year	A historical test year sets rates using data from a 12-month period that ends prior to a general rate case filing. A hybrid test year allows an update to historical data for "known and measurable" changes that occur subsequent to the historical test year.	MD, MO, NJ, WV
Utility plant recovery mechanisms	Allows recovery of the full return on utility plant costs during the construction period, instead of capitalizing an allowance for funds used during construction ("AFUDC"). In addition, some states allow the utility to seek pre-approval of certain capital projects and associated costs. In this pre-approval process, the PUC may assess the prudency of such projects.	CA, IL, KY, PA, TN, VA
Expense mechanisms	Allows changes in certain operating expenses, which may fluctuate based on conditions beyond the utility's control, to be recovered outside of a general rate case proceeding or deferred until the next general rate case proceeding.	CA, HI, IL, IN, MD, MO, NJ, PA, TN, VA
Revenue stability mechanisms	Adjusts rates periodically to ensure that a utility recovers the revenues authorized in its general rate case, regardless of sales volume, including recognition of declining sales resulting from reduced consumption, while providing an incentive for customers to use water more efficiently.	CA, IL
Consolidated tariffs	Use of a unified rate structure for water systems owned and operated by a single utility, which may or may not be physically interconnected. The consolidated tariff pricing structure may be used fully or partially in a state, and is generally used to moderate the price impact of periodic fluctuations in local costs, while lowering administrative costs for customers. Pennsylvania and West Virginia also permit a blending of water and wastewater revenue requirements.	CA, IA, IL, IN, KY, MD, MO, NJ, PA, VA, WV
Deferred accounting	A regulator's willingness to defer recognition of financial impacts when setting rates for utilities.	All

(c) U.S. SEC-filing data[1]



(d) Wikipedia Article[3]

Figure 1.2.: Examples of real-world tables exhibiting diverse structures and contexts.

in regulatory documents, or a scientist interpreting results in a research paper, tables are typically embedded within larger documents. Their full meaning often depends on surrounding textual context. Understanding how LLMs can effectively process and integrate information from such mixed-format documents remains a highly relevant but sparsely studied field of research.

Summarized, the core challenges for Table QA systems of mixed-format documents are as follows:

- **Structural Misalignment:** The two-dimensional layout of tables contrasts with the sequential nature of LLM input, making spatial relationships harder to interpret after serialization.
- **Table Heterogeneity:** Real-world tables vary significantly in format, content, sparsity, and complexity, making generalizable solutions difficult to develop.
- **Contextual Dependency:** The interpretation of a table often relies on surrounding text, yet most prior work processes tables in isolation, ignoring document-level context.

[1] `https://www.sec.gov/Archives/edgar/data/1410636/000141063624000050/awk-20231231.htm`, visited on 03/28/2025

[2] `https://de.wikipedia.org/wiki/Tour_de_France_1983`, visited on 03/28/2025

[3] Sui et al. (2024)

1.2. Objectives and Contribution

To address these challenges, this thesis investigates the capabilities of LLMs for Table QA, but in contrast to other studies, does not treat tables as isolated entities but rather examines them in the context of full documents, where they coexist with unstructured text. The research is guided by the following key research questions (RQs):

(RQ1.) *How effectively do LLMs perform in Table QA for both standalone tables and tables embedded in full documents?*

(RQ2.) *Which table representation method yields the best performance for answering questions in both isolated and document-embedded contexts?*

- **(a)** *Which serialization format best facilitates LLM-based answer generation for table-related questions?*
- **(b)** *Which format most effectively supports the retrieval of relevant information from full documents containing tables?*

(RQ3.) *What are the key components required to build an effective question-answering pipeline for tables embedded within documents?*

To address these questions, this thesis introduces a comprehensive framework with the following core contributions:

(i) **Evaluation Dataset for Table QA on Full Documents:** A manually annotated dataset of financial documents containing both text and tables, with accompanying questions and ground-truth answers.

(ii) **Retrieval-Augmented Generation (RAG)-Pipeline for Table QA on Full Documents:** A comprehensive pipeline tailored for the specific Table QA task on documents containing both text and tables, incorporating table-aware mechanisms.

(iii) **TabTree Table Serialization:** A novel table serialization method that models tables as directed tree structures based on column headers and row labels, generating serialized representations optimized for LLM processing.

1.3. Structure

This thesis is organized as follows: Chapter 2 introduces the theoretical foundations and key concepts of this thesis, covering the concept of RAG and fundamentals of Ta-

ble QA, along with a review of related research within the field. Chapter 3 details the proposed framework, with a particular focus on the novel TabTree table serialization approach. Formal definitions and task descriptions are provided, followed by in-depth explanations of the mechanisms within the TabTree approach. Chapter 4 presents the details about the utilized datasets, implementation details, experimental setup and results. Finally, Chapter 5 summarizes the thesis's key findings, discusses limitations, and suggests directions for future research.

2. Background and Related Work

This chapter provides an overview of key concepts and terminology essential for addressing the research questions outlined in the introduction. In particular, it discusses RAG, which constitutes the core framework underlying the methods developed in this thesis (Section 2.1). After introducing the fundamentals of RAG, Section 2.2 examines the principles of Question Answering (QA), with a particular focus on Table Question Answering (Table QA) as a specialized subdomain. Finally, Section 2.3 presents a comprehensive review of existing research in related fields.

2.1. Retrieval Augmented Generation

As outlined in Chapter 1, recent advances in artificial intelligence have led to significant progress in the development of LLMs (Minaee et al., 2024). These models demonstrate remarkable capabilities across various NLP tasks, including general and table-based question answering. However, despite their success, LLMs exhibit several notable limitations:

- **Hallucinations**: LLMs occasionally generate incorrect or fabricated content, as they rely solely on learned patterns within training data rather than verifiable knowledge sources (Huang et al., 2025).
- **Knowledge Constraints**: The knowledge within LLMs is static, restricted to the information available during their training phase. Consequently, they struggle with knowledge-intensive tasks requiring access to more recent or specialized data (Lewis et al., 2021).
- **Lack of Referencing**: Since LLMs operate as generative models, they are not able to cite sources, making it challenging to verify the credibility of their outputs.
- **Suboptimal Performance on Long Contexts**: LLMs face difficulties processing lengthy documents containing large amounts of information. Even models with large context windows often fail to retain and synthesize information from distant

parts of the input, leading to issues such as the *'lost in the middle'* phenomenon (Liu et al., 2023).

RAG[4] has emerged as a robust solution to these challenges by integrating external knowledge sources into the text generation process. This approach serves as the foundation for establishing chatbots and real-world LLM applications. Instead of relying on pre-trained knowledge, RAG dynamically retrieves relevant information from an external knowledge source based on the user's query and incorporates it into the input prompt before passing it to the LLM for response generation. These external knowledge sources may include local databases, document stores, or web-based search engines. Figure 2.1 illustrates the outlined RAG workflow.

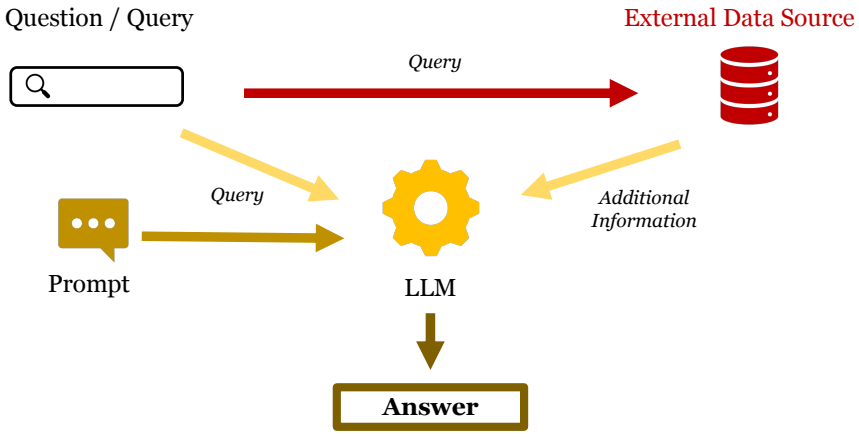


Figure 2.1.: Overview of a RAG pipeline: The system retrieves relevant knowledge from an external data source before integrating it into the input prompt for LLM text generation.

RAG mitigates several of the aforementioned limitations of LLMs: Studies such as Shuster et al. (2021) demonstrate that RAG significantly reduces hallucinations by grounding model responses in factual data. However, Xu et al. (2024) highlight potential conflicts between an LLM's intrinsic knowledge and retrieved external information, which can lead to inconsistencies in responses. By enabling access to up-to-date, query-specific information, RAG facilitates more contextually relevant answers and supports explicit source attribution. Moreover, it improves efficiency in long-context scenarios by narrowing the input to the most relevant segments of retrieved documents.

Numerous extensions and optimizations of the basic RAG framework have been proposed in recent years, as reviewed comprehensively by Y. Gao et al. (2024). This thesis, however, focuses on the original RAG approach – commonly referred to as Vanilla RAG

[4]The explanations regarding RAG and its components are primarily based on the work of Y. Gao et al. (2024), with additional insights supported by cross-references.

or Naive RAG – which comprises three primary stages: **Indexing**, **Retrieval**, and **Text Generation**. These stages are discussed in detail below.

2.1.1. Indexing

Indexing constitutes the initial stage of a RAG pipeline, wherein external knowledge sources are transformed into a structured and retrievable format. This work is limited to the use of a local document store as the external source. The primary goal of indexing is to preprocess, segment, and encode the documents into a form that supports efficient retrieval during inference. The indexing stage involves the key steps of preprocessing, document chunking, and embedding generation. The resulting embeddings are subsequently leveraged during the retrieval phase to compute similarity between a user query and the indexed document chunks. An overview of the indexing process is provided in Figure 2.2.

Figure 2.2.: Overview of the indexing stage of a RAG pipeline, including preprocessing, document chunking, embedding generation and document storage.

Preprocessing Documents are typically available in unstructured formats such as HTML, PDF, Microsoft Word, etc. The preprocessing step aims to extract meaningful textual content from these documents and remove irrelevant metadata or layout information. For example, when processing HTML documents, elements such as CSS styles or navigation links may be removed to improve textual coherence and reduce noise.

Document Chunking Given the context length limitations of LLMs, as well as their performance degradation on long passages, documents might be split into smaller, semantically coherent units known as chunks. Chunking is a non-trivial task, particularly when dealing with heterogeneous documents that include tables, figures, or other non-textual content. In practice, two main approaches are widely established:

- **Fixed-Size Chunking:** Introduced alongside early RAG frameworks, this method segments documents into uniform chunks based on a fixed number of tokens. Extensions of this method include hierarchical splitting techniques that recursively

segment documents using syntactic separators, as well as approaches that exploit structural metadata, such as HTML headers, to determine chunk boundaries.

- **Semantic Chunking:** Initially proposed by Kamradt (2024), semantic chunking partitions a document based on based on shifts in meaning rather than arbitrary token counts. The method identifies semantically distinct segments by computing pairwise distances between embeddings of different parts of the initial documents. This enables chunking at points of high semantic divergence. While more computationally intensive than fixed-size methods, semantic chunking is promising in contexts where structural variation or content complexity is high.

Recent research (Qu et al., 2024) suggests that semantic chunking is not always outperforming fixed-size chunking. However, in the context of documents containing tabular content, our preliminary experiments indicated that fixed-size methods perform poorly. Consequently, we adopt a modified semantic chunking strategy, discussed in further detail in Section 4.2.

Embeddings The core of RAG systems is the ability to measure semantic similarity between queries and document chunks, typically achieved by comparing vector representations, referred to as embeddings, using distance metrics. Traditional retrieval systems relied on sparse embeddings, such as term-frequency vectors in a bag-of-words model, and apply scoring functions like BM25 (Robertson et al., 2009) that emphasize exact term overlap and frequency-based relevance. In contrast, modern approaches employ dense embeddings trained on large corpora to capture deeper semantic relationships.

Early advancements in dense representation learning are marked by models such as Sentence-BERT (Reimers et al., 2019). More recently, state-of-the-art embedding models include the open-source *BGE* family (J. Chen et al., 2024) and *GTE* family (Z. Li et al., 2023), as well as proprietary solutions like OpenAI's *text-embedding-3* models (OpenAI, 2024c) and Cohere's *Embed* series (Cohere, 2023). Comprehensive benchmarking initiatives such as MTEB (Muennighoff et al., 2023) provide comparative evaluations of embedding models across a wide range of tasks and domains.

Indexing Optimisations Several recent extensions enhance the indexing stage of the RAG pipeline. These include hierarchical indexing and multi-representation indexing. Hierarchical indexing, as presented by Sarthi et al. (2024), organizes documents into a tree-like structure of summaries at varying levels of granularity, enabling the retrieval system to handle both high-level and fine-grained queries effectively. Multi-

representation indexing involves summarizing documents and comparing queries to these summaries rather than the full content. In this thesis, we adapt this strategy for handling tabular data, generating summaries during indexing as one of the proposed approaches for retrieval (see Section 4.2).

2.1.2. Retrieval

The retrieval process typically involves transforming the query into an embedding representation using the same strategy applied during indexing, comparing it against the stored embeddings of document chunks, by using a similarity metric, and selecting the top-ranked results for inclusion in the prompt passed to the LLM for answer generation. An overview of this process is depicted in Figure 2.3.

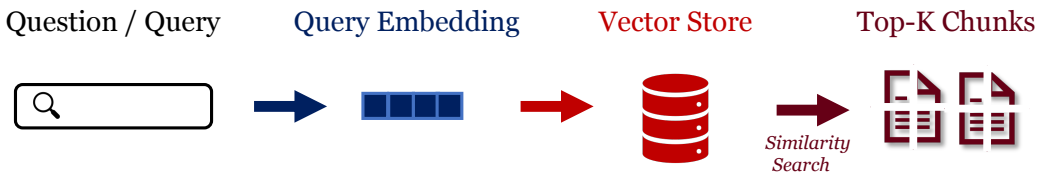


Figure 2.3.: Overview of the retrieval stage of a RAG pipeline including embedding generation of the user query, comparing embeddings against previously stored document chunk embeddings and extracting most relevant matches.

Similarity Search The core task in the retrieval stage is to compute the similarity between the query embedding and the precomputed embeddings of the indexed document chunks. In traditional sparse retrieval systems, such as those based on bag-of-words representations, this similarity is assessed using scoring functions, which rely on term frequency and word overlap, e.g., BM25 (Robertson et al., 2009). In contrast, recent dense retrieval approaches represent queries and documents as continuous vectors in high-dimensional space, enabling the use of distance metrics such as cosine similarity, dot product, or Euclidean distance to capture semantic similarity.

A core challenge for dense retrieval methods is the computational cost of comparing a query against all indexed embeddings. To address this, approximate nearest neighbor (ANN) techniques are widely adopted to enable efficient similarity search at scale (Douze et al., 2025; Boytsov et al., 2013). Key hyperparameters in this process include the number of top-k results returned and similarity thresholds used to filter out irrelevant matches.

Hybrid retrieval approaches aim to combine the complementary strengths of sparse and dense methods. Techniques like reciprocal rank fusion (RRF, Cormack et al., 2009) merge results from both modalities to improve coverage and ranking quality. Additionally, post-retrieval filtering using metadata such as source, date, or domain is often applied to further tailor results to specific task requirements.

Retrieval Optimisations Several advanced retrieval strategies have been proposed to enhance the quality and relevance of retrieved content. Query translation and expansion methods, such as step-back-prompting (Zheng et al., 2024), multi-query generation (Ma et al., 2023) or RAG-Fusion (Rackauckas, 2024), try to rewrite the user input to improve retrieval effectiveness. Query decomposition techniques break down complex questions into simpler subqueries to improve precision. The HyDe approach (L. Gao et al., 2022) takes a different path by using an LLM to generate a hypothetical document from the query, whose embedding is then used for retrieval instead of the query itself. Retrieval routing dynamically directs the query to different data sources (Xiaoqian Li et al., 2023), while iterative retrieval alternates between retrieval and generation, using the model's response to refine the query in subsequent rounds.

2.1.3. Text Generation

The final stage of a RAG pipeline involves the generation of text, wherein the retrieved document chunks are incorporated into the input prompt provided to a LLM[5]. This prompt typically includes task-specific instructions – for instance, in question answering, it contains directives to produce an appropriate response to a given question – alongside the contextual information derived from the retrieved documents.

Prompt Engineering Prompt engineering refers to the strategic construction of input prompts to effectively steer LLM behaviour toward desired outputs. Within the framework of RAG, this process includes the integration of retrieved passages with the user's query, potentially augmented with task-specific instructions, to guide the LLM in producing accurate responses. As emphasized by Minaee et al. (2024), constructing effective prompts necessitates a nuanced understanding of both the nature of the user query and the operational characteristics and constraints of LLMs.

[5]We do not present the architectural structure and inner workings of LLMs in this thesis, as these lie beyond its scope. For a comprehensive overview, refer to Minaee et al. (2024). Instead, we focus on their functional role in the RAG context.

Two prompt engineering strategies are particularly relevant and employed in the methods developed in this thesis (see Section 4.2 for details):

- **Chain-of-Thought (CoT) Prompting:** Chain-of-Thought prompting enhances the reasoning capabilities of LLMs by instructing them to explicitly enumerate intermediate steps leading to the final answer, rather than responding with a direct output by incorporating reasoning instructions as 'let's think step-by-step' to the prompt (Wei et al., 2023). A further evolution of this idea is the *Tree-of-Thoughts* framework (Yao et al., 2023), which generalizes CoT by enabling the model to explore multiple reasoning paths.
- **Few-Shot Prompting:** Introduced by Brown et al. (2020), this method involves providing the model with a small number of input-output examples within the prompt to guide its behaviour. This contextual demonstration allows the model to infer task structure and desired output formatting without explicit re-training.

Fine-Tuning Fine-tuning represents an alternative or complementary strategy to prompt engineering for adapting LLMs to specific tasks. It involves continuing the training of a pre-trained LLM on a domain-specific corpus or task-specific dataset to tailor the model's behaviour and outputs (Minaee et al., 2024). Fine-tuning is particularly effective in achieving greater alignment with specific use-case requirements, especially when consistent output formats are necessary (Du et al., 2022).

In addition, fine-tuning enables LLMs to acquire knowledge that is absent from their original training data, thereby partially addressing the knowledge constraints of LLMs. For this reason, it is sometimes considered an alternative to RAG. However, unlike RAG, fine-tuning does not offer dynamic knowledge updates and typically requires costly re-training whenever the source data changes. Furthermore, it only partially addresses other LLM limitations such as hallucinations, lack of referencing, and suboptimal performance on long contexts.

As emphasized by Y. Gao et al. (2024), combining RAG with fine-tuning is increasingly regarded as a best-practice strategy, uniting the adaptability of retrieval with the specialization achieved through fine-tuning. Nevertheless, due to computational and resource constraints, fine-tuning is not employed in this thesis.

2.2. Table Question Answering

This section provides an introduction to Table Question Answering (Table QA). We start by introducing the general concept of Question Answering (QA), then turn to the specific challenges that arise when the supporting context of a question answering framework is presented in tabular form.

Question Answering QA refers to the task of interpreting natural language questions and producing accurate, concise answers to these questions. It enables "users to directly and efficiently interact with large-scale and heterogeneous knowledge sources" (Nan et al., 2021). Early QA systems date back to the 1960s and 1970s and were often designed for highly constrained domains. For instance, BASEBALL (Green Jr et al., 1961) answered questions about American baseball games by querying a small database of statistics (e.g., 'Who pitched for the Yankees on July 4, 1960?'), while LUNAR (Woods, 1973) enabled geologists to ask questions about lunar rock and lunar soil composition retrieved from Apollo missions (e.g., 'What is the aluminum content of rock sample 15415?').

Over the decades, QA research has diversified across multiple areas, including the nature of questions and answers, the format of contextual knowledge, and the target domain.

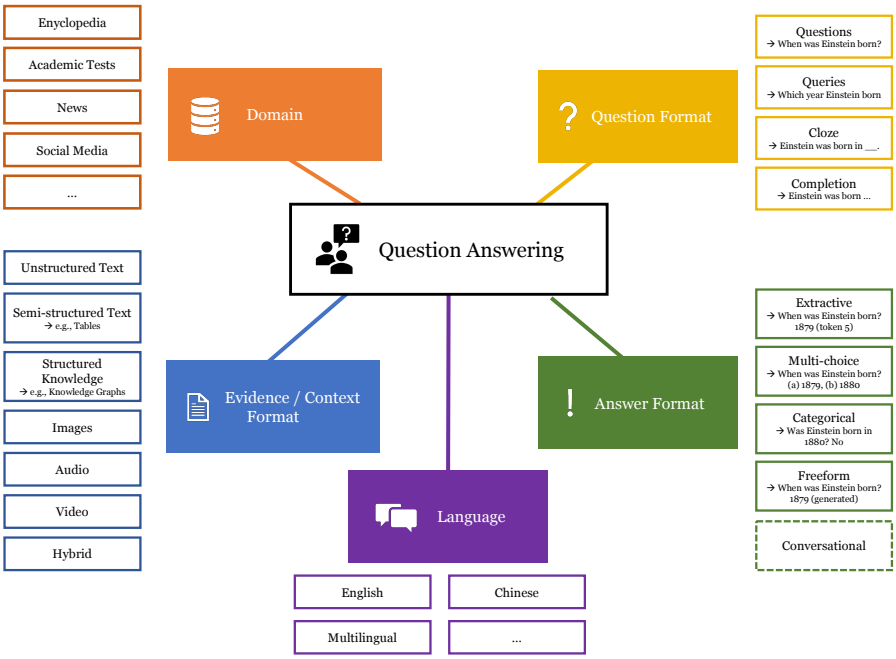


Figure 2.4.: Overview of QA task categories and classification criteria, adapted from Rogers et al. (2023).

Rogers et al. (2023) provide a comprehensive taxonomy of QA tasks, visualized in Figure 2.4.

QA systems offer broad applicability across diverse domains: in healthcare, they can assist medical professionals by surfacing relevant clinical guidelines or patient-specific insights during diagnosis (Hetz et al., 2024). Within the legal domain, they might facilitate the rapid retrieval of case law or statutory references from large corpora of legal documents (Chalkidis et al., 2020). Furthermore, educational applications include intelligent tutoring systems that deliver personalized, context-aware feedback to learners. In customer service, QA systems power conversational agents capable of resolving user queries autonomously. In enterprise environments, they potentially enhance knowledge management by allowing employees to query internal documentation using natural language.

Table Question Answering Table Question Answering (Table QA) is a specialized subdomain of QA in which the supporting context or evidence to answer questions is provided in tabular form. This includes structured database tables as well as semi-structured sources such as web tables, spreadsheets, or tables embedded within documents. While database tables typically adhere to strict schema constraints, non-database tables are more loosely structured and heterogeneous in format (Jin et al., 2022).

Jin et al. (2022) propose a taxonomy of Table QA tasks along two primary axes: domain scope and answer format, visualized in Figure 2.5. In a closed-domain setting, questions are answerable from a fixed and known set of tables, whereas open-domain Table QA may require retrieval across a diverse corpus of tables or documents incorporating both text and tables. Regarding the answer format, free-form QA aims to generate natural

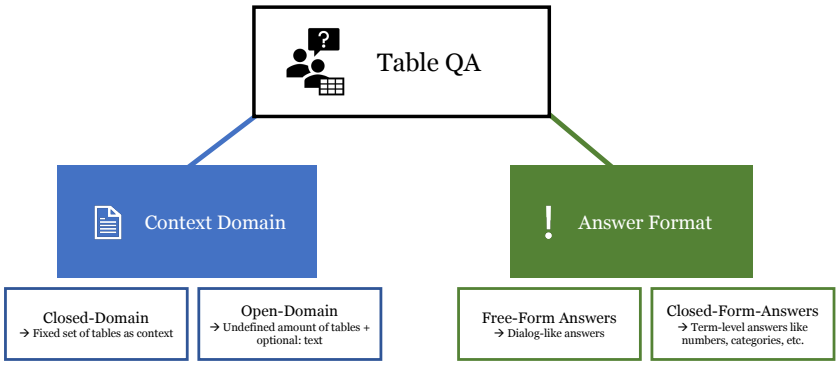


Figure 2.5.: Table QA taxonomy, adapted from Jin et al. (2022).

language responses, while non-free-form QA targets constrained outputs such as spans, numbers, or multiple-choice options.

This thesis primarily focuses on open-domain Table QA, particularly in scenarios involving an undefined number of tables embedded within full documents. Nonetheless, we also investigate closed-domain settings to a certain extent to comprehensively evaluate the developed methods. In terms of output format, the scope is limited to closed-form answers, excluding free-form generation due to the complexity and unreliability of its evaluation (Rogers et al., 2023).

Table 2.1 presents an overview of representative Table QA datasets, categorized by their modality, table-only and tables with additional annotation, domain scope, and answer format.

Table-only			**Non-table-only**		
Dataset	**Closed-domain**	**Question Type**	**Dataset**	**Closed-domain**	**Question Type**
WikiTableQuestions (Pasupat et al., 2015)	Yes	Closed form	FeTaQA (Nan et al., 2021)	Yes	Free form
SQA (Iyyer et al., 2017)	Yes	Closed form	FinQA (Z. Chen et al., 2022)	Yes	Closed form
WikiSQL (Zhong et al., 2017)	Yes	Closed form	TAT-QA (F. Zhu et al., 2021)	Yes	Closed form
Spider (T. Yu et al., 2019)	Yes	Closed form	HybridQA (W. Chen et al., 2021a)	Yes	Closed form
HiTab (Cheng et al., 2022)	Yes	Closed form	TabMCQ (Jauhar et al., 2016)	Yes	Multi-Choice
AIT-QA (Katsis et al., 2021)	Yes	Closed form	GeoTSQA (Xiao Li et al., 2021)	Yes	Multi-Choice
–	–	–	OTT-QA (W. Chen et al., 2021b)	No	Closed form
–	–	–	NQ-tables (Herzig et al., 2021)	No	Closed form
–	–	–	Open-WikiTable (Kweon et al., 2023)	No	Closed form
–	–	–	*SEC-Filing Tables*	*No*	*Closed form*

Table 2.1.: Overview of Table QA datasets, adapted from Jin et al. (2022).

Given the research questions outlined in Chapter 1, this thesis prioritizes non-table-only Table QA datasets that combine tabular and textual information and operate in an open-domain setting. Based on this taxonomy, only three publicly available datasets meet these criteria: OTT-QA (W. Chen et al., 2021b), NQ-Tables (Herzig et al., 2021), and Open-WikiTable (Kweon et al., 2023). OTT-QA includes open-domain questions requiring joint reasoning over Wikipedia tables and textual passages. NQ-Tables extends the Natural Questions dataset (Kwiatkowski et al., 2019) to cover user queries answered via Wikipedia tables. Open-WikiTable reformulates and enriches questions

from WikiSQL (Zhong et al., 2017) and WikiTableQuestions (Pasupat et al., 2015) into an open-domain context, incorporating table retrieval and multistep reasoning.

While these datasets provide a useful benchmark, their reliance on Wikipedia articles limits their applicability. First, Wikipedia content typically covers general knowledge and does not represent specialized domains. Second, Wikipedia articles tend to be relatively short and can often be accommodated within the context window of LLMs. As a result, retrieval-based approaches may not be strictly necessary, reducing the practical relevance of experiments based solely on these datasets.

To address these limitations, we introduce a new dataset – SEC-Filing Tables – described in detail in Section 4.1. It consists of financial filings and presents a realistic challenge for Table QA systems operating in open-domain, non-table-only settings. Additionally, we include the widely used WikiTableQuestions dataset (Pasupat et al., 2015) in our experiments, enabling comparison with established baselines and assessment across a range of question complexities.

2.3. Related Work

Based on the introduction to Table QA, this section presents methods and models developed to address the Table QA task. We proceed chronologically, tracing the development of the field up to the current use of LLMs as core engines for Table QA. Figure 2.6 visually summarises the developments in the area of table QA. Furthermore, current research for the task of serialization of tables is discussed, a key aspect of applying LLMs to Table QA.

2.3.1. Table QA

Semantic Parsing Table QA tasks, especially in closed-domain and non-free-form settings, have been addressed primarily through semantic parsing approaches. Semantic parsing involves converting a natural language question into an executable logical form (e.g., SQL), which can then be executed against a table to retrieve an answer (Jin et al., 2022). Prior to the use of neural models, these approaches largely relied on handcrafted features. Over time, they were replaced by neural architectures such as Seq2SQL (Zhong et al., 2017), which employs an LSTM-based model with reinforcement learning to generate SQL queries from natural language input. The generation of database queries from

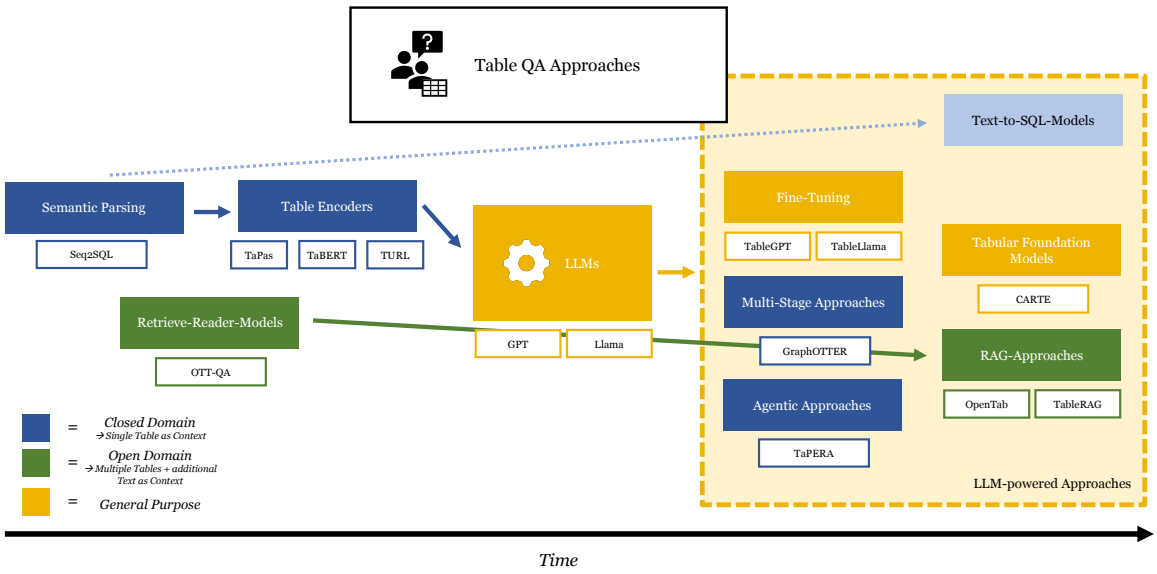


Figure 2.6.: Evolution of Table QA approaches

natural language remains a central problem, as demonstrated by the ongoing research in the Text-to-SQL domain (X. Zhu et al., 2024).

Table Encoders The introduction of the Transformer architecture (Vaswani et al., 2017) and the rise of pre-trained language models such as BERT (Devlin et al., 2019) significantly transformed the Table QA landscape. Numerous approaches have been proposed to transfer the semantic capabilities of pre-trained language models to tabular data. These typically involve serializing tables into linear text and applying masked language modeling techniques, while incorporating structural information from the tables. Notable models in this category include TaPas (Herzig et al., 2020), TaBERT (P. Yin et al., 2020), and TURL (Deng et al., 2020).

Retrieve-Reader-Models Until this point, most advances in Table QA have focused on closed-domain scenarios. In the open-domain setting, the retrieve-reader paradigm – originally developed in open-domain QA over text-only documents – has become increasingly relevant for Table QA. The retrieve-reader architecture can be considered a predecessor of the RAG framework (see Section 2.1) and is therefore based on the similar core idea: A retriever first identifies candidate table segments, followed by a downstream reader model that generates the final answer (Jin et al., 2022). In their work introducing the OTT-QA dataset, W. Chen et al. (2021b) implemented a fusion-based reader that combines multiple relevant textual and tabular segments into a unified block, along with a cross-block reader employing a sparse attention mechanism to generate answers.

LLMs as Table QA Engine Due to their strong generalization and multi-task capabilities (Minaee et al., 2024), LLMs have recently attracted significant attention in the context of Table QA. These models are increasingly replacing purpose-built, table-optimized architectures. A comprehensive survey by Fang et al. (2024) highlights a growing amount of research evaluating the performance of general-purpose LLMs on Table QA tasks. The findings suggest that LLMs not explicitly optimized for tabular data often achieve competitive performance, especially when combined with reasoning techniques such as chain-of-thought prompting. Nevertheless, several limitations remain, particularly regarding table structure understanding and representation of categorical and numerical data, as discussed in the same work.

In a detailed analysis, Sui et al. (2024) examined the ability of GPT-3.5 and GPT-4 to comprehend tabular structures. By evaluating structure-based tasks such as cell lookup and row retrieval in isolation, they concluded that LLMs demonstrate only a basic structural understanding of tables, indicating room for further improvement.

Fine-Tuning Fine-tuning has been shown to enhance the performance of LLMs in Table QA settings: For instance, TableGPT (P. Li et al., 2023), which fine-tunes GPT-2, and TableLlama (T. Zhang et al., 2024), based on LLaMA 2, both report improved effectiveness over general purpose LLMs. However, as noted by Fang et al. (2024), fine-tuning remains complex and lacks out-of-the-box usability. Moreover, optimal fine-tuning strategies for heterogeneous tabular inputs are not yet well understood. As demonstrated in the TabLLM study (Hegselmann et al., 2023), fine-tuned model performance can be highly sensitive to prompt design and table serialization format. For these reasons, the approach taken in this thesis uses LLMs as core engines without additional fine-tuning.

Multi-Stage Approaches Recent research has shifted toward multi-stage, reasoning-centric methods for Table QA, moving beyond approaches that generate answers in a single step. For instance, Sui et al. (2024) propose a self-augmented prompting strategy, which proceeds in multiple phases: first identifying relevant table regions, then generating intermediate reasoning steps, and finally producing the answer using an LLM. Similarly, Q. Li et al. (2024) introduce GraphOTTER, a graph-based framework that converts tables into undirected graphs. The model executes step-wise reasoning using predefined intermediate operations, constructing an explicit reasoning path yielding the final answers to the initial questions.

Agentic Approaches The aforementioned models rely on the LLM for reasoning and answer generation, without incorporating external tools. However, such approaches often struggle with numerical reasoning, due to the inherent limitations of LLMs in interpreting numerical values semantically (Fang et al., 2024). To address this issue, recent research explores hybrid agentic systems, i.e., modular systems where the LLM acts as a planner and delegator rather than a monolithic solver. For example, TaPERA (Zhao et al., 2024) employs a modular framework with three components: a content planner that decomposes the question into sub-questions, an execution-based table reasoner that generates executable Python code for each sub-question, and an answer generator that synthesizes the final response.

Tabular Foundation Model CARTE (Kim et al., 2024) challenges the suitability of LLM architectures for tabular reasoning fundamentally. Instead, it proposes a dedicated tabular foundation model based on graph neural networks. This model represents tables as graphs, with string embeddings for entries and column names, and employs graph attention mechanisms to contextualize values within their tabular structure. The authors argue that such architectures may offer a more principled and scalable foundation for all table-related tasks.

RAG Approaches In open-domain settings, research increasingly expands the retriever-reader paradigm into more advanced RAG frameworks. Kong et al. (2024) propose OpenTab, a system that includes a BM25 retriever, an LLM-based code generator for table parsing, and a generation module that synthesizes answers from executed code results.

S.-A. Chen et al. (2024) introduce TableRAG, a framework designed specifically for handling long tables which might exceed LLM context-window constraints. Their method retrieves relevant tabular content through a three-stage process: Tabular Query Expansion, Schema Retrieval, and Cell Retrieval. In the first stage, an LLM is prompted to infer relevant structural elements of the table, such as column names or data types, based on the input question. These components are then used to guide the identification of pertinent cell values for retrieval. The question and selected schema components are encoded and compared to cell values, resulting in a refined set of candidate cells. This filtered information is subsequently passed to an LLM for final answer generation.

2.3.2. Table Serialization

As outlined in Chapter 1, LLMs are sequential models, which necessitates transforming tabular data into linear, text-based representations. This transformation process is referred to as table serialization (Fang et al., 2024). A variety of serialization techniques have been proposed in the literature, which are summarized in Table 2.2.

Method	**Description**	**Example**
DFLoader	Python code where a dictionary is loaded as a Pandas dataframe	`pd.DataFrame({name:['helen'], age:[47] })`
JSON	Row number as indexes, with each row represented as a dictionary of keys (column names) and values	`{"0": {"name": "helen", "age": "47"}}`
Data Matrix	Dataframe as a list of lists, where the first item is the column header	`[[','name','age'], [0, 'helen', 47]]`

Markdown	Rows are line-separated, columns are separated by '|'	`| | name | age |`
		`|:-|:-----|----:|`
		`|0 |helen | 47 |`

LaTeX	Rows are separated by '\\ \hline', columns are separated by '&'	`name & age \\ \hline` `helen & 47`
CSV	Rows are line-separated, columns are separated by comma, semicolon, tab, etc.	`, name, age` `0, helen, 47`
Attribute-Value Pairs	Concatenation of paired columns and cells {c : v}	`name:helen ; age:47`

HTML	HTML element for tabular data	`<table><thead><tr><th></th>`
		`<th>name</th><th>age</th></tr>`
		`</thead><tbody><tr><th>0</th>`
		`<td>helen</td><td>47</td>`
		`</tr></tbody></table>`

Sentences	Rows are converted into sentences using templates	`name is helen, age is 47`

Table 2.2.: Overview of table serialization methods used in the literature, adapted from Fang et al. (2024).

Numerous studies have demonstrated that model performance in Table QA is highly sensitive to the choice of table serialization. Lu et al. (2024) consolidate findings across several studies and conclude that markup-based formats, such as HTML, Markdown, CSV and JSON, consistently yield superior performance. This may be attributed to the high exposure of LLMs to such formats during pretraining (Lu et al., 2024).

Due to the sequential nature of LLMs and their extensive exposure to natural language during training, several works have explored converting table rows into sentence-like templates. For example, B. Yu et al. (2023) serialize rows into linearized descriptions

such as "Row one's name is Michael, the age is 17...". Hegselmann et al. (2023) and Gong et al. (2020) use similar templates, applying structures like "<attribute> is <value>" or variations with additional context. Jaitly et al. (2023) further refine this approach by introducing sentence structures that emphasize feature importance and feature combinations based on inter-feature correlation. Dinh et al. (2022) suggest a compact key-value serialization format: "<column>=<value>, ...".

