

Master's Thesis

**Enhancing Retrieval and LLM
Understanding of Structured Data for
Financial Table Question Answering**

Shanthini Malarvizhi

Examiners: Prof. Dr. Hannah Bast, Prof. Dr. Joschka

Boedecker

Advisers: Sebastian Walter,
Dr. Christoph Kommer,
Christian Schreckenberger,
Sonali Jha



Albert-Ludwigs-Universität Freiburg

Technische Fakultät

Institut für Informatik

Professur für Algorithmen und Datenstrukturen

February 13th, 2026

Declaration

I hereby declare that I am the sole author and composer of my thesis and that no other sources or learning aids, other than those listed, have been used. Furthermore, I declare that I have acknowledged the work of others by providing detailed references of said work.

I hereby also declare that my Thesis has not been prepared for another examination or assignment, either wholly or in part.

Germany, 13.02.2026

Place, Date



Signature

Abstract

Though the recent advances in Retrieval-Augmented Generation (RAG) and Large Language Models (LLMs) have improved reasoning over structured data, there are still challenges in retrieving and interpreting information from heterogeneous and multi-table datasets, especially in the financial domain.

This thesis examines three retrieval strategies for tables and different generation techniques using LLMs within a unified RAG-based Table QA framework. In the table retrieval stage, a SPARQL-based hybrid approach that combines retrieval over RDF-based knowledge graphs using SPARQL queries with semantic similarity filtering via FAISS vector search is proposed. On top of this, the study also compares BM25 retrieval and embedding-based retrieval methods across various table representations, such as linearized and context-augmented formats. In the generation stage, LLMs are evaluated for their ability to understand and reason over retrieved tabular data, which are presented in formats like `HTML`, `Markdown`, `Plaintext`, and `RDF triples`. In addition, different prompt designs for LLMs were experimented with to enhance the reasoning performance.

Experiments show that retrieval performance strongly depends on the choice of table representation, retrieval model, and dataset characteristics. Using tables in `plaintext` format, our dense retrieval approach improves Average Recall@30 by approximately 7–10% compared to the TableRAG baseline across finance and open-domain datasets. Furthermore, the SPARQL-based hybrid retrieval method supports fine-grained retrieval using `RDF triples` and achieves comparable or better performance, especially for questions involving multiple tables. Finally, the answer generation step results show that retrieval-augmented prompting with tables or `RDF triples` significantly improves LLM accuracy compared to zero-shot prompting. We also find that presenting tables in `Markdown` or `HTML` format improves LLM performance, and the effective prompt design further enhances the results of the GPT-4o model by placing the context before instructions and adding few-shot examples for `RDF triples`. Overall, using the best-performing LLM and prompt design for each dataset leads to significant improvement over the TableRAG baseline.

Contents

1. Introduction	1
1.1. Motivation	2
1.2. Problem Definition	3
1.3. Chapter Overview	4
2. Related Work	5
3. Background	9
3.1. Information Retrieval	9
3.2. Tables	10
3.3. Retrieval Augmented Generation	11
3.4. Table Retrieval	12
3.4.1. BM25 Retrieval	12
3.4.2. Dense Retrieval	13
3.4.3. Knowledge Graphs	16
3.5. Evaluation Metrics	19
3.5.1. Table Retrieval Metrics	19
3.5.2. Answer Generation Metrics	20
4. Approach	23
4.1. Table Linearization Formats	24
4.2. Context-Augmented Table Formats	28
4.3. Table Retrieval	32
4.3.1. BM25 Retrieval Method	32
4.3.2. Semantic Retrieval Method	33
4.4. SPARQL-based Hybrid Retrieval Method	34
4.5. Answer Generation	37
5. Experiments	39
5.1. Datasets and Benchmarks	39

5.2.	Table Retrieval using Linearized Table Formats	40
5.2.1.	BM25	40
5.2.2.	Semantic Retrieval	44
5.3.	Tables Retrieval using Context-Augmented Table formats	48
5.3.1.	BM25	48
5.3.2.	Semantic Retrieval	50
5.4.	SPARQL-based Hybrid Retrieval	53
5.4.1.	Comparison with Table Retrieval Methods Using Linearized Table Formats	54
5.5.	Answer Generation using Table Retrieval Results	56
5.5.1.	Prompt Settings	60
5.6.	Answer Generation using SPARQL-based Retrieval Results	66
5.6.1.	Prompt Settings	68
5.7.	Discussion	73
5.7.1.	Effect of Table Representation on Retrieval Performance	73
5.7.2.	Impact of Context Augmented Table Formats	74
5.7.3.	SPARQL-based Hybrid Retrieval	75
5.7.4.	Table-based Answer Generation and Prompt Design	75
5.7.5.	RDF Triple-Based Answer Generation and Prompt Design	76
5.7.6.	Summary	77
5.8.	Evaluation Against Existing Method	77
6.	Conclusion	81
	Bibliography	83
A.	Appendix	89
A.1.	Evaluation Against Additional Existing Method	89

List of Figures

1.	Illustrative distinction between (a) a normal table and (b) a hierarchical table. In the hierarchical table, the first row (highlighted in green) represents a category header, while the subsequent rows (highlighted in blue) contain data values associated with that category.	10
2.	Retrieval-Augmented Generation workflow.	11
3.	Illustration of an RDF triple connecting a <i>subject</i> to an <i>object</i> via a <i>predicate</i>	16
4.	Overview of our two-stage system.	23
5.	Illustration of different table linearization strategies applied to a hierarchical table. The first row (highlighted in green) represents a category header, and the subsequent rows (highlighted in blue) contain data values associated with that category.	26
6.	Illustration of different table linearization strategies applied to a normal table (no hierarchy).	27
7.	Illustration of different surrounding text strategies applied to the same table.	31
8.	Example of converting tables into RDF triples. (a) Normal Table to RDF conversion - The first column values are used as subjects, subsequent columns are mapped to predicates, and cell values are mapped as objects. (b) Hierarchical Table to RDF conversion - The category row (green) is written using <code>ex: group</code> , and the following rows (blue) are written as subjects, predicates, objects.	35
9.	Overview of the SPARQL-based hybrid retrieval pipeline combining SPARQL querying with semantic filtering.	36
10.	BM25 retrieval performance across different table formats for Enterprise Dataset Benchmark 1.	41
11.	BM25 retrieval performance across different table formats for Enterprise Dataset Benchmark 2.	42

12.	BM25 retrieval performance across different table formats for FinQA Dataset Benchmark.	42
13.	BM25 retrieval performance across different table formats for WikiTableQuestions Dataset Benchmark.	43
14.	Top-3 embedding models per format based on Average Recall@30 for Enterprise Dataset Benchmark 1.	44
15.	Top-3 embedding models per format based on Average Recall@30 for Enterprise Dataset Benchmark 2.	45
16.	Top-3 embedding models per format based on Average Recall@30 for the FinQA dataset benchmark.	46
17.	Top-3 embedding models per format based on Average Recall@30 for the WikiTableQuestions dataset benchmark.	47
18.	BM25 retrieval performance across different context-augmented table formats for Enterprise Dataset Benchmark 1.	48
19.	BM25 retrieval performance across different context-augmented table formats for Enterprise Dataset Benchmark 2.	49
20.	BM25 retrieval performance across different context-augmented table formats for FinQA Dataset Benchmark 2.	50
21.	Top-3 embedding models per context-augmented table format based on Average Recall@30 for Enterprise Dataset Benchmark 1.	51
22.	Top-3 embedding models per context-augmented table format based on Average Recall@30 for Enterprise Dataset Benchmark 2.	51
23.	Top-3 embedding models per context-augmented table format based on Average Recall@30 for FinQA Dataset Benchmark.	52
24.	Comparison of chunking strategies between the baseline and proposed method.	90