However, with few exceptions involving additional fine-tuning (Gong et al., 2020), markup-based formats (e.g., HTML, JSON, CSV) have generally been found to outperform sentence-based formats. Similarly, generating natural language sentences from individual table rows using LLMs, as analyzed by Hegselmann et al. (2023) using GPT-3, has demonstrated limited effectiveness, primarily due to hallucination issues. Nonetheless, given the rapid progress in LLM capabilities, these earlier findings may not fully reflect the performance of current models.

Overall, existing sentence-based methods exhibit limited structural diversity, as they mainly rely on simple attribute-value patterns. Consequently, they often fall short in representing the structural heterogeneity of real-world tables. To overcome these limitations, we propose a novel sentence-based serialization strategy, TabTree, which is presented in detail in Chapter 3.

Table Serialization for Retrieval Only few studies have systematically explored table representation strategies in retrieval-centric, open-domain Table QA settings. A notable exception is the work of Roychowdhury et al. (2024), which partially aligns with the objectives of this thesis. Their study examined the effectiveness of different table representations in contexts combining textual and tabular content. Their study, which used Markdown-serialized tables, found that table-only embeddings generally outperformed those combining text and table. Moreover, row-level embeddings yielded better retrieval performance than full-table embeddings. The inclusion of table headers further improved retrieval accuracy. However, their findings were constrained by the use of embedding models with a maximum input length of 512 tokens, which does not account for the capabilities of more recent models supporting significantly larger context windows.

3. Table QA Framework

This chapter describes the conceptual methodology developed to address the research objective of establishing an effective and reliable Table QA framework for documents containing both text and tabular content. It starts with a detailed overview of the research goals and the proposed framework in Section 3.1. The formal task definition and relevant terminology are introduced in Section 3.2. The chapter concludes with the core contribution: a tree-based table serialization method, TabTree, detailed in Section 3.3.

3.1. Overview and Objectives

As outlined, existing research on Table QA using LLMs predominantly focuses on a plain version of the Table QA task, assuming that the relevant table associated with a given question is preidentified. Moreover, comprehensive research focuses on table serialization techniques for input to LLMs. However, these serialization methods have primarily been explored within the context of this plain Table QA task and leave room for further research.

The core objectives of this thesis aim to address these challenges and are as follows:

- **Develop a comprehensive framework for Table QA on documents containing both text and tables**: Create a framework capable of accurately and reliably answering questions about tables embedded within documents, leveraging LLMs. This framework will address the following subtasks:

    1. Identify and retrieve relevant document chunks associated with a given table-related question. Ideally, the retrieved chunks contain a relevant table that provides the information to answer the question.

    2. Generate the correct answer to a given question based on the retrieved document chunk, ensuring the information from the relevant table is utilized effectively.

- **Explore and extend table serialization methods:** Investigate different table serialization techniques in the context of extended Table QA tasks for large documents. Additionally, introduce a novel serialization method called the **TabTree** approach, which conceptualizes a table as a directed tree where each node represents a table cell. The TabTree approach consists of the following core components:

  1. Create two subtrees based on column headers and row labels, with individual nodes representing table cells.
  2. Clearly distinguish context information of column headers and row labels from value cells.
  3. Merge the two subtrees at the value cell level to form a complete table representation.
  4. Use pre-order depth-first-search traversal to serialize the table, incorporating different variants for displaying context and value cell information.

Figure 3.1 illustrates a representative use case of the proposed objectives: Given a collection of documents containing tables, a question is posed concerning the content of a specific table. The framework's task is to accurately generate the corresponding answer, incorporating any form of table serialization within the framework.

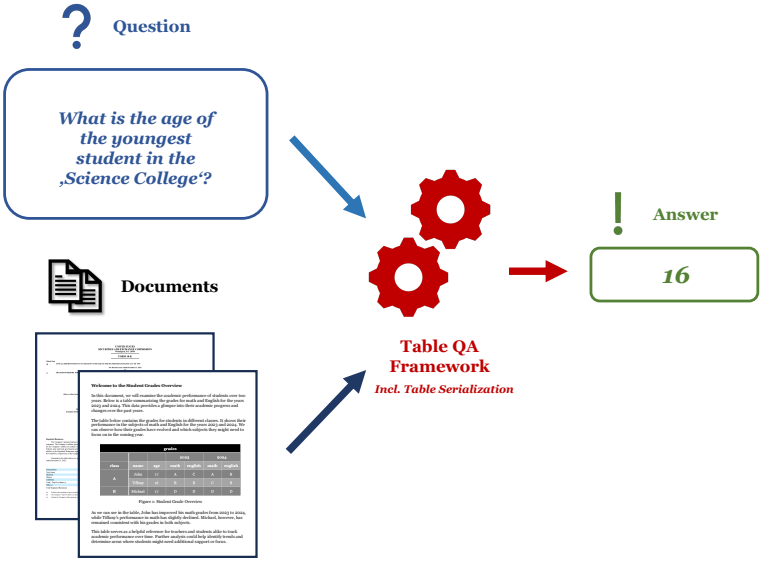


Figure 3.1.: An illustrative example for a potential use case of the Table QA Framework developed within this work. Given a corpus of documents, a question is asked related to a table within the documents. The objective of the Table QA Framework is to accurately generate the correct answer based on the relevant table.

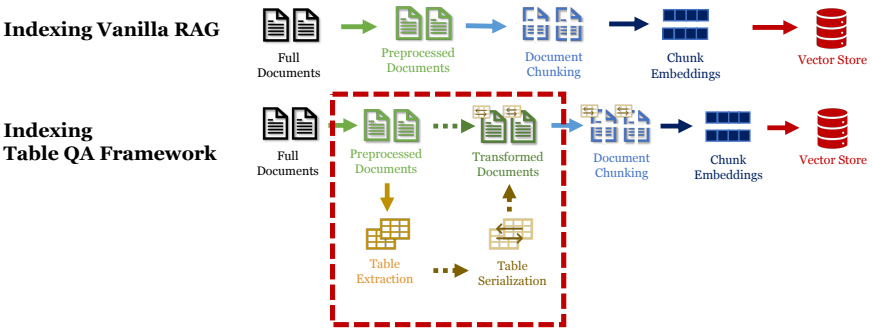


Figure 3.2.: Comparison of a vanilla RAG indexing approach versus the indexing approach of the Table QA framework of this thesis. Highlighted in red are the modifications to the approach of this thesis: the table serialization process.

To achieve the outlined objectives, a RAG pipeline was developed, specifically tailored to the requirements of table question answering for long documents. The RAG pipeline follows the typical structure of a standard RAG pipeline consisting of three main components: indexing, retrieval and text generation.

The main focus of this work is placed within the indexing phase, specifically concerning table serialization. Figure 3.2 highlights the differences between the indexing approach used in a standard RAG pipeline, as outlined in Section 2.1, and the indexing strategy proposed in this work. The additional component, table serialization, is indicated by red dashed lines.

The indexing and retrieval components of the developed Table QA framework are based on the principle of the vanilla RAG approach, as detailed in Section 2.1.2 and Section 2.1.3. The primary differences arise from the use of modified chunks during retrieval,

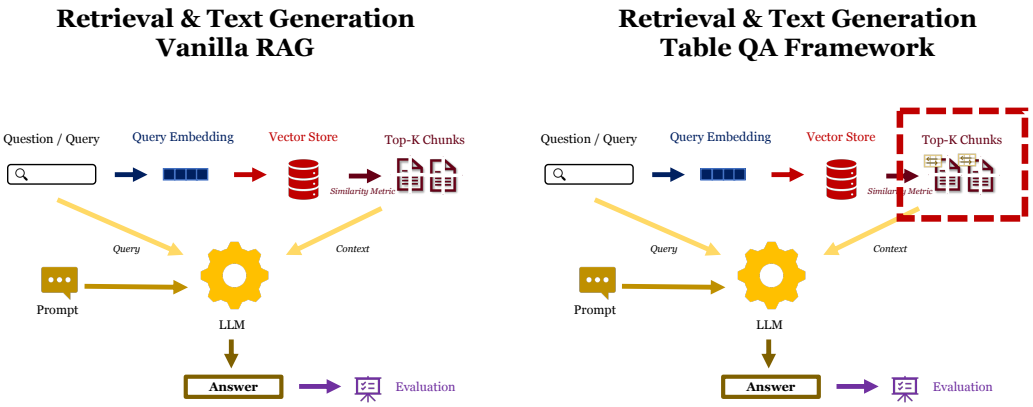


Figure 3.3.: Comparison of a vanilla RAG retrieval and text generation approach versus the retrieval and text generation approach of the Table QA framework of this thesis.

which also affect text generation. Unlike the standard RAG approach, the chunks in the Table QA Framework include serialized tables instead of their initial representation. These serialized tables are subsequently incorporated into the LLM for text generation as part of the RAG pipeline. Figure 3.3 illustrates the distinction between the two approaches, the differences highlighted within the dashed red box.

3.2. Document Model and Task Definition

This section defines the formal foundations necessary for a detailed description of the methods employed in this thesis. It starts by introducing the fundamental terminology, including the document and table models, which specify the structural requirements for the documents and tables used in this work. Building on this foundation, the section concludes with a formal definition of the core tasks of this thesis, comprising multiple subtasks as intermediate steps.

3.2.1. Document Model

A document is understood as a collection of content that forms a coherent unit of information. In this work, the focus is placed on textual documents, that is, documents of written sources. Multimedia documents, including images, audio, video, or other similar formats, are not considered within this work.

Documents are assumed to consist of a sequence of distinct unit elements e_i, where each element occurs as a sequence of characters.

Definition 3.1 (Document)**.** *Formally, a document d is defined as:*

$$d := \langle e_1, \ldots, e_l \rangle$$

where e_i denotes an individual unit element, and l represents the total number of elements within the document.

Unit elements typically consist of individual sentences, paragraphs within documents, or tables. Based on the emphasis on tabular data in this work, tables are treated as separate entities within the provided document framework. Thus, a unit element e_i can be one of the following:

- Text p_i (any form of consecutive sequences of characters)
- Table t_i (refer to Definition 3.3)

Depending on the scope of analysis, it may be beneficial to move beyond viewing the document as a single sequence of characters and represent it rather as a bunch of structured, semantically coherent sections. We therefore define chunks as structural subdivisions of a document that ideally represent its different semantic segments.

Definition 3.2 (Chunk)**.** *Given a document d, a chunk c is defined as a contiguous subsequence of unit elements of a document d. Formally:*

$$c := \langle e_i, \ldots, e_j \rangle \quad \textit{with } 1 \leq i, j \leq n$$

A document d can be partitioned into segments of varying lengths by applying different chunking strategies. The selection of an appropriate chunking method depends on the specific requirements of the application context. Figure 3.4 illustrates an example of a document and its potential division into chunks.

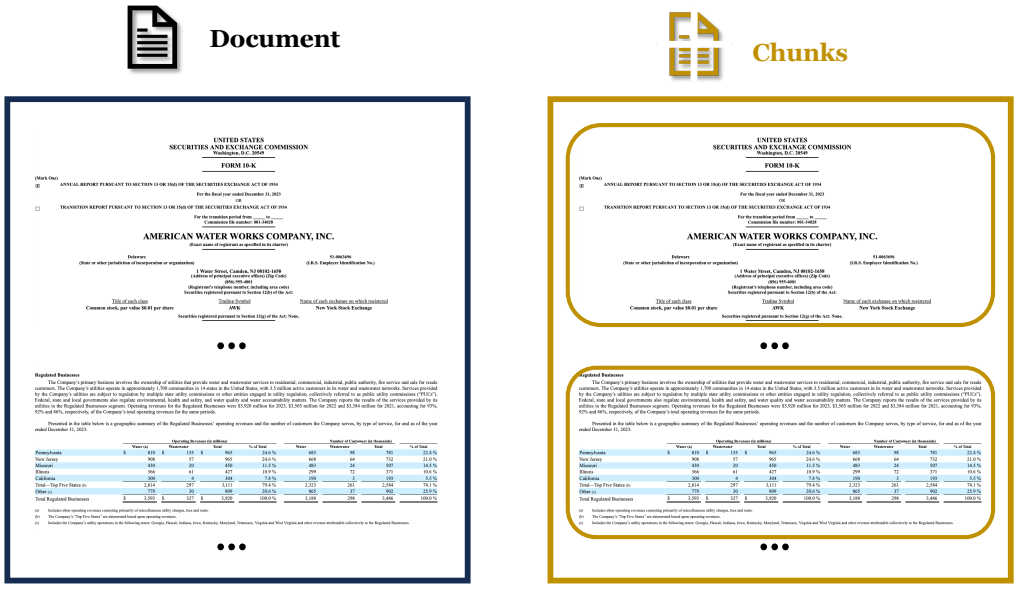


Figure 3.4.: A sample document and its potential segmentation into chunks. Note: Individual chunks may contain both tabular and textual content, as well as a split in between tables may be possible.

In specific cases, such as when a paragraph or table exceeds the specified maximum chunk size, a unit element may not fit entirely within a single chunk and becomes fragmented. In such instances, we assume, that a unit element e_i is divided into a tuple (e_i^1, e_i^2), thus preserving the integrity of the previous definition of a chunk.

3.2.2. Table Model

As already outlined, tables play a pivotal role in this work and are thus treated as unit elements with special significance. In general, a table is a structured representation of data, organized into rows and columns.

Definition 3.3 (Table)**.** *A table t is a structured representation of n rows and m columns, denoted by:*

$$t = \begin{bmatrix} \gamma_{11} & \cdots & \gamma_{1m} \\ \vdots & & \vdots \\ \gamma_{n1} & \cdots & \gamma_{nm} \end{bmatrix},$$

where each entry γ_{ij} represents a cell at the intersection of the i-th row r_i and the j-th column c_j.

Throughout this chapter, explanations will be illustrated using a sample table. Figure 3.5 presents this sample table.

	j = 1	j = 2	j = 3	j = 4	j = 5	j = 6	j = 7
i = 1	grades						
i = 2				2023		2024	
i = 3	class	name	age	math	english	math	english
i = 4	A	John	17	A	C	A	B
i = 5		Tiffany	16	B	B	C	B
i = 6	B	Michael	17	D	D	D	D

Figure 3.5.: Sample table used to support the explanations of the concepts of the table model and the subsequently presented TabTree model. The indices i and j are used to specify rows, columns, and cells in the table.

The primary components of a table are cells formed at the intersection of rows and columns. A single cell typically lacks standalone interpretability and derives meaning from its surrounding context. This context is established through the labeling of rows and columns.

The labeling of columns usually occurs in the initial rows of a table, forming what are referred to as the column headers. Similarly, rows are often assigned labels that identify and describe the entirety of the row. These labels, typically located in the first columns on the left-hand side of the table, are called row labels. Within this thesis we define column headers row labels as subsequences of the row indices r_i and column indices c_j.

Definition 3.4 (Row Labels & Column Headers)**.** *Given a table t with rows $r_1, \ldots, r_n$ and columns $c_1, \ldots, c_m$, the sequence of column headers h_t and the sequence of row labels l_t of a table t are defined as:*

$$h_t := \langle r_1, \ldots, r_{i^*} \rangle$$
$$l_t := \langle c_1, \ldots, c_{j^*} \rangle$$

where i^* represents the last row index of a column header in t, and j^* denotes the last column index of a row label in t.

For the sample table in Figure 3.5 the following applies: The first three rows can be identified as header rows, while the first two columns serves as labels corresponding to the associated rows. Consequently, it follows that $h_t = \langle r_1, r_2, r_3 \rangle$ and $l_t = \langle c_1, c_2 \rangle$.

This work assumes that row headers and column headers always form a contiguous sequence, starting from the beginning of the table and continuing up to the point where only proper cell values remain. As a result, intermediate headers or labels within the table are not supported. For example, Figure 3.6 illustrates a table with a column header appearing in the middle, which falls outside the scope of this study.

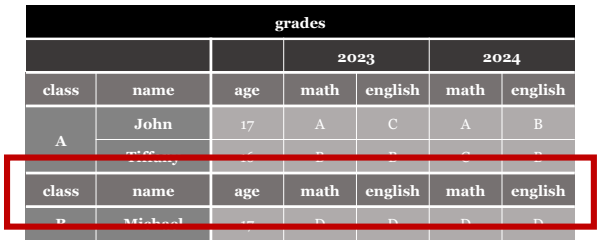


Figure 3.6.: A sample table, with an intermediate header highlighted by the red box, illustrates a table structure not considered in this work. This thesis focuses exclusively on tables with headers and labels placed at the beginning of the table.

From now on, cells that belong to column headers or row labels will be referred to as context cells, while cells representing specific values within this context will be called value cells. Cells that represent a column header are referred to as column header cells, while those that represent a row label are termed row label cells. There are cells, where column headers and row labels overlap, resulting in cells that belong to both a row of a column header and a column of row label. These overlapping cells are referred to as context-intersection cells. Figure 3.7 illustrates examples of these terms using the sample table from Figure 3.5.

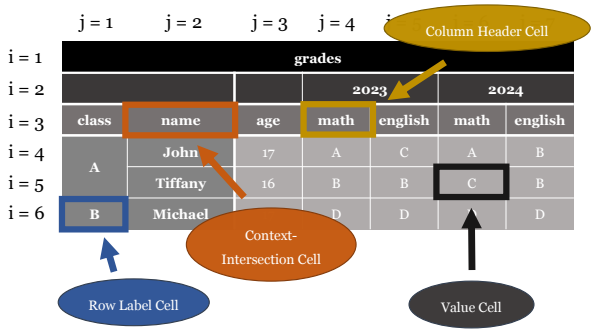


Figure 3.7.: Illustration of the terms used within the table model: Column Header Cell, Row Label Cell, Value Cell and Context-Intersection Cell

In addition to the basic structure of rows, columns, and cells, tables can include rowspans and colspans, where a single cell extends across multiple rows or columns. This feature allows for the representation of hierarchical relationships and facilitates grouping of related data, ideally enabling a more compact and intuitive visualisation of complex data. Similarly, row labels and column labels can themselves be structured hierarchically, using spanning cells to group attributes or records under shared categories.

Definition 3.5 (Cell). *A cell γ_{ij} within a table t is defined as a tuple:*

$$\gamma_{ij} = \left(s^c_{ij}, s^r_{ij}, t_{ij}\right),$$

where t_{ij} denotes the concrete value of the corresponding cell, s^c_{ij} represents the colspan and s^r_{ij} represents the rowspan of this cell.

Definition 3.6 (Row- & Colspan). *For each cell γ_{ij}, its colspan s^c_{ij} and its rowspan s^r_{ij} is defined as follows:*

$$s^c_{ij} := \left[s^c_{prev}(i,j), s^c_{next}(i,j)\right],$$
$$s^r_{ij} := \left[s^r_{prev}(i,j), s^r_{next}(i,j)\right].$$

Here, $s^c_{prev}(i,j)$ and $s^r_{prev}(i,j)$ represent the columns or rows preceding the current cell γ_{ij} that are spanned by it, while $s^c_{next}(i,j)$ and $s^r_{next}(i,j)$ denote the columns or rows following the current cell that are spanned by it.

It is worth noting that if a cell γ_{ij} is part of a col- or rowspan, the value t_{ij} is propagated to all cells covered by the corresponding col- or rowspan, ensuring consistency among the spanned cells.

Figure 3.8 presents the application of Definition 3.3 up to Definition 3.6, specifically highlighting two concrete cell values, γ_{24} and γ_{51}. For γ_{24} yields: The cell belongs to a colspan encompassing the two cells $(2, 4)$ and $(2, 5)$. Consequently, the span vector $s^r_{ij} = [0, 1]$, since the cell $(2, 5)$ is the only additional cell in this colspan. Furthermore, the cell $(2, 4)$ does not extend across multiple rows, resulting in a rowspan vector $s^c_{ij} = [0, 0]$. Finally, the value of this cell is specified as '2023'. The corresponding tuple for γ_{51} can be derived in an analogous manner.

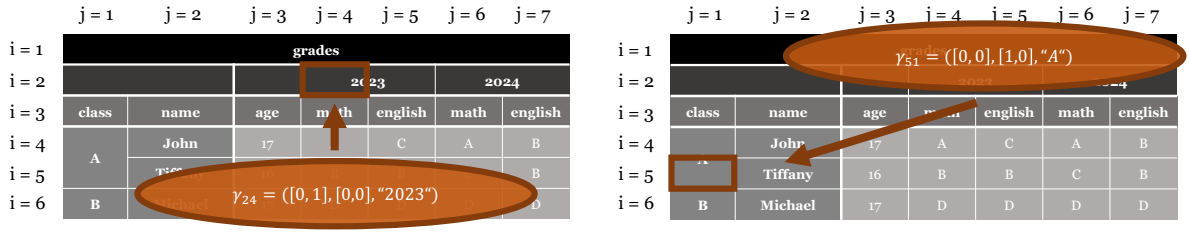


Figure 3.8.: Sample table with application of Definition 3.3 to Definition 3.6

To enable the use of tables in LLMs, it is necessary to convert them into a natural language string representation. The serialization of tables is formally defined below.

Definition 3.7 (Table Serialization)**.** *Given a table t, the mapping* `serialize`*, which transforms the table t into any string representation p_t is defined as follows:*

$$\texttt{serialize}\colon t \mapsto p_t$$

Figure 3.9 presents an example of a possible serialization of the table from Figure 3.5. The serialization in this case is carried out in HTML format.

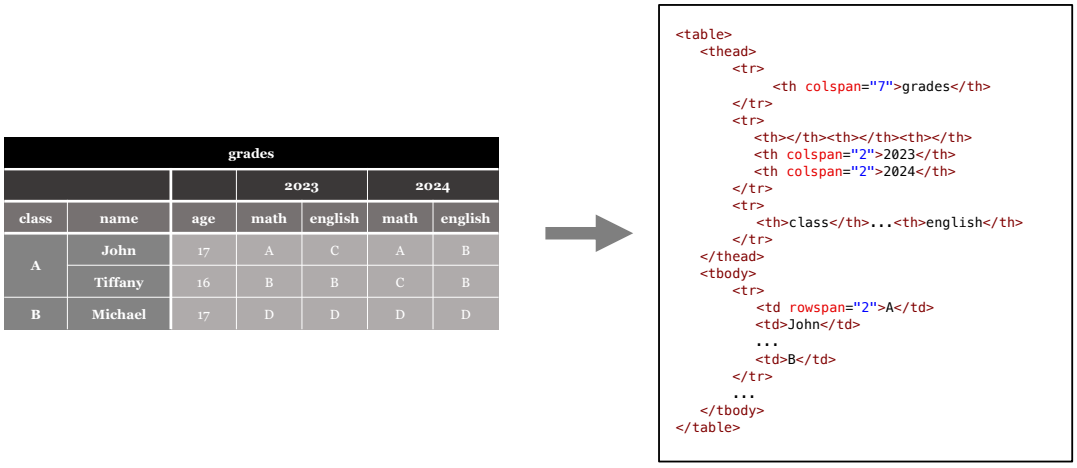


Figure 3.9.: Exemplary serialization of the sample table from Figure 3.5 using HTML formatting

Given that tables within a document or chunk can be reliably identified, the process of serializing tables can be extended to entire chunks.

Definition 3.8 (Chunk Serialization)**.** *Let $c = \langle e_1, \ldots, e_l \rangle$ be a chunk of a document d. Then, the mapping* `serialize-chunk` *is defined as a procedure that serializes each table in c while maintaining the order of all elements:*

$$\texttt{serialize-chunk}: \; c = \langle e_1, \ldots, e_l \rangle \mapsto \langle e'_1, \ldots, e'_l \rangle = p_c$$
$$\text{where} \quad e'_j = \begin{cases} \texttt{serialize}(e_j), & \text{if } e_j \text{ is a table} \\ e_j, & \text{otherwise} \end{cases}$$

and $p_c = \langle e'_1, \ldots, e'_l \rangle$ represents the ordered serialized chunk.

3.2.3. Task Definition

Building on the terminology introduced, it is now possible to formally describe the task that the models and concepts developed in this work are designed to address.

First it is essential, to clearly define what constitutes a question and an answer: A question is characterized as a query expressed as natural language string, intended to elicit an answer, which is likewise represented as a natural language string. Assessing whether a given answer adequately addresses its corresponding question is non-trivial, as correctness can span a spectrum from fully correct to fully incorrect, with intermediate cases of partial correctness.

For instance, the response *approximately 2 million people* to the question *What is the population of Paris?* is neither completely incorrect nor fully accurate. According to the French National Institute of Statistics and Economic Studies INSEE, the estimated population of Paris in 2024 was 2,087,577 inhabitants (INSEE, 2024). Consequently, if exact responses were required, this response would be deemed incorrect. Furthermore, ambiguity arises regarding the geographical scope of 'Paris', as the term may refer to either the administrative city or the broader metropolitan region. In 2021, INSEE estimated the population of the Paris metropolitan area to be 10,890,751 inhabitants (INSEE, 2025).

In this thesis, we constrain the problem space to questions formulated such that their answers can be categorized as either correct or incorrect based upon provided context to this question. This restriction necessitates that answers must take one of the following forms: a numerical value, a word or phrase, or a categorical label. Questions requiring

extended responses in the form of sentences or detailed explanations, are excluded from the scope of this work.

Importantly, the provided context to a question may not always contain the relevant information needed to answer the question, and in such cases, the context cannot substantively contribute to the answer generation process.

Definition 3.9 (Question)**.** *A question q is a natural language string which elicit an answer derived from a given context θ.*

Definition 3.10 (Answer)**.** *Given a question q and a context θ, an answer a_q is a response to q, represented as a discrete string. The truth value of an answer can be either True or False.*

Having established the foundational definitions of questions and answers, we can now proceed to formally define the different tasks addressed in this work, starting with the Table QA task starting with the Table QA task as it is commonly applied in existing research.

Definition 3.11 (Table QA)**.** *Given a table t as context $\theta = t$ and a natural language question q, the task of a Table QA model is to generate an answer a_q to the question q. This can be expressed as the mapping*

$$\texttt{tab-qa}: \ (t, q) \mapsto a_q$$

The `tab-qa` *procedure might involve any form of serialization of the table. Formally, this can be expressed as:*

$$\texttt{tab-qa}: \ (t, q) \mapsto (\texttt{serialize}(t), q) = (p_t, q) \mapsto a_q$$

where p_t denotes the string representation of the serialized table t.

As previously noted, chunks of documents, which may include multiple tables, can also serve as the context for answering questions in this work. Under these conditions, the Table QA task can be extended to a chunk-based task. Figure 3.10 illustrates an example of the Chunk-based Table QA task.

Definition 3.12 (Chunk-based Table QA)**.** *Given a chunk c, the chunk-based Table QA task is defined as:*

$$\texttt{tab-qa-chunk}: \ (c, q) \mapsto \big(\texttt{serialize-chunk}(c), q\big) \mapsto a_q$$

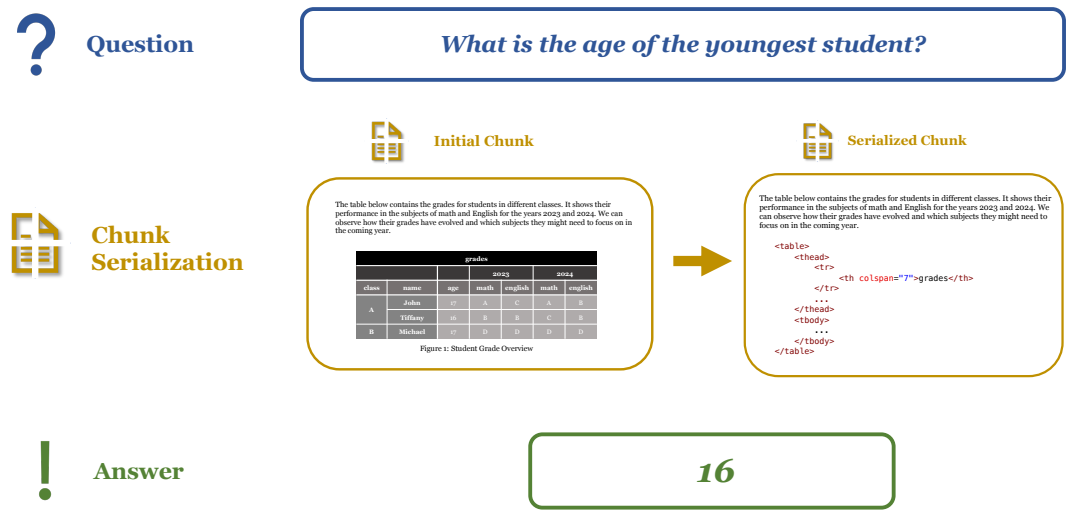


Figure 3.10.: Sample illustration of the Chunk-based Table QA task

The chunk displayed contains the sample table from Figure 3.5 along with a potential serialization of this table in HTML format. Given the serialization and the question q: 'What is the age of the youngest student?', the task is to identify the correct answer a_q, which is '16'.

As this work addresses Table QA not only focusing on the process of answer generation but also the retrieval of the relevant document or chunk within a document, we need to formally define the subtask of retrieving relevant chunks corresponding to given questions.

Definition 3.13 (Chunk Retrieval)**.** *Given a document d composed of multiple chunks, and a question q, the task of chunk retrieval is to identify and retrieve a chunk $c \in d$ that is most relevant to answering the question q. Ideally, the retrieved chunk c contains the information necessary to generate a correct answer to q. Formally, this task can be expressed as:*

$$\texttt{retrieve-chunk}: \ (d, q) \mapsto c \quad \textit{with} \quad c \in d$$

Figure 3.11 outlines the process of chunk retrieval as introduced in Definition 3.13. Specifically, given a document divided into distinct chunks, the main objective is to identify the chunk that contains the table which provides the information to answer the given sample question.

It should be noted that depending on the nature of the question q and the tables present within a document, multiple chunks may potentially provide the correct answer a_q. For

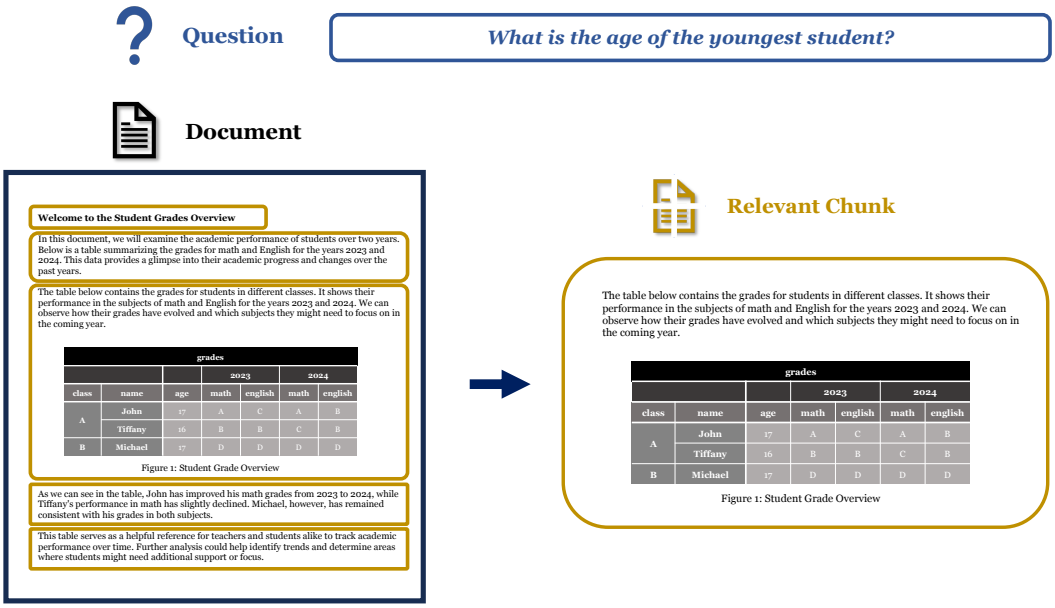


Figure 3.11.: Sample illustration of the Chunk Retrieval task as part of the Table QA process

example, key figures within a financial document might be represented within multiple tables. As a result, the mapping `retrieve-chunk` is not guaranteed to be unique.

When extending the scope from a single document d to a corpus D, the task of chunk retrieval can be generalized to involve retrieving a chunk within the entire corpus.

Definition 3.14 (Chunk-Retrieval on Corpus)**.** *Let D be a corpus of documents and q a question. Then the retrieval of a chunk is defined as follows::*

$$\texttt{retrieve-chunk-corpus} : (\bigcup_{k=1}^{n} d_k, q) \mapsto c \quad \textit{with} \quad t \in c$$

Combining the various subtasks yields the definition of Table QA on Full Documents, which involves first retrieving an ideally highly relevant chunk (i.e., `retrieve-chunk`) and then answering the question based on the retrieved chunk (i.e., performing `tab-qa-chunk`).

Definition 3.15 (Table QA on Full Documents)**.** *Given a document d and a question q, the `tab-qa-full` task for answering questions on full documents is defined as follows:*

$$\texttt{tab-qa-full} := \texttt{tab-qa-chunk} \circ \texttt{retrieve-chunk} : (d, q) \mapsto a_q$$

When extending the task to an entire corpus D, the definition can be generalized as follows:

Definition 3.16 (Table QA on Corpus)**.** *Given a document corpus D and a question q, the task of answering the question on a corpus is defined as:*

$$\begin{aligned} \texttt{tab-qa-full-corpus} &:= \texttt{tab-qa-chunk} \circ \texttt{retrieve-chunk-corpus} \\ &= (D, q) \mapsto a_q \end{aligned}$$

Figure 3.12 illustrates the integration of the previously described individual subtasks, leading to the `tab-qa-full-corpus` task. Given a sample question q, the tasks `retrieve-chunk-corpus` and `tab-qa-chunk` are executed sequentially to derive the answer a_q. As outlined in Definition 3.12, the `tab-qa-chunk` task incorporates the serialization of the table within the retrieved chunk as an intermediate step.

In scenarios where the prerequisite is a single document rather than an entire corpus, the chunk retrieval would occur at document level instead of corpus level. Consequently, the `retrieve-chunk-corpus` task would be replaced by the `retrieve-chunk` task.

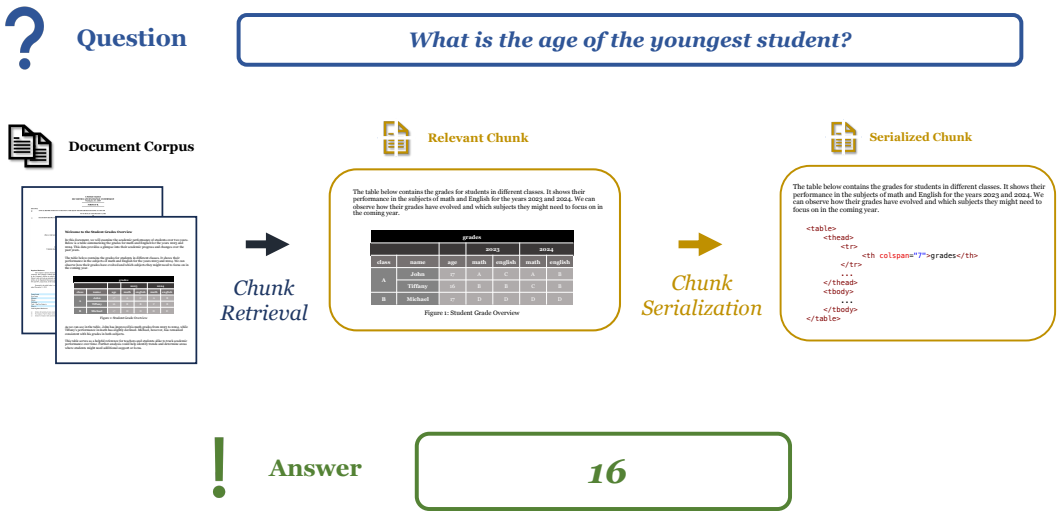


Figure 3.12.: Sample illustration of the Table QA Task on a corpus of documents combining the several subtasks of chunk retrieval, table serialization and chunk-based answer generation.

3.3. TabTree Table Serialization

A key focus of this work is the exploration of table serialization within documents into string representations, as described in Definition 3.8. This section introduces the

TabTree method, which conceptualizes a table as a tree structure, enabling multiple serialization approaches based on this framework.

This section is organized as follows: First, the methodology for constructing the tree structure in the TabTree model is explained. We then present the serialization strategies derived from this representation. Next, we introduce an approach for detecting column headers and row labels. Finally, we conclude with a brief complexity analysis of the proposed method.

3.3.1. TabTree Model

Motivation In many cases, tables consist of a structure where individual cell values cannot be represented solely by the intersection of a single row and a single column: Instead, they may rely on multiple rows or columns for proper interpretation. For instance, the sample table in Figure 3.5 illustrates how nested structures, often implemented through rowspans and colspans, complicate the assignment of individual cell values.

So, how might humans navigate such a structure to clearly assign individual cell contents? For example, how could the assignment for cell γ_{57} in the table from Figure 3.5 be determined? In other words, how do humans derive which grade 'Tiffany' received in her English classes in 2024? To explore this, we refer to Figure 3.13 and outline a possible approach for resolving the assignment.

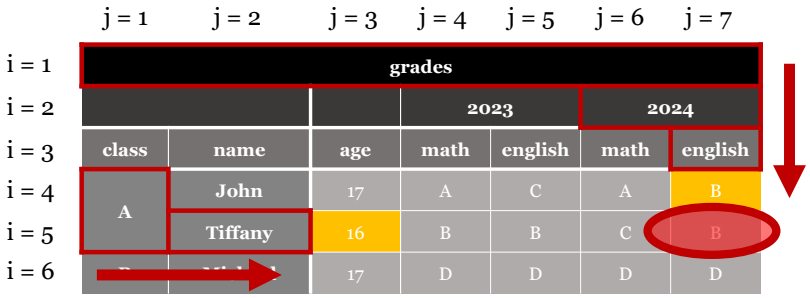


Figure 3.13.: Exemplary derivation of the TabTree model

Humans would likely approach the table's nested structure systematically to assign values, primarily by identifying the context within the table that guides us to the target. This context emerges from the interaction between column headers and row labels.

First, we would establish the table's overall context before locating the content corresponding to the value 'Tiffany'. This two-step process involves starting at the top-level

column header in row r_1, identified as 'grades'. From this point, we likely would navigate down the hierarchy level by level until reaching a cell that is no longer a column header but contains a specific value derived from the combination of column headers and row labels. This process is depicted by the red arrow in the upper-right part of the figure.

In row r_2, the entry for $j = 7$ is associated with the value '2024', representing context information for the table. Similarly, in row r_3, the value 'english' might be identified. Proceeding further down, the first value cell that is neither a column header nor a row label is encountered. It corresponds to the row label class 'A' and the student name 'John', marked in yellow. At this point, the hierarchical traversal for rows ends.