List of Tables

1.	Embedding models employed for vector-based retrieval, along with their output dimensions.	33
2.	Example rows of ontology-mapped table column names to RDF predicates.	34
3.	Summary of datasets and benchmarks used in experiments.	40
4.	SPARQL-based hybrid retrieval results across benchmarks, showing the number of questions sampled, the top-K triples retrieved, and the achieved average recall.	53
5.	Enterprise Dataset Benchmark 1 Retrieval Comparison	55
6.	Enterprise Dataset Benchmark 2 Retrieval Comparison	55
7.	FinQA Benchmark Retrieval Comparison	55
8.	WikiTableQuestions Benchmark Retrieval Comparison	55
9.	Weighted Accuracy (%) for Enterprise Dataset Benchmark 1 across zero-shot, BM25 retrieval, and Dense retrieval for different table formats.	57
10.	Weighted Accuracy (%) for Enterprise Dataset Benchmark 2 (multiple ground-truth tables per question) across zero-shot, BM25 retrieval, and Dense retrieval for different table formats.	58
11.	Weighted Accuracy (%) for the FinQA Benchmark across zero-shot, BM25 retrieval, and Dense retrieval for different table formats.	59
12.	Weighted Accuracy (%) for the WikiTableQuestions Benchmark across zero-shot, BM25 retrieval, and Dense retrieval for different table formats.	59
13.	Performance (in %) under different prompt settings for the Enterprise Dataset Benchmark 1.	62
14.	Performance (in %) under different prompt settings for the Enterprise Dataset Benchmark 2.	63
15.	Performance (in %) under different prompt settings for the FinQA Benchmark.	64
16.	Performance (in %) under different prompt settings for the WikiTableQuestions Benchmark.	65

17.	Answer generation performance (% weighted accuracy) using SPARQL-retrieved RDF triples as context in three presentation approaches: (1) sequential triples, (2) table-grouped triples, and (3) table- and subject-grouped triples.	67
18.	Answer generation performance (% weighted accuracy) for Approach 1 using SPARQL-retrieved RDF triples as context across different prompt settings.	69
19.	Answer generation performance (% weighted accuracy) for Approach 2 using SPARQL-retrieved RDF triples as context across different prompt settings.	70
20.	Answer generation performance (% weighted accuracy) for Approach 3 using SPARQL-retrieved RDF triples as context across different prompt settings.	71
21.	Performance results (Average Recall@30 in %) between baseline and our proposed retrieval approaches.	79
22.	Performance comparison (weighted accuracy in %) between the baseline and proposed answer generation approaches using GPT-4o.	80
23.	Comparison results for Enterprise Benchmark 1 (100 QA) between baseline and proposed retrieval approaches.	91
24.	Comparison results for Enterprise Benchmark 2 between baseline and proposed retrieval approaches.	92

List of Algorithms

1. Dense Retrieval for Document Chunks 15

1. Introduction

Table Question Answering (Table QA) [1] aims at answering natural language inquiries correctly by retrieving and utilizing data organized within tables. This approach becomes more instrumental in domains such as healthcare, finance, and transport, where the majority of operational information is stored in table-like spreadsheets or relational databases. By enabling users to ask questions directly in natural language, Table QA systems can be used by non-expert users to efficiently access and interpret structured data, thereby prompting data-driven understanding and reducing the need for manual querying and formal expertise in query languages like SQL.

A table provides two types of complementary information: *structural*, which is represented by rows, columns, and schemas, and *textual*, which consists of captions, headers, titles, and related contextual descriptions. Understanding how these two types of information interact is necessary to respond to queries from tables accurately. Despite this, some earlier methods have only examined textual and structural information individually [2], which prevents them from revealing intricate relationships in the data.

When the pertinent data is distributed across a large collection of tables rather than being confined to a single source, the problem in Table QA becomes even more challenging. This necessitates efficient methods for retrieving and evaluating the right data from potentially several tables [3]. Similarly, the traditional keyword-based retrieval models like BM25, which perform well when the user’s query closely matches table headers or cell entries, are prone to faltering as they deal with paraphrased questions, synonyms, or specialized phrases. Embedding-based techniques, which project user queries and tables into a shared semantic space, have been developed to overcome the limitations of keyword-based retrieval. As a result, meaning beyond precise lexical matching can be captured by the system.

Table QA has been significantly enhanced by the latest advances in Large Language Models (LLMs), which allow for more complex reasoning over structured data. One of the factors driving the development in this direction is Retrieval-Augmented Generation (RAG), which combines both retrieval and generation by enabling the

LLMs to use the retrieved tables or document segments from external knowledge sources as evidence to ground the model outputs, which substantially improves factual accuracy and enhances the overall reliability of the generated responses [4]. Nevertheless, scaling these approaches to complex, multi-table, or real-world datasets remains challenging, as current retrieval approaches often perform well only on simplified or curated benchmarks [1].

1.1. Motivation

Despite the substantial advancements in Table QA, achieving reliable, interpretable, and generalizable retrieval from structured data remains a persistent challenge. The existing retrieval methods [5, 6], including keyword-based search, embedding-based semantic similarity, and knowledge graph (KG) reasoning [7, 8], have all demonstrated effectiveness when tested separately in specific contexts, such as single- or multi-table benchmarks, or in representations that provide surrounding textual context. In order to assess these approaches, different datasets, preprocessing strategies, and experimental setups were used. As a result, there is a limited understanding of how various methods compare in consistent settings or how their advantages and disadvantages change depending on the data representation and contextual information level in the same setup.

The necessity to thoroughly examine various retrieval and generation techniques within a single RAG-based Table QA experimental framework spurred this thesis. In the retrieval step, this work extends beyond evaluating *keyword-based retrieval* and *embedding-based semantic similarity* by proposing a *hybrid symbolic reasoning method* that uses QLever to initially retrieve potential RDF triples from a KG using SPARQL queries, which are further filtered to keep only the pertinent ones based on semantic similarity measured through FAISS-based vector search. The comparison takes into account various formats for representing tables, such as HTML, CSV, and Markdown, among others, as well as by using the surrounding text, which is either presented alone or in conjunction with tables. The experimental evaluation focuses mainly on financial-domain datasets in both single-table and multi-table QA settings, with additional testing performed on an open-domain benchmark.

As part of the response generation step, the study explores the LLM’s ability to understand, contextualize, and reason over tabular inputs. The purpose of this step is to enhance the factual grounding, reliability, and transparency of the LLM-generated answer by analyzing the impact of retrieved context data and different prompt

designs.

Overall, this comprehensive analysis shows the effects of table representation, contextual information, and dataset features on retrieval performance as well as the ability of large language models to interpret, reason, and factually ground answers over tables, with a focus on prompt designs. This ultimately guides the development of more reliable, interpretable, and context-aware RAG-based Table QA systems for complex real-world data.

1.2. Problem Definition

Let's consider a user query denoted as Q and the knowledge source as a corpus of tables, which is represented as

$$T = \{T_1, T_2, T_3, \dots, T_n\}.$$

The objective of the system is to identify relevant evidence from this table corpus and generate contextually grounded, accurate answers. This process can be divided into two interdependent tasks:

- **Table Retrieval:** For the given query Q , the system must identify a subset of tables that are most relevant to answer the query. This process is represented as,

$$\text{Retrieval}(Q, \mathcal{T}) = \mathcal{T}'$$

$$\mathcal{T}' = \{T'_1, T'_2, T'_3, \dots, T'_m\}, \quad \text{where } \mathcal{T}' \subseteq \mathcal{T}, \quad m \leq n$$

If all tables in \mathcal{T} are relevant, then m may be equal to n . The retrieval process should only choose informative tables that maximize relevance while reducing noise.

- **LLM-based Inference:** Once the relevant subset of tables \mathcal{T}' are retrieved, the system combines the query Q with the retrieved tables to generate an answer A . This can be formally expressed as:

$$A = \text{LLM_Inference}(\text{PROMPT}(Q, \text{Retrieval}(Q, \mathcal{T}))),$$

Where $\text{Retrieval}(Q, \mathcal{T})$ denotes the subset of tables most relevant to the query, and $\text{PROMPT}(Q, \text{Retrieval}(Q, \mathcal{T}))$ denotes the constructed input prompt combining the query and retrieved table subset under different prompt designs.