Next, we search for the relevant row to answer the initial question, following a similar logic. Rows r_4 and r_5 reveal the value of class 'A' and the value 'Tiffany' in row r_5 providing the relevant context information regarding the provided question. The neighbouring column c_3 then represents the first non-context cell containing Tiffany's age, i.e., '16'.

By combining the identified column headers and row labels, we determine their intersection, locating the cell γ_{57}. Thus, γ_{57} is defined by the combination of column headers 'grades' $\rightarrow$ '2024' $\rightarrow$ 'english' and the row labels 'A' $\rightarrow$ 'Tiffany'.

This approach mirrors the structure of a graph, which forms the foundation of the TabTree method. In the TabTree model, a table is conceptualized as a directed tree data structure, i.e., a connected, acyclic and directed graph. The TabTree is constructed by generating two independent directed trees: one based on the column headers and another on the row labels. Both trees are then merged at the level of value cells, such as the cell γ_{57} from the introductory example. To facilitate the distinction between the components of the two subtrees, colouring of the nodes is implemented.

Preprocessing Table Modifications Before defining the Column Header Tree, the Row Label Tree, and the final TabTree, we establish some general assumptions that apply to all subsequent definitions. To ensure coherence in the following definitions, slight modifications to the general table model (see Definition 3.3) are required.

Specifically, colspans and rowspans that extend across both row labels and column headers must be divided into separate parts, with each part covering only the row label cells or column header cells individually.

Furthermore, value cells of the table with colspans or rowspans that span only across other value cells are also modified: these cells are stripped of their column and row

spans, with their content duplicated across the respective spans. This adjustment can be justified by the assumption that hierarchical structures occur only at the context-cell level and not at the value-cell level – an approach consistent with the general table model, which distinguishes between context and value content. Algorithm 3.1 provides a formal description of the two modifications described.

Algorithm 3.1 TabTree Table Modifications

Input: Table $t = \{\gamma_{11}, \ldots, \gamma_{nm}\}$ with rows $r_1, \ldots, r_n$, columns $c_1, \ldots, c_m$, column header sequence $h_t = \langle r_1, \ldots, r_{i^*} \rangle$, and row label sequence $l_t = \langle c_1, \ldots, c_{j^*} \rangle$ as defined in Section 3.2.2

Output: Modified table $\tilde{t} = \{\tilde{\gamma}_{11}, \ldots, \tilde{\gamma}_{nm}\}$

for $i = 1$ to n **do**
  for $j = 1$ to m **do**
    ▷ ***Adjust spans for column header cells***
    if $i \leq i^*$ and $j > j^*$ **then**
      $\tilde{s}^c_{ij} = (\tilde{s}^c_{\text{prev}}(i,j), \tilde{s}^c_{\text{next}}(i,j))$ with $\tilde{s}^c_{\text{prev}}(i,j) \leftarrow \min\left(j - j^* - 1, s^c_{\text{prev}}(i,j)\right)$, $\tilde{s}^c_{\text{next}}(i,j) \leftarrow s^c_{\text{next}}$
      $\tilde{s}^r_{ij} = s^r_{ij}$
    ▷ ***Adjust spans for row label cells***
    else if $i > i^*$ and $j \leq j^*$ **then**
      $\tilde{s}^c_{ij} = s^c_{ij}$
      $\tilde{s}^r_{ij} = (\tilde{s}^r_{\text{prev}}(i,j), \tilde{s}^r_{\text{next}}(i,j))$ with $\tilde{s}^r_{\text{prev}}(i,j) \leftarrow \min\left(i - i^* - 1, s^r_{\text{prev}}(i,j)\right)$, $\tilde{s}^r_{\text{next}}(i,j) \leftarrow s^r_{\text{next}}(i,j)$
    ▷ ***Adjust spans for context-intersection cells***
    else if $i \leq i^*$ and $j \leq j^*$ **then**
      $\tilde{s}^c_{ij} = (\tilde{s}^c_{\text{prev}}(i,j), \tilde{s}^c_{\text{next}}(i,j))$ with $\tilde{s}^c_{\text{prev}}(i,j) \leftarrow s^c_{\text{prev}}(i,j)$, $\tilde{s}^c_{\text{next}}(i,j) \leftarrow \min\left(j^* - j, s^c_{\text{next}}(i,j)\right)$
      $\tilde{s}^r_{ij} = (\tilde{s}^r_{\text{prev}}(i,j), \tilde{s}^r_{\text{next}}(i,j))$ with $\tilde{s}^r_{\text{prev}}(i,j) \leftarrow s^r_{\text{prev}}(i,j)$, $\tilde{s}^r_{\text{next}}(i,j) \leftarrow \min\left(i^* - i, s^r_{\text{next}}(i,j)\right)$
    ▷ ***Remove spans from value cells***
    else if $i > i^*$ and $j > j^*$ **then**
      $\tilde{s}^c_{ij} \leftarrow (0,0)$
      $\tilde{s}^r_{ij} \leftarrow (0,0)$
    end if
    $\tilde{\gamma}_{ij} \leftarrow (\tilde{s}^c_{ij}, \tilde{s}^r_{ij}, t_{ij})$ ▷ Combine the adjusted spans into $\tilde{\gamma}_{ij}$
  end for
end for
return $\tilde{t} = \{\tilde{\gamma}_{11}, \ldots, \tilde{\gamma}_{nm}\}$

Some of the required modifications are illustrated in Figure 3.14, i.e., the split of cells spanning across row label cells and column header cells. The provided sample table

doesn't contain any row- or colspans within its value cells, therefore no changes occur for the value cells of the sample table.

For the split of cells across row label cells and column header cells, consider row r_1, which spans across all columns of the table. Since r_1 partly shares cells the row labels l_t of the table, i.e., the cells γ_{11} and γ_{12}, the colspans of r_1's cells need adjustment. Specifically, all cells γ_{1j} with a column index $j \leq j^* = 2$ (the maximum index of row labels columns) are updated with a new colspan that spans only up to j^*. In this example, this adjustment affects the cell $\gamma_{11} = ([0, 6], [0, 0], \text{'grades'})$, which is modified to $\tilde{\gamma}_{11} = ([0, 1], [0, 0], \text{'grades'})$.

For columns with $j > j^* = 1$, the colspan is also revised. These cells receive a new colspan range with respect to previous columns. For instance, the cell $\gamma_{14} = ([3, 3], [0, 0], \text{'grades'})$ is adjusted to span only up to column c_3, resulting in $\tilde{\gamma}_{14} = ([1, 3], [0, 0], \text{'grades'})$. Similarly, for all column indices $j > 1$ in the first row of the example table, the value of $s^c_{\text{prev}}(1, j)$ is reduced by 1.

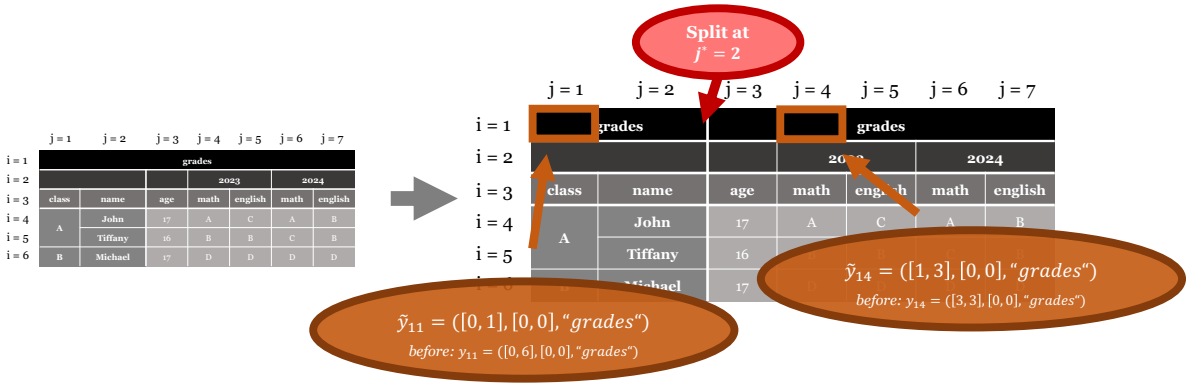


Figure 3.14.: Exemplary illustration of the effects of the introduced modifications from Prerequisites 3.1.

This same adjustment applies to any potential rowspans spanning both context cells and value cells, though this scenario does not occur in the example table. It is also worth noting that no changes are made to cells that do not span across both value cells and context cells.

Definition of Column Header Tree & Row Label Tree We now proceed with the definition of the Column Header Tree and a Row Label Tree. Figure 3.15 illustrates exemplary how these two trees are derived from the modified sample table presented in Figure 3.14.

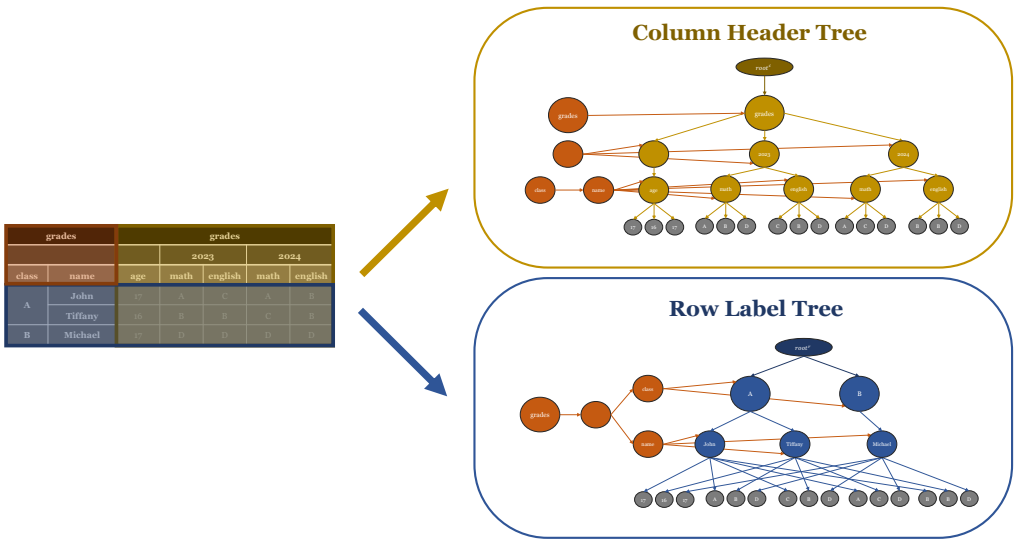


Figure 3.15.: Derivation of the Column Header Trees and the Row Label Trees based on the modified sample table from Figure 3.14

Definition 3.17 (Column Header Tree & Row Label Tree)**.** *Let t be a table with rows $r_1, \ldots, r_n$, columns $c_1, \ldots, c_m$, column headers $h_t = \langle r_1, \ldots, r_{i^*} \rangle$ and row labels $l_t = \langle c_1, \ldots, c_{j^*} \rangle$. We assume that Algorithm 3.1 is applied, i.e., $\tilde{t} = \{\tilde{\gamma}_{11}, \ldots, \tilde{\gamma}_{nm}\}$ with $\tilde{\gamma}_{ij} = (\tilde{s}^c_{ij}, \tilde{s}^r_{ij}, t_{ij})$.*

Then, the Column Header Tree and Row Label Tree are defined as tuples $T_c := (V_c, E_c, \phi_c)$ and $T_r := (V_r, E_r, \phi_r)$, respectively. Each tree consists of:

- *V: the set of nodes,*
- *E: the set of edges,*
- *$\phi : V \to \{$`yellow, blue, gray, orange`$\}$: the node colouring function*

Here, (V, E, ϕ) refers to (V_c, E_c, ϕ_c) for the Column Header Tree and (V_r, E_r, ϕ_r) for the Row Label Tree.

The node colouring function ϕ_r and ϕ_c enables the differentiation and filtering of nodes within the TabTree model. The meanings of node colours are as follows:

- `yellow`: Column Header Cell
- `blue`: Row Label Cell
- `gray`: Value Cell
- `orange`: Context-Intersection Cell

The definitions of the Column Header Tree and Row Label Tree, as introduced in Definition 3.17, do not specify their actual structure or the process of their formation. Therefore, the following Algorithm 3.2 and Algorithm 3.3 outline the construction of these trees based on the established table modifications described in Algorithm 3.1 and the interpretation of the colour assignments provided above[6].

Both the Column Header Tree and the Row Label Tree are constructed using a similar approach, with the key distinction being their orientation: the Column Header Tree is built from top to bottom, traversing the rows of the original table, while the Row Label Tree is constructed from left to right, across the table's columns. It is important to note that the construction process for both trees is deterministic.

Algorithm 3.2 Construction of the Column Header Tree

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:
   a) 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`.
   b) Connect each cell v_{new} to its corresponding parent node from the preceding column header, or v_{root^c} for the first row.
   c) 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`.
   d) 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.
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:
   a) Create leaf nodes $v_{\text{new}} = \tilde{\gamma}_{ij}$ and colour them `gray`.
   b) Connect v_{new} to its parent column header node.

[6]Some of the algorithms discussed in this chapter are presented using a mixed approach in between rigorous pseudocode and textual descriptions. A rigourous pseudocode representation, while comprehensive, would significantly extend the description and potentially reduce the clarity of the explanations.

Algorithm 3.3 Construction of the Row Label Tree

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:
   a) 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`.
   b) Connect each cell v_{new} to its corresponding parent node from the preceding column header, or v_{root^r} for the first column.
   c) 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`.
   d) 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.
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:
   a) Create leaf nodes $v_{\text{new}} = \tilde{\gamma}_{ij}$ and colour them `gray`.
   b) Connect v_{new} to its parent row label node.

Definition of TabTree Finally, we can define the TabTree model by merging the two trees V_c and V_r at the leaf-node level, yielding a deterministic representation of a table. Figure 3.16 presents the final TabTree model for the sample table presented in Figure 3.5.

Since the leaf-level nodes of both trees are similar, it is sufficient to construct the union of the node sets V_c and V_r, as well as the edge sets E_c and E_r. Note that the TabTree contains two 'roots', corresponding to the respective starting points of the two constructed subtrees. Additionally, cells located at the intersection of column headers and row labels appear twice, as part of each subtree.

The colouring of nodes in the TabTree model is determined by combining the colouring function ϕ_c and ϕ_r. The Column Header Tree and the Row Label Tree only overlap at the leaf-node level, where nodes are assigned the same colour, `gray`. As a result, the union of the respective colouring functions does not modify the initial colouring of the two subtrees.

Definition 3.18 (TabTree Model)**.** *Let* $T_c = (V_c, E_c, \phi_c)$ *be a Column Label Tree and* (V_r, E_r, ϕ_r) *a Row Label Tree. The TabTree* $T_{TabTree} := (V_{TabTree}, E_{TabTree}, \phi_{TabTree})$ *is defined as follows:*

$$V_{TabTree} := V_c \cup V_r$$
$$E_{TabTree} := E_c \cup E_r$$
$$\phi_{TabTree} : V_{TabTree} \to \{\texttt{yellow}, \texttt{blue}, \texttt{gray}, \texttt{orange}\},$$
$$v \mapsto \phi_{TabTree}(v) = \begin{cases} \phi_c(v), & \text{if } v \in V_c, \\ \phi_r(v), & \text{otherwise}, \end{cases}$$

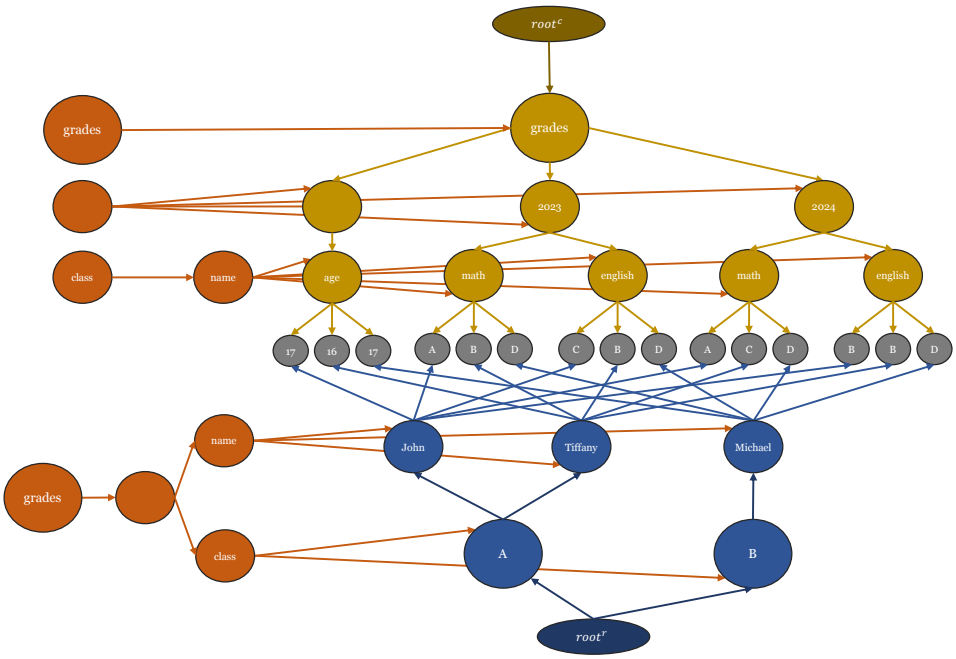


Figure 3.16.: TabTree model for the sample table from Figure 3.5

3.3.2. TabTree String Generation

The TabTree model, as outlined above, provides the foundation for the development of various strategies to generate serialized representations of tables, i.e., capturing the serialization task specified in Definition 3.7. In this section, we propose a general conceptual approach for traversing a TabTree model and, based upon this approach, multiple concrete implementations.

It is crucial to emphasize that both the general concept and the specific implementations discussed represent only a subset of the possible approaches to serializing the TabTree

model, and are by no means exhaustive. The serialization methods introduced will be systematically evaluated in Chapter 4.

Core Idea As an initial step, this section introduces the fundamental conceptual approach – the core idea – underlying the generation of string representations for tables within the TabTree model. Figure 3.17 illustrates the various components of this core concept.

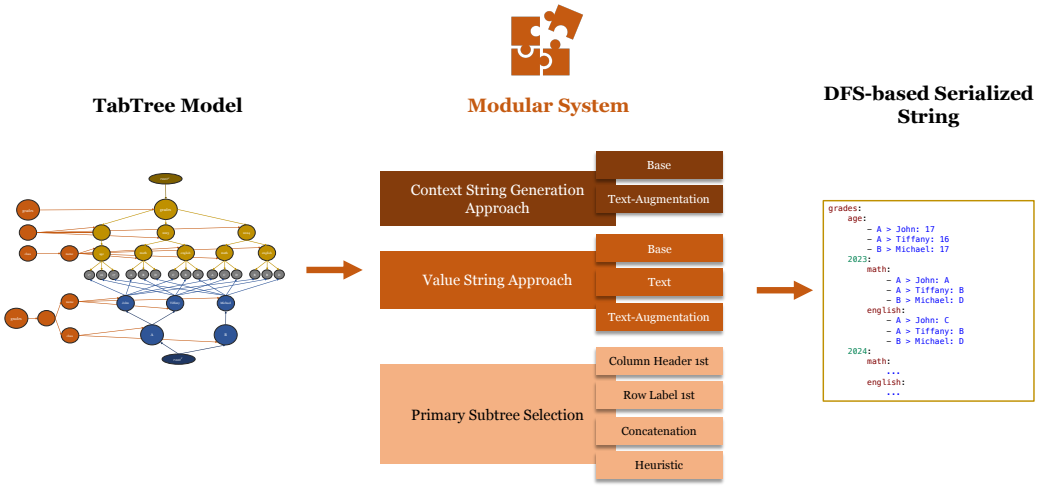


Figure 3.17.: Modular system approach for string generation using the TabTree model

The primary objective of the core idea of string representation within the TabTree model is to mitigate the limitations of the two-dimensional structure of tables when working with LLMs by effectively leveraging the tree structure in the TabTree model. To achieve this, we utilize the hierarchical structure of the TabTree model and traverse through the two subtrees of the TabTree using a depth-first search (DFS) approach, generating various strings at both the context cell and value cell levels, thus covering the entire table. For both, context cells and value cells, distinct strategies for string representation strategies are proposed.

Furthermore, the selection of the traversal's starting point, i.e., determining which of the two subtrees within the TabTree model serves as the root, constitutes a configurable parameter. Ultimately, string generation in the TabTree model follows a modular system, comprising three key components: **Context String Generation, Value String Generation, and Primary Tree Selection**, as depicted in Figure 3.17. These components are further elaborated in subsequent sections.

The discussion now proceeds with a detailed algorithmic explanation of the DFS-based traversal approach, incorporating the various elements of the modular system.

Prerequisites for DFS-based Serialization Before we get into the details of the core idea and its concrete implementations, it is necessary to define two concepts relevant to the following discussion: the inverted TabTree and the filtered TabTree. An inverted tree generally reverses the direction of all edges in a directed tree.

Definition 3.19 (Reversed TabTree)**.** *Given a TabTree $T = (V, E, \phi)$, the reversed TabTree $\overleftarrow{T} := (\overleftarrow{V}, \overleftarrow{E}, \overleftarrow{\phi})$ is defined as follows:*

$$\begin{aligned}\overleftarrow{V} &:= V \\ \overleftarrow{E} &:= \{(v_{end}, v_{start}) \mid (v_{start}, v_{end}) \in E\} \\ \overleftarrow{\phi} &:= \phi\end{aligned}$$

A filtered TabTree based on a given colour set S_{colour} only considers nodes of the initial TabTree model matching one of the colours of S_{colour}, excluding also edges which connect at least one node not having a colour within S_{colour}.

Definition 3.20 (Filtered TabTree)**.** *Let $S_{colour} \subset \{\texttt{yellow}, \texttt{blue}, \texttt{orange}, \texttt{gray}\}$ denote a set of colours, and let $T = (V, E, \phi)$ represent the initial tree. A filtered tree $T_{S_{colour}}$ is defined as:*

$$T_{S_{colour}} = \left(V_{S_{colour}}, E_{S_{colour}}, \phi_{S_{colour}}\right),$$

where:

$$\begin{aligned}V_{S_{colour}} &:= \{v \in V \mid \phi(v) \in S_{colour}\}, \\ E_{S_{colour}} &:= \{(u, v) \in E \mid u, v \in V_{S_{colour}}\}, \\ \phi_{S_{colour}} &:= \phi\end{aligned}$$

Note that both concepts, filtered and reverse TabTrees can be combined, i.e., providing $\overleftarrow{T}_{S_{\text{colour}}} = (\overleftarrow{V}_{S_{\text{colour}}}, \overleftarrow{E}_{S_{\text{colour}}}, \overleftarrow{\phi}_{S_{\text{colour}}})$.

DFS-based Serialization We can now proceed with the description of the core idea: Let us assume, that a table t is represented as a TabTree, i.e., $T_{\text{TabTree}} := (V_{\text{TabTree}}, E_{\text{TabTree}}, \phi_{\text{TabTree}})$, or equivalently as a combination of a Column Header Tree $T_c = (V_c, E_c, \phi_c)$ and a Row Label Tree $T_r = (V_r, E_r, \phi_r)$. From this we generate a string representation p_t, following these high-level steps: Initialize a serialization string, select a primary subtree and a secondary subtree, generate a reversed version of the secondary subtree and finally perform a pre-order DFS traversal of the primary subtree incorpo-

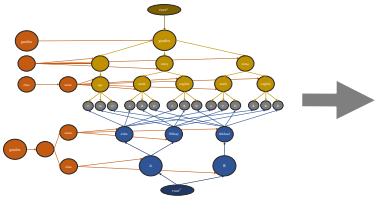
rating context string generations for context cells and value string generations for value cells. Algorithm 3.4 outlines the details of these several steps.

Algorithm 3.4 TabTree Serialization – Core Idea

1. **Initialize Serialization String**: Initialize the serialization string p_t as an empty string.
2. **Primary Subtree Selection:** Select a direction colour $CLR \in \{\texttt{yellow}, \texttt{blue}\}$. If $CLR = \texttt{yellow}$, define a primary subtree $T_1 = T_c$ (Column Header Tree) and a secondary subtree $T_2 = T_r$ (Row Label Tree). If $CLR = \texttt{blue}$, then $T_1 = T_r$ and $T_2 = T_c$. The secondary subtree's main node colour is denoted as $\overline{CLR}$, where $\overline{\texttt{yellow}} = \texttt{blue}$ and $\overline{\texttt{blue}} = \texttt{yellow}$. This choice determines which tree is traversed first to establish the context.
3. **Reverse Tree Construction:** Construct the reverse TabTree $\overleftarrow{T_2}$ of the secondary subtree T_2. This reversed tree is later on used for traversing back up the secondary tree from value cells.
4. **Context String Generation:** Perform a DFS traversal of the primary subtree T_1, starting at its root $v_{\text{root}}^{T_1}$. For each non-leaf node $v \in V_1$ with $\phi_1(v) = CLR$ do the following:
   a) Generate a context string $p_{\text{context}}(v)$ that captures the context of v within T_1. Append the context string $p_{\text{context}}(v)$ to the output string p_t.
   b) Optionally, include context-intersection connections (edges to nodes coloured `orange`) to the generation process of the context string to enhance the contextual information.
5. **Value String Generation:** Once the DFS traversal of the primary tree arrives at a value cell / value node, i.e., $v \in V_{\text{TabTree}}$ with $\phi_{\text{TabTree}}(v) = \texttt{gray}$, perform the following steps in order to generate a string representation for the value cell:
   a) Traverse the reversed secondary subtree $\overleftarrow{T_2}$ filtered on $\overline{CLR}$ from v towards its root $v_{\text{root}}^{T_2}$.
   b) Construct a value string $p_{\text{value}}(v)$ by aggregating information from the nodes encountered during this traversal. Append the value string $p_{\text{value}}(v)$ to the output string p_t. Optionally, include context-intersection connections (edges to nodes coloured `orange`) to enrich the value string with additional context.

Figure 3.18 illustrates an exemplary serialization of the sample table from Figure 3.5 based upon its representation as TabTree model (refer to Figure 3.16). The serialization follows the proposed core idea, for both Column Header Tree and Row Label Tree as primary subtree T_1. The presented approach represents a baseline approach for

generating context and value strings of the core idea, without yet accounting for context-intersection connections. The detailed strategies for constructing context strings and value strings are discussed below.



Column Header Tree as Primary Subtree

```
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:
            ...
```

Row Label Tree as Primary Subtree

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

Figure 3.18.: Base serialization of the sample table of Figure 3.5 for the Column Header Tree as primary subtree and the Row Label Tree as primary subtree.

To illustrate the serialization process, we describe the case where the Column Header Tree serves as the primary subtree ($T_1 = T_c$) and consequently the direction colour is set to $CLR =$ `yellow`:

- The DFS traversal of the Column Header Tree begins at the root node $v_{\text{root}}^{T_2} = v_{\text{root}}^{c}$, which has a single outgoing edge leading to the node with the value 'grades'.
- Since the 'grades' node has the same colour (`yellow`) as the root node, a context string is generated, which is in this case 'grades:'. This string is appended to the so far empty serialized string p_t.
- Continuing the DFS traversal, the next visited node is a `yellow`-coloured node with an empty value. In this case, no modification is made to p_t, as an empty string is added, and the traversal proceeds.
- The following node visited is labeled 'age', again a context node coloured in `yellow`. The context string 'age:' is generated and appended to p_t.
- The next node in the traversal, labeled '17', corresponds to the leftmost `gray`-coloured node in Figure 3.18. At this point, the first traversal of the reversed secondary subtree ($\overleftarrow{T_2} = T_r$) begins. While traversing to the root $v_{\text{root}}^{T_2} = v_{\text{root}}^{r}$, the nodes labeled 'John' and 'A' are passed. The values of these row labels (i.e., nodes with the colour `blue`), are concatenated in reverse order using the separator '>', and the resulting string is joined with the value '17' via a colon. This produces the value string: 'A > John: 17'. This string is appended to p_t.

- Now the DFS traversal of the primary tree continues and proceeds to the node directly to the right, labeled '16'. As before, the reversed secondary subtree ($\overleftarrow{T_2} = T_r$) is traversed up to $v_{\text{root}}^{T_2} = v_{\text{root}}^{r}$, the row labels passed are concatenated in reverse order and joined with the current value to form the value string: 'A > Tiffany: 16'. This value string is again appended to p_t.
- The DFS traversal continues analogously, incrementally constructing the serialized string p_t. The resulting structure, highlighted in yellow, is depicted in Figure 3.18.

The presented core idea for string generation out of a TabTree model does not provide a detailed specification of how context strings $p_{\text{context}}(v)$ and value strings $p_{\text{value}}(v)$ are generated. In Figure 3.18, only one possible approach is applied, while several other methods are conceivable. Below, various approaches for generating both context strings and value strings are introduced.

Context String Generation The context string generation represents one of the three fundamental components of the modular system for string generation within the TabTree model (see Figure 3.17). Its primary purpose is to encode the tabular context, either the column headers or row labels, depending on the chosen primary subtree T_1.

We propose two distinct strategies for generating a context string, $p_{\text{context}}(v)$ with $v \in V_1$, which can optionally incorporate context-intersection nodes as described in the core idea process of Algorithm 3.4. Additionally, we consider the alternative of omitting context strings entirely, i.e., skipping step four in Algorithm 3.4 and generating only value strings.

As a result, there are five possible approaches to context string generation:

1. Two primary strategies,
2. Each with or without the integration of context-intersection nodes, and
3. One approach that assigns empty strings to all context nodes.

A formal description of the four non-empty approaches is provided in Algorithm 3.6, Algorithm 3.7, Algorithm 3.9, and Algorithm 3.10.

Prerequisites for Context String Generation In order to elaborate the details of the different approaches of context string generation, we introduce definitions of the relationships between nodes in $T = (V, E, \phi)$, i.e., parent, sibling, and child nodes.

Definition 3.21 (Parent Node, Sibling Nodes & Children Nodes)**.** *Given a directed tree as a TabTree model, i.e., $T = (V, E, \phi)$, and a node $v \in V$, a parent node, sibling nodes and children nodes are defined as follows:*

- *A parent node v_{parent} is the unique node $u \in V$ such that $(u, v) \in E$.*
- *Two nodes v_1 and v_2 are called siblings or sibling nodes if they share the same parent node v_{parent}, i.e., (v_{parent}, v_1), $(v_{parent}, v_2) \in E$ and $v_1 \neq v_2$. The set of siblings of v is:*

$$V_{siblings}(v) = \{v_{sibling} \in V \mid (v_{parent}, v), (v_{parent}, v_{sibling}) \in E, v_{sibling} \neq v\}$$

- *Children or child nodes of v are the nodes $v_{child} \in V$ such that $(v, v_{child}) \in E$. The set of children of v is:*

$$V_{children}(v) = \{v_{child} \in V \mid (v, v_{child}) \in E\}$$

We further introduce notations to represent strings derived from sequences of nodes and sets of nodes. For a sequence of nodes $\pi = \langle v_1, \ldots, v_k \rangle$, its string representation p_π^{seq} is constructed by concatenating the values t_v of each node v in the provided order, using a designated delimiter `sep`. The delimiter may vary across the sequence and is therefore position-dependent, i.e., $\texttt{sep}(v_l)$ for $l = 1, \ldots, k$. If there is only one node $v_l \in \pi$, then the sequence string only consists of the value of this node. Algorithm 3.5 formally outlines the process of generation sequence strings.

For a set of nodes $V = \{v_1, \ldots, v_k\}$, the corresponding string representation p_V is generated by converting the set into a sequence by random ordering and applying the

Algorithm 3.5 Sequence String Representation

Input: Set of nodes V, Sequence of nodes $\pi = \langle v_1, \ldots, v_k \rangle$ with $v_l = \tilde{\gamma}_{v_l} = (\tilde{s}^c_{v_l}, \tilde{s}^r_{v_l}, t_{v_l}) \in V \ \forall l = 1, \ldots, k$; Separator approach `sep`

Output: String representation of π

1: **if** $|\pi| = 1$ **then**
2: $\quad p_\pi^{\text{seq}} \leftarrow <t_{v_1}>$
3: **else**
4: $\quad p_\pi^{\text{seq}} \leftarrow <t_{v_1}> <\texttt{sep}(v_1)> \ldots <\texttt{sep}(v_{k-1})> <t_{v_k}>$
5: **end if**
6: **return** p_π^{seq}

previously introduced definition, i.e., we use a random permutation $\sigma : \{1, \ldots, k\} \to \{1, \ldots, k\}$ and generate the string representation for the sequence $\pi_{V_\sigma} = \langle v_{\sigma(1)}, \ldots, v_{\sigma(k)} \rangle$ as described in Algorithm 3.5, i.e.,

$$p_V := p^{\text{seq}}_{\pi_{V_\sigma}}$$

Examples of a few options for selecting a separator approach and the corresponding construction of a sequence string are presented in Table 3.1. In order to illustrate the proposed approaches, we use a sequence of nodes from the example of the TabTree model of Figure 3.16, i.e., nodes with the values 'grades', 'name', and 'John'.

Separator Approach	**Separator Definition**	**Sequence String p^{seq}_{π}**
Comma	$\texttt{sep}(v_l) = \text{','}$	grades, name, John
Comma + Colon	$\texttt{sep}(v_l) = \begin{cases} \text{','} & \text{if } l \neq k-1 \\ \text{':'} & \text{else} \end{cases}$	grades, name: John
Comma + Is	$\texttt{sep}(v_l) = \begin{cases} \text{','} & \text{if } l \neq k-1 \\ \text{'is'} & \text{else} \end{cases}$	grades, name is John
Hierarchy + Colon	$\texttt{sep}(v_l) = \begin{cases} \text{'>'} & \text{if } l \neq k-1 \\ \text{':'} & \text{else} \end{cases}$	grades > name: John

Table 3.1.: Examples of separator selection for generating a string representation of a sequence of nodes in the TabTree model.

Context String Generation without Context-Intersection Cells With the introduced formalisms in place, we can now formally define the two distinct approaches for generating the context string $p_{\text{context}}(v)$: a *Base* approach and a *Text-Augmentation* approach. Both methods may optionally incorporate context-intersection nodes, represented by nodes highlighted in `orange`. Initially, we will focus on the case in which context-intersection nodes are not taken into account.

When context-intersection nodes are excluded, we focus on the primary subtree without context-intersection connections, omitting all nodes marked in `orange`. Additionally, we exclude value nodes, i.e., nodes marked in `gray`, retaining only the context nodes. In other words, we first consider the colour set $S_{\text{colour}} = \{CLR\}$ and only use the filtered primary subtree $T_{1|S_{\text{colour}}}$.

Figure 3.19 displays the two proposed strategies for context string generation without context-intersection nodes using two examples from the TabTree model of Figure 3.16. The first example considers the context node with the value '2023', where the Column Header Tree is selected as the primary subtree T_1. The second example focuses on the context node 'John', where the Row Label Tree is chosen as the primary subtree. In both cases, the respective nodes are highlighted with a red circle within their subtrees, and the generated context strings are displayed on the right-hand side of the figure.

1. **Base**: The context string is generated by concatenating the value of the node with a colon. This process is formally described in Algorithm 3.6.

2. **Text Augmentation**: Generate a detailed, natural-language description of v by incorporating its siblings and children. The general procedure is outlined below, while Algorithm 3.7 provides a formal, detailed description:

   a) Using the primary tree T_1 and the associated direction colour CLR, we define the colour set $S_{\text{colour}} = \{\text{CLR}\}$. This allows us to filter the primary tree to retain only context nodes, resulting in $T_{1|S_{\text{colour}}}$.

   b) In the filtered primary subtree of v, we identify all siblings, $V^v_{\text{siblings}} = V_{1|S_{\text{colour}}|\text{siblings}}(v)$, and all children, $V^v_{\text{children}} = V_{1|S_{\text{colour}}|\text{children}}(v)$ of v.

   c) For both sets, V^v_{siblings} and V^v_{children}, we generate their respective string representations, $p_{V^v_{\text{siblings}}}$ and $p_{V^v_{\text{children}}}$, using the designated separator approach described above. These generated sequence strings are then combined with the value t_v of the initial context node v to construct the final context string representation $p_{\text{context}}(v)$.

   Note that the set of children only considers context nodes due to the colour filtering of the primary tree T_1, i.e., column header nodes or row label nodes, and does not include value nodes.

Algorithm 3.6 Context String Generation – Base

Input: Context node $v = (\tilde{s}^c_v, \tilde{s}^r_v, t_v)$

Output: Context string $p_{\text{context}}(v)$

1: $p_{\text{context}}(v) \leftarrow <t_v>$:

2: **return** $p_{\text{context}}(v)$

Algorithm 3.7 Context String Generation – Text-Augmentation

Input: Context node $v = (\tilde{s}_v^c, \tilde{s}_v^r, t_v)$, Primary tree T_1, Direction colour CLR, Separator approach `sep`

Output: Context string $p_{\text{context}}(v)$

▷ ***Filter primary tree***

Define colour set $S_{\text{colour}} = \{CLR\}$

Filter primary tree on S_{colour} to obtain $T_{1|S_{\text{colour}}}$

▷ ***Generate primary tree string*** p_{T_1}

if $CLR =$ `yellow` **then**

    $p_{T_1} \leftarrow$ 'column header'

else

    $p_{T_1} \leftarrow$ 'row label'

end if

Identify siblings of v, i.e., $V^v_{\text{siblings}} = V_{1|S_{\text{colour}}|\text{siblings}}(v)$

Generate string representation of V^v_{siblings}, i.e., $p_{V^v_{\text{siblings}}}$, by applying Algorithm 3.5.

Identify children of v, i.e., $V^v_{\text{children}} = V_{1|S_{\text{colour}}|\text{children}}(v)$

Generate string representation of V^v_{children}, i.e., $p_{V^v_{\text{children}}}$, by applying Algorithm 3.5.

▷ ***Construct*** $p_{context}(v)$

if $|V^v_{\text{siblings}}| > 0$ and $|V^v_{\text{children}}| > 0$ **then**

    $p_{\text{context}}(v) \leftarrow$ The <p_{T_1}> <t_v> has siblings <$p_{V^v_{\text{siblings}}}$>.
    The children of <t_v> are <$p_{V^v_{\text{children}}}$>.

else if $|V^v_{\text{siblings}}| = 0$ and $|V^v_{\text{children}}| > 0$ **then**

    $p_{\text{context}}(v) \leftarrow$ The <p_{T_1}> <t_v> has no siblings.
    The children of <t_v> are <$p_{V^v_{\text{children}}}$>.

else if $|V^v_{\text{siblings}}| > 0$ and $|V^v_{\text{children}}| = 0$ **then**

    $p_{\text{context}}(v) \leftarrow$ The <p_{T_1}> <t_v> has siblings <$p_{V^v_{\text{siblings}}}$>.
    The values of <t_v> are:

else

    $p_{\text{context}}(v) \leftarrow$ The values of the <p_{T_1}> <t_v> are:

end if

return $p_{\text{context}}(v)$

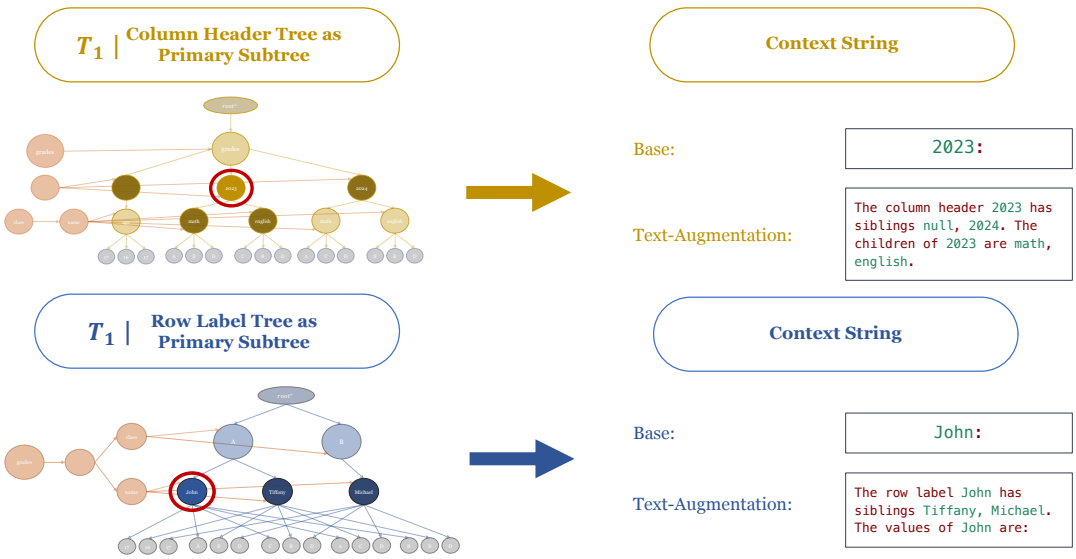


Figure 3.19.: Examples of context string generation without incorporating context-intersection nodes.

Context-Intersection Sequences Next, we extend the previous representations by incorporating context-intersection nodes and edges, specifically the nodes coloured in `orange`. For a given context node v, we include the sequence of connected context-intersection nodes in the generation of the context string $p_{\text{context}}(v)$. A formal description of the procedure is provided in Algorithm 3.8.

We define the colour set $S_{\text{colour}} = \{\text{CLR}, \texttt{orange}\}$ and construct the corresponding filtered tree $T_{S_{\text{colour}}}$ along with its reversed version $\overleftarrow{T}_{S_{\text{colour}}}$. Subsequently, we derive the sequence of connected context-intersection nodes, denoted as $\pi_{\text{context}}(v)$, by identifying the unique longest path originating from node v in the filtered reversed tree. Nodes with an empty value and those with values duplicating the value of a preceding node are excluded from the resulting sequence.

The sequence order, beginning with the initial context node v, is often suboptimal for string generation. Therefore, the sequence is reversed to produce the final output of the process.

Figure 3.20 shows an example of how the respective sequence of connected context-intersection nodes can be derived from the filtered reverse tree for a specific node v. The subtrees shown represent the reverse primary trees filtered for the colours CLR and `orange`. For the Column Header Tree as the primary key and the node with the value '2023' as the example node, only a single node with empty content is connected as a context-intersection node, which is later on discarded. As a result, the context-intersection sequence is given as $\pi_{\text{context}}(v) = \langle 2023 \rangle$.

Algorithm 3.8 Context-Intersection Sequence Generation

Input: Context node v, Colour set $S_{\text{colour}} = \{\text{CLR}, \texttt{orange}\}$, TabTree T

Output: Sequence of connected context nodes $\pi_{\text{context}}(v) = \langle v_{\text{context}|k}, \ldots, v_{\text{context}|1}, v \rangle$

1: Generate filtered reverse tree $\overleftarrow{T}_{S_{\text{colour}}} = (\overleftarrow{V}_{S_{\text{colour}}}, \overleftarrow{E}_{S_{\text{colour}}}, \overleftarrow{\phi}_{S_{\text{colour}}})$
2: Initialize sequence $\pi_{\text{context}}(v)$ with initial node v, i.e., $\pi_{\text{context}}(v) \leftarrow \langle v \rangle$
3: Set $v_{\text{current}} \leftarrow v$
4: Initialize $l \leftarrow 1$
5: **while** True **do**
6: Find unique node $v_{\text{context}|l}$ such that:
7: $v_{\text{context}|l} = u$ with $(v_{\text{current}}, u) \in \overleftarrow{E}_{S_{\text{colour}}}$ and $\overleftarrow{\phi}_{S_{\text{colour}}}(u) = \texttt{orange}$
8: **if** no such node u exists **then**
9: **break**
10: **else**
11: **if** value of $v_{\text{context}|l}$ is not empty or equal to value of v_{current} **then**
12: Append $v_{\text{context}|l}$ to $\pi_{\text{context}}(v)$, i.e., $\pi_{\text{context}}(v) \leftarrow \pi_{\text{context}}(v) + v_{\text{context}|l}$
13: **end if**
14: Set $v_{\text{current}} \leftarrow v_{\text{context}|l}$
15: Set $l \leftarrow l + 1$
16: **end if**
17: **end while**
18: Reverse $\pi_{\text{context}}(v)$, i.e., $\pi_{\text{context}}(v) \leftarrow \langle v_{\text{context}|k}, \ldots, v_{\text{context}|1}, v \rangle$
19: **return** $\pi_{\text{context}}(v)$

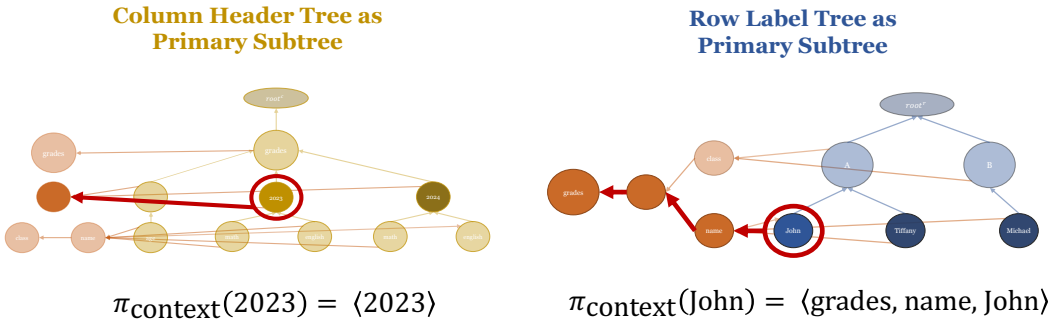


Figure 3.20.: Exemplary derivation of the context-intersection sequence for the case of the Column Header Tree as primary tree and the Row Label Tree as primary tree.

Contrary, for the Row Label Tree as the primary tree and the node 'John' as an example the sequence ⟨ 'John', 'name', '', 'grades' ⟩ is formed, which is then transformed to $\pi_{\text{context}}(v) = \langle$ 'grades', 'name', 'John' $\rangle$ due to empty value filtering and reversing.

Context String Generation with Context-Intersection Cells We use the generated sequence of connected context-intersection nodes to create the two missing variants of the four approaches for generating the context string $p_{\text{context}}(v)$: Their construction is described below, Figure 3.21 illustrates the procedures and their outcomes using the same examples presented in Figure 3.19.

1. **Base with Context-Intersection**: Generate the context string by generating a sequence string representation of the items of the context-intersection sequence $\pi_{\text{context}}(v)$ using a predefined separator `sep`, i.e., generate $p^{\text{seq}}_{\pi_{\text{context}}(v)}$ by applying Algorithm 3.5 and concatenate it with a colon. See Algorithm 3.9 for a formal description.

2. **Text-Augmentation with Context-Intersection**: Preserve the context string generation process from the text-augmentation approach, as outlined in Algorithm 3.7, and extend its string representation by a more sophisticated description of each context node v using information from the context-intersection $\pi_{\text{context}}(v)$. Again the general outline of the procedure is described below, while Algorithm 3.10 provides a formal, detailed description:

   a) First, generate the context sequence $\pi_{\text{context}}(v)$ by applying Algorithm 3.8, with the node v subsequently excluded, resulting in the modified sequence $\tilde{\pi}_{\text{context}}(v) = \pi_{\text{context}}(v) \setminus v$. If this modified sequence satisfies $|\tilde{\pi}_{\text{context}}(v)| = 0$, i.e., there is no context-intersection node with a distinct and non-empty value for the given context node v, the context string is generated by applying Algorithm 3.7, as in the case of *Text-Augmentation* without considering any context-intersection.

   b) If there is at least one context-intersection node with a distinct and non-empty value for the given context node v, i.e., $|\tilde{\pi}_{\text{context}}(v)| > 0$, the context string is also generated in a manner similar to the approach described in Algorithm 3.7, with adjustments made to incorporate information about the context-intersection.

   c) Therefore, first, the string representation $p^{\text{seq}}_{\tilde{\pi}_{\text{context}}(v)}$ is constructed based on Algorithm 3.5. Next, a natural language sentence is constructed using the following structure: *The column header/row label <*t_v*> represents <*$p^{seq}_{\tilde{\pi}_{context}(v)}$*>*. This sentence is then prepended to the string generated by Algorithm 3.7, which does not consider context-intersection nodes.

Algorithm 3.9 Context String Generation – Base with Context-Intersection

Input: Context node $v = (\tilde{s}^c_v, \tilde{s}^r_v, t_v)$, Primary tree T_1, Direction colour CLR, Separator approach `sep`

Output: Context string $p_{\text{context}}(v)$

1: Generate context-intersection sequence $\pi_{\text{context}}(v)$ by applying Algorithm 3.8
2: Generate sequence string representation $p^{\text{seq}}_{\pi_{\text{context}}(v)}$ by applying Algorithm 3.5
3: $p_{\text{context}}(v) \leftarrow <p^{\text{seq}}_{\pi_{\text{context}}(v)}>$:
4: **return** $p_{\text{context}}(v)$

Algorithm 3.10 Context String Generation – Text-Augmentation with Context-I.

Input: Context node $v = (\tilde{s}^c_v, \tilde{s}^r_v, t_v)$, Primary tree T_1, Direction colour CLR, Separator approach `sep`

Output: Context string $p_{\text{context}}(v)$

1: Observe $V^v_{\text{siblings}}, V^v_{\text{children}}, p_{V^v_{\text{siblings}}}, p_{V^v_{\text{children}}}, p_{T_1}$ by applying Algorithm 3.7
2: Generate context-intersection sequence $\pi_{\text{context}}(v)$ by applying Algorithm 3.8
3: Remove v from $\pi_{\text{context}}(v)$, i.e., $\tilde{\pi}_{\text{context}}(v) \leftarrow \pi_{\text{context}}(v) \setminus v$
4: **if** $|\tilde{\pi}_{\text{context}}(v)| = 0$ **then** ▷ No relevant context-intersection nodes found
5: $p_{\text{context}}(v) \leftarrow$ Output of Algorithm 3.7
6: **return** $p_{\text{context}}(v)$
7: **else**
8: Generate sequence string representation $p^{\text{seq}}_{\tilde{\pi}_{\text{context}}(v)}$ by applying Algorithm 3.5
9: **if** $|V^v_{\text{siblings}}| > 0$ and $|V^v_{\text{children}}| > 0$ **then** ▷ Construct $p_{\text{context}}(v)$
10: $p_{\text{context}}(v) \leftarrow$ The $<p_{T_1}>$ $<t_v>$ represents $<p^{\text{seq}}_{\tilde{\pi}_{\text{context}}(v)}>$.
The $<p_{T_1}>$ $<t_v>$ has siblings $<p_{V^v_{\text{siblings}}}>$.
The children of $<t_v>$ are $<p_{V^v_{\text{children}}}>$.
11: **else if** $|V^v_{\text{siblings}}| = 0$ and $|V^v_{\text{children}}| > 0$ **then**
12: $p_{\text{context}}(v) \leftarrow$ The $<p_{T_1}>$ $<t_v>$ represents $<p^{\text{seq}}_{\tilde{\pi}_{\text{context}}(v)}>$.
The $<p_{T_1}>$ $<t_v>$ has children $<p_{V^v_{\text{children}}}>$.
13: **else if** $|V^v_{\text{siblings}}| > 0$ and $|V^v_{\text{children}}| = 0$ **then**
14: $p_{\text{context}}(v) \leftarrow$ The $<p_{T_1}>$ $<t_v>$ represents $<p^{\text{seq}}_{\tilde{\pi}_{\text{context}}(v)}>$.
The $<p_{T_1}>$ $<t_v>$ has siblings: $<p_{V^v_{\text{siblings}}}>$.
The values of $<t_v>$ are:
15: **else**
16: $p_{\text{context}}(v) \leftarrow$ The $<p_{T_1}>$ $<t_v>$ represents $<p^{\text{seq}}_{\tilde{\pi}_{\text{context}}(v)}>$.
The values of the $<p_{T_1}>$ $<t_v>$ are:
17: **end if**
18: **return** $p_{\text{context}}(v)$
19: **end if**

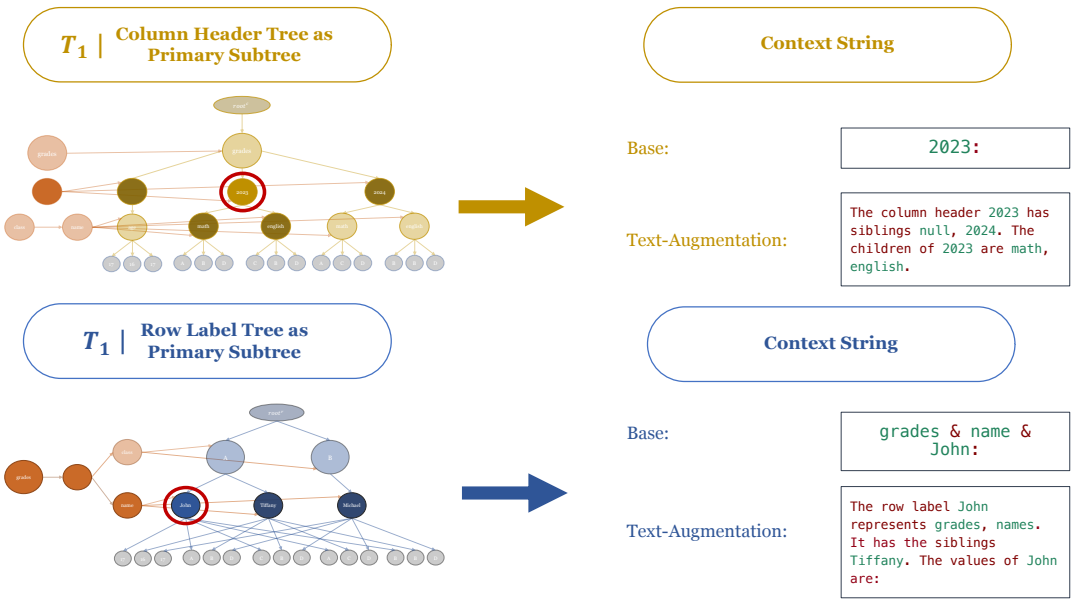


Figure 3.21.: Examples of context string generation incorporating context-intersection nodes.

Value String Generation The second core component of the modular system for string generation within the TabTree model, as depicted in Figure 3.17, is Value String Generation. As the name implies, this component refers to the task of generating string representations of the table's value-containing elements, i.e., the value cells.

As outlined in Algorithm 3.4, the generation of value cell strings is enclosed within the string representation of context cells. In other words, each generated value cell string is preceded by one or more previously generated context cell strings, except in the case of the empty context approach.

The primary objective of value cell generation within this framework is to capture information not yet covered by the context string generation. This includes, on the one hand, establishing a semantic connection to the enclosing context cells, and on the other hand, integrating relevant information from context nodes in the secondary subtree T_2, which have not yet been incorporated into the string generation process at this point.

To address these requirements, we propose three distinct approaches for value string generation $p_{\text{value}}(v)$ for a value node v, which may be extended to incorporate context-intersection nodes, analogous to the concepts of context string generation. Consequently, this results in a total of six possible options for generating a value string.

Value Sequence In general, all value string generation approaches rely, as outlined in Algorithm 3.4, on traversing the reversed secondary subtree $\overleftarrow{T_2}$ of the TabTree model. This traversal produces a sequence of values, which we refer to as the value sequence.

Starting from a given value node v, the traversal proceeds toward the root node $v_{\text{root}}^{T_2}$, considering only nodes of the secondary colour $\overline{CLR}$, i.e., excluding context-intersection nodes coloured in `orange`. This colour-based filtering ensures the uniqueness of the resulting path. For practical purposes, similar to the treatment of context-intersection node sequences (see Algorithm 3.8), we consider the reverse of this path, ultimately yielding the value sequence $\pi_{\text{value}}(v)$.

Algorithm 3.11 formally outlines the process for generating $\pi_{\text{value}}(v)$. The application of this algorithm is exemplified in Figure 3.22, which visualizes the approach for two value nodes from the TabTree sample of Figure 3.16. For both the Column Header Tree and the Row Label Tree as secondary subtrees, the resulting value sequences are depicted for an exemplary value node using red arrows. Notably, the textual representations of these sequences, as shown in the figure, correspond to their final reversed versions.

Algorithm 3.11 Value Sequence Generation

Input: Value node v, Direction colour $\overline{CLR}$, TabTree subtree T_2

Output: Sequence of nodes connected to the value node v of the secondary subtree, $\pi_{\text{value}}(v) = \langle v_k, \ldots, v_1, v \rangle$

1: Define colour set $S_{\text{colour}} = \{\overline{CLR}, \texttt{gray}\}$
2: Generate filtered reverse tree $\overleftarrow{T}_{2|S_{\text{colour}}} = (\overleftarrow{V}_{2|S_{\text{colour}}}, \overleftarrow{E}_{2|S_{\text{colour}}}, \overleftarrow{\phi}_{2|S_{\text{colour}}})$
3: Initialize sequence $\pi_{\text{value}}(v) \leftarrow \langle\ \rangle$
4: Set $v_{\text{current}} \leftarrow v$
5: Initialize $l \leftarrow 1$
6: **while** $v_{\text{current}} \neq v_{\text{root}}^{T_2}$ **do**
7: Append v_{current} to $\pi_{\text{value}}(v)$, i.e., $\pi_{\text{value}}(v) \leftarrow \pi_{\text{value}}(v) + v_l$
8: Find unique node v_l such that:
9: $v_l = u$ with $(v_{\text{current}}, u) \in \overleftarrow{E}_{2|S_{\text{colour}}}$
10: Set $v_{\text{current}} \leftarrow v_l$
11: Set $l \leftarrow l + 1$
12: **end while**
13: Reverse $\pi_{\text{value}}(v)$, i.e., $\pi_{\text{value}}(v) \leftarrow \langle v_k, \ldots, v_1, v \rangle$
14: **return** $\pi_{\text{value}}(v)$

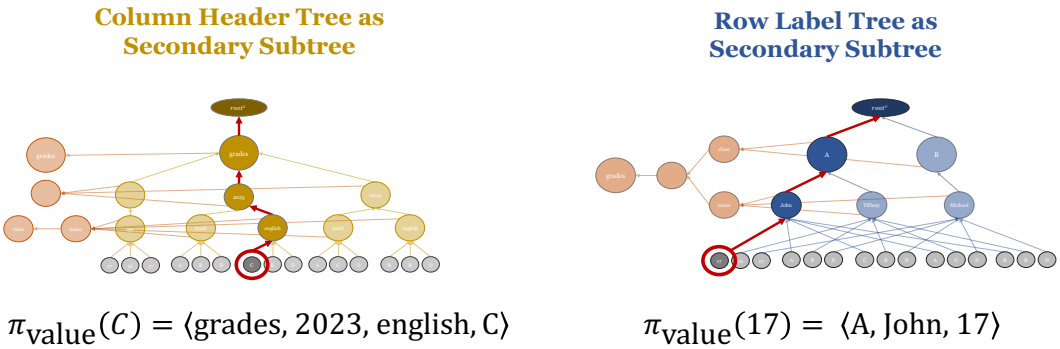


Figure 3.22.: Exemplary derivation of the value sequence for the case of the Column Header Tree as secondary subtree and the Row Label Tree as secondary subtree.

Value String Generation without Context-Intersection Cells Based on the value sequence $\pi_{\text{value}}(v)$ for a value node $v = \tilde{\gamma}_{ij}$, we propose three options for value string generation without considering context-intersection cells, detailed below. Exemplary realizations of these approaches are presented in Figure 3.23: The figure demonstrates string generations for a value node in two scenarios: one where the Column Header Tree serves as the secondary subtree, and another where the Row Label Tree is used as the secondary subtree.

1. **Base**: This approach generates the value string by applying Algorithm 3.5 for sequence string representation, using the value sequence $\pi_{\text{value}}(v)$. The procedure is formally described in Algorithm 3.12.

2. **Text:** This approach extends the base method by wrapping the value sequence $\pi_{\text{value}}(v)$ in a natural language sentence. Specifically, it employs the 'Comma + Is' separator approach (refer to Table 3.1) to construct the sequence string representation $p^{\text{seq}}_{\pi_{\text{value}}(v)}$. Furthermore, to facilitate a more comprehensive understanding of the values context within the table, the parent node within the primary subtree $v_{\text{parent}|T_1}$ with value $t_{v_{\text{parent}|T_1}}$ and the corresponding row index i and column index j are included into the constructed sentence. Depending on the choice of primary or secondary subtree, sentences are thus constructed using the following template: *The value of column header / row label <$t_{v_{parent|T_1}}$> and the row label / column header combination <$p^{seq}_{\pi_{value}(v)}$> (row index: <i>, column index: <j>).*

3. **Text Augmentation**: The text augmentation approach constitutes an extension of the previous text approach. In this approach, in addition to incorporating the context information derived from the traversed nodes within the secondary subtree

of the TabTree model, the context from the primary subtree associated with the node $v = \tilde{\gamma}_{ij}$ is also utilized to enhance the generation of the value string.

The approach is based on the idea of unifying all information related to a specific table cell or node within the TabTree model into a single natural language sentence, allowing language models to capture the whole context of data more easily. It is, however, important to note that the primary subtree context is already captured through the context string generation process. Consequently, this approach introduces a degree of redundancy as some information is repeated.

The procedural steps for the text augmentation approach are detailed below and formally outlined in Algorithm 3.14:

a) A value sequence is generated for the reversed primary subtree using Algorithm 3.11, analogous to the sequence generation for the secondary subtree. As a result, two value sequences are obtained: one for the secondary subtree ($\pi_{\text{value}|\overleftarrow{T_2}}(v)$) and another for the primary subtree ($\pi_{\text{value}|\overleftarrow{T_1}}(v)$).

b) Node v is excluded from both sequences to yield modified sequences $\tilde{\pi}_{\text{value}|\overleftarrow{T_2}}(v) = \pi_{\text{value}|\overleftarrow{T_2}}(v) \setminus \{v\}$ and $\tilde{\pi}_{\text{value}|\overleftarrow{T_1}}(v) = \pi_{\text{value}|\overleftarrow{T_1}}(v) \setminus \{v\}$. Subsequently, string representations of these sequences are created applying Algorithm 3.5, resulting in $p^{\text{seq}}_{\tilde{\pi}_{\text{value}|\overleftarrow{T_2}}(v)}$ and $p^{\text{seq}}_{\tilde{\pi}_{\text{value}|\overleftarrow{T_1}}(v)}$.

c) A value string is constructed as a sentence that incorporates the sequence string representations from the previous step. This captures the contexts of both the column header and row label, with the value t_v of node v appended. For example, if the Column Header Tree serves as the secondary subtree and the Row Label Tree as the primary subtree, the resulting representation is: *The value of the column header combination <$p^{seq}_{\tilde{\pi}_{value|\overleftarrow{T_2}}(v)}$> and row label combination <$p^{seq}_{\tilde{\pi}_{value|\overleftarrow{T_1}}(v)}$> is <$t_v$> (row index: <i>, column index: <j>).*

   In cases where the Row Label Tree is the secondary subtree and the Column Header Tree is the primary subtree, the terms 'column header' and 'row label' are interchanged accordingly.

Algorithm 3.12 Value String Generation – Base

Input: Value node v, Secondary colour $\overline{CLR}$, Secondary subtree T_2, Separator approach `sep`

Output: Value string $p_{\text{value}}(v)$

1: Generate value sequence $\pi_{\text{value}}(v)$ by applying Algorithm 3.11
2: Generate sequence string representation $p^{\text{seq}}_{\pi_{\text{value}}(v)}$ by applying Algorithm 3.5
3: $p_{\text{value}}(v) \leftarrow <p^{\text{seq}}_{\pi_{\text{value}}(v)}>$
4: **return** $p_{\text{value}}(v)$

Algorithm 3.13 Value String Generation – Text

Input: Value node $v = \tilde{\gamma}_{ij}$, Secondary colour $\overline{CLR}$, Secondary subtree T_2, Primary subtree T_1

Output: Value string $p_{\text{value}}(v)$

1: Select separator approach `sep` ← 'Comma + Is' (see Table 3.1)
2: **if** $\overline{CLR}$ = `yellow` **then** ▷ Generate secondary tree string p_{T_2}
3: $p_{T_2} \leftarrow$ 'column header'
4: **else**
5: $p_{T_2} \leftarrow$ 'row label'
6: **end if**
7: Get parent node of v in the primary subtree T_1, i.e., $v_{\text{parent}|T_1}$ with value $t_{v_{\text{parent}|T_1}}$
8: Generate value sequence $\pi_{\text{value}}(v)$ by applying Algorithm 3.11
9: Generate sequence string $p^{\text{seq}}_{\pi_{\text{value}}(v)}$ by applying Algorithm 3.5 using `sep`
10: $p_{\text{value}}(v) \leftarrow$ The value of the $<p_{T_1}>$ $<t_{v_{\text{parent}|T_1}}>$ and the $<p_{T_2}>$ combination $<p^{\text{seq}}_{\pi_{\text{value}(v)}}>$ (row index: $<i>$, column index: $<j>$).
11: **return** $p_{\text{value}}(v)$

Algorithm 3.14 Value String Generation – Text-Augmentation

Input: Value node $v = (\tilde{s}^c_v, \tilde{s}^r_v, t_v) = \tilde{\gamma}_{ij}$, Secondary colour $\overline{CLR}$, Primary colour CLR, Secondary subtree T_2, Primary subtree T_1

Output: Value string $p_{\text{value}}(v)$

1: **if** $\overline{CLR}$ = `yellow` **then** ▷ Generate secondary tree string p_{T_2}
2: $p_{T_2} \leftarrow$ 'column header'
3: $p_{T_1} \leftarrow$ 'row label'
4: **else**
5: $p_{T_2} \leftarrow$ 'row label'
6: $p_{T_1} \leftarrow$ 'column header'
7: **end if**

8: Generate value sequence $\pi_{\text{value}|\overleftarrow{T_2}}(v)$ for T_2 and colour $\overleftarrow{CLR}$ by applying Algorithm 3.11
9: Generate value sequence $\pi_{\text{value}|\overleftarrow{T_1}}(v)$ for T_1 and colour CLR by applying Algorithm 3.11
10: Remove v from $\pi_{\text{value}|\overleftarrow{T_2}}(v)$, i.e., $\tilde{\pi}_{\text{value}|\overleftarrow{T_2}}(v) = \pi_{\text{value}|\overleftarrow{T_2}}(v) \setminus \{v\}$
11: Remove v from $\pi_{\text{value}|\overleftarrow{T_1}}(v)$, i.e., $\tilde{\pi}_{\text{value}|\overleftarrow{T_1}}(v) = \pi_{\text{value}|\overleftarrow{T_1}}(v) \setminus \{v\}$
12: Generate sequence string $p^{\text{seq}}_{\tilde{\pi}_{\text{value}|\overleftarrow{T_2}}(v)}$ by applying Algorithm 3.5
13: Generate sequence string $p^{\text{seq}}_{\tilde{\pi}_{\text{value}|\overleftarrow{T_1}}(v)}$ by applying Algorithm 3.5
14: $p_{\text{value}}(v) \leftarrow$ The value of the $<p_{T_1}>$ combination $<p^{\text{seq}}_{\tilde{\pi}_{\text{value}|\overleftarrow{T_1}}(v)}>$ and the $<p_{T_2}>$ combination $<p^{\text{seq}}_{\tilde{\pi}_{\text{value}|\overleftarrow{T_2}}(v)}>$ is $<t_v>$ (row index: $<i>$, column index: $<j>$).
15: **return** $p_{\text{value}}(v)$

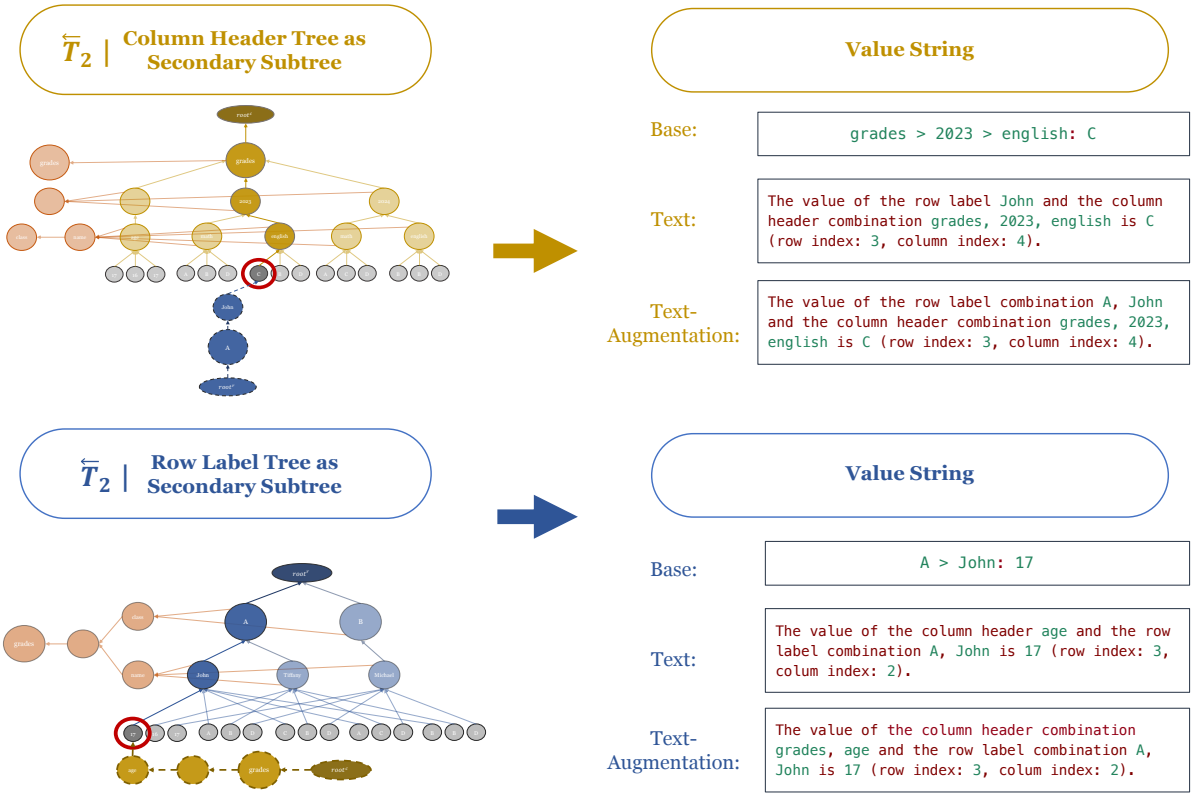


Figure 3.23.: Examples of value string generation without incorporating context-intersection nodes.

Value String Generation with Context-Intersection Cells Similar to the generation of context strings, it is also possible to incorporate `orange` context-intersection nodes into the approaches described for value string generation. To achieve this, we propose a

modified version of the value sequence generation procedure (see Algorithm 3.11). This adaptation generates the context-intersection sequence for each node encountered during the traversal from the value node v to the root of the subtrees and stores it as sequence of sequences – in contrast to the original version, where only single nodes were appended to the value sequence. Algorithm 3.15 formally defines the modified procedure of string generation of nested sequences, while Figure 3.24 illustrates its practical application using the same examples as presented in Figure 3.22.

Algorithm 3.15 Value Sequence Generation with Context-Intersection

Input: Value node v, Direction colour $\overline{CLR}$, TabTree subtree T_2

Output: Sequence of sequences of nodes connected to the value node v of the secondary subtree, $\pi_{\text{value}}(v) = \langle \pi_{\text{context}}(v_k), \ldots, \pi_{\text{context}}(v_1), \langle v \rangle \rangle$

1: Define colour set $S_{\text{colour}} = \{\overline{CLR}, \texttt{gray}\}$
2: Generate filtered reverse tree $\overleftarrow{T}_{2|S_{\text{colour}}} = (\overleftarrow{V}_{2|S_{\text{colour}}}, \overleftarrow{E}_{2|S_{\text{colour}}}, \overleftarrow{\phi}_{2|S_{\text{colour}}})$
3: Initialize sequence $\pi_{\text{value}}(v)$ with a sequence of the containing only the initial node v, i.e., $\pi_{\text{context}}(v) \leftarrow \langle \langle v \rangle \rangle$
4: Set $v_{\text{current}} \leftarrow v$
5: Initialize $l \leftarrow 1$
6: **while** True **do**
7: Find unique node v_l such that:
8: $v_l = u$ with $(v_{\text{current}}, u) \in \overleftarrow{E}_{2|S_{\text{colour}}}$
9: **if** $v_{\text{l}} = v_{\text{root}}^{T_2}$ **then**
10: break
11: **else**
12: Generate context-intersection sequence $\pi_{\text{context}}(v_l)$ by applying Algorithm 3.8, i.e., $\pi_{\text{context}}(v_l) = \langle v_{\text{context}|h_l|l}, \ldots, v_{\text{context}|1|l}, v_l \rangle$
13: Append $\pi_{\text{context}}(v_l)$ to $\pi_{\text{value}}(v)$, i.e., $\pi_{\text{value}}(v) \leftarrow \pi_{\text{value}}(v) + \pi_{\text{context}}(v_l)$
14: Set $v_{\text{current}} \leftarrow v_l$
15: Set $l \leftarrow l + 1$
16: **end if**
17: **end while**
18: Reverse $\pi_{\text{value}}(v)$, i.e., $\pi_{\text{value}}(v) \leftarrow \langle \pi_{\text{context}}(v_k), \ldots, \pi_{\text{context}}(v_1), \langle v \rangle \rangle$
19: **return** $\pi_{\text{value}}(v)$

Based on the resulting sequence of context-intersection sequences, value string representations can be generated through the iterative application of Algorithm 3.5. Initially, string representations are constructed at the lowest level, specifically for the context-

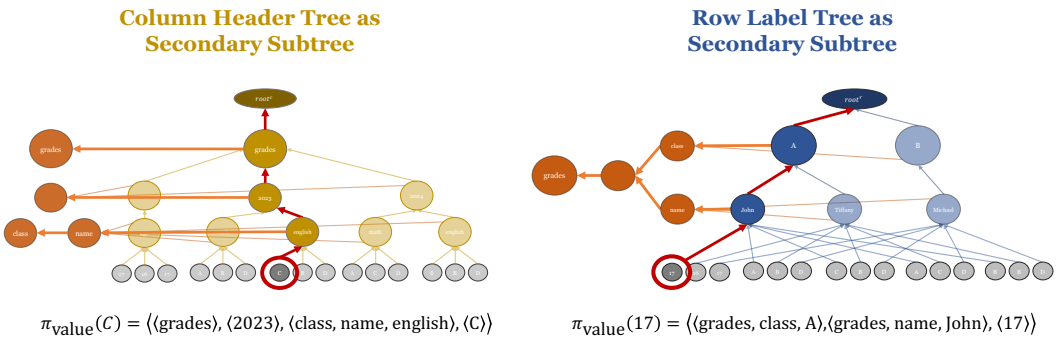


Figure 3.24.: Exemplary derivation of the value sequence incorporating context-intersection nodes for the case of the Column Header Tree as secondary subtree and the Row Label Tree as secondary subtree.

intersection connections. Subsequently, the resulting context-intersection connections are combined into a unified string by reapplying Algorithm 3.5. This iterative process is formally defined in Algorithm 3.16.

Algorithm 3.16 Nested Sequence String Representation

Input: Nested sequence of nodes $\pi = \langle \pi_1, \ldots, \pi_k \rangle$, First-level separator approach `sep-1`, Lower-level separator approach `sep-2`

Output: String representation of p_π^{seq}

1: Initialize sequence string representation p_π^{seq} as empty string
2: Initialize $l = 1$
3: **for** $l \leq k$ **do**
4: Generate sequence string representation $p_{\pi_l}^{\text{seq}}$ by applying Algorithm 3.5 using `sep-2`
5: **if** $l < k$ **then**
6: Append $p_{\pi_l}^{\text{seq}}$ to p_π^{seq} in combination with `sep-1`, i.e., $p_\pi^{\text{seq}} \leftarrow <p_\pi^{\text{seq}}><\texttt{sep-1}><p_{\pi_l}^{\text{seq}}>$
7: **else if** $l = k$ **then**
8: Append only $p_{\pi_l}^{\text{seq}}$ to p_π^{seq}, i.e., $p_\pi^{\text{seq}} \leftarrow <p_\pi^{\text{seq}}><p_{\pi_l}^{\text{seq}}>$
9: **end if**
10: **end for**
11: **return** p_π^{seq}

The implementation of the three proposed methods incorporating context-intersection nodes – **Base with Context-Intersection**, **Text with Context-Intersection**, and **Text-Augmentation with Context-Intersection** – follow the same procedure as outlined in Algorithm 3.12, Algorithm 3.13, and Algorithm 3.14. The key difference lies

in the generation of sequence strings, which is achieved using Algorithm 3.16 instead of Algorithm 3.5. Figure 3.25 provides an example implementation of these methods in this setting.

The figure illustrates string generation for a value node with context-intersection nodes again under two scenarios: one where the Column Header Tree is utilized as the secondary subtree and another where the Row Label Tree is used as the secondary subtree.