The resulting response A should be factually correct by directly using the information from the context of the retrieved tables.

1.3. Chapter Overview

Having defined the problem context and the objectives of this thesis, the remaining chapters are organized as follows:

- **Chapter 2** provides a comprehensive review of the latest prior research works related to Table QA, highlighting key advancements and existing challenges that motivate this work.
- **Chapter 3** presents the theoretical background necessary to understand the core concepts, different retrieval methods, and metrics used for evaluation.
- **Chapter 4** describes the proposed system architecture and methodology, detailing preprocessing techniques, retrieval strategies, and LLM-based answer generation.
- **Chapter 5** outlines the datasets used and the experimental setup. It also presents and discusses the key findings, limitations, and a comparison with the existing method.
- **Chapter 6** concludes the thesis by summarizing the main contributions, final insights, and suggesting potential directions for future research.

2. Related Work

The task of question answering over structured data, such as tables and semi-structured data, has attracted considerable research attention. These approaches involve two key steps, namely, *retrieval* and *generation*. The retrieval step focuses on retrieving pertinent data, for which various techniques have been developed as discussed below. The generation step discusses how these data should be presented to the LLMs to produce accurate and hallucination-free answers.

In the work on **parsing and vectorizing semi-structured data for RAG** [7], documents are chunked based on section boundaries such as tables, text blocks, and images, which are then embedded using the *text – embedding – ada – 002* model, with the resulting representations stored in vector databases (Pinecone) for efficient retrieval. Alternatively, a tabular conversational system that uses dual-encoder embeddings and fine-grained cell-level ranking for table retrieval is demonstrated by **cTBLS** [9] approach, which achieves higher accuracy than BM25 baselines. Another method introduced a **syntactic- and structure-aware table retrieval** [10] that outperformed TF-IDF, BM25, and Bi-encoder baselines by explicitly encoding both syntactic and structural information, and aggregating embeddings using dot products and max-sum operations. These approaches highlight the importance of maintaining relational and structural data across tables to improve retrieval accuracy and support downstream reasoning.

With focus on the financial domain, a multi-path retrieval approach named **Fin-Sage** [11] combines metadata matching, dense embeddings, BM25, and HyDE with query paraphrasing and document re-ranking using a fine-tuned *BAAI/bge – reranker – v2 – Gemma* model to calculate a final score as the sum of a text match score and a time bonus derived from metadata to improve question answering. In case of hybrid documents containing both tabular and narrative text, **HD-RAG** [8] method integrates BM25 with embedding- and LLM-based retrieval using general and hierarchical table summaries.

All of the aforementioned techniques work well for single-table QA from hybrid sources, but they are insufficient for queries that require reasoning across multiple

tables. To address this challenge, the **Join-aware multi-table retrieval** [12] approach combines semantic similarity with join feasibility optimization, which improves table retrieval by 9.3% (F1 score) and end-to-end multiple table QA by 5.4%.

More recently, **TableRAG** [13] improves RAG for questions over heterogeneous documents by preserving tables using an SQL database and text in FAISS index instead of linearizing tables into text. At query time, questions are decomposed into subqueries, and relevant text or table content is retrieved. Finally, this enables accurate multi-hop reasoning and outperforms standard RAG methods on public benchmarks and the HeteQA dataset. Similarly, **Table-Augmented Generation (TAG)** [14] integrates database execution with language model reasoning for answering natural-language questions over structured data. It decomposes the task into query synthesis, query execution, and answer generation, which effectively unifies Text-to-SQL and RAG. By directly executing database queries, TAG allows LLMs to reliably handle tasks like aggregation and ranking, and also achieves better performance than existing methods.

Structured-GraphRAG [6] is another interesting retrieval strategy that goes beyond embedding- or keyword-based approaches. It transforms structured datasets into Knowledge Graphs, after which queries are translated into Cypher, using which graph segments are retrieved precisely. Compared to raw data processing, this method achieves 98% faster execution and also enhances answer accuracy by maintaining low-density KGs.