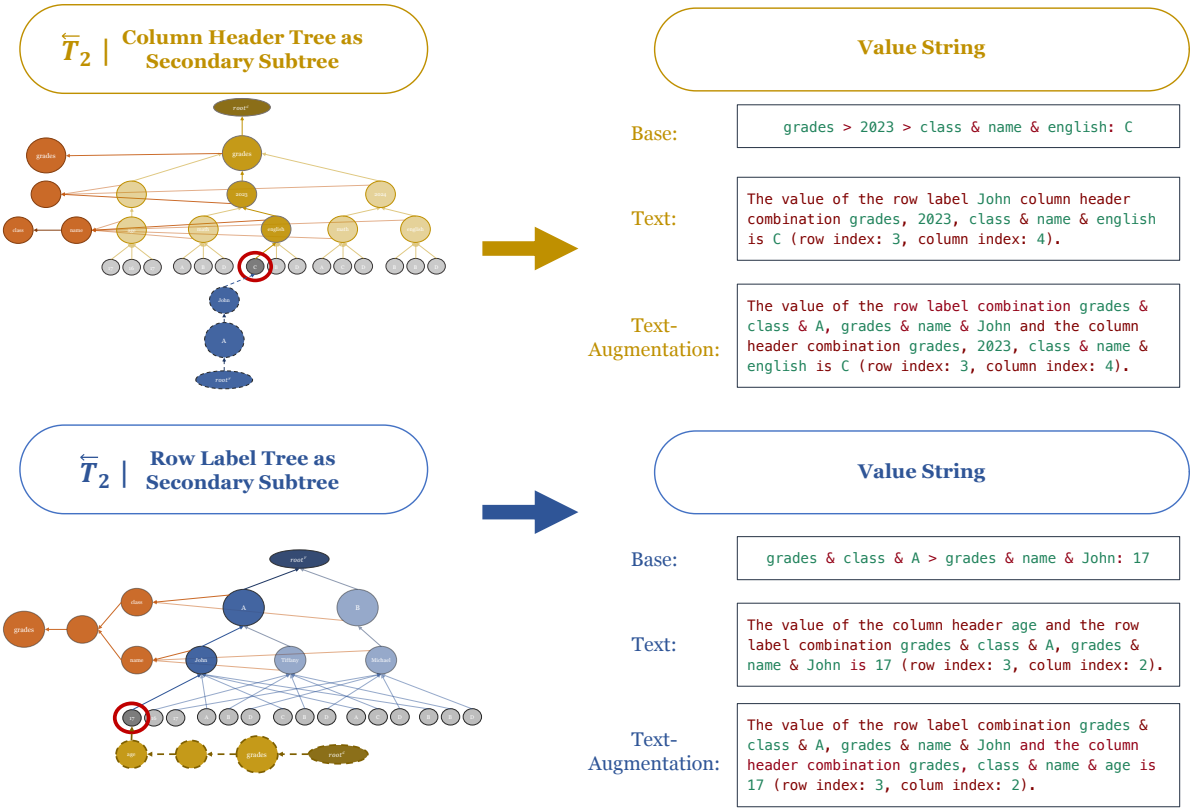


Figure 3.25.: Examples of value string generation incorporating context-intersection nodes.

Primary Subtree Selection The third component of the modular system for string generation in the TabTree model is the selection of the primary subtree. As detailed in Algorithm 3.4, the serialization of a table depends on the choice of the primary subtree T_1 and the secondary subtree T_2, which are distinguished by the primary colour CLR and the secondary colour $\overline{CLR}$, respectively.

In some cases, a representation derived from the Column Header Tree as primary subtree may better capture the table's structure, while in others, a representation starting from the Row Label Tree as primary subtree is more appropriate. Determining the optimal

choice, however, remains ambiguous and depends on the specific characteristics of the table. For this reason, we propose four different approaches for solving the issue of primary subtree selection:

1. **Column Header Tree as Primary Subtree:** Always select the Column Header Tree T_c as the primary subtree.
2. **Row Label Tree as Primary Subtree:** Always select the Row Label Tree T_r as the primary subtree.
3. **Concatenated Representation:** Generate string representations using both the Column Header Tree and the Row Label Tree as primary subtrees, and concatenate the resulting strings. Note, that this approach introduces content duplication.
4. **Heuristic-Based Selection:** Select the primary subtree based on the number of column header rows (i^*) and row label columns (j^*). Specifically:
   - If $i^* > j^*$, select the Column Header Tree (T_c) as the primary subtree.
   - If $j^* > i^*$, select the Row Label Tree (T_r) as the primary subtree.
   - If $i^* = j^*$, select the primary subtree randomly to avoid bias.

3.3.3. Column Header- and Row Label-Detection

The preceding discussions on constructing the TabTree model from a table and generating its string representation are based on the assumption that the column headers and row labels of a table are pre-identified. However, in various settings, this assumption does not hold, and the differentiation of column headers and row labels from value content is far from trivial. For instance, certain table representations do not support the assignment of row labels. For example, HTML tables don't incorporate any form of row labelling, only column headers can be specified using the `<thead>` and `<th>` tags.

Since the TabTree model relies on the concepts of column headers and row labels, it becomes essential to reliably detect these elements from a plain table. The task of detecting column headers and row labels, referred to as context detection, can be formally defined as follows:

Definition 3.22 (Context Detection)**.** *Let $t = \{\gamma_{11}, \ldots, \gamma_{nm}\}$ be a table with rows $r_1, \ldots, r_n$ and columns $c_1, \ldots, c_m$. Then the task* `detect-context` *is defined as:*

$$\texttt{detect-context}: \; t \mapsto (h_t, l_t)$$

where $h_t = \langle r_1, \ldots, r_{i^*} \rangle$ *represents the sequence of column headers and* $l_t = \langle c_1, \ldots, c_{j^*} \rangle$ *the sequence of row labels of* t*.*

It is important to note that the correct sequence of column headers and row labels is typically unknown.

Approach In general, the task of context detection in a table is highly intricate and can be approached through a variety of methods. However, as this thesis primarily focuses on the broader task of applying the TabTree model for table QA within large documents, we limit the discussion of this aspect and introduce a relatively straightforward approach, which is formally described in Algorithm 3.17.

The approach is based on the idea of iteratively querying a LLM to determine whether a row or column should be classified as a column header or row label. This process begins with the first row or column and proceeds sequentially through the table. At each step, the content of the current row or column and additional contextual information are incorporated into the prompt provided to the LLM. We outline three variations for incorporating this additional contextual information:

1. **Full Table**: Include the entire table as additional context in the prompt.
2. **1-Range**: Provide, if available, the preceding row or column and the subsequent row or column as context, excluding the remainder of the table.
3. **2-Range**: Provide, if available, the two preceding rows or columns and the two subsequent rows or columns as context, excluding the remainder of the table.

The iterative process terminates as soon as a row or column is identified as non-header or non-label. At this point, the range from the first row or column to the index of the breaking condition is returned as the detected column headers h_t or row labels l_t, respectively.

Figure 3.26 illustrates an exemplary application of the proposed prompting strategies for row r_3 from the sample table shown in Figure 3.26. In this example, it is assumed that rows r_1 and r_2 have already been identified as column header rows. In the third step, the task is to determine whether r_3 should also be classified as a column header row. Ideally, the detection model would classify r_3 positively and proceed to analyze r_4. At r_4, the model should recognize that this row contains value cells, thereby terminating the process. Ultimately, the model would return the sequence $\langle r_1, r_2, r_3 \rangle$ as the detected column header rows.

Algorithm 3.17 Table Context Detection

Input: Table t with rows $r_1, \ldots, r_n$, columns $c_1, \ldots, c_m$; Large language model LLM

Output: Column headers h_t or row labels l_t

Initialize $h_t \leftarrow \emptyset$ or $l_t \leftarrow \emptyset$
Initialize $i \leftarrow 0$
while True **do**
    $i \leftarrow i + 1$
    Query LLM with c_i or r_i using designated prompting strategy
    if LLM determines c_i is not a column header or r_i a row label **then**
        break
    end if
end while
if $i > 0$ **then**
    $h_t \leftarrow \langle r_1, \ldots r_i \rangle$ or $l_t \leftarrow \langle c_1, \ldots c_i \rangle$
end if
return h_t or l_t

Figure 3.26.: Exemplary prompts of the three proposed strategies for context detection: Full-Table, 1-Range, and 2-Range.

3.3.4. Complexity Analysis

This section presents a high-level complexity analysis of the TabTree approach, focusing on its three primary components: Context Detection (see Section 3.3.3), TabTree Model Generation (see Section 3.3.1), and TabTree String Generation (see Section 3.3.2). Throughout the analysis, we assume a table t with n rows and m columns.

Context Detection Context detection involves iterative querying of an LLM for each row and column until a non-header row and a non-label column are identified. To capture the computational cost of a single LLM query, we introduce the variable L, representing the runtime complexity of one such call. This abstraction accounts for several contributing factors of an LLM call, including model size, prompt and output token lengths, and system-level execution conditions (Stamatescu, 2024). In the worst case, each of the n rows and m columns may be queried, yielding a complexity of

$$\mathcal{O}((n+m) \cdot L).$$

Empirical observations suggest, however, that the number of header rows and label columns is typically small. In the datasets evaluated in this work (see Section 4.1), the maximum observed indices for column headers and row labels are 5 and 6, respectively, while the averages are 0.22 and 0.71. We therefore assume a practical upper bound $K \in \mathbb{N}$ on the number of column headers and row labels, and simplify the complexity for context detection to

$$\mathcal{O}(L).$$

TabTree Model Generation TabTree model generation comprises two main operations: preprocessing (see Algorithm 3.1) and the construction of the Column Header Tree and Row Label Tree (Algorithms 3.2 and 3.3). Each of these steps operates linearly in the size of the input table, that is, each table cell is visited once. Accordingly, the complexity for TabTree model generation is

$$\mathcal{O}(n \cdot m).$$

TabTree String Generation Based on a TabTree model $T = (V, E, \phi)$, string generation involves three key steps: selection of the primary subtree, construction of the reversed secondary subtree, and a preorder DFS traversal of the primary subtree. During traversal, context and value strings are generated for each node (see Algorithm 3.4).

Reversing the tree involves inverting all edge directions and has therefore a complexity of

$$\mathcal{O}(n \cdot m).$$

The DFS traversal visits every node in T, which also yields a complexity of

$$\mathcal{O}(n \cdot m).$$

For each visited node, either a context string or a value string is generated. Context string generation may involve additional traversal through context-intersection nodes in the reversed secondary subtree. While this introduces a theoretical cost of $\mathcal{O}(n+m)$, it is bounded in practice by the small number of column headers and row labels K, and therefore treated as $\mathcal{O}(1)$. Value string generation similarly involves traversing to the root of the reversed secondary subtree – potentially including context-intersection nodes – under the same theoretical and empirical bounds. Consequently, string generation per node can be treated as a constant-time operation in practical scenarios.

The total complexity for string generation is thus:

$$\underbrace{\mathcal{O}(n \cdot m)}_{\text{Reverse Tree Construction}} + \left(\underbrace{\mathcal{O}(n \cdot m)}_{\text{DFS Traversal}} \cdot \underbrace{\mathcal{O}(1)}_{\text{String Generation}} \right).$$

Full TabTree Complexity Combining the components yields the overall complexity of the TabTree approach:

$$\underbrace{\mathcal{O}(L)}_{\text{Context Detection}} + \underbrace{\mathcal{O}(n \cdot m)}_{\text{Model Generation}} + \underbrace{\mathcal{O}(n \cdot m)}_{\text{String Generation}} = \mathcal{O}(L) + \mathcal{O}(n \cdot m).$$

In practice, the runtime cost of LLM calls L typically dominate, since real-world tables tend to be modest in size. This observation is supported by the datasets examined in this work (see Section 4.1.1 and Section 4.1.2). As a result, context detection constitutes the most computationally expensive component of the TabTree approach. This distinguishes the TabTree approach from template-based baselines such as HTML, CSV or Markdown, which exhibit a complexity of $\mathcal{O}(n \cdot m)$. However, if the table context is provided in advance, the overall computational cost becomes comparable to that of these baseline methods.

4. Experimental Evaluation

In this chapter, the proposed framework introduced in Chapter 3, is systematically evaluated, with a particular emphasis on the TabTree method developed for table serialization. The chapter is structured as follows: First, we provide a comprehensive description of the datasets employed for evaluation, followed by an overview of the implementation details of the RAG pipeline outlined in Section 3.1. Next, we describe the evaluation setup, including the metrics and baseline models utilized. We then, finally, present the concrete results obtained from the evaluation.

The evaluation focuses on the tasks defined in Section 3.2.3: Vanilla Table QA (Definition 3.11), Chunk Retrieval (Definition 3.13), and Table QA on Full Documents (Definition 3.15). As described in Section 3.2.3, these tasks can be extended from individual documents to document corpora (see Definitions 3.14 and 3.16). While the primary focus of this work is on the former tasks, the extended tasks are also considered, although to a more limited extent.

4.1. Datasets

To evaluate the proposed framework, each of the five tasks requires a set of input questions $q_1, \ldots, q_n$ accompanied by their respective contexts $\theta_1, \ldots, \theta_n$. The specific type of these contexts and the expected outputs vary across tasks, as summarized in Table 4.1.

The Table QA task, as discussed in Section 2.3, has been well explored in prior work, resulting in several publicly available datasets for evaluation. In contrast, table-based chunk retrieval and Table QA on full documents have received comparatively less attention, and currently no commonly used datasets exist for these tasks.

To evaluate the performance of the vanilla Table QA task, we employ the WikiTableQuestions dataset (Pasupat et al., 2015), a standard benchmark in this domain ensuring comparability with existing studies. For the remaining two tasks, we constructed a custom dataset through manual annotation of U.S. SEC Form 10-K filings from two publicly

Task	**Question Context θ**	**Output**	**Evaluation Dataset**
Table QA	Table t	Answer a_q	WikiTableQuestions
Chunk Retrieval	Document d	Document chunk c	SEC-Filing Tables
Table QA on Full Documents	Document d	Answer a_q	SEC-Filing Tables
Chunk Retrieval on Corpus	*Document corpus D*	*Document chunk c*	*SEC-Filing Tables*
Table QA on Corpus	*Document corpus D*	*Answer a_q*	*SEC-Filing Tables*

Table 4.1.: Summary of the evaluation tasks, highlighting differences in context, output, and corresponding evaluation datasets.

traded companies. The WikiTableQuestions dataset was considered unsuitable for these tasks due to the nature of the Wikipedia articles linked with its tables. These articles typically contain minimal additional content beyond the relevant tables, thereby failing to meet the requirements of a heterogeneous document context that includes both tabular and textual content.

In order to conduct a comprehensive analysis, the questions within both datasets were categorized according to different question types. This categorization was performed using a LLM-based approach for the WikiTablesQuestion dataset (see Figure A.5 for utilized prompt template) and performed manually for the SEC-Filing Tables dataset. The questions were classified into the following categories:

1. **Lookup**: Questions requiring the direct retrieval of a value from a table without additional computational processing.
2. **Advanced Lookup**: Questions necessitating operations such as counting, sorting, ranking (including identification of minimum or maximum values), or basic comparisons between values.
3. **Boolean**: Questions requiring a binary response (e.g., yes/no or true/false).
4. **Calculation**: Questions that involve arithmetic operations, including summation, percentage calculation, subtraction, or other complex mathematical computations.
5. **Position-Related**: Questions concerning the relative positioning of items within a table (e.g., *'Who ranked immediately after Turkey?'*). This category is exclusive to the WikiTableQuestions dataset. In the manually annotated SEC-Filing Tables dataset, the focus is only on content-related questions.

4.1.1. WikiTableQuestions

The WikiTableQuestions dataset (Pasupat et al., 2015) is a widely used benchmark for Table QA, designed to evaluate the ability of models to answer natural language questions using semi-structured tables. It consists of 22,003 manually annotated question-answer pairs, with each question beeing associated with one of a total of 2,108 different tables of a Wikipedia document. As a prerequisite, Pasupat et al. (2015) defined that tables must have at least 8 rows and 5 columns in order to be included in the data set. The data is provided in TSV format, with the corresponding Wikipedia articles and tables also available in HTML format. Additionally, an official evaluator is available to assess performance using execution accuracy (see Section 4.3 for further details). Two illustrative example documents, including their corresponding tables and questions, can be found in the Appendix (Figure A.1 and Figure A.2).

A comprehensive analysis of the WikiTablesQuestion dataset, which comprises 22,003 questions, would require substantial computational resources. Therefore, this thesis adopts a sampling approach, selecting four random samples, each containing 50 questions related to distinct tables. Additionally, some tables within the dataset exhibit malformed HTML structures, including inconsistencies in the number of defined cells within individual rows and columns. Such tables are excluded from our analysis, and only those with fully valid HTML structures are considered.

Table 4.2 presents key metrics for the 200 distinct tables corresponding to the eight selected samples. The table properties reported include the number of rows, number of columns, character count (excluding table structure), and token count as processed by the Tiktoken `o200k_base` tokenizer[7]. Token counts are provided for two representa-

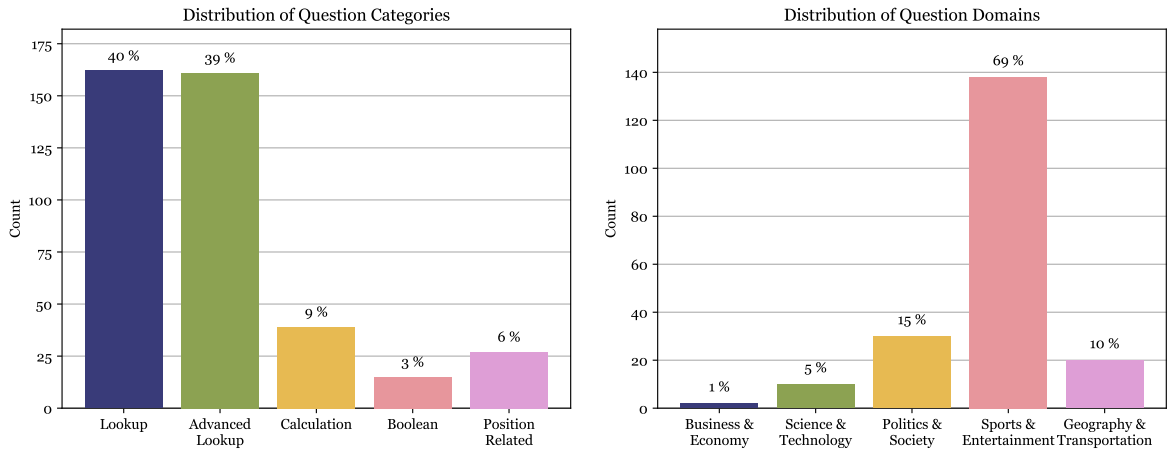
	# Rows	# Columns	# Characters	# Tokens	
				HTML	**TabTree Text**[8]
Avg ± Std	24.4 ± 24.6	6.5 ± 1.9	1,688 ± 2,409	1,513 ± 1,775	7,273 ± 13,573
Minimum[9]	7	3	124	265	385
Maximum	196	17	18,118	14,549	143,478

Table 4.2.: Summary of table characteristics (after preprocessing) for $n = 200$ samples from the WikiTableQuestions dataset (Pasupat et al., 2015).

[7] `https://github.com/openai/tiktoken`, visited on 03/28/2025

[8] Refer to Table 4.5 for the specification of the selected components of the modular system approach utilized in the TabTree approach.

[9] Pasupat et al. (2015) set a prerequisite of at least 8 rows and 5 columns, though these dimensions might be reduced due to preprocessing (see Section 4.2).



(a) Question category distribution (b) Distribution of question / table domains

Figure 4.1.: Distributions of question categories and question / table domains among 200 samples from the WikiTableQuestions dataset

tions: filtered HTML (excluding attributes) and CSV format. Additionally, Figure 4.1 illustrates the distribution of the 200 samples across question categories and their classification into five distinct domains.

4.1.2. SEC-Filing Tables

The SEC Form 10-K is a comprehensive annual report that publicly traded companies in the United States are required to file with the Securities and Exchange Commission (SEC). This report provides detailed information about a company's financial performance, risks, and business operations, and includes both textual descriptions and a series of tables presenting key financial figures. All filings are publicly accessible through the SEC's EDGAR database[10].

To address the challenge of analysing Table QA on full documents, we constructed a dataset based on the SEC Form 10-K filings from 2023 for two companies: American Water Works[11] and Uber[12]. This dataset comprises a total of 310 manually curated questions, of which 209 pertain to American Water Works and 101 to Uber. Each entry in the dataset includes the question, the assigned category, the answer without units, a separate column for units, and a search reference. The search reference consists of a regular expression designed to locate the specific table containing the answer to the

[10] `https://www.sec.gov/edgar/search/`, visited on 03/28/2025

[11] `https://www.sec.gov/Archives/edgar/data/1410636/000141063624000050/awk-20231231.htm`, visited on 03/28/2025

[12] `https://www.sec.gov/Archives/edgar/data/1543151/000154315124000012/uber-20231231.htm`, visited on 03/28/2025

question. Examples of two tables from these documents along with their associated questions are presented in the Appendix in Figure A.3 and Figure A.4. The complete dataset is available on GitHub[13].

Table 4.3 provides a comprehensive summary of the key metrics related to both the general and tabular content of the two documents. The reported metrics encompass the token count for the pre-processed documents (refer to Section 4.2 for explanations of the document preprocessing methodology), the number of tables per document, and statistics regarding the number of rows, columns, and tokens, based on the similar filtered HTML approach (i.e., excluding attributes) as in Table 4.3.

Document	# Doc Tokens	# Tables	Table Statistics				
				# Rows	# Cols	# Tokens (HTML)	# Tokens (TabTree Text)[14]
American Water Works	194,264	114	Avg ± Std	10.6 ± 8.0	9.6 ± 5.6	781.6 ± 680.0	2298.2 ± 2843.0
			Min	1	1	54	74
			Max	46	26	3,490	18,169
Uber	185,120	103	Avg ± Std	10.4 ± 8.1	9.0 ± 5.1	722.6 ± 708.6	1917.9 ± 2023.6
			Min	1	2	63	110
			Max	48	31	3,974	10,124

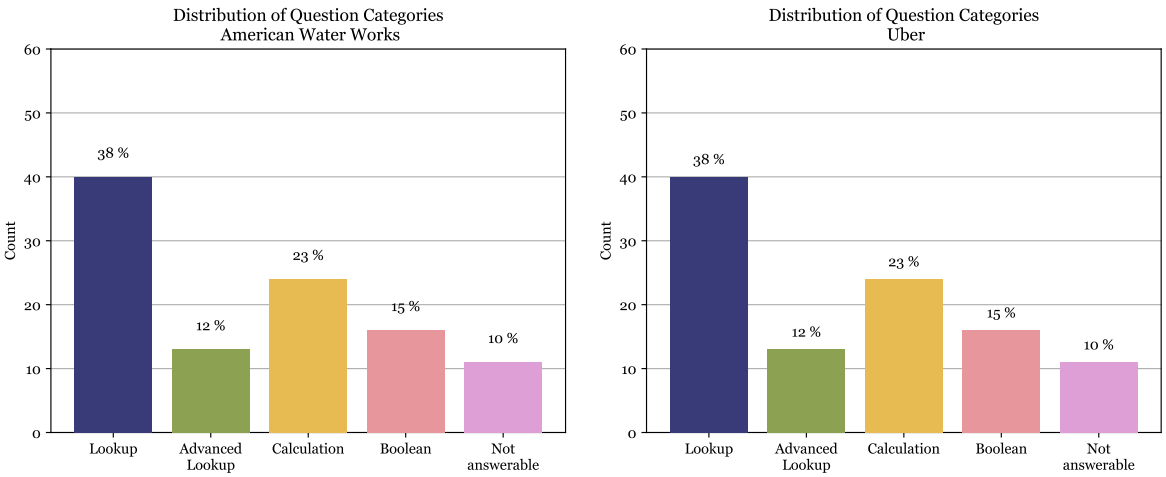
Table 4.3.: Summary of document and table characteristics of the SEC-Filing Tables dataset

Analogous to the WikiTableQuestions dataset, the questions in the SEC-Filing Tables dataset were classified into the previously introduced categories. Notably, position-related questions were intentionally excluded, as the focus of this study is on content-related questions rather than structural aspects. Furthermore, we included 11 negative examples for each of the two documents – questions that cannot be answered unambiguously based on the available information (refer to Section 3.2.3 for a detailed discussion on the criteria for question answerability). In contrast to the WikiTableQuestions dataset, the classification process was performed manually rather than utilizing a LLM-based approach. Figure 4.2 provides an overview of the distribution of question types for the two documents.

It is important to note that the SEC-Filing Tables dataset exhibits an imbalanced distribution of questions – 209 for American Water Works and 101 for Uber. Moreover, the dataset is limited to only two documents. This imbalance and limited scope are

[13] `https://raw.githubusercontent.com/gsindlinger/TabTree-Table-QA-on-Large-Documents/refs/heads/main/SEC_Filing_Tables_Dataset.csv`, visited on 03/28/2025

[14] Refer to Table 4.5 for the specification of the selected components of the modular system approach utilized in the TabTree approach.



(a) American Water Works (b) Uber

Figure 4.2.: Question category distribution for the SEC-Filing Tables dataset

intentional design choices, reflecting the primary objective of this study to concentrate on document-based tasks rather than corpus-based tasks. This design decision implies that the questions associated with American Water Works constitute the principal focus of the analysis, while those pertaining to Uber serve as supplementary data to provide additional insights into findings related to document corpora.

4.2. Implementation Details

This section describes the main components of the framework implemented for this thesis. As outlined in Section 3.1, the implementation is based on a vanilla RAG approach and was developed using Python 3.12. Figure 4.3 provides an overview of the software architecture.

Document Loader and Document Service For the WikiTablesQuestion dataset, an automated data extraction mechanism was developed to scrape information directly from the GitHub repository of Pasupat et al. (2015). In contrast, the SEC-Filing dataset involved manual acquisition of SEC-Filing documents along with a locally stored version of the SEC-Filing Tables questions. To enhance efficiency during iterative analyses, a local storage solution was implemented for both the WikiTablesQuestion and SEC-Filing Tables datasets.

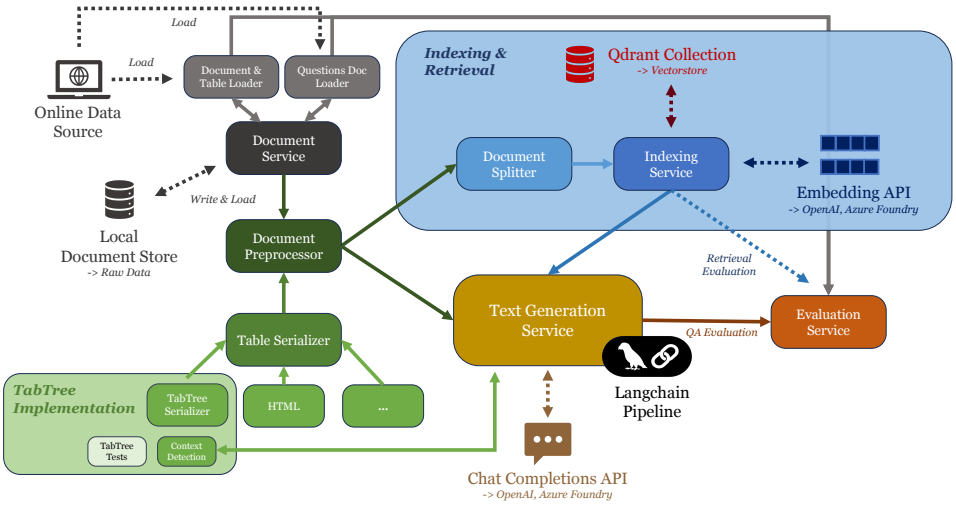


Figure 4.3.: Software architecture of the Table QA Framework

Document Preprocessing Following the data loading process, a systematic preprocessing procedure was implemented to prepare the documents and tables for subsequent evaluation. HTML documents and tables were parsed using the Beautiful Soup library[15], enabling the removal of irrelevant information. Concretely, we discarded all non-visible HTML elements and all HTML attributes. Additionally, design-centric HTML tags[16] were unwrapped. In contrast, structural HTML tags such as headings, list items, and particularly table structures were preserved.

Table Serialization Following the preprocessing phase, the document tables were subjected to a systematic table serialization procedure. For all baseline approaches (including HTML, JSON, and others), this process was executed utilizing the Pandas library[17]. In contrast, a distinct methodology was applied for the TabTree approach, as elaborated in the subsequent paragraph. During the serialization process, redundant data, including empty and duplicate rows and columns, was removed. The refined tables were subsequently converted into serialized strings through Pandas serialization methods, such as `to_html`, `to_csv`, and `to_json`.

TabTree Implementation To establish the TabTree model, tables were, contrary to the approaches for the baselines, first parsed into a custom tabular data structure, yielding a better capturing of column spans and rowspans. Subsequently, column headers

[15] `https://www.crummy.com/software/BeautifulSoup/bs4/doc/`, visited on 03/28/2025

[16] Tags unwrapped: `span, a, b, strong, i, em, u, small, mark, abbr, acronym, del, ins, code, div`

[17] `https://pandas.pydata.org/`, visited on 03/28/2025

and row labels for each table were determined using the 2-Range context detection approach described in Section 3.3.3 and the rather small *GPT-4o-mini* (OpenAI, 2024a) model[18]. The determined headers were stored locally to minimize the evaluation time across various subtasks associated with Table QA on full documents. Detailed prompts used for context detection are provided in the Appendix (see Figure A.12, Figure A.13 and Figure A.14).

Leveraging the column headers and row labels, the TabTree model, along with the corresponding serializations, was generated following the outlined algorithms. For the construction and processing of the multiple trees in the TabTree framework, the NetworkX library[19] is used. The quality of the TabTree method's implementation was ensured through unit testing, achieving a test coverage of 86 % for TabTree related code.

Indexing & Retrieval To facilitate the retrieval-based tasks of Chunk Retrieval and Table QA on Full Documents, an indexing and retrieval pipeline was developed. This pipeline initially segments the preprocessed documents into manageable chunks, subsequently generates embeddings via external API access, and stores these embeddings in a local Qdrant vectorstore (encapsulated within a Docker container). For generating embeddings, we evaluated the OpenAI's *text-embedding-3-small* model (OpenAI, 2024c) with embedding dimension of 1,536 and maximum input token length of 8,191.

For document chunking, a semantic splitting strategy was employed, as initially proposed by Kamradt (2024). Building upon this approach, our chunking methodology was structured as follows:

1. **Sentence Splitting**: The document (excluding tables) was segmented into individual sentences using the sentence tokenizer provided by the NLTK framework[20]. Irrelevant information, such as page numbers, was excluded by retaining only sentences with a character length of ≥ 3.

2. **Table Integration**: All serialized tables were incorporated into the list of sentences, employing different strategies based on the chosen serialization approach:

    - **Baseline Approaches (HTML, JSON, etc.)**: Entire tables were treated as single sentences to prevent intra-table splits. While this approach pre-

[18]Based on an evaluation of three context detection methods, Full Table, 1-Range, and 2-Range, using $n = 21$ manually created samples, the 2-Range approach demonstrated the best performance. The results of this analysis are presented in Table A.1

[19]`https://networkx.org/`, visited on 03/28/2025

[20]`https://www.nltk.org/`, visited on 03/28/2025

served table integrity, it occasionally led to chunks exceeding the maximum input length, requiring subsequent greedy splitting of the serialized tables.

- **TabTree Serialization**: When the TabTree serialization output contained fewer than 18,000 characters[21], the table was treated as a single sentence. If this threshold is exceeded, the NLTK sentence tokenizer was applied to the serialized tables, leveraging the inherent sentence-based structure defined by the TabTree method. Initially, we also explored the application of the NLTK sentence tokenizer to all TabTree-serialized tables. However, this approach was ultimately abandoned due to suboptimal performance.
- **Optional Context Augmentation**: To enhance contextual understanding, the sentence immediately preceding each table was optionally prepended to the serialized table.
- **Table Summarization**: As an alternative strategy, we explored the use of LLMs to generate concise summaries of tables. These summaries are subsequently employed in place of the full serialized tables for the indexing- and retrieval-related tasks[22].

3. **Semantic Chunking**: The `SemanticChunker` implementation from the Langchain framework[23], combined with the selected embedding model, was utilized to perform semantic chunking. We adopted a 0.93-percentile breakpoint threshold to identify semantically coherent chunk boundaries, and any resulting chunk exceeding 20,000 characters was split greedily at the nearest available boundary.

   This approach occasionally caused undesired splits within tables (intra-tabular) and between semantically related tables. To mitigate this, we stored all *related tables* alongside the split chunks and systematically reintroduced them during retrieval and answer generation as optional additional context.

For the retrieval stage, a standard dense retrieval approach was employed, utilizing cosine similarity as distance metric. The top k document chunks were extracted for $k = 1, 3, 5$, each subject to a minimum similarity score threshold of 0.5 to ensure relevance in the retrieval results. We have not chosen a larger k for the retrieval because in a

[21] The maximum token input length of the embedding model used, *text-embedding-3-small*, is 8,191 tokens. Based upon the rule of thumb of 1 token $\approx$ 4 characters (see `https://platform.openai.com/tokenizer`, visited on 03/28/2025), this corresponds to approximately 24,000 characters. The applied threshold of 18,000 characters represents a trade-off between maximizing token utilization and ensuring compliance with embedding model token window constraints.

[22] See Figure A.9 for utilized prompt template.

[23] `https://python.langchain.com/docs/introduction/`, visited on 03/28/2025

RAG setting, in which the selected chunks are entered into an LLM via the prompt, the context would otherwise be too large or excessive.

Text Generation The answer generation for the various tasks addressed in this thesis was implemented using a LLM pipeline based on the Langchain framework[23]. Distinct prompt templates were designed for the different text generation tasks for both the vanilla Table QA task and the Table QA task on full documents. For the latter, two variants were employed: one incorporating related tables retrieved alongside the main chunk, and one excluding them. All prompts were designed following a zero-shot prompting strategy combined with chain-of-thought reasoning, which was facilitated by incorporating the directive 'Think step-by-step' into the prompt. The full text of the three distinct prompts is provided in the Appendix in Figure A.6 to Figure A.8.

The experimental evaluation was conducted using four different LLMs, the specifications and relevant characteristics of which are summarized in Table 4.4. The OpenAI models were accessed via the OpenAI API[24], whereas the remaining models were accessed through the Inference API provided by the Microsoft Azure Foundry Hub[25]. For all models, the temperature parameter was set to 0 to ensure stable output, while all other parameters were retained at their default settings.

Model	**Provider**	**Parameters (in Billion)**	**Context Window (# Tokens)**
GPT-4o (OpenAI, 2024b)	OpenAI	*~1,800*[26] *(Mixture Of Experts)*	128k
GPT-4o-mini (OpenAI, 2024a)	OpenAI	*~8*	128k
Llama 3.3 70B Instruct (Grattafiori et al., 2024)[27]	Meta AI	70	128k
Phi 4 (Abdin et al., 2024)	Microsoft	14	16k

Table 4.4.: LLM Specifications

[24] `https://platform.openai.com/`, visited on 03/28/2025

[25] `https://ai.azure.com/`, visited on 03/28/2025

[26] The model specifications for OpenAI models are proprietary; the figures presented are based on publicly available estimates (refer to Josh Howarth (2024) for further details).

[27] Llama 3.3 Model Card: `https://github.com/meta-llama/llama-models/blob/main/models/llama3_3/MODEL_CARD.md`, visited on 03/28/2025

4.3. Evaluation Setup

The evaluation of the Table QA framework developed in this thesis was conducted using a set of metrics tailored to the different characteristics of the retrieval and answer generation subtasks. The following section provides an explanation of the selected evaluation metrics, including the reasoning behind their selection. In addition, the baselines against which the proposed TabTree model was compared are presented.

Retrieval Metrics As discussed, the SEC-Filing Tables dataset employed for retrieval-based tasks was constructed to include a search reference allowing for retrieving relevant tables within the original document for given queries. It is important to note that the assignment of a search reference to a table is not always unique, due to the presence of fairly similar tables within the SEC filing documents.

Despite this ambiguity, the evaluation focuses on a binary query scenario, where the goal is to determine whether the search reference can be successfully retrieved at least once among the top k most similar document chunks for a given query. Given this setting, the performance of the retrieval system was evaluated using the following metrics:

- **Recall@k**: Reflects a binary measurement whether a relevant chunk is retrieved within the top k chunks. It is calculated as:

$$R@k = \frac{\text{Number of queries with a relevant chunk in the top } k \text{ results}}{\text{Total number of queries}}$$

  In this binary retrieval system, the definition of Recall@k corresponds to the **Hit Rate** metric.

- **Mean Reciprocal Rank@k (MRR@k)**: MRR focuses on the position of the first relevant chunk among the top k chunks. It is calculated as:

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

  where N is the amount of all queries, rank_i is the rank of the first relevant chunk for the i-th query, and $\mathbb{I}(\cdot)$ is an indicator function that is 1 if the relevant chunk is within the top k positions, otherwise 0.

Other metrics such as Precision, F1-score, Mean Average Precision (mAP), and Normalized Discounted Cumulative Gain (nDCG) are frequently applied in RAG applications

(Y. Gao et al., 2024; Nguyen, 2023)), but not suitable for the described binary retrieval setting.

Table QA Metrics For the evaluation of the question answering capabilities of the Table QA framework in this work, we focused on accuracy-based metrics rather than text-based metrics such as BLEU, ROUGE, or embedding-based strategies like BERTScore. This choice is justified by the nature of the two datasets used: the WikiTablesQuestion and the SEC-Filing Tables datasets, which contain fixed-form answers (i.e., categorical and numerical) rather than open-ended text responses. This approach aligns with the status quo in the literature, as discussed by Pasupat et al. (2015).

We used the following two metrics:

- **Execution Accuracy**: This metric compares the predicted answer of a question with the ground truth after applying a standardizing procedure (e.g., answer parsing) to both. Since answers may differ in formatting, such as upper or lower case, or different data types for numbers, a direct Exact Match accuracy could produce misleading results. Therefore, we adopt the evaluation methodology of the WikiTableQuestions dataset (Pasupat et al., 2015).
- **(Macro) F1-score**: In the question answering setting, the F1-score is calculated as the harmonic mean of precision and recall based on the word overlap between the ground truth response and the predicted answer. The overall F1-score is then calculated as the average over all data samples. For this metric, we apply basic parsing (i.e., converting all characters to lowercase).

Evaluated Approaches and Baselines Our experimental analysis encompassed a variety of implementations of the TabTree model as well as multiple baselines. To identify the most effective approaches for the context string and value string, we evaluated five distinct combinations, as summarized in Table 4.5. Full table serializations for the sample table of Figure 3.5 for each combination using the primary subtree selection approach *Column Header Tree* can be found in the Appendix in Figure A.15 to Figure A.19.

The selection of the **Primary Subtree** was performed independently across the four options (refer to Section 3.3.2 for details): *Column Header Tree* as primary subtree, *Row Label Tree* as primary subtree, *Concatenated* representation, and *Heuristic*-based selection.

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-Augmentation w/ Context-Intersection	Text-Augmentation	✅	Text-Augmentation	✅
Context-Empty	Empty	❌	Text-Augmentation	✅

Table 4.5.: Overview of the evaluated combinations of context string and value string approaches for the TabTree model.

In accordance with current research in the field, we benchmark the performance of the TabTree model against the following baseline serialization formats: **HTML, JSON Records**[28], **CSV**, and **Markdown**.

4.4. Evaluation Results

This section presents the results of the evaluation conducted for the different components of the proposed Table QA framework. The initial focus is on the vanilla Table QA task, followed by an analysis of Chunk Retrieval and Table QA on Full Documents. For each subtask, we start by evaluating the optimal configuration of the modular components of the TabTree system (refer to Figure 3.17). These configurations are then benchmarked against the baseline methods.

4.4.1. Table QA

TabTree Primary Subtree Approaches Table 4.6 reports the performance of the four different strategies for primary subtree selection within the TabTree model (see Section 3.3.2) using the *Text* approach for context and value string generation, evaluated on the WikiTableQuestions dataset using two LLMs: *GPT-4o-mini* and *Phi-4*. The

[28]The serialization method JSON Records refers to the Pandas `to_json` method and its parameter `orient`, which specifies the JSON formatting strategy. See `https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.to_json.html` (visited on 03/28/2025) for details.

comparison includes execution accuracy, F1-score, and error count, i.e., number of serializations exceeding the model's context window. Token count statistics based on the `o200k_base` tokenizer are also included as reference for estimating computational cost.

From Table 4.6 we observe, that the *Concatenate* approach yields the highest execution accuracy (0.65 ± 0.03) and F1-score (0.69 ± 0.03) for *GPT-4o-mini*, although at the cost of high average token length ($13{,}370 \pm 24{,}501$) and elevated error count of 3 for *GPT-4o-mini* and 38 for *Phi-4*. This makes it impractical for real-world applications, particularly with models that have a smaller context window. For *Phi-4*, the *Column Header Tree* approach performs best with an execution accuracy of 0.60 ± 0.02 and F1-score of 0.64 ± 0.01, likely benefiting from its rather condensed serialization ($4{,}479 \pm 7{,}283$ tokens) and correspondingly lower error count of 8.

Overall, all approaches perform within a narrow margin, ranging from 0.53 to 0.62 (*GPT-4o-mini*) and 0.49 to 0.60 (*Phi-4*) in terms of execution accuracy. The *Heuristic* approach provides a strong compromise, ranking consistently second-best across both LLMs while maintaining a moderate token count. As a result, it is adopted as the default primary subtree selection method for further TabTree evaluations.

Approach	**GPT-4o-mini**			**Phi-4**			**Token Count**
	Execution Acc.	**F1-score**	**Error Count**[29]	**Execution Acc.**	**F1-score**	**Error Count**	
Heuristic	0.62 ± 0.06	0.66 ± 0.04	0	0.55 ± 0.05	0.60 ± 0.06	22	$8{,}120 \pm 16{,}965$
Column Header Tree	0.60 ± 0.07	0.65 ± 0.07	0	$\mathbf{0.60 \pm 0.02}$	$\mathbf{0.64 \pm 0.01}$	8	$4{,}479 \pm 7{,}283$
Row Label Tree	0.53 ± 0.02	0.57 ± 0.02	0	0.53 ± 0.06	0.57 ± 0.07	24	$8{,}835 \pm 18{,}275$
Concatenate	$\mathbf{0.65 \pm 0.03}$	$\mathbf{0.69 \pm 0.03}$	3	0.49 ± 0.09	0.54 ± 0.08	38	$13{,}370 \pm 24{,}501$

Table 4.6.: Vanilla Table QA results (Average ± Std) for TabTree primary subtree selection on WikiTableQuestions (4 samples, 50 questions each) using *GPT-4o-mini* and *Phi-4* models.

TabTree Context- & Value-String Approaches Building on the *Heuristic* strategy for primary subtree selection, the next evaluation examines different combinations of context string and value string generation within the TabTree model, as shown in Table 4.7. The same metrics are reported as in the previous analysis.

[29]Error Count refers to the amount of table serializations which exceeded the LLM context window size.

As shown in Table 4.7, the *Text* and *Text-Augmented with Context-Intersection* approaches emerge as the most effective TabTree approaches when using *GPT-4o-mini*, both achieving an average execution accuracy of 0.62. For *Phi-4*, the *Context Empty* strategy performs best, with an average execution accuracy of 0.58, likely due to its substantially reduced token count (3,745 ± 6,809) and lower occurrence of LLM context window overflows of only 2.

Across both models, all *Text*-based strategies outperform the *Base* approach, which achieves the lowest execution accuracy for both models (0.50 for *GPT-4o-mini* and 0.48 for *Phi-4*). Among the *Text*-based variants, performance differences are minor (within ± 0.02 execution accuracy), suggesting no significant advantage of the advanced methods of the TabTree model, i.e., incorporating context-intersection nodes or using text-augmentation for value string generation.

TabTree Approach	GPT-4o-mini			Phi-4			Token Count
	Execution Acc.	F1-score	Error Count	Execution Acc.	F1-score	Error Count	
Base	0.50 ± 0.07	0.52 ± 0.09	0	0.48 ± 0.09	0.52 ± 0.09	0	1,031 ± 1,459
Text	**0.62 ± 0.06**	0.64 ± 0.04	0	0.55 ± 0.05	0.60 ± 0.06	22	8,120 ± 16,965
Text w/ Context-I.	0.60 ± 0.06	0.65 ± 0.05	0	0.56 ± 0.09	0.58 ± 0.09	23	8,882 ± 18,003
Text-Augm. w/ Context-I.	**0.62 ± 0.06**	**0.66 ± 0.06**	0	0.57 ± 0.08	0.59 ± 0.08	18	8,232 ± 17,825
Context Empty	0.57 ± 0.03	0.60 ± 0.04	0	**0.58 ± 0.05**	**0.62 ± 0.04**	2	3,745 ± 6,809

Table 4.7.: Vanilla Table QA results (Average ± Std) for TabTree modular system combinations on WikiTableQuestions (4 samples, 50 questions each) using the *Heuristic* primary subtree selection approach and *GPT-4o-mini* and *Phi-4* models.

Comparison with Baselines To contextualize the performance of the TabTree approach, its best-performing configuration, is compared against the baselines *HTML*, *CSV*, *JSON Records*, and *Markdown*. The evaluation was conducted across multiple LLMs (see Table 4.4) on 4 samples of 50 questions of the WikiTableQuestion dataset each. Figure 4.4 presents the average execution accuracy for each combination of serialization format and LLM, along with corresponding standard deviations. Comprehensive numerical results, including token count statistics for each format, are provided in the Appendix (Table A.2).

Across all LLMs, the baseline serializations consistently outperform the TabTree approach, with a minimum margin of 0.09 in execution accuracy. Among the baselines, *HTML* yields the highest accuracy across the board, achieving 0.71 (*Llama 3.3 70B Instruct*), 0.72 (*Phi-4*), 0.71 (*GPT-4o-mini*), and 0.76 (*GPT-4o*). However, the differences among the baseline methods themselves remain modest: for each LLM, the maximum performance gap between baselines does not exceed 0.06.

Figure 4.4 also reveals the influence of different LLMs on Table QA performance. *GPT-4o* consistently achieves the highest accuracy across all serialization approaches, outperforming other models by at least 0.05 in execution accuracy. Meanwhile, *Llama 3.3 70B Instruct*, *Phi-4*, and *GPT-4o-mini* perform within a narrow range of 0.66 to 0.71. Notably, smaller models such as *Phi-4* and *GPT-4o-mini* demonstrate competitive accuracy while offering lower inference costs, making them well-suited for deployment in resource-constrained scenarios. These findings align with prior work, which similarly observed that baseline formats, particularly HTML, tend to outperform sentence-based representations in Table QA tasks.

Beyond assessing effectiveness, it is also important to consider the computational cost associated with each method. The baseline serializations remain relatively lightweight, with average token counts consistently below 1,560 (see Table A.2). In contrast, the TabTree approach produces significantly larger representations, averaging 8,232 tokens.

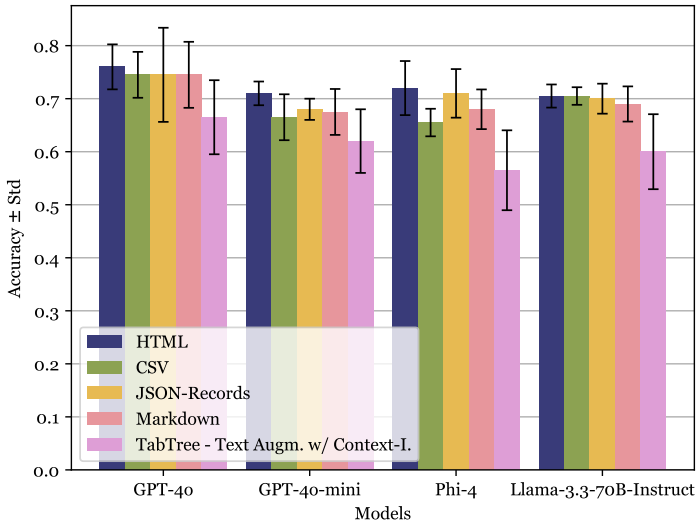


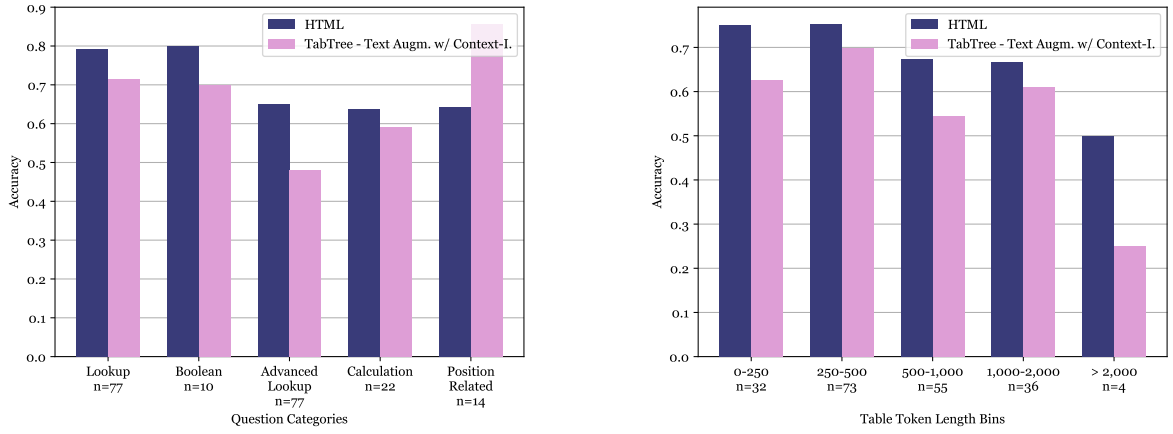
Figure 4.4.: LLM Model Comparison on Vanilla Table QA for the baseline approaches *HTML*, *CSV*, *JSON Records*, *Markdown* and the best performing TabTree approach *Text Augmentation with Context-Intersection* on WikiTableQuestions (4 samples, 50 questions each).

This increased computational cost, coupled with its lower effectiveness relative to baseline methods, limits the practicality of the TabTree approach.

Question Category & Table Token Length Analysis To conclude the evaluation of the vanilla Table QA task, we analyze the performance of LLMs across different question categories and varying table lengths. Figure 4.5 reports execution accuracy segmented by question type and table token count. Results are presented for both the best-performing baseline format (*HTML*) and the best TabTree configuration (*Text-Augmentation with Context-Intersection*), using the *GPT-4o-mini* model.

As shown in Figure 4.5, execution accuracy varies substantially across question categories. Simple *lookup* and *boolean* questions are answered most reliably with an execution accuracy of 0.79 and 0.80 for *HTML*, and 0.71 and 0.70 for the TabTree approach, respectively. In contrast, more complex questions, such as *advanced lookup* and *calculation*, show reduced performance, with execution accuracies of 0.65 and 0.64 for *HTML*, and 0.48 and 0.59 for the TabTree approach. Notably, TabTree performs well on *position-based* questions with an execution accuracy of 0.86, likely benefiting from its explicit inclusion of row and column indices (see Section 3.3.2). In contrast, *HTML* achieves an execution accuracy of only 0.64 in this category.

Regarding table token length, Figure 4.5 shows a performance degradation for *HTML* as table size increases. For shorter tables (less than 500 tokens), the execution accuracy is



(a) Question Category Analysis

(b) Table Token Length Analysis

Figure 4.5.: Table QA results by Question Category and Table Token Length for the approaches *HTML* and the best performing TabTree approach *Text Augmentation with Context-Intersection* on WikiTableQuestions (4 samples, 50 questions each).

0.75, decreasing to 0.67 for medium-sized tables (500 to 2,000 tokens). Due to the small number of samples ($n = 4$), results for tables exceeding 2,000 tokens are not considered reliable. For the TabTree approach, no consistent pattern emerges with respect to table length. Execution accuracy varies between table sizes, indicating possible robustness to variation in table size.

4.4.2. Chunk Retrieval

For the Chunk Retrieval task, the SEC-Filing Tables dataset was utilized, excluding all non-answerable questions, i.e., questions that do not correspond to a unique, retrievable table.

Methodology The evaluation considers two distinct matching strategies to search for ground truth data within retrieved content: *Match within Chunk*, and *Match in Related Table*. This distinction arises due to the nature of the chunking process employed during document splitting. Specifically, tables that should semantically belong to a chunk may be separated if the chunk length limit of 18,000 characters is exceeded. To mitigate this, we maintain metadata linking each chunk to potentially corresponding relevant tables. In the *Match in Related Table* method, the retrieval assessment is performed not only over the chunk itself, but also over all related tables. This method is thus more exhaustive and yields higher recall than the *Match within Chunk* approach.

Additionally, each serialization strategy is evaluated with and without *optional context augmentation*, which involves including preceding textual sentences before tables during the embedding generation phase.

Retrieval performance is measured using chunk recall at top-k for $k \in \{1, 3, 5\}$, and MRR at $k = 5$. We limit the retrieval parameter to $k \leq 5$ due to the substantial size of individual chunks[30]. For $k = 3$, the resulting retrieval context already partially exceeds the context window of the utilized LLMs when including it into the downstream text generation step of the RAG pipeline (see Table 4.12).

TabTree Primary Subtree Approaches As in the previous section, we begin the Chunk Retrieval analysis by evaluating the effect of different primary subtree selection strategies: Table 4.8 presents the results of the four different primary subtree strategies

[30]The median number of chunks per document across all analysed approaches (TabTree and Baselines) is ≈ 181 and therefore most chunks contain roughly $\pm$ 1,000 tokens (excluding related tables).

of the TabTree model, using both the *Base* and *Text* context and value string generation approaches (see Table 4.5). For this analysis, we use the *Match within Chunk* method and omit preceding sentence context during embedding generation, i.e., not employing context augmentation.

The results reveal that, in the TabTree *Base* configuration, the *Heuristic* and *Column Header Tree* strategies yield the strongest performance, with MRR@5 scores of 0.43 and 0.42, respectively. The *Row Label Tree* approach follows closely at 0.40, while the *Concatenate* strategy performs significantly worse, with an MRR@5 of 0.29. These findings suggest that, aside from the *Concatenate* variant, the choice of primary subtree approach has only a moderate impact on retrieval performance within the TabTree Base setting.

In contrast, retrieval effectiveness for the TabTree *Text* configuration is substantially lower across all primary subtree approaches. The highest MRR@5 achieved is 0.14 with the *Row Label Tree* approach, followed by 0.11 with the *Heuristic* strategy.

Given that the *Heuristic* approach performs best for the TabTree Base setting and second-best for the TabTree Text setting, we select it as the default primary subtree selection strategy for subsequent retrieval evaluations – similarly to the vanilla Table QA task.

Primary Subtree Approach	**TabTree - Base**				**TabTree - Text**			
	R@1	**R@3**	**R@5**	**MRR@5**	**R@1**	**R@3**	**R@5**	**MRR@5**
Heuristic	**0.27**	0.57	**0.71**	**0.43**	0.06	0.14	0.22	0.11
Column Header Tree	0.26	**0.58**	0.70	0.42	0.05	0.11	0.20	0.09
Row Label Tree	0.25	0.52	0.66	0.40	**0.07**	0.20	**0.27**	**0.14**
Concatenate	0.14	0.42	0.57	0.29	0.02	0.12	0.18	0.08

Table 4.8.: Retrieval results for TabTree primary subtree selection on SEC-Filing tables excluding non-answerable questions ($n = 288$) on match within related table method.

TabTree Context- & Value-String Approaches Following the selection of the *Heuristic* strategy as the primary subtree approach, we evaluate the various combinations of context string and value string generation within the TabTree model, as defined in Table 4.5. Table 4.9 presents retrieval results across these combinations for both evaluation strategies, *Match within Chunk* and *Match within Related Table*, as well as for setups that either include or exclude preceding sentences during embedding generation.

The impact of the latter is illustrated in Figure 4.6, visualizing Recall@1, Recall@3, and Recall@5 for selected TabTree variants – specifically, *Base*, *Text-Augmentation with Context-Intersection*, and *Context Empty* – under both settings with and without inclusion of preceding sentence for embedding creation using *Match within Related Table* setup.

A few key trends emerge: Most notably, the *TabTree Base* configuration outperforms all other approaches in both evaluation settings, achieving an MRR@5 of 0.43 (*Match within Chunk*) and 0.46 (*Match within Related Table*), and Recall@5 scores of 0.71 and 0.74, respectively. In the more permissive *Match within Related Table* setting, the four

TabTree Approach	**w/ prec. sent.?**	**Match Within Chunk**				**Match Within Related Table**			
		R@1	**R@3**	**R@5**	**MRR@5**	**R@1**	**R@3**	**R@5**	**MRR@5**
Base	❌	**0.27**	**0.57**	**0.71**	**0.43**	**0.30**	**0.62**	**0.74**	**0.46**
	✅	0.19	0.49	0.64	0.35	0.21	0.55	0.67	0.38
Text	❌	0.06	0.14	0.22	0.11	0.09	0.27	0.40	0.19
	✅	0.10	0.22	0.27	0.16	0.19	0.51	0.62	0.35
Text w/ Context-I.	❌	0.05	0.13	0.20	0.10	0.09	0.26	0.39	0.19
	✅	0.10	0.22	0.27	0.16	0.20	0.53	0.66	0.37
Text-Aug. w/ Context-I.	❌	0.08	0.21	0.27	0.15	0.12	0.33	0.43	0.24
	✅	0.09	0.24	0.34	0.18	0.18	0.48	0.66	0.35
Context Empty	❌	0.09	0.25	0.36	0.18	0.15	0.35	0.48	0.26
	✅	0.10	0.30	0.40	0.21	0.20	0.55	0.72	0.39

Table 4.9.: Retrieval results for TabTree modular system combinations on SEC-Filing Tables excluding non-answerable questions ($n = 288$) using the heuristic primary subtree selection approach.

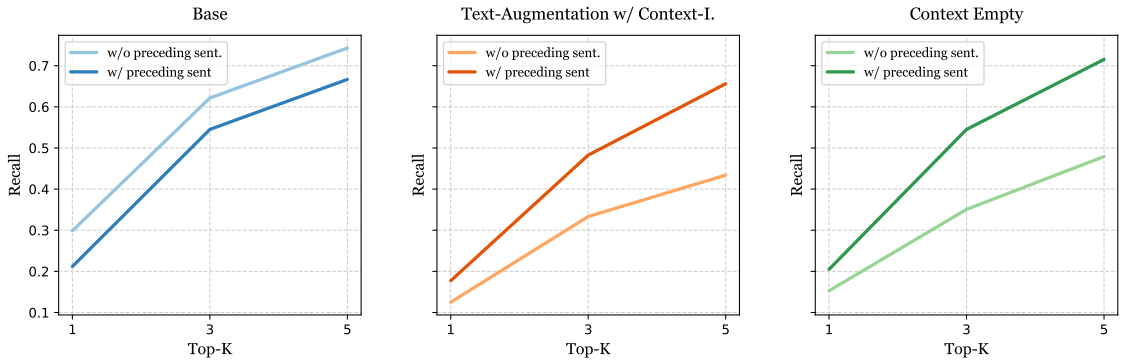


Figure 4.6.: Comparison of retrieval results for match within related tables for approaches with and without including preceding sentence during indexing of tables for top performing TabTree approaches on SEC-Filing Tables excluding non-answerable questions ($n = 288$).

other text-based approaches variants come close to *TabTree Base*, with Recall@5 values between 0.62 and 0.72.

Among the text-based approaches, the choice of evaluation strategy has a considerable impact. Switching from *Match within Chunk* to *Match within Related Table* yields MRR@5 gains of at least 0.08 and up to 0.21. For example, the *Text with Context-Intersection* approach with preceding sentence inclusion improves from an MRR@5 of 0.16 to 0.37. This performance gap is likely due to the higher token counts associated with text-based serializations (see Table 4.7), which lead to more semantic disrupted chunks. Consequently, including related tables during retrieval appears essential for reliable performance in RAG pipelines using these approaches.

Another important factor is the inclusion of preceding sentence context during embedding generation. For text-based TabTree variants, this addition substantially improves retrieval performance, whereas it has the opposite effect on the TabTree *Base* configuration, as shown in Figure 4.6. For both *Text-Augmentation with Context-Intersection* and *Context Empty*, Recall@k improves across all $k \in \{1, 3, 5\}$ when predeceding sentences are included. The difference is particularly notable for Recall@5, which increases by 0.23 and 0.24, respectively. In contrast, for TabTree *Base*, the inclusion of the preceding sentence in front of tables for embedding generation results in a drop in Recall@5 from 0.74 to 0.67.

Comparison with Baselines Table 4.10 presents the retrieval results for various baseline serialization formats, each evaluated both with and without the inclusion of preceding sentences during embedding generation. For reference, results for the best-performing TabTree variant *Base* are also included. Additionally, the performance of an approach that uses table summaries instead of full tables for embedding generation, while retaining the raw table data as related tables, is reported.

Across all baselines, performance is fairly comparable: In the *Match within Chunk* setting, MRR@5 scores range from 0.29 to 0.35, while Recall@5 scores span 0.51 to 0.61. Under the *Match within Related Table* setting, MRR@5 improves to between 0.34 and 0.48, and Recall@5 rises to between 0.63 and 0.70. Among the baselines, Markdown performs slightly better than others, achieving the highest MRR@5 scores for the related table setting – 0.46 with and 0.48 without preceding sentence inclusion.

Notably, the inclusion of preceding sentences during indexing has minimal effect on retrieval performance for baseline methods. In both evaluation settings, the change in

MRR@5 due to context inclusion does not exceed 0.04, suggesting that this augmentation offers limited benefit for these formats.

In comparison, the *TabTree Base* approach without preceding sentence inclusion outperforms all baseline approaches by a notable margin under the *Match within Chunk* setting. With an MRR@5 of 0.43, it exceeds the best-performing baseline JSON Records without preceding sentence by 0.08. It also performs among the best under the *Match within Related Table* strategy, achieving the highest Recall@5 of all approaches of 0.74, which is 0.04 points higher than the best baselines (*Markdown*, *HTML*, and *JSON Records*).

Finally, the structurally different *Table Summary* approach achieves slightly better retrieval performance than the baselines, with a Recall@5 of 0.71 and an MRR@5 of 0.47. While it ranks just behind *TabTree Base* in terms of Recall@5, this improvement comes at the cost of requiring an additional LLM call to generate a summary for each table prior to indexing, making it more resource-intensive in practice.

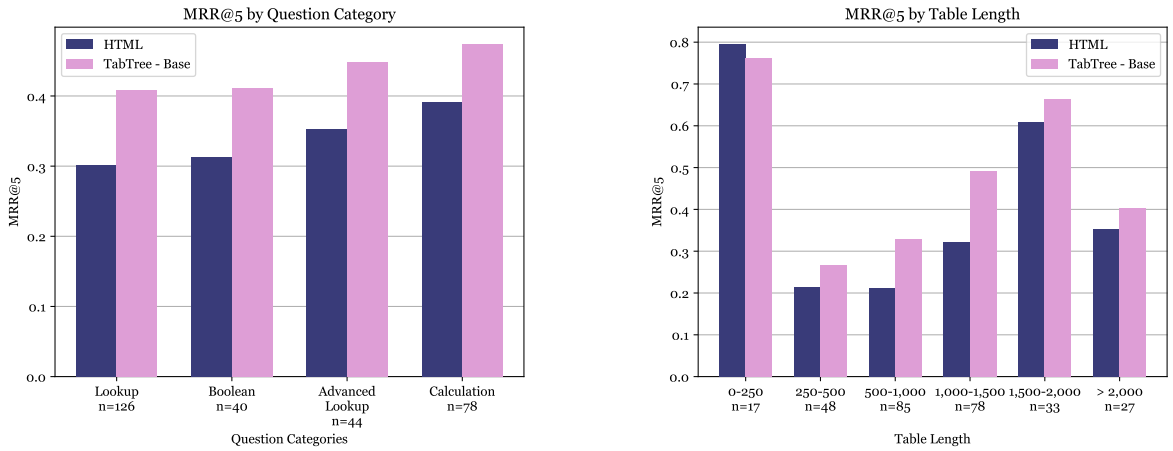
Approach	**w/ prec. sentence?**	**Match Within Chunk**				**Match Within Related Table**			
		R@1	**R@3**	**R@5**	**MRR@5**	**R@1**	**R@3**	**R@5**	**MRR@5**
HTML	❌	0.19	0.46	0.57	0.33	0.20	0.52	0.66	0.37
	✅	0.15	0.41	0.55	0.29	0.17	0.53	**0.70**	0.36
CSV	❌	0.18	**0.51**	0.58	0.34	0.19	**0.57**	0.64	0.37
	✅	0.14	0.46	0.57	0.30	0.15	0.52	0.63	0.34
JSON Records	❌	0.20	0.47	**0.61**	**0.35**	0.22	0.54	**0.70**	0.39
	✅	0.16	0.45	0.57	0.31	0.17	0.54	0.68	0.36
Markdown	❌	0.19	0.37	0.52	0.30	0.33	0.56	**0.70**	0.46
	✅	**0.21**	0.36	0.51	0.30	**0.36**	0.55	0.69	**0.48**
TabTree - Base	❌	**0.27**	**0.57**	**0.71**	**0.43**	0.30	**0.62**	**0.74**	0.46
	✅	0.19	0.49	**0.64**	0.35	0.21	0.55	0.67	0.38
Table Summary	❌	–	–	–	–	0.32	**0.63**	**0.71**	0.47

Table 4.10.: Retrieval results for baseline approaches, best performing TabTree approach and the Table Summary approach with and without including preceding sentence during indexing of tables on SEC-Filing Tables excluding non-answerable questions ($n = 288$).

Question Category & Table Token Length Analysis As with the vanilla Table QA task, we conclude the retrieval evaluation with an analysis of performance across different question types and table token lengths. Figure 4.7 shows MRR@5 results, segmented by question category and table size, for the best-performing baseline (*HTML*) and the best TabTree configuration (*Base*).

With respect to question categories, we find, that retrieval performance appears to increase with question complexity: For both *HTML* and TabTree Base, the lowest MRR@5 scores are observed for *lookup* questions (0.30 for *HTML*, 0.41 for *TabTree Base*), followed by *boolean* (0.31 and 0.42), then *advanced lookup* (0.35 and 0.44), and the highest scores for *calculation* questions (0.39 and 0.47). This trend may reflect the more extensive semantic cues in more complex questions, while simpler questions as lookup questions are semantically more difficult to distinguish.

In contrast, the analysis of table token length does not reveal a consistent trend. The highest MRR@5 scores are observed for short tables (up to 250 tokens), with 0.80 for *HTML* and 0.76 for TabTree Base. This is followed by a sharp drop in the 250-500 token range (0.21 for *HTML*, 0.27 for *TabTree Base*) and a secondary peak in the 1,500-2,000 token range. Overall, the most reliable finding is that very short tables are retrieved with a fairly high effectiveness.



(a) Question Category Analysis

(b) Table Token Length Analysis

Figure 4.7.: Retrieval results by Question Category and Table Token Length for the approaches *HTML* and the best performing TabTree approach *Base* on SEC-Filing Tables excluding non-answerable questions ($n = 288$).

4.4.3. Table QA on Full Documents

Finally, we present the results for the Table QA task on full documents using the SEC-Filing dataset. This task combines the retrieval and answer generation components of the proposed RAG framework and integrates the configurations established in the previous subtasks. Specifically, the top-$k = 3$ chunks are retrieved per question, and performance is measured using the same metrics as for the vanilla Table QA task: execution accuracy and F1-score. Additionally, we report the number of *window exceed errors*,

which indicate how often the retrieved content, together with the prompt, surpassed the models context window[31].

Chunk-only Results We begin by evaluating the setup in which only the retrieved chunks are used as LLM input, excluding related tables. Table 4.11 presents results for the two baseline formats, *HTML* and *Markdown*, each evaluated with and without the inclusion of preceding sentences during embedding generation. In addition, three TabTree variants are included: *Base*, *Text*, and *Text-Augmentation with Context-Intersection*. For each of these TabTree configurations, we selected the version that achieved the best retrieval performance with regard to preceding sentence inclusion: *Base* without preceding sentences, and the other two with (as determined in Table 4.9).

Across all methods, execution accuracy remains modest not exceeding 0.53 for *GPT-4o-mini* and 0.48 for *Phi-4*, meaning that, at best, roughly half of the questions are answered correctly. The best-performing approach is *TabTree Base*, which also showed the highest retrieval effectiveness (see Table 4.10). *TabTree Base* achieves execution accuracies of 0.53 for *GPT-4o-mini* and 0.48 for *Phi-4*. The next-best performance is observed for *HTML* without preceding sentences, with scores of 0.45 and 0.40, followed by the three remaining baseline approaches *HTML* with preceding sentence inclusion and both *Markdown* setups, with each execution accuracies of 0.39 and 0.35. Lastly, the two text-based TabTree variants perform significantly worse than others, with execution accuracies of just 0.24 and 0.27 (*GPT-4o-mini*), and 0.21 and 0.26 (*Phi-4*).

Retrieval Serialization	w/ prec. sentence?	GPT-4o-mini			Phi-4		
		Exec. Acc.	F1-score	Wind. Exceed	Exec. Acc.	F1-score	Wind. Exceed
HTML	❌	0.45	0.45	0	0.40	0.40	0
	✅	0.39	0.39	0	0.35	0.34	0
Markdown	❌	0.39	0.39	0	0.35	0.35	0
	✅	0.39	0.39	0	0.35	0.35	0
TabTree - Base	❌	**0.53**	**0.53**	0	**0.48**	**0.49**	0
TabTree - Text	✅	0.24	0.24	0	0.21	0.21	0
TabTree - Text Augm. w/ Context-I.	✅	0.27	0.27	0	0.26	0.26	0

Table 4.11.: Table QA on Full Documents results for different table serialization approaches on SEC-Filing Tables ($n = 310$), excluding related tables from retrieved chunks, using *GPT-4o-mini* and *Phi-4* with retrieval parameter $k = 3$.

[31] In such cases, the content was truncated to fit the window, which may still allow for correct answer generation. Token limits for each LLM are listed in Table 4.4.

When comparing LLMs, *GPT-4o-mini* consistently outperforms *Phi-4* across all configurations, though the margin is relatively small and never exceeds 0.05. This suggests a slight overall advantage for *GPT-4o-mini* in the full-document RAG setting.

A key observation of Table 4.11 is, that execution accuracy closely aligns with retrieval performance. Across most approaches, execution accuracy mirrors or slightly exceeds the corresponding Recall@3 values from the retrieval step. For instance, *HTML* without context achieves 0.45 execution accuracy compared to 0.46 Recall@3, while *Markdown* scores 0.39 accuracy with Recall@3 values of 0.37 (without) and 0.36 (with context)[32].

These results highlight a central insight: when relevant chunks are successfully retrieved, LLMs are typically able to generate correct answers. This suggests that the primary bottleneck in full-document Table QA lies not in the answer generation step, but in the retrieval process.

Chunks with Related Tables Next, Table 4.12 presents the results for Table QA on full documents in the setting, in which retrieved chunks are augmented with associated related tables. Building on previous results, we evaluate selected combinations that reflect strong performance for the previously presented subtasks: For the indexing and retrieval stage, we reuse the baselines from Table 4.11, *HTML* and *Markdown*, each without preceding sentence inclusion, and add the best-performing TabTree configuration (*Base*), as well as the Table Summary approach as an alternative retrieval strategy. For the serialization of related tables for answer generation, we consider four approaches: *HTML*, *Markdown*, *TabTree Text*, and *TabTree Text-Augmentation with Context-Intersection*.

Consistent with findings from the chunk-only setup, *TabTree Base* remains the most effective approach in this augmented setting. When paired with related tables serialized in *HTML*, it achieves the highest execution accuracy across all evaluated combinations: 0.55 for *GPT-4o-mini* and 0.51 for *Phi-4*. Other *TabTree Base* variants also perform well: Using *TabTree Text* for related tables yields execution accuracies of 0.55 (*GPT-4o-mini*) and 0.50 (*Phi-4*), while using *TabTree Text-Augmentation with Context-Intersection* results in execution accuracies of 0.53 and 0.49, respectively.

[32]The pattern of execution accuracy exceeding Recall@3 may be explained by the exclusion of non-answerable questions in the retrieval evaluation. These questions might be answered more correctly than others on average, thereby raising the overall execution accuracy comparing to Recall@3.

Retrieval Serialization	w/ prec. sentence?	Related Table Serialization	GPT-4o-mini			Phi-4		
			Exec. Acc.	F1-score	Wind. Exceed	Exec. Acc.	F1-score	Wind. Exceed
HTML	❌	HTML	0.46	0.47	0	0.43	0.43	44
Markdown	❌	Markdown	0.47	0.47	16	0.41	0.41	136
		HTML	0.45	0.46	0	0.44	0.44	77
TabTree - Base	❌	HTML	**0.55**	**0.56**	1	**0.51**	**0.52**	35
		TabTree - Text	**0.55**	0.55	1	0.50	0.50	168
		TabTree - Text Augm. w/ Context-I.	0.53	0.53	1	0.49	0.49	202
Table Summary	❌	HTML	0.39	0.39	0	0.35	0.35	47
		TabTree - Text Augm. w/ Context-I.	0.36	0.37	0	0.32	0.32	98

Table 4.12.: Table QA on Full Documents results from different retrieval and related table serialization strategies on SEC-Filing Tables ($n = 310$), including related tables from retrieved chunks, using *GPT-4o-mini* and *Phi-4* with retrieval parameter $k = 3$.

Trailing behind are the baseline retrieval methods, *Markdown* and *HTML*, with execution accuracies ranging between 0.45 and 0.47 for *GPT-4o-mini*, and between 0.41 and 0.44 for *Phi-4*, depending on the format used to serialize related tables.

In contrast, the *Table Summary* approach consistently yields the weakest results among all evaluated methods. When related tables are serialized in *HTML*, execution accuracy drops to 0.39 (*GPT-4o-mini*) and 0.35 (*Phi-4*). When serialized using *TabTree Text-Augmentation with Context-Intersection*, scores decrease further to 0.36 and 0.35. These outcomes suggest that summarizing tables for retrieval offers no added benefit in the RAG-based Table QA on full documents setting.

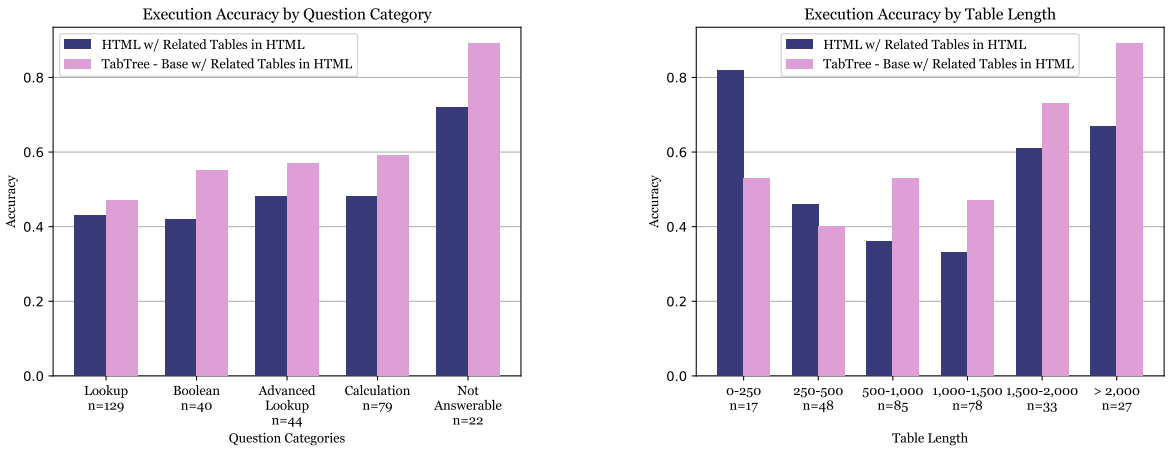
Comparing these results to the chunk-only setup in Table 4.11 reveals that the inclusion of related tables leads to only marginal gains in answer accuracy. For most configurations, the improvement in execution accuracy does not exceed 0.02 for *GPT-4o-mini* and 0.03 for *Phi-4*. An exception is found with the *Markdown* format, which shows a more noticeable increase: execution accuracy rises from 0.39 to 0.47, when related tables are also serialized in Markdown, and to 0.45, when serialized in HTML (using *GPT-4o-mini*). For *Phi-4*, the respective improvements are 0.06 and 0.09. Despite these gains, the overall impact of related table augmentation remains modest.

Finally, it is important to highlight the trade-offs introduced by the related-table augmentation. Including related tables increases prompt length considerably, frequently resulting in context window overflows, particularly for smaller models like *Phi-4*. This

is reflected in the high number of window exceed instances reported for *Phi-4* in Table 4.12. Even for *GPT-4o-mini*, despite its 128k token limit, the resulting prompt occasionally exceeds the context window, most notably in the *Markdown-Markdown* setup, which produces 16 overflows.