Extending beyond text and table-based retrieval, **RAG-Anything** [15] presents a unified RAG framework for heterogeneous data, including text, tables, images, and equations. It breaks documents into modality-aware units while preserving their relationships and builds two Knowledge Graphs, namely, a text-based and a cross-modal knowledge graph. These graphs are jointly indexed and retrieved using a hybrid of graph traversal and dense search, which enables structured and modality-aware question answering.

In the answer generation step of the task of question answering over structured data, even though LLMs have demonstrated impressive text understanding capabilities, reasoning over tabular data still remains challenging, particularly for multi-step or calculation-based questions.

In order to highlight the LLM’s potential for enhancing domain-specific reasoning using vector knowledge bases, the study **A Method for Parsing and Vectorization of Semi-Structured Data used in Retrieval-Augmented Generation** [7]

evaluated using GPT-4.0 on fifty questions, both in zero-shot settings and with retrieved context. It was seen that these vector representations significantly improve the LLM’s ability to generate accurate and technically detailed answers.

Focused on prompting strategies to guide LLMs in reasoning over structured data, **HD-RAG** [8] employs the prompting strategy called RECAP which stands for Restate the question (R), Extract relevant data (E), Compute the answer (C), Answer the question (A), and Present the calculation or result (P). The first four steps are applied to every question, while the final answer depends on the question type. For computation-based questions, use step P, whereas descriptive or reasoning questions use step A. This approach helps LLMs to systematically process the input and generate accurate responses. Similarly, **cTBLS** [9], a tabular conversational system, combines dialogue history, top-ranked table cells, and the current question into context for GPT-3.5 answer generation by maintaining continuity across interactions and incorporating relevant retrieved table information into reasoning.

To further examine the impact of table presentation on LLM comprehension, the study **Table meets LLM** [16] investigates seven task types like table partitioning, cell lookup, reverse lookup, column and row retrieval, size detection, and merged cell detection. Various table representations, such as CSV, JSON, XML, Markdown, HTML, and XLSX, are evaluated to assess how such formatting affects the performance of LLMs. Additionally, this paper shows a comparison between two prompting strategies, namely manual prompting and self-augmented prompting, in which the model first extracts relevant table information before generating answers. The key findings indicate that HTML formatting, role-based instructions, and careful input ordering substantially improve comprehension, and self-augmented prompting consistently outperforms the manual prompting approach.

Building on all previously mentioned insights, our approach integrates retrieval and answer generation into a unified pipeline. In order to understand how different table representations and text surrounding tables affect retrieval performance, a thorough analysis was conducted first for the retrieval step. Here, the tables are processed into multiple formats such as `HTML`, `markdown`, `plaintext`, among others, and are also augmented with surrounding text like headings, pretext, and posttext. Then, the retrieval methods like BM25, dense embeddings, and symbolic SPARQL retrieval over RDF triples in a knowledge graph, combined with vector search, were experimented on the processed formats. Next, for the answer generation step, two types of representations are used; the retrieved data can either be tables or SPARQL-retrieved RDF triples. These retrieved contexts are presented in different formats.

Additionally, we experiment with various prompt designs, including reordering, few-shot prompting, explicit identifiers, and zero-shot as a baseline. This work helps us understand the LLM’s ability to comprehend and reason over tables with an emphasis on various prompt designs and shows the impact of retrieved structured data on LLM reasoning.

3. Background

This chapter provides the foundational concepts and techniques that form the basis of this thesis, which includes information retrieval methods, table structures, knowledge graphs, the BM25 method, vector-based similarity search, and evaluation metrics. These concepts establish the technical context for subsequent chapters of this thesis.

3.1. Information Retrieval

Information Retrieval (IR) is commonly defined as the task of finding material, typically textual and unstructured, that satisfies a user's information need from a large digital collection [17]. Historically, IR has focused on document retrieval from large text corpora instead of querying relational databases directly.