Question Category & Table Token Length Analysis To conclude our evaluation of Table QA on full documents, we analyze, similarly to the previous subtasks, performance across different question types and table token lengths. Figure 4.8 shows execution accuracy segmented by category and table size for two retrieval strategies, *HTML* and *TabTree Base*, with related tables serialized in *HTML* in both cases. These selected approaches correspond to the first and fourth rows of Table 4.12.

With respect to questions categories, we observe, that full-document Table QA performance appears to mirror retrieval quality, with execution accuracy increasing alongside question complexity. For both *HTML* and *TabTree Base*, the lowest scores are observed for *lookup* questions (0.43 and 0.47, respectively), followed by *boolean* (0.42 and 0.55), then *advanced Lookup* (0.48 and 0.57), and the highest for *calculation* questions (0.48 and 0.59). In contrast, the vanilla Table QA task (see Figure 4.5) showed a reverse trend, with more complex questions being harder to answer correctly. This reversal



(a) Question Category Analysis

(b) Table Token Length Analysis

Figure 4.8.: Table QA on Full Documents results by Question Category and Table Token Length for the best performing approaches TabTree Base with related tables serialized in *HTML* and *HTML* with related tables serialized in *HTML* on SEC-Filing Tables ($n = 310$) using the *GPT-4o-mini* model and retrieval parameter $k = 3$. Note: For table token length analysis, non-answerable questions without corresponding ground truth table were excluded.

suggests that in the full-document RAG setting, retrieval, rather than generation, is the primary factor influencing overall performance.

A noteworthy observation in the analysis of question category effectiveness is the performance on *non-answerable* questions. These are answered significantly more reliably than other categories, with execution accuracies of 0.72 for *HTML* and 0.89 for *TabTree Base* as retrieval serializations.

Turning to table token length, we find patterns similar to those observed in the chunk retrieval subtask: the effect of table size on performance is somewhat inconsistent. For *HTML*, execution accuracy is highest for very short tables (≤ 250 tokens) at 0.82, then decreases for mid-length tables before recovering for longer ones. The lowest performance occurs in the 1,000-1,500 token range (0.33), while tables over 2,000 tokens reach 0.67.

TabTree Base exhibits a similar pattern, with accuracy lowest at 0.40 for 250-500 table tokens and peaking at 0.89 for tables exceeding 2,000 tokens. An explanation for this behaviour could be that the condensed format of the *TabTree Base* serialization facilitates more accurate retrieval, while *HTML*-based related table serialization enriches the generation context, especially for larger tables.

5. Conclusion and Future Work

This thesis presented a framework for Table Question Answering (Table QA) in contexts where tables are not treated as isolated artifacts, but as integral parts of documents containing both structured and unstructured content. The framework incorporates a Retrieval-Augmented Generation (RAG) pipeline designed to handle table-aware retrieval and answer generation challenges. A key contribution of this thesis is the introduction of a novel table serialization approach, TabTree, that models tables as directed trees based on column headers and row labels. These trees are translated into natural language to provide serialized table representations using a modular, template-based approach. The proposed framework alongside the TabTree serialization and standard serialization baselines were evaluated empirically.

5.1. Key Findings

This thesis was guided by a set of research questions introduced in Chapter 1. In the following, each question is revisited and answered based on the findings from the empirical evaluation presented in Chapter 4.

(RQ1.) *How effectively do LLMs perform in Table QA for both standalone tables and tables embedded in full documents?*

LLMs demonstrate high effectiveness in answering questions related to standalone tables, with performance largely invariant across different table serialization formats. In particular, the best performing LLM, *GPT-4o*, achieved up to 76 % accuracy on the *WikiTableQuestions* dataset, with all other approaches achieving at least 67 % for *GPT-4o*. Other models also performed competitively, however, smaller models such as *Phi-4* were occasionally limited by their shorter context window.

In contrast, when tables are embedded in full documents and relevant information must first be retrieved from these documents, performance drops

considerably. Even in the best-performing setup, only about half of the questions were answered correctly.

(RQ2.) *Which table representation method yields the best performance for answering questions in both isolated and document-embedded contexts?*

(a) *Which serialization format best facilitates LLM-based answer generation for table-related questions?*

When relevant tables are provided directly, traditional serialization approaches such as HTML, CSV, JSON, and Markdown yield comparable results, with HTML slightly outperforming the others. Sentence-based representations like TabTree, underperform in this context and impose additional computational costs due to increased token lengths.

(b) *Which format most effectively supports the retrieval of relevant information from full documents containing tables?*

In contrast to its performance in answer generation, the minimalist variant of TabTree (*TabTree Base*) outperforms traditional serialization formats for retrieval tasks. This suggests that a condensed representation of table content, stripped of structural tokens, is more advantageous to semantic matching of document chunks. Even compared to table summaries, *TabTree Base* yielded superior retrieval performance, underscoring the value of concise but complete table serialization.

(RQ3.) *What are the key components required to build an effective question-answering pipeline for tables embedded within documents?*

Empirical results show that retrieval quality is the critical limiting factor in a RAG-pipeline for Table QA on full documents. When retrieval fails, due to irrelevant or incomplete chunks, the answering component cannot compensate, even when the LLM is otherwise capable. An analysis of retrieval performance shows that when retrieving up to five chunks per question, approximately 30% of retrieved sets contained only irrelevant content. When the retrieval limit was reduced to three chunks, the proportion of entirely irrelevant chunk increased to 43 %. Conversely, when relevant table chunks are successfully retrieved, LLMs reliably generate accurate answers. These findings underscore that the retrieval component is the key component in Table QA over mixed-format documents.

5.2. Limitations & Future Work

While this thesis offers several insights and introduces new ideas for handling Table QA on full documents, it also reveals limitations that present opportunities for future work. These can be grouped into the following areas:

- **Retrieval Improvements:** The retrieval component employed in this thesis is limited to a standard dense retrieval approach and does not reflect the full range of techniques explored in recent research. Future work could explore hybrid approaches that integrate sparse signals, e.g., keyword-based scoring, implement reranking mechanisms, or apply advanced methods such as query rewriting and retrieval routing.
- **Representation and Encoding:** A recurring challenge in this work was handling long serialized tables or document chunks that exceed the input limits of embedding models and LLMs. These cases were approached in a mostly greedy and naive manner. More sophisticated handling methods could be considered in future work, including multi-stage approaches for selecting relevant content. Additionally, only one embedding model was analysed in this thesis due to limited resources. To assess the generalizability of the results, future studies could include a broader range of embedding models.
- **Text Generation Improvements:** Although the results for answering questions based on provided ground truth tables showed effectiveness of around 70%, there remains room for optimization. The current pipeline follows a single-route architecture, meaning that all queries are processed through the same fixed sequence of components without dynamically selecting between different response mechanisms such as direct answering versus code execution. Furthermore, the thesis does not make use of multi-stage approaches to identify relevant table content first before generating answers.

  The thesis also exclusively used general-purpose LLMs without domain-specific fine-tuning. Particularly in the case of the sentence-based format of TabTree, fine-tuning could enhance model understanding of the underlying tabular data. Finally, while the RAG approach provides a partial workaround for the loss of spatial structure in table serialization, it does not address the fundamental architectural limitations. Future work may benefit from incorporating tabular foundation models combating this problem.

A. Appendix

A.1. Algorithms

Algorithm A.1 Column- / Rowspan-based Cell Backtracing

Input: Table $t = \{\gamma_{11}, \ldots, \gamma_{nm}\}$, Cell to backtrace $\gamma_{ij} = (s^c_{ij}, s^r_{ij}, t_{ij})$

Output: Backtraced cell $\gamma_{\text{orig}|ij}$

▷ ***Return 'origin cell', i.e., most upper left cell with same content as given cell***

1: $i_{\text{new}} = i - s^r_{\text{prev}}(i, j)$

2: $j_{\text{new}} = j - s^c_{\text{prev}}(i, j)$

3: **return** $\gamma_{\text{orig}|ij} \leftarrow \gamma_{i_{\text{new}} j_{\text{new}}}$

Algorithm A.2 Column Header Tree Construction

Input: Table $t = \{\gamma_{11}, \ldots, \gamma_{nm}\}$ with rows $r_1, \ldots, r_n$, columns $c_1, \ldots, c_m$, column headers $h_t = \langle r_1, \ldots, r_{i^*} \rangle$, and row labels $l_t = \langle c_1, \ldots, c_{j^*} \rangle$

Output: Column Header Tree $T_c := (V_c, E_c, \phi_c)$

▷ ***Apply table modifications of Algorithm 3.1***

$\tilde{t} \leftarrow \{\tilde{\gamma}_{11}, \ldots, \tilde{\gamma}_{nm}\}$ with $\tilde{\gamma}_{ij} = (\tilde{s}^c_{ij}, \tilde{s}^r_{ij}, t_{ij})$.

▷ ***Initialize tree with a single root node***

Initialize $V_c \leftarrow \{v_{\text{root}^c}\}$, $E_c \leftarrow \emptyset$

Set $\phi_c(v_{\text{root}^c}) \leftarrow$ `yellow`

for each column header row $r_i \in h_t$ **where** $i = 1, \ldots, i^*$ **do**

▷ ***Create column header nodes with child-parent connections***

  $V_{\text{new}} \leftarrow \{\tilde{\gamma}_{\text{orig}|ij} \mid \tilde{s}^c_{\text{prev}}(i,j) = 0, j = j^* + 1, \ldots, m\}$ ▷ Apply Algorithm A.1

  $V_c \leftarrow V_c \cup V_{\text{new}}$

  Set $\phi_c(v) \leftarrow$ `yellow` for all $v \in V_{\text{new}}$

  if $i = 1$ **then** ▷ Connect first row to root

    $E_{\text{new}} \leftarrow \{(v_{\text{root}^c}, v_{\text{new}}) \mid v_{\text{new}} \in V_{\text{new}}\}$

  else ▷ Connect child nodes to parent nodes using Algorithm A.1

    $E_{\text{new}} \leftarrow \{(v_{\text{prev}}, v_{\text{new}}) \mid v_{\text{new}} = \tilde{\gamma}_{\text{orig}|ij} \in V_{\text{new}}, v_{\text{prev}} = \tilde{\gamma}_{\text{orig}|(i-1)j}\}$

  end if

  $E_c \leftarrow E_c \cup E_{\text{new}}$

▷ ***Add context-intersection nodes using Algorithm A.1***

  $V_{\text{ref}} \leftarrow \{\tilde{\gamma}_{\text{orig}|ij} \mid \tilde{s}^c_{\text{prev}}(i,j) = 0, j = 1, \ldots, j^*\}$

  $V_c \leftarrow V_c \cup V_{\text{ref}}$

  Set $\phi_c(v) \leftarrow$ `orange` for all $v \in V_{\text{ref}}$

  $E^1_{\text{ref}} \leftarrow \{(v_{\text{left}}, v_{\text{right}}) \mid$
  $v_{\text{left}} = \tilde{\gamma}_{\text{orig}|ij} \in V_{\text{ref}}, v_{\text{right}} = \tilde{\gamma}_{\text{orig}|i(j+\tilde{s}^c_{\text{next}}(i,j))}, j + \tilde{s}^c_{\text{next}}(i,j) \leq j^*\}$

  $E^2_{\text{ref}} \leftarrow \{(v_{\text{ref}}, v_{\text{new}}) \mid v_{\text{ref}} = \tilde{\gamma}_{\text{orig}|ij} \in V_{\text{ref}}, j + \tilde{s}^c_{\text{next}}(i,j) = j^*, v_{\text{new}} \in V_{\text{new}}\}$

  $E_c \leftarrow E_c \cup E^1_{\text{ref}} \cup E^2_{\text{ref}}$

end for

▷ ***Add value cells as leaf nodes using Algorithm A.1***

$V_{\text{new}} \leftarrow \{\tilde{\gamma}_{\text{orig}|(i^*+k)(j^*+l)} \mid k = 0, \ldots, n - i^*, l = 1, \ldots, m - j^*\}$

$V_c \leftarrow V_c \cup V_{\text{new}}$

Set $\phi_c(v) \leftarrow$ `gray` for all $v \in V_{\text{new}}$

$E_{\text{new}} \leftarrow \{(v_{\text{prev}}, v_{\text{new}}) \mid v_{\text{new}} = \tilde{\gamma}_{ij} \in V_{\text{new}}, v_{\text{prev}} = \tilde{\gamma}_{\text{orig}|i^*j}\}$

$E_c \leftarrow E_c \cup E_{\text{new}}$

Algorithm A.3 Row Label Tree Construction

Input: Table $t = \{\gamma_{11}, \ldots, \gamma_{nm}\}$ with rows $r_1, \ldots, r_n$, columns $c_1, \ldots, c_m$, column headers $h_t = \langle r_1, \ldots, r_{i^*} \rangle$, and row labels $l_t = \langle c_1, \ldots, c_{j^*} \rangle$

Output: Row Label Tree $T_r := (V_r, E_r, \phi_r)$

▷ ***Apply table modifications of Algorithm 3.1***

$\tilde{t} \leftarrow \{\tilde{\gamma}_{11}, \ldots, \tilde{\gamma}_{nm}\}$ with $\tilde{\gamma}_{ij} = (\tilde{s}^c_{ij}, \tilde{s}^r_{ij}, t_{ij})$.

▷ ***Initialize tree with a single root node***

Initialize $V_r \leftarrow \{v_{\text{root}^r}\}$, $E_r \leftarrow \emptyset$

Set $\phi_c(v_{\text{root}^r}) \leftarrow$ `blue`

for each row label column $c_j \in l_t$ **where** $j = 1, \ldots, j^*$ **do**

▷ ***Create row label nodes with child-parent connections***

  $V_{\text{new}} \leftarrow \{\tilde{\gamma}_{\text{orig}|ij} \mid \tilde{s}^r_{\text{prev}}(i,j) = 0, i = i^*, \ldots, n\}$ ▷ Apply Algorithm A.1

  $V_r \leftarrow V_r \cup V_{\text{new}}$

  Set $\phi_r(v) \leftarrow$ `blue` for all $v \in V_{\text{new}}$

  if $i = 1$ **then** ▷ Connect first row to root

    $E_{\text{new}} \leftarrow \{(v_{\text{root}^r}, v_{\text{new}}) \mid v_{\text{new}} \in V_{\text{new}}\}$

  else ▷ Connect child nodes to parent nodes using Algorithm A.1

    $E_{\text{new}} \leftarrow \{(v_{\text{prev}}, v_{\text{new}}) \mid v_{\text{new}} = \tilde{\gamma}_{\text{orig}|ij} \in V_{\text{new}}, v_{\text{prev}} = \tilde{\gamma}_{\text{orig}|i(j-1)}\}$

  end if

  $E_c \leftarrow E_c \cup E_{\text{new}}$

▷ ***Add context-intersection nodes using Algorithm A.1***

  $V_{\text{ref}} \leftarrow \{\tilde{\gamma}_{\text{orig}|ij} \mid \tilde{s}^r_{\text{prev}}(i,j) = 0, i = 1, \ldots, i^*\}$

  $V_c \leftarrow V_c \cup V_{\text{ref}}$

  Set $\phi_c(v) \leftarrow$ `orange` for all $v \in V_{\text{ref}}$

  $E^1_{\text{ref}} \leftarrow \{(v_{\text{up}}, v_{\text{down}}) \mid$
  $v_{\text{up}} = \tilde{\gamma}_{\text{orig}|ij} \in V_{\text{ref}}, v_{\text{down}} = \tilde{\gamma}_{\text{orig}|(i+\tilde{s}^r_{\text{next}}(i,j))j}, i + \tilde{s}^r_{\text{next}}(i,j) \leq i^*\}$

  $E^2_{\text{ref}} \leftarrow \{(v_{\text{ref}}, v_{\text{new}}) \mid v_{\text{ref}} = \tilde{\gamma}_{\text{orig}|ij} \in V_{\text{ref}}, i + \tilde{s}^r_{\text{next}}(i,j) = i^*, v_{\text{new}} \in V_{\text{new}}\}$

  $E_c \leftarrow E_c \cup E^1_{\text{ref}} \cup E^2_{\text{ref}}$

end for

▷ ***Add value cells as leaf nodes using Algorithm A.1***

$V_{\text{new}} \leftarrow \{\tilde{\gamma}_{\text{orig}|(i^*+k)(j^*+l)} \mid k = 0, \ldots, n - i^*, l = 1, \ldots, m - j^*\}$

$V_c \leftarrow V_c \cup V_{\text{new}}$

Set $\phi_c(v) \leftarrow$ `gray` for all $v \in V_{\text{new}}$

$E_{\text{new}} \leftarrow \{(v_{\text{prev}}, v_{\text{new}}) \mid v_{\text{new}} = \tilde{\gamma}_{\text{orig}|ij} \in V_{\text{new}}, v_{\text{prev}} = \tilde{\gamma}_{\text{orig}|ij^*}\}$

$E_c \leftarrow E_c \cup E_{\text{new}}$

A.2. Dataset Examples

A.2.1. WikiTablesQuestions

WIKIPEDIA
The Free Encyclopedia

Donate Create account Log in

List of highest-grossing openings for films

3 languages

Article Talk Read Edit View history Tools

From Wikipedia, the free encyclopedia

The following is a list of the **highest-grossing opening weekends for films**. The list is dominated by recent films due to steadily increasing marketing budgets, and modern films opening on more screens. Another contributing factor is inflation not being taken into account. *Jaws*, the first "summer blockbuster", introduced the screen saturation strategy.

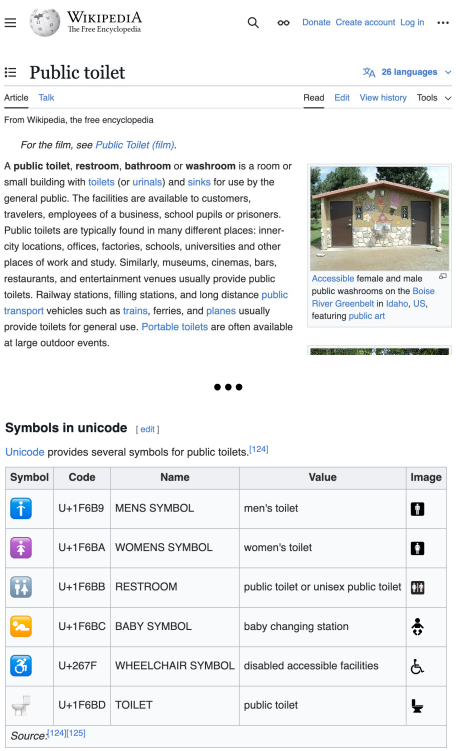
Biggest worldwide openings since 2002

This list charts films that had openings in excess of $200 million worldwide. Since films do not open on Fridays in many markets, the 'opening' is taken to be the gross between the first day of release and the first Sunday. Figures prior to the year 2002 are not available.

Rank	Film	Year	Opening (USD)[1]
1	*Harry Potter and the Deathly Hallows – Part 2*	2011	$483,189,427
2	*Harry Potter and the Half-Blood Prince*	2009	$394,022,354
3	*Marvel's The Avengers*	2012	$392,538,708
4	*Transformers: Dark of the Moon*	2011	$382,425,000
5	*Spider-Man 3*	2007	$381,660,892
6	*Iron Man 3*	2013	$372,553,677
7	*Pirates of the Caribbean: On Stranger Tides*	2011	$350,653,677
8	*Pirates of the Caribbean: At World's End*	2007	$343,972,864
9	*The Twilight Saga: Breaking Dawn - Part 2*	2012	$340,667,634
10	*Harry Potter and the Order of the Phoenix*	2007	$332,715,157
11	*Harry Potter and the Deathly Hallows – Part 1*	2010	$330,017,372
12	*Star Wars Episode III: Revenge of the Sith*	2005	$303,949,700
13	*The Twilight Saga: Breaking Dawn - Part 1*	2011	$291,022,261
14	*The Twilight Saga: New Moon*	2009	$274,939,137
15	*Indiana Jones and the Kingdom of the Crystal Skull*	2008	$272,150,927
16	*The Lord of the Rings: The Return of the King*	2003	$250,000,534
17	*The Dark Knight Rises*	2012	$248,887,295
18	*Avatar*	2009	$241,571,046
19	*The Da Vinci Code*	2006	$232,074,037
20	*2012*	2009	$230,470,474
21	*The Twilight Saga: Eclipse*	2010	$228,893,465
22	*The Hobbit: An Unexpected Journey*	2012	$222,617,303
23	*Iron Man 2*	2010	$220,766,270
24	*Transformers: Revenge of the Fallen*	2009	$219,921,171
25	*Ice Age: Dawn of the Dinosaurs*	2009	$218,430,064
26	*The Hunger Games*	2012	$211,785,747
27	*Alice in Wonderland*	2010	$210,101,023
28	*Harry Potter and the Prisoner of Azkaban*	2004	$207,197,367
29	*The Hobbit: The Desolation of Smaug*	2013	$204,845,197
30	*War of the Worlds*	2005	$203,061,125
31	*Man of Steel*	2013	$201,981,486
32	*The Matrix Revolutions*	2003	$201,390,000
33	*Iron Man*	2008	$201,174,140

Question	Answer
in terms of gross what movie is above toy story 3?	Pirates of the Caribbean: At World's End
the biggest worldwide opening since 2002 is held by what movie?	Marvel's The Avengers
what was the number of twilight movies that made this ranked list?	3
what is the average opening weekend gross between iron man 3 and the dark knight?	$166,278,034
only this movie had a worldwide opening of $390+ million in 2012?	Marvel's The Avengers
which film had the least opening weekend?	Indiana Jones and the Kingdom of the Crystal Skull
name one that ranked after man of steel.	Alice in Wonderland

Figure A.1.: Example of a Wikipedia document (List of highest-grossing openings for films) containing a table along with corresponding questions from the WikiTablesQuestions dataset (Pasupat et al., 2015)



Public toilet

26 languages

Article Talk Read Edit View history Tools

From Wikipedia, the free encyclopedia

For the film, see Public Toilet (film).

A **public toilet**, **restroom**, **bathroom** or **washroom** is a room or small building with toilets (or urinals) and sinks for use by the general public. The facilities are available to customers, travelers, employees of a business, school pupils or prisoners. Public toilets are typically found in many different places: inner-city locations, offices, factories, schools, universities and other places of work and study. Similarly, museums, cinemas, bars, restaurants, and entertainment venues usually provide public toilets. Railway stations, filling stations, and long distance public transport vehicles such as trains, ferries, and planes usually provide toilets for general use. Portable toilets are often available at large outdoor events.

Accessible female and male public washrooms on the Boise River Greenbelt in Idaho, US, featuring public art

Symbols in unicode [edit]

Unicode provides several symbols for public toilets.[124]

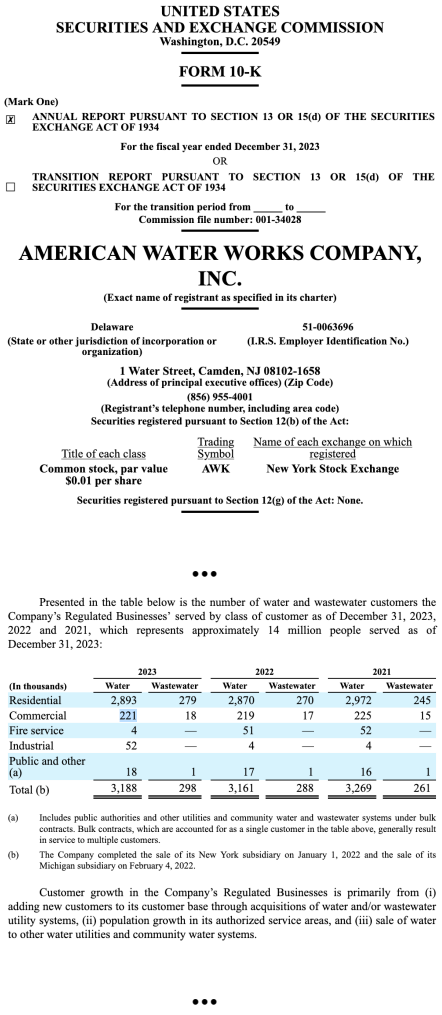
Symbol	Code	Name	Value	Image
	U+1F6B9	MENS SYMBOL	men's toilet	
	U+1F6BA	WOMENS SYMBOL	women's toilet	
	U+1F6BB	RESTROOM	public toilet or unisex public toilet	
	U+1F6BC	BABY SYMBOL	baby changing station	
	U+267F	WHEELCHAIR SYMBOL	disabled accessible facilities	
	U+1F6BD	TOILET	public toilet	

Source:[124][125]

Question	Answer
what is the code for shower facilities?	U+1F6BF
what is the code for a unisex restroom?	U+1F6BB
is a water closet the same as a toilet?	No

Figure A.2.: Example of a Wikipedia document (Public toilet) containing a table along with corresponding questions from the WikiTablesQuestions dataset (Pasupat et al., 2015)

A.2.2. SEC-Filing Tables



UNITED STATES
SECURITIES AND EXCHANGE COMMISSION
Washington, D.C. 20549

FORM 10-K

(Mark One)
☒ ANNUAL REPORT PURSUANT TO SECTION 13 OR 15(d) OF THE SECURITIES EXCHANGE ACT OF 1934

For the fiscal year ended December 31, 2023
OR
☐ TRANSITION REPORT PURSUANT TO SECTION 13 OR 15(d) OF THE SECURITIES EXCHANGE ACT OF 1934

For the transition period from _____ to _____
Commission file number: 001-34028

AMERICAN WATER WORKS COMPANY, INC.
(Exact name of registrant as specified in its charter)

Delaware	51-0063696
(State or other jurisdiction of incorporation or organization)	(I.R.S. Employer Identification No.)

1 Water Street, Camden, NJ 08102-1658
(Address of principal executive offices) (Zip Code)
(856) 955-4001
(Registrant's telephone number, including area code)
Securities registered pursuant to Section 12(b) of the Act:

Title of each class	Trading Symbol	Name of each exchange on which registered
Common stock, par value $0.01 per share	AWK	New York Stock Exchange

Securities registered pursuant to Section 12(g) of the Act: None.

•••

Presented in the table below is the number of water and wastewater customers the Company's Regulated Businesses' served by class of customer as of December 31, 2023, 2022 and 2021, which represents approximately 14 million people served as of December 31, 2023:

	2023		2022		2021	
(In thousands)	Water	Wastewater	Water	Wastewater	Water	Wastewater
Residential	2,893	279	2,870	270	2,972	245
Commercial	221	18	219	17	225	15
Fire service	4	—	51	—	52	—
Industrial	52	—	4	—	4	—
Public and other (a)	18	1	17	1	16	1
Total (b)	3,188	298	3,161	288	3,269	261

(a) Includes public authorities and other utilities and community water and wastewater systems under bulk contracts. Bulk contracts, which are accounted for as a single customer in the table above, generally result in service to multiple customers.

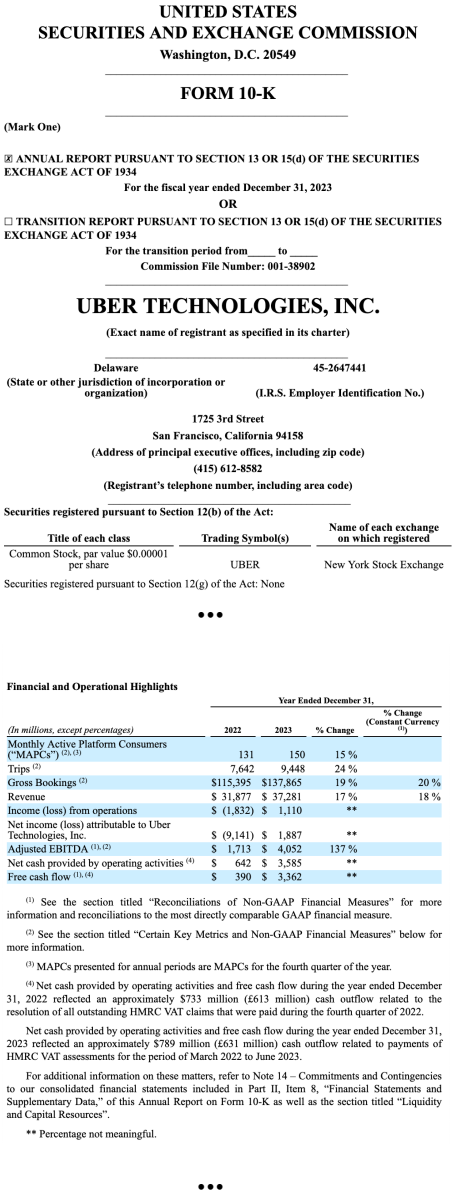
(b) The Company completed the sale of its New York subsidiary on January 1, 2022 and the sale of its Michigan subsidiary on February 4, 2022.

Customer growth in the Company's Regulated Businesses is primarily from (i) adding new customers to its customer base through acquisitions of water and/or wastewater utility systems, (ii) population growth in its authorized service areas, and (iii) sale of water to other water utilities and community water systems.

•••

Question	Answer	Unit	Search Reference
How many customers did American Water Works serve in the commercial area in the water sector in 2023?	221	thousands	Commercial.*?221
How many customers did American Water Works serve in the commercial area in the water sector in 2021?	225	thousands	Commercial.*?225
How many customers did American Water Works serve in the class of residential customers in the water sector in 2022?	2870	thousands	Residential.*?2,?870
How many fire service customers did American Water Works have in the wastewater sector in 2022?	0		Fire service.*?—
Did American Water Works have any fire service customers in the wastewater sector in 2021 to 2023?	No		Fire service.*?—
Did the amount of customers for American Water Works increase from 2022 to 2023 in both sectors water and wastewater?	Yes		Total.*?288
What's the class of customers with the smallest amount of customers for American Water Works in 2023 in the water sector?	Fire Service		Fire service.*?4
Did the class of customers with the smallest amount of customers for American Water Works change from 2022 to 2023?	Yes		Fire service.*?4.*?Industrial.*?4
How many customers did American Water Works loose from 2021 to 2022 in the commercial area in the water sector?	6	thousands	Commercial.*?225
What's been the average amount of customers of American Water Works in the wastewater sector in residential area in the years 2021 to 2023?	264.7	thousands	Residential.*?245
What's the difference of the total amount of customers for American Water Works in the area of wastewater from 2023 to 2021?	37	thousands	Total.*?261
What's the amount of customers which American Water Works served in the Water sector for the class of Police offices?	*None*		

Figure A.3.: Excerpt from the SEC Form 10-K document of American Water Works, illustrating a table alongside related questions from the SEC-Filing Tables dataset. The search reference is based on a regular expression designed to identify tables within the document. However, it is important to note that this reference may not be unique, as multiple tables can contain overlapping or relevant information.



UNITED STATES
SECURITIES AND EXCHANGE COMMISSION
Washington, D.C. 20549

FORM 10-K

(Mark One)

☒ ANNUAL REPORT PURSUANT TO SECTION 13 OR 15(d) OF THE SECURITIES EXCHANGE ACT OF 1934
For the fiscal year ended December 31, 2023
OR
☐ TRANSITION REPORT PURSUANT TO SECTION 13 OR 15(d) OF THE SECURITIES EXCHANGE ACT OF 1934
For the transition period from_____ to _____
Commission File Number: 001-38902

UBER TECHNOLOGIES, INC.
(Exact name of registrant as specified in its charter)

Delaware	**45-2647441**
(State or other jurisdiction of incorporation or organization)	**(I.R.S. Employer Identification No.)**

1725 3rd Street
San Francisco, California 94158
(Address of principal executive offices, including zip code)
(415) 612-8582
(Registrant's telephone number, including area code)

Securities registered pursuant to Section 12(b) of the Act:

Title of each class	**Trading Symbol(s)**	**Name of each exchange on which registered**
Common Stock, par value $0.00001 per share	UBER	New York Stock Exchange

Securities registered pursuant to Section 12(g) of the Act: None

•••

Financial and Operational Highlights

	Year Ended December 31,			
(In millions, except percentages)	**2022**	**2023**	**% Change**	**% Change (Constant Currency [1])**
Monthly Active Platform Consumers ("MAPCs") [2], [3]	131	150	15 %	
Trips [2]	7,642	9,448	24 %	
Gross Bookings [2]	$115,395	$137,865	19 %	20 %
Revenue	$ 31,877	$ 37,281	17 %	18 %
Income (loss) from operations	$ (1,832)	$ 1,110	**	
Net income (loss) attributable to Uber Technologies, Inc.	$ (9,141)	$ 1,887	**	
Adjusted EBITDA [1], [2]	$ 1,713	$ 4,052	137 %	
Net cash provided by operating activities [4]	$ 642	$ 3,585	**	
Free cash flow [1], [4]	$ 390	$ 3,362	**	

[1] See the section titled "Reconciliations of Non-GAAP Financial Measures" for more information and reconciliations to the most directly comparable GAAP financial measure.

[2] See the section titled "Certain Key Metrics and Non-GAAP Financial Measures" below for more information.

[3] MAPCs presented for annual periods are MAPCs for the fourth quarter of the year.

[4] Net cash provided by operating activities and free cash flow during the year ended December 31, 2022 reflected an approximately $733 million (£613 million) cash outflow related to the resolution of all outstanding HMRC VAT claims that were paid during the fourth quarter of 2022.

Net cash provided by operating activities and free cash flow during the year ended December 31, 2023 reflected an approximately $789 million (£631 million) cash outflow related to payments of HMRC VAT assessments for the period of March 2022 to June 2023.

For additional information on these matters, refer to Note 14 – Commitments and Contingencies to our consolidated financial statements included in Part II, Item 8, "Financial Statements and Supplementary Data," of this Annual Report on Form 10-K as well as the section titled "Liquidity and Capital Resources".

** Percentage not meaningful.

•••

Question	Answer	Unit	Search Reference
How many monthly active users did Uber have in 2022?	131	mio.	Monthly Active Platform Consumers ("MAPCs").*?131
How many people did use Uber on a monthly basis in 2023?	150	mio.	Monthly Active Platform Consumers ("MAPCs").*?50
How many trips were booked by Uber in 2023?	9448	mio.	Trips.*?9,?448
How much did the amount of booked trips on Uber change from 2022 to 2023?	24	%	Trips.*?24
What's the sum of booked trips of Uber in the years 2022 and 2023 combined?	17090	mio.	Trips.*?7,?642.*?9,?448
How many monthly active users did Uber have in 2019?	*None*		

Figure A.4.: Excerpt from the SEC Form 10-K document of Uber, illustrating a table alongside related questions from the SEC-Filing Tables dataset. The search reference is based on a regular expression designed to identify tables within the document. However, it is important to note that this reference may not be unique, as multiple tables can contain overlapping or relevant information.