Although traditional IR emphasizes unstructured data, the distinction between structured and unstructured information is often blurred. Unstructured data generally refers to content without a clear machine-readable structure, whereas structured data is organized into defined schemas like relational databases. In practice, textual data contains latent linguistic structure along with explicit organizational cues such as headings, sections, and markup (e.g., HTML tags).

This observation has motivated research into *semi-structured search* where both structural cues and textual content guide retrieval. For instance, querying documents whose titles contain "Java" while the body discusses "threading". Beyond semi-structured retrieval, *structured data retrieval* targets highly organized sources including relational databases, tabular data, and knowledge graphs.

Structured IR is increasingly important as most of the information in web and enterprise systems exists in structured or semi-structured forms rather than free text. Retrieving effectively from these sources requires methods that extend beyond lexical keyword matching and include schema awareness, semantic representations, and hybrid strategies that combine structured and unstructured data. This thesis follows in this direction by focusing specifically on table retrieval and knowledge graph-based retrieval of structured data.

3.2. Tables

Tables are one of the most widely used forms of structured data representation [18]. They organize information into a grid of rows and columns [2] where each row typically represents an entity, record, or observation, and each column corresponds to a specific attribute or variable. This tabular structure enables compact storage, human interpretation, and direct alignment with relational database schemas. Broadly, tables can be categorized into the following two types [19] [5],

- **Normal tables:** are flat two-dimensional structures in which each row contains an independent observation and each column represents a specific value corresponding to the observation. These are the most common types of tables that are often found in spreadsheets, databases, and open-domain resources such as Wikipedia.
- **Hierarchical tables:** are more complex structures that go beyond simple row and column structure. These include representations that capture hierarchical relationships within the data, like categories, subcategories, and subtotals, which makes them harder to linearize and retrieve effectively.

Name	City	Age	Salary
Smith	Freiburg	35	\$395
David	Berlin	41	\$295
Brown	Hamburg	22	\$260
Taylor	Munich	29	\$350

(a)

Account	2016	2017	2018
Net Income	615	585	595
Profit	671	654	695
Costs	562	678	706
Sales	1233	1332	1401

(b)

Figure 1.: Illustrative distinction between (a) a normal table and (b) a hierarchical table. In the hierarchical table, the first row (highlighted in green) represents a category header, while the subsequent rows (highlighted in blue) contain data values associated with that category.

Understanding these table structures is critical for effective information retrieval. For retrieval-based table QA, capturing both the content and the relationships within tables is essential, particularly in the context of LLMs. Structure-aware representations allow retrieval systems not only to identify relevant tables but also to provide models with rich contextual information that enables more accurate reasoning over tabular data [2].

3.3. Retrieval Augmented Generation

Retrieval-Augmented Generation (RAG) [20] enhances LLMs by retrieving the relevant information from a knowledge base and using that information to generate more accurate and context-aware responses. It works by integrating two main components, namely, retrieval and generation. The retrieval component searches the knowledge base to get the relevant context, which guides the generation process. Then, the generation component uses the retrieved information as an input to the LLM to enhance the accuracy by ensuring the responses are factually grounded.