A.3. Prompts

```
Classify a question into one of the following categories based on the provided
tabular data:

1. **Lookup** – Directly retrieves a value from the table without any additional
operations.
2. **Advanced Lookup** – Involves counting, sorting, ranking (including mininmum /
maximum), or simple comparison of different values.
3. **Boolean** – Requires a yes/no or true/false answer.
4. **Calculation** – Requires arithmetic operations like summation, percentage
calculation, subtractions, or any other sophisticaded math calculations.
5. **Position Related** – Asks about the next, previous, or any other positioned item
in the table in relation to a given one (e.g., "Who ranked right after Turkey?").

**Provide concise reasoning (within 500 tokens) and a final classification.**

---

**Input:**
- **Question:** {Question}
- **Table Data:** {Table}

---

### **Output Format:**
1. **Step-by-step reasoning** (keywords, operations, and table structure).
2. **Final classification:**
   **Answer: <category>**

---

### **Examples**

#### **Example 1**
**Question:** *What was the total regulated business operating revenue of American
Water Works in 2023?*
**Output:**
- Retrieves a single value from the "Total Regulated Businesses" row under "Operating
Revenues."
- No calculations or sorting required → **Lookup**.
**Answer: Lookup**

... (more examples for each category – refer to codebase on GitHub)
```

Figure A.5.: Prompt template for categorization of questions for the WikiTableQuestions dataset

```
You are tasked with answering questions based on the provided table data. Your task
is to give the correct answer based on the given table data. Only use the information
presented. If you can't find a correct answer based upon the provided data indicate
so.

Think step-by-step, but in a short and comprehensive manner (max 500 tokens). Provide
your final conclusion to the question as a single statement using the following
format:
Answer: <Answer>

Provide the answer <Answer> as one of the following:
- A number for questions about quantities.
- A word or phrase for questions about categories, names, etc.
- A semicolon separated list if the question asks for multiple answers (e.g. Brazil;
Argentina; Germany).
- 'None' if the context doesn't provide information about the given question.

Example:
- What was the sales revenue of Company Y from 2022 to 2024?
  <Step-by-step reasoning>
  Answer: 2,500
- What is the main product of Company Z?
  <Step-by-step reasoning>
  Answer: ZetaDrug
- Which clubs have won the german Bundesliga in the last five years?
  <Step-by-step reasoning>
  Answer: Bayern Munich; Bayer Leverkusen
- What's stock market price of Company X in 2021?
  <Step-by-step reasoning>
  Answer: None

{table_title} # E.g.: 'Table Title: Public Toilet', empty string if not existant

Table data: {table}

Question: {question}
```

Figure A.6.: Prompt template for answer generation of the vanilla Table QA task (Definition 3.11)

```
You are tasked with answering questions based on document chunks and related tables.
Your goal is to provide the most accurate answer using only the provided data. **Do
not assume or invent information.** If no sufficient answer can be found, indicate
so.

Think step-by-step, but in a short and comprehensive manner (max 600 tokens). After
the reasoning, provide your final conclusion to the question as a single statement
using the following format:
Answer: <Answer>

### **Instructions**:
1. **Analyze the provided document chunks carefully.**
2. **Clearly indicate if neither the chunks nor the tables provide enough data.**
3. Provide the answer <Answer> after your step-by-step reasoning as one of the
following:
  - A number (without units) for questions about quantities.
  - A word or phrase for questions about categories, names, etc.
  - Yes or No for questions that can be answered with a boolean.
  - 'None' if the context doesn't provide information about the given question.

### **Examples**:
- **What was the revenue of Company Y in 2022?**
  <Step-by-step reasoning>
  Answer: 2,500

- **What is the main product of Company Z?**
  <Step-by-step reasoning>
  Answer: ZetaDrug

- **What was Company X's stock price in 2021?**
  <Step-by-step reasoning>
  Answer: None

### Question:
{question}

### **Context**:
{context}
```

Figure A.7.: Prompt template for answer generation of the RAG-based task of Table QA on Full Documents (Definition 3.15) and Table QA on Corpus (Definition 3.16 without including related tables of retrieved chunks.

```
You are tasked with answering questions based on document chunks and related tables.
Your goal is to provide the most accurate answer using only the provided data. **Do
not assume or invent information.** If no sufficient answer can be found, indicate
so.

Think step-by-step, but in a short and comprehensive manner (max 600 tokens). After
the reasoning, provide your final conclusion to the question as a single statement
using the following format:
Answer: <Answer>

### **Instructions**:
1. **Analyze the provided document chunks and related tables carefully.**
2. **Clearly indicate if neither the chunks nor the tables provide enough data.**
3. Provide the answer <Answer> after your step-by-step reasoning as one of the
following:
  - A number (without units) for questions about quantities.
  - A word or phrase for questions about categories, names, etc.
  - Yes or No for questions that can be answered with a boolean.
  - 'None' if the context doesn't provide information about the given question.

### **Examples**:
- **What was the revenue of Company Y in 2022?**
  <Step-by-step reasoning>
  Answer: 2,500

- **What is the main product of Company Z?**
  <Step-by-step reasoning>
  Answer: ZetaDrug

- **What was Company X's stock price in 2021?**
  <Step-by-step reasoning>
  Answer: None

### Question:
{question}

### **Context**:
{context}

### **Related Tables**:
{related_tables}
```

Figure A.8.: Prompt template for answer generation of the RAG-based task of Table QA on Full Documents (Definition 3.15) and Table QA on Corpus (Definition 3.16 including related tables of retrieved chunks.

```
You are an AI assistant tasked with summarizing tables in a **coherent, structured,
and concise manner** for retrieval-based applications. Your goal is to generate a
short but meaningful summary that retains the most relevant details while being easy
to understand.

### **Instructions:**
- **Preceding Sentence:** {preceding_sentence}
  *(This is the sentence that directly precedes the table. It may provide context,
but sometimes it may not be helpful.)*
- **Table Summary:**
  - The summary **must begin with:** "Presented in this table..."
  - Provide a **coherent summary** in **a maximum of 3 sentences**.
  - Capture the **columns, row structure, and intent** of the table.
  - Highlight **key insights or trends** if applicable.
- **Limit the output to 500 tokens** while ensuring clarity and completeness.

Here is the table that needs summarization:
{table}

Table Summary: Presented in this table <concise summary in 3 sentences>.
```

Figure A.9.: Prompt template for generating table summaries for the tasks of Chunk Retrieval (Definition 3.13) and Chunk Retrieval on Corpus (Definition 3.14)

```
Column header rows serve as a structural component at the top of a dataset or table,
defining the meaning and context of the columns below them. Therefore the probability
of a row being header row decreases by increasing row index. It also might happen
that a table has no column header rows.

Column header rows typically contain labels that describe the type of data in each
column, acting as a guide for interpreting the table's contents. In tables with
hierarchical structures, column header rows can span multiple levels, using features
like colspans to group related columns visually. This layered organization allows for
the representation of complex relationships between columns, improving clarity and
facilitating detailed analysis of multidimensional data.
```

Figure A.10.: Description of column header rows used in prompt templates for Table Context Detection (see Figure A.12, Figure A.13, and Figure A.14).

```
Row label columns act as a structural component in a table, typically positioned on
the left side, and provide descriptive or categorical identifiers for each row. The
probability of column beeing a label column therefore decrease by increasing column
index. It also might happen that a table has no row label columns.

Row label columns define the context or grouping of the data within the rows,
allowing for easier reference and understanding of the dataset. In hierarchical
tables, row label columns can represent multiple levels of categorization, employing
features like rowspans to visually group related rows under shared labels.
This structure enables the organization of nested relationships within the data,
enhancing readability and supporting more detailed analysis of grouped or
hierarchical information.
```

Figure A.11.: Description of row label columns used in prompt templates for Table Context Detection (see Figure A.12, Figure A.13, and Figure A.14).

```
You are tasked with analyzing table data to determine whether the provided
{row_or_column} should be classified as a {column-header-row_or_row-label-column} or
not.

Instructions:
1. Carefully review the table, provided in HTML format. The table lacks predefined
headers, and all rows are wrapped in <tr> tags.
2. Consider typical properties of a {column-header-row_or_row-label-column}
{row_or_column} when making your determination. For {column-header-row_or_row-label-
column} {row_or_column}s typically yield: {row_label_or_column_header_description}
3. End your response with one of the following:
   - "Answer: Yes" if the {row_or_column} represents a {column-header-row_or_row-
label-column} {row_or_column}.
   - "Answer: No" if the {row_or_column} contains actual records or entries (e.g.,
numerical values, measurements, metrics or attributes corresponding to a category).
4. Reason your answer with maximum 500 tokens.

{row_or_column} to analyze:
{row_or_column_data}

The {row_or_column} index of the provided {row_or_column} is {index}.

Table data:
{table}
```

Figure A.12.: Prompt template for Full Table approach to Table Context Detection (see Definition 3.22), incorporating either the column header row description (Figure A.10) or the row label column description (Figure A.11).

```
You are tasked with analyzing table data to determine whether the provided
{row_or_column} should be classified as a {column-header-row_or_row-label-column} or
not.

Instructions:
1. Review the provided {row_or_column} and the next and previous {row_or_column}.
2. Consider typical properties of a {column-header-row_or_row-label-column}
{row_or_column} when making your determination. For {column-header-row_or_row-label-
column} {row_or_column}s typically yield: {row_label_or_column_header_description}
3. End your response with one of the following:
   - "Answer: Yes" if the {row_or_column} represents a {column-header-row_or_row-
label-column} {row_or_column}.
   - "Answer: No" if the {row_or_column} contains actual records or entries (e.g.,
numerical values, measurements, metrics or attributes corresponding to a category).
4. Reason your answer with maximum 500 tokens.

{row_or_column} to analyze:
{row_or_column_data}

The {row_or_column} index of the provided {row_or_column} is {index}.

{previous_row_or_column_data} # depends on index of previous row or column, e.g., for
                              # first row / column no previous row / column exist

{next_row_or_column_data}     # depends on index of next row or column, e.g., if end
                              # of table is reached, no next row /column exist
```

Figure A.13.: Prompt template for 1-Range approach to Table Context Detection (see Definition 3.22), incorporating either the column header row description (Figure A.10) or the row label column description (Figure A.11).

```
You are tasked with analyzing table data to determine whether the provided
{row_or_column} should be classified as a {column-header-row_or_row-label-column} or
not.

Instructions:
1. Review the provided {row_or_column} and the next and previous {row_or_column}s.
2. Consider typical properties of a {column-header-row_or_row-label-column}
{row_or_column} when making your determination. For {column-header-row_or_row-label-
column} {row_or_column}s typically yield: {row_label_or_column_header_description}
3. End your response with one of the following:
   - "Answer: Yes" if the {row_or_column} represents a {column-header-row_or_row-
label-column} {row_or_column}.
   - "Answer: No" if the {row_or_column} contains actual records or entries (e.g.,
numerical values, measurements, metrics or attributes corresponding to a category).
4. Reason your answer with maximum 500 tokens.

{row_or_column} to analyze:
{row_or_column_data}

The {row_or_column} index of the provided {row_or_column} is {index}.

# depends on index of previous row or column, e.g., for first row / column no
previous row / column exist
{previous_row_or_column_data}
{second_previous_row_or_column_data}

# depends on index of next row or column, e.g., if end of table is reached no next
row /column exist
{next_row_or_column_data}
{second_next_row_or_column_data}
```

Figure A.14.: Prompt template for 2-Range approach to Table Context Detection (see Definition 3.22), incorporating either the column header row description (Figure A.10) or the row label column description (Figure A.11).

A.4. TabTree Sample Serializations

```
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 A.15.: Table serialization of the sample table from Figure 3.5 employing the TabTree Base approach.

```
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, colum index: 2).
        The value of the column header age and the row label combination A, Tiffany
is 16 (row index: 4, colum index: 2).
        The value of the column header age and the row label combination B, Michael
is 17 (row index: 5, colum 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, colum index: 3).
        The value of the column header math and the row label combination A, Tiffany
is B (row index: 4, colum index: 3).
        The value of the column header math and the row label combination B, Michael
is D (row index: 5, colum index: 3).
      The column header english has siblings math. The values of english are:
        The value of the column header english and the row label combination A, John
is C (row index: 3, colum index: 4).
        The value of the column header english and the row label combination A,
Tiffany is B (row index: 4, colum index: 4).
        The value of the column header english and the row label combination B,
Michael is D (row index: 5, colum index: 4).
    The column header 2024 has siblings null, 2023. The children of 2024 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, colum index: 5).
        The value of the column header math and the row label combination A, Tiffany
is C (row index: 4, colum index: 5).
        The value of the column header math and the row label combination B, Michael
is D (row index: 5, colum index: 5).
      The column header english has siblings math. The values of english are:
        The value of the column header english and the row label combination A, John
is B (row index: 3, colum index: 6).
        The value of the column header english and the row label combination A,
Tiffany is B (row index: 4, colum index: 6).
        The value of the column header english and the row label combination B,
Michael is D (row index: 5, colum index: 6).
```

Figure A.16.: Table serialization of the sample table from Figure 3.5 employing the Tab-Tree Text approach.

```
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 age and the row label combination grades &
class & A, grades & name & John is 17 (row index: 3, colum index: 2).
        The value of the column header age and the row label combination grades &
class & A, grades & name & Tiffany is 16 (row index: 4, colum index: 2).
        The value of the column header age and the row label combination grades &
class & B, grades & name & Michael is 17 (row index: 5, colum 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 math and the row label combination grades &
class & A, grades & name & John is A (row index: 3, colum index: 3).
        The value of the column header math and the row label combination grades &
class & A, grades & name & Tiffany is B (row index: 4, colum index: 3).
        The value of the column header math and the row label combination grades &
class & B, grades & name & Michael is D (row index: 5, colum index: 3).
      The column header english represents class and name. The column header english
has siblings math. The values of the column header english are:
        The value of the column header english and the row label combination grades &
class & A, grades & name & John is C (row index: 3, colum index: 4).
        The value of the column header english and the row label combination grades &
class & A, grades & name & Tiffany is B (row index: 4, colum index: 4).
        The value of the column header english and the row label combination grades &
class & B, grades & name & Michael is D (row index: 5, colum index: 4).
    The column header 2024 has siblings null, 2023. The children of 2024 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 math and the row label combination grades &
class & A, grades & name & John is A (row index: 3, colum index: 5).
        The value of the column header math and the row label combination grades &
class & A, grades & name & Tiffany is C (row index: 4, colum index: 5).
        The value of the column header math and the row label combination grades &
class & B, grades & name & Michael is D (row index: 5, colum index: 5).
      The column header english represents class and name. The column header english
has siblings math. The values of the column header english are:
        The value of the column header english and the row label combination grades &
class & A, grades & name & John is B (row index: 3, colum index: 6).
        The value of the column header english and the row label combination grades &
class & A, grades & name & Tiffany is B (row index: 4, colum index: 6).
        The value of the column header english and the row label combination grades &
class & B, grades & name & Michael is D (row index: 5, colum index: 6).
```

Figure A.17.: Table serialization of the sample table from Figure 3.5 employing the Tab-Tree Text with Context-Intersection approach.

```
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, colum 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, colum 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, colum 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
and the row label combination grades & class & A, grades & name & John is A (row
index: 3, colum index: 3).
        The value of the column header combination grades, 2023, class & name & math
and the row label combination grades & class & A, grades & name & Tiffany is B (row
index: 4, colum index: 3).
        The value of the column header combination grades, 2023, class & name & math
and the row label combination grades & class & B, grades & name & Michael is D (row
index: 5, colum index: 3).
      The column header english represents class and name. The column header english
has siblings math. The values of the column header english are:
        The value of the column header combination grades, 2023, class & name &
english and the row label combination grades & class & A, grades & name & John is C
(row index: 3, colum index: 4).
        The value of the column header combination grades, 2023, class & name &
english and the row label combination grades & class & A, grades & name & Tiffany is
B (row index: 4, colum index: 4).
        The value of the column header combination grades, 2023, class & name &
english and the row label combination grades & class & B, grades & name & Michael is
D (row index: 5, colum index: 4).
    The column header 2024 has siblings null, 2023. The children of 2024 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, 2024, class & name & math
and the row label combination grades & class & A, grades & name & John is A (row
index: 3, colum index: 5).
        The value of the column header combination grades, 2024, class & name & math
and the row label combination grades & class & A, grades & name & Tiffany is C (row
index: 4, colum index: 5).
        ...# CONTINUING
```

Figure A.18.: Table serialization of the sample table from Figure 3.5 employing the Tab-Tree Text-Augmentation with Context-Intersection approach.

```
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,
colum 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, colum 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, colum index: 2).

The value of the column header combination grades, 2023, class & name & math and the
row label combination grades & class & A, grades & name & John is A (row index: 3,
colum index: 3).

The value of the column header combination grades, 2023, class & name & math and the
row label combination grades & class & A, grades & name & Tiffany is B (row index: 4,
colum index: 3).

The value of the column header combination grades, 2023, class & name & math and the
row label combination grades & class & B, grades & name & Michael is D (row index: 5,
colum index: 3).

The value of the column header combination grades, 2023, class & name & english and
the row label combination grades & class & A, grades & name & John is C (row index:
3, colum index: 4).

The value of the column header combination grades, 2023, class & name & english and
the row label combination grades & class & A, grades & name & Tiffany is B (row
index: 4, colum index: 4).

The value of the column header combination grades, 2023, class & name & english and
the row label combination grades & class & B, grades & name & Michael is D (row
index: 5, colum index: 4).

The value of the column header combination grades, 2024, class & name & math and the
row label combination grades & class & A, grades & name & John is A (row index: 3,
colum index: 5).

The value of the column header combination grades, 2024, class & name & math and the
row label combination grades & class & A, grades & name & Tiffany is C (row index: 4,
colum index: 5).

The value of the column header combination grades, 2024, class & name & math and the
row label combination grades & class & B, grades & name & Michael is D (row index: 5,
colum index: 5).
 ...# CONTINUING
```

Figure A.19.: Table serialization of the sample table from Figure 3.5 employing the Tab-Tree Context-Empty approach.

A.5. 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 A.1.: Evaluation results for Context Detection on $n = 21$ manually annotated samples created for tables from the WikiTableQuestions and SEC-Filings datasets. (a) illustrates how evaluation metrics are calculated, while (b) presents the corresponding results.

Approach	**Execution Accuracy**				**Token Count**
	GPT-4o	**GPT-4o-mini**	**Phi-4**	**Llama-3.3-70B-Instruct**	
HTML	**0.76 ± 0.04**	**0.71 ± 0.02**	**0.72 ± 0.05**	**0.71 ± 0.02**	1,531 ± 1,835
CSV	0.74 ± 0.04	0.67 ± 0.04	0.66 ± 0.03	0.70 ± 0.02	809 ± 1,047
JSON-Records	0.75 ± 0.09	0.68 ± 0.02	0.71 ± 0.05	0.70 ± 0.03	1,560 ± 2,245
Markdown	0.74 ± 0.06	0.68 ± 0.04	0.68 ± 0.04	0.69 ± 0.03	1,131 ± 1,397
TabTree - Text-Augm. w/ Context-I.	0.67 ± 0.07	0.62 ± 0.06	0.57 ± 0.08	0.60 ± 0.07	8,232 ± 17,825

Table A.2.: Vanilla Table QA results for the baseline approaches HTML, CSV, JSON Records, Markdown and the best performing TabTree approach *Text Augmentation with Context-Intersection* on WikiTableQuestions (4 samples, 50 questions each).

Bibliography

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).

Boytsov, Leonid et al. (2013). "Engineering Efficient and Effective Non-metric Space Library". en. In: *Similarity Search and Applications*. Ed. by Nieves Brisaboa et al. Berlin, Heidelberg: Springer, pp. 280–293. ISBN: 978-3-642-41062-8. DOI: 10.1007/978-3-642-41062-8_28.

Brown, Tom B. et al. (July 2020). *Language Models are Few-Shot Learners*. arXiv:2005.14165 [cs]. DOI: 10.48550/arXiv.2005.14165. URL: http://arxiv.org/abs/2005.14165 (visited on 03/22/2025).

Carlyle, Thomas (1842). *Chartism*. en. Chapman and Hall.

Chalkidis, Ilias et al. (Oct. 2020). *LEGAL-BERT: The Muppets straight out of Law School*. arXiv:2010.02559 [cs]. DOI: 10.48550/arXiv.2010.02559. URL: http://arxiv.org/abs/2010.02559 (visited on 03/23/2025).

Chen, Si-An et al. (Dec. 2024). *TableRAG: Million-Token Table Understanding with Language Models*. arXiv:2410.04739 [cs]. DOI: 10.48550/arXiv.2410.04739. URL: http://arxiv.org/abs/2410.04739 (visited on 03/25/2025).

Chen, Jianlv et al. (June 2024). *BGE M3-Embedding: Multi-Lingual, Multi-Functionality, Multi-Granularity Text Embeddings Through Self-Knowledge Distillation*. arXiv:2402.03216 [cs]. DOI: 10.48550/arXiv.2402.03216. URL: http://arxiv.org/abs/2402.03216 (visited on 03/21/2025).

Chen, Wenhu (Jan. 2023). *Large Language Models are few(1)-shot Table Reasoners*. arXiv:2210.06710 [cs]. DOI: 10.48550/arXiv.2210.06710. URL: http://arxiv.org/abs/2210.06710 (visited on 03/19/2025).

Chen, Wenhu et al. (May 2021a). *HybridQA: A Dataset of Multi-Hop Question Answering over Tabular and Textual Data*. arXiv:2004.07347 [cs]. DOI: 10.48550/arXiv.2004.07347. URL: http://arxiv.org/abs/2004.07347 (visited on 03/24/2025).

Chen, Wenhu et al. (Feb. 2021b). *Open Question Answering over Tables and Text*. arXiv:2010.10439 [cs]. DOI: 10.48550/arXiv.2010.10439. URL: http://arxiv.org/abs/2010.10439 (visited on 03/24/2025).

Chen, Zhiyu et al. (May 2022). *FinQA: A Dataset of Numerical Reasoning over Financial Data*. arXiv:2109.00122 [cs]. DOI: 10.48550/arXiv.2109.00122. URL: http://arxiv.org/abs/2109.00122 (visited on 03/24/2025).

Cheng, Zhoujun et al. (Mar. 2022). *HiTab: A Hierarchical Table Dataset for Question Answering and Natural Language Generation*. arXiv:2108.06712 [cs]. DOI: 10.48550/arXiv.2108.06712. URL: http://arxiv.org/abs/2108.06712 (visited on 03/24/2025).

Chui, Michael et al. (2018). "Notes from the AI frontier: Insights from hundreds of use cases". In: *McKinsey Global Institute* 2.267, pp. 1–31. URL: https://www.mckinsey.com/featured-insights/artificial-intelligence/notes-from-the-ai-frontier-applications-and-value-of-deep-learning.

Cohere (Feb. 2023). *Introducing Embed v3*. en-US. URL: https://cohere.com/blog/introducing-embed-v3 (visited on 03/21/2025).

Cormack, Gordon V. et al. (July 2009). "Reciprocal rank fusion outperforms condorcet and individual rank learning methods". In: *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*. SIGIR '09. New York, NY, USA: Association for Computing Machinery, pp. 758–759. ISBN: 978-1-60558-483-6. DOI: 10.1145/1571941.1572114. URL: https://doi.org/10.1145/1571941.1572114 (visited on 03/21/2025).

Deng, Xiang et al. (Dec. 2020). *TURL: Table Understanding through Representation Learning*. arXiv:2006.14806 [cs]. DOI: 10.48550/arXiv.2006.14806. URL: http://arxiv.org/abs/2006.14806 (visited on 03/25/2025).

Devlin, Jacob et al. (May 2019). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. arXiv:1810.04805 [cs]. DOI: 10.48550/arXiv.1810.04805. URL: http://arxiv.org/abs/1810.04805 (visited on 03/23/2025).

Dinh, Tuan et al. (2022). *LIFT: Language-Interfaced Fine-Tuning for Non-Language Machine Learning Tasks*. arXiv: 2206.06565 [cs.LG]. URL: https://arxiv.org/abs/2206.06565.

Douze, Matthijs et al. (Feb. 2025). *The Faiss library*. arXiv:2401.08281 [cs]. DOI: 10.48550/arXiv.2401.08281. URL: http://arxiv.org/abs/2401.08281 (visited on 03/21/2025).

Du, Xinya et al. (Nov. 2022). *Retrieval-Augmented Generative Question Answering for Event Argument Extraction*. arXiv:2211.07067 [cs]. DOI: 10.48550/arXiv.2211.07067. URL: http://arxiv.org/abs/2211.07067 (visited on 03/22/2025).

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).

Gao, Luyu et al. (Dec. 2022). *Precise Zero-Shot Dense Retrieval without Relevance Labels*. arXiv:2212.10496 [cs]. DOI: 10.48550/arXiv.2212.10496. URL: http://arxiv.org/abs/2212.10496 (visited on 03/21/2025).

Gao, Yunfan et al. (Mar. 2024). *Retrieval-Augmented Generation for Large Language Models: A Survey*. arXiv:2312.10997 [cs]. DOI: 10.48550/arXiv.2312.10997. URL: http://arxiv.org/abs/2312.10997 (visited on 03/20/2025).

Gong, Heng et al. (Dec. 2020). "TableGPT: Few-shot Table-to-Text Generation with Table Structure Reconstruction and Content Matching". In: *Proceedings of the 28th International Conference on Computational Linguistics*. Ed. by Donia Scott et al. Barcelona, Spain (Online): International Committee on Computational Linguistics, pp. 1978–1988. DOI: 10.18653/v1/2020.coling-main.179. URL: https://aclanthology.org/2020.coling-main.179.

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).

Green Jr, Bert F et al. (1961). "Baseball: an automatic question-answerer". In: *Papers presented at the May 9-11, 1961, western joint IRE-AIEE-ACM computer conference*, pp. 219–224.

Hegselmann, Stefan et al. (Mar. 2023). *TabLLM: Few-shot Classification of Tabular Data with Large Language Models*. arXiv:2210.10723 [cs]. DOI: 10.48550/arXiv.2210.10723. URL: http://arxiv.org/abs/2210.10723 (visited on 03/25/2025).

Herzig, Jonathan et al. (2020). "TAPAS: Weakly Supervised Table Parsing via Pre-training". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. arXiv:2004.02349 [cs], pp. 4320–4333. DOI: 10.18653/v1/2020.acl-main.398. URL: http://arxiv.org/abs/2004.02349 (visited on 03/24/2025).

Herzig, Jonathan et al. (June 2021). *Open Domain Question Answering over Tables via Dense Retrieval*. arXiv:2103.12011 [cs]. DOI: 10.48550/arXiv.2103.12011. URL: http://arxiv.org/abs/2103.12011 (visited on 03/24/2025).

Hetz, M. J. et al. (Dec. 2024). "Superhuman performance on urology board questions using an explainable language model enhanced with European Association of Urology guidelines". English. In: *ESMO Real World Data and Digital Oncology* 6. Publisher: Elsevier. ISSN: 2949-8201. DOI: 10.1016/j.esmorw.2024.100078. URL: https:

//www.esmorwd.org/article/S2949-8201(24)00056-0/fulltext (visited on 03/23/2025).

Huang, Lei et al. (Mar. 2025). "A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions". In: *ACM Transactions on Information Systems* 43.2. arXiv:2311.05232 [cs], pp. 1–55. ISSN: 1046-8188, 1558-2868. DOI: 10.1145/3703155. URL: http://arxiv.org/abs/2311.05232 (visited on 03/20/2025).

INSEE (Nov. 2024). *Population estimates - All - Ville de Paris*. URL: https://www.insee.fr/en/statistiques/serie/001760155#Tableau (visited on 02/13/2025).

— (Jan. 2025). *Comparateur de territoires Région d'Île-de-France*. URL: https://www.insee.fr/fr/statistiques/1405599?geo=AAV2020-001+UU2020-00851+REG-11 (visited on 02/13/2025).

Iyyer, Mohit et al. (July 2017). "Search-based Neural Structured Learning for Sequential Question Answering". In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Regina Barzilay et al. Vancouver, Canada: Association for Computational Linguistics, pp. 1821–1831. DOI: 10.18653/v1/P17-1167. URL: https://aclanthology.org/P17-1167/ (visited on 03/24/2025).

Jaitly, Sukriti et al. (2023). *Towards Better Serialization of Tabular Data for Few-shot Classification with Large Language Models*. arXiv: 2312.12464 [cs.LG]. URL: https://arxiv.org/abs/2312.12464.

Jauhar, Sujay Kumar et al. (Feb. 2016). *TabMCQ: A Dataset of General Knowledge Tables and Multiple-choice Questions*. arXiv:1602.03960 [cs]. DOI: 10.48550/arXiv.1602.03960. URL: http://arxiv.org/abs/1602.03960 (visited on 03/24/2025).

Jin, Nengzheng et al. (July 2022). *A Survey on Table Question Answering: Recent Advances*. arXiv:2207.05270 [cs]. DOI: 10.48550/arXiv.2207.05270. URL: http://arxiv.org/abs/2207.05270 (visited on 03/23/2025).

Josh Howarth (Aug. 2024). *Number of Parameters in GPT-4 (Latest Data)*. en. URL: https://explodingtopics.com/blog/gpt-parameters (visited on 03/07/2025).

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

Katsis, Yannis et al. (June 2021). *AIT-QA: Question Answering Dataset over Complex Tables in the Airline Industry*. arXiv:2106.12944 [cs]. DOI: 10.48550/arXiv.2106.12944. URL: http://arxiv.org/abs/2106.12944 (visited on 03/24/2025).

Kim, Myung Jun et al. (May 2024). *CARTE: Pretraining and Transfer for Tabular Learning*. arXiv:2402.16785 [cs]. DOI: 10.48550/arXiv.2402.16785. URL: http://arxiv.org/abs/2402.16785 (visited on 03/24/2025).

Kong, Kezhi et al. (Apr. 2024). *OpenTab: Advancing Large Language Models as Open-domain Table Reasoners*. arXiv:2402.14361 [cs]. DOI: 10.48550/arXiv.2402.14361. URL: http://arxiv.org/abs/2402.14361 (visited on 03/25/2025).

Kweon, Sunjun et al. (May 2023). *Open-WikiTable: Dataset for Open Domain Question Answering with Complex Reasoning over Table*. arXiv:2305.07288 [cs]. DOI: 10.48550/arXiv.2305.07288. URL: http://arxiv.org/abs/2305.07288 (visited on 03/24/2025).

Kwiatkowski, Tom et al. (2019). "Natural Questions: A Benchmark for Question Answering Research". In: *Transactions of the Association for Computational Linguistics* 7. Ed. by Lillian Lee et al. Place: Cambridge, MA Publisher: MIT Press, pp. 452–466. DOI: 10.1162/tacl_a_00276. URL: https://aclanthology.org/Q19-1026/ (visited on 03/24/2025).

Lewis, Patrick et al. (Apr. 2021). *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. arXiv:2005.11401 [cs]. DOI: 10.48550/arXiv.2005.11401. URL: http://arxiv.org/abs/2005.11401 (visited on 03/20/2025).

Li, Peng et al. (Oct. 2023). *Table-GPT: Table-tuned GPT for Diverse Table Tasks*. arXiv:2310.09263 [cs]. DOI: 10.48550/arXiv.2310.09263. URL: http://arxiv.org/abs/2310.09263 (visited on 03/25/2025).

Li, Qianlong et al. (Dec. 2024). *GraphOTTER: Evolving LLM-based Graph Reasoning for Complex Table Question Answering*. arXiv:2412.01230 [cs]. DOI: 10.48550/arXiv.2412.01230. URL: http://arxiv.org/abs/2412.01230 (visited on 03/25/2025).

Li, Xiao et al. (Jan. 2021). *TSQA: Tabular Scenario Based Question Answering*. arXiv:2101.11429 [cs]. DOI: 10.48550/arXiv.2101.11429. URL: http://arxiv.org/abs/2101.11429 (visited on 03/24/2025).

Li, Xiaoqian et al. (Dec. 2023). *From Classification to Generation: Insights into Crosslingual Retrieval Augmented ICL*. arXiv:2311.06595 [cs]. DOI: 10.48550/arXiv.2311.06595. URL: http://arxiv.org/abs/2311.06595 (visited on 03/21/2025).

Li, Zehan et al. (Aug. 2023). *Towards General Text Embeddings with Multi-stage Contrastive Learning*. arXiv:2308.03281 [cs]. DOI: 10.48550/arXiv.2308.03281. URL: http://arxiv.org/abs/2308.03281 (visited on 03/21/2025).

Liu, Nelson F. et al. (Nov. 2023). *Lost in the Middle: How Language Models Use Long Contexts*. arXiv:2307.03172 [cs]. DOI: 10.48550/arXiv.2307.03172. URL: http://arxiv.org/abs/2307.03172 (visited on 03/20/2025).

Lu, Weizheng et al. (Feb. 2024). *Large Language Model for Table Processing: A Survey*. en. DOI: 10.1007/s11704-024-40763-6. URL: https://arxiv.org/abs/2402.05121v3 (visited on 03/19/2025).

Ma, Xinbei et al. (Oct. 2023). *Query Rewriting for Retrieval-Augmented Large Language Models*. arXiv:2305.14283 [cs]. DOI: 10.48550/arXiv.2305.14283. URL: http://arxiv.org/abs/2305.14283 (visited on 03/21/2025).

Minaee, Shervin et al. (Feb. 2024). *Large Language Models: A Survey*. arXiv:2402.06196 [cs]. DOI: 10.48550/arXiv.2402.06196. URL: http://arxiv.org/abs/2402.06196 (visited on 03/20/2025).

Muennighoff, Niklas et al. (Mar. 2023). *MTEB: Massive Text Embedding Benchmark*. arXiv:2210.07316 [cs]. DOI: 10.48550/arXiv.2210.07316. URL: http://arxiv.org/abs/2210.07316 (visited on 03/21/2025).

Nan, Linyong et al. (2021). *FeTaQA: Free-form Table Question Answering*. arXiv: 2104.00369 [cs.CL]. URL: https://arxiv.org/abs/2104.00369.

Nguyen, Isabelle (Jan. 2023). *Evaluating RAG Part I: How to Evaluate Document Retrieval*. en. URL: https://www.deepset.ai/blog/rag-evaluation-retrieval (visited on 03/07/2025).

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).

— (Oct. 2024b). *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).

— (Mar. 2024c). *New embedding models and API updates*. en-US. URL: https://openai.com/index/new-embedding-models-and-api-updates/ (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).

Qu, Renyi et al. (Oct. 2024). *Is Semantic Chunking Worth the Computational Cost?* arXiv:2410.13070 [cs]. DOI: 10.48550/arXiv.2410.13070. URL: http://arxiv.org/abs/2410.13070 (visited on 03/20/2025).

Rackauckas, Zackary (Feb. 2024). "RAG-Fusion: a New Take on Retrieval-Augmented Generation". In: *International Journal on Natural Language Computing* 13.1. arXiv:2402.03367 [cs], pp. 37–47. ISSN: 23194111. DOI: 10.5121/ijnlc.2024.13103. URL: http://arxiv.org/abs/2402.03367 (visited on 03/21/2025).

Reimers, Nils et al. (Aug. 2019). *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks*. arXiv:1908.10084 [cs]. DOI: 10.48550/arXiv.1908.10084. URL: http://arxiv.org/abs/1908.10084 (visited on 03/21/2025).

Richardson, Ben (June 2022). *Excel Facts & Statistics: Original Research - Acuity Training*. URL: https://www.acuitytraining.co.uk/news-tips/new-excel-facts-statistics-2022/ (visited on 03/19/2025).

Robertson, Stephen et al. (Apr. 2009). "The Probabilistic Relevance Framework: BM25 and Beyond". In: *Found. Trends Inf. Retr.* 3.4, pp. 333–389. ISSN: 1554-0669. DOI: 10.1561/1500000019. URL: https://doi.org/10.1561/1500000019 (visited on 03/20/2025).

Rogers, Anna et al. (Oct. 2023). "QA Dataset Explosion: A Taxonomy of NLP Resources for Question Answering and Reading Comprehension". In: *ACM Computing Surveys* 55.10. arXiv:2107.12708 [cs], pp. 1–45. ISSN: 0360-0300, 1557-7341. DOI: 10.1145/3560260. URL: http://arxiv.org/abs/2107.12708 (visited on 03/22/2025).

Roychowdhury, Sujoy et al. (Aug. 2024). *Evaluation of Table Representations to Answer Questions from Tables in Documents : A Case Study using 3GPP Specifications*. arXiv:2408.17008 [cs]. DOI: 10.48550/arXiv.2408.17008. URL: http://arxiv.org/abs/2408.17008 (visited on 03/25/2025).

Ruan, Yucheng et al. (Aug. 2024). *Language Modeling on Tabular Data: A Survey of Foundations, Techniques and Evolution*. arXiv:2408.10548 [cs]. DOI: 10.48550/arXiv.2408.10548. URL: http://arxiv.org/abs/2408.10548 (visited on 02/19/2025).

Sarthi, Parth et al. (Jan. 2024). *RAPTOR: Recursive Abstractive Processing for Tree-Organized Retrieval*. arXiv:2401.18059 [cs]. DOI: 10.48550/arXiv.2401.18059. URL: http://arxiv.org/abs/2401.18059 (visited on 03/20/2025).

Shuster, Kurt et al. (Apr. 2021). *Retrieval Augmentation Reduces Hallucination in Conversation*. arXiv:2104.07567 [cs]. DOI: 10.48550/arXiv.2104.07567. URL: http://arxiv.org/abs/2104.07567 (visited on 03/20/2025).

Shwartz-Ziv, Ravid et al. (June 2021). *Tabular Data: Deep Learning is Not All You Need*. en. URL: https://arxiv.org/abs/2106.03253v2 (visited on 03/19/2025).

Stamatescu, Luca (Apr. 2024). *The LLM Latency Guidebook: Optimizing Response Times for GenAI Applications | Microsoft Community Hub*. en-US. URL: https://techcommunity.microsoft.com/blog/azure-ai-services-blog/the-llm-latency-guidebook-optimizing-response-times-for-genai-applications/4131994 (visited on 04/01/2025).

Sui, Yuan et al. (July 2024). *Table Meets LLM: Can Large Language Models Understand Structured Table Data? A Benchmark and Empirical Study*. arXiv:2305.13062 [cs].

DOI: 10.48550/arXiv.2305.13062. URL: http://arxiv.org/abs/2305.13062 (visited on 02/12/2025).

Vaswani, Ashish et al. (2017). *Attention Is All You Need*. arXiv: 1706.03762 [cs.CL]. URL: https://arxiv.org/abs/1706.03762.

Wei, Jason et al. (Jan. 2023). *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*. arXiv:2201.11903 [cs]. DOI: 10.48550/arXiv.2201.11903. URL: http://arxiv.org/abs/2201.11903 (visited on 03/06/2025).

Woods, William A (1973). "Progress in natural language understanding: an application to lunar geology". In: *Proceedings of the June 4-8, 1973, national computer conference and exposition*, pp. 441–450.

Xu, Rongwu et al. (June 2024). *Knowledge Conflicts for LLMs: A Survey*. arXiv:2403.08319 [cs]. DOI: 10.48550/arXiv.2403.08319. URL: http://arxiv.org/abs/2403.08319 (visited on 03/20/2025).

Yao, Shunyu et al. (Dec. 2023). *Tree of Thoughts: Deliberate Problem Solving with Large Language Models*. arXiv:2305.10601 [cs]. DOI: 10.48550/arXiv.2305.10601. URL: http://arxiv.org/abs/2305.10601 (visited on 03/22/2025).

Yin, Pengcheng et al. (May 2020). *TaBERT: Pretraining for Joint Understanding of Textual and Tabular Data*. en. arXiv:2005.08314 [cs]. DOI: 10.48550/arXiv.2005.08314. URL: http://arxiv.org/abs/2005.08314 (visited on 03/24/2025).

Yin, Shukang et al. (Nov. 2024). "A Survey on Multimodal Large Language Models". In: *National Science Review* 11.12. arXiv:2306.13549 [cs], nwae403. ISSN: 2095-5138, 2053-714X. DOI: 10.1093/nsr/nwae403. URL: http://arxiv.org/abs/2306.13549 (visited on 03/20/2025).

Yu, Bowen et al. (2023). *Unified Language Representation for Question Answering over Text, Tables, and Images*. arXiv: 2306.16762 [cs.CL]. URL: https://arxiv.org/abs/2306.16762.

Yu, Tao et al. (Feb. 2019). *Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task*. arXiv:1809.08887 [cs]. DOI: 10.48550/arXiv.1809.08887. URL: http://arxiv.org/abs/1809.08887 (visited on 03/24/2025).

Zhang, Shuo et al. (Feb. 2020). *Web Table Extraction, Retrieval and Augmentation: A Survey*. en. arXiv:2002.00207 [cs]. DOI: 10.48550/arXiv.2002.00207. URL: http://arxiv.org/abs/2002.00207 (visited on 03/19/2025).

Zhang, Tianshu et al. (Apr. 2024). *TableLlama: Towards Open Large Generalist Models for Tables*. arXiv:2311.09206 [cs]. DOI: 10.48550/arXiv.2311.09206. URL: http://arxiv.org/abs/2311.09206 (visited on 03/25/2025).

Zhao, Yilun et al. (Aug. 2024). "TaPERA: Enhancing Faithfulness and Interpretability in Long-Form Table QA by Content Planning and Execution-based Reasoning". In: *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Lun-Wei Ku et al. Bangkok, Thailand: Association for Computational Linguistics, pp. 12824–12840. DOI: 10.18653/v1/2024.acl-long.692. URL: https://aclanthology.org/2024.acl-long.692/ (visited on 03/25/2025).

Zheng, Huaixiu Steven et al. (Mar. 2024). *Take a Step Back: Evoking Reasoning via Abstraction in Large Language Models*. arXiv:2310.06117 [cs]. DOI: 10.48550/arXiv.2310.06117. URL: http://arxiv.org/abs/2310.06117 (visited on 03/21/2025).

Zhong, Victor et al. (Nov. 2017). *Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning*. arXiv:1709.00103 [cs]. DOI: 10.48550/arXiv.1709.00103. URL: http://arxiv.org/abs/1709.00103 (visited on 03/24/2025).

Zhu, Fengbin et al. (June 2021). *TAT-QA: A Question Answering Benchmark on a Hybrid of Tabular and Textual Content in Finance*. arXiv:2105.07624 [cs]. DOI: 10.48550/arXiv.2105.07624. URL: http://arxiv.org/abs/2105.07624 (visited on 03/24/2025).

Zhu, Xiaohu et al. (Oct. 2024). *Large Language Model Enhanced Text-to-SQL Generation: A Survey*. arXiv:2410.06011 [cs]. DOI: 10.48550/arXiv.2410.06011. URL: http://arxiv.org/abs/2410.06011 (visited on 03/23/2025).