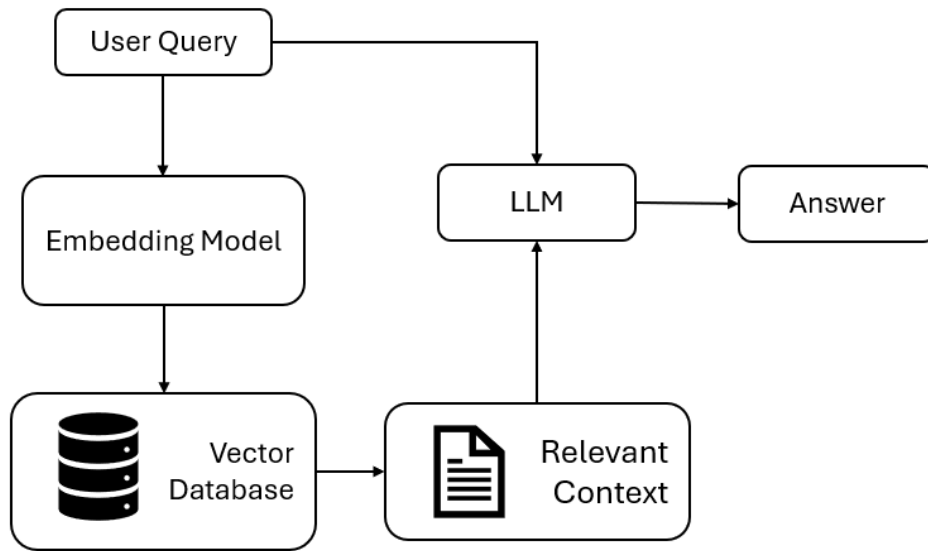


Figure 2.: Retrieval-Augmented Generation workflow.

The workflow of the RAG, as shown above in the Figure 2, begins with a user's question that is processed by an embedding model to create a query embedding. This is sent to the vector database that contains pre-indexed embeddings of data or documents, and the vector store finds the most relevant information related to the query using text similarity search. Finally, a prompt is created using the retrieved information (relevant context) and the query for the LLM to generate a response.

3.4. Table Retrieval

Table retrieval is a critical component of table-based question answering (QA) systems [12]. Here, given a natural language query, the goal is to identify the most relevant tables from a large collection. Unlike document retrieval, which primarily operates over unstructured text, table retrieval must handle both text and the structure of tables. As the retrieval stage directly influences the downstream answer generation task, improved retrieval accuracy significantly increases the final response accuracy. This highlights that retrieval has become a foundational step in retrieval-augmented QA pipelines.

In this work, we focus on two retrieval methods, namely, sparse retrieval (BM25) and dense retrieval, which leverage embedding-based semantic similarity.

3.4.1. BM25 Retrieval

In the information retrieval domain, BM25 (Best Match 25 or Okapi BM25) is a highly influential ranking function derived from the Okapi BM model. It also serves as a foundational baseline for numerous modern ranking algorithms [21, 22]. It belongs to the family of probabilistic term-based models and extends the classical Term Frequency–Inverse Document Frequency (TF–IDF) approach. This is a core component of search engines, question answering systems, and many retrieval pipelines. BM25 estimates the relevance of a document to a query by treating both as unordered collections of words under the *bag-of-words* assumption. Instead of modeling word order or proximity, it relies on the frequency and rarity of query terms as signals of relevance, which strikes a balance between computational efficiency and strong retrieval performance.

BM25 builds on two fundamental components,

- **Term Frequency (TF)**: measures how often a term appears in a document,

$$\text{TF}(t, d) = f(t, d) \tag{1}$$

where $f(t, d)$ is the frequency of term t in document d .

- **Inverse Document Frequency (IDF)**: downweights terms that occur in many documents and highlights the rare and more discriminative ones,

$$\text{IDF}(t) = \log \left(\frac{N - n_t + 0.5}{n_t + 0.5} + 1 \right) \tag{2}$$

where N is the total number of documents in the corpus and n_t is the number of documents containing t . The smoothing ensures numerical stability and avoids division by zero.

TF captures how important a term is in a single document, whereas IDF captures how unique a term is across the entire corpus. BM25 combines these two signals into a single scoring function that balances how often a term appears with how long the document is. Therefore, the final BM25 score for a document d with respect to a query Q is,

$$\text{BM25}(d, Q) = \sum_{t \in Q} \text{IDF}(t) \cdot \frac{f(t, d) \cdot (k_1 + 1)}{f(t, d) + k_1 \cdot \left(1 - b + b \cdot \frac{|d|}{\text{avgdl}}\right)} \quad (3)$$

where $|d|$ is the document length, avgdl is the average length in the corpus, and k_1 and b are tunable parameters for term frequency and length normalization, respectively. This scoring ensures that the influence of each term saturates with increasing frequency, and longer documents are treated fairly.

Although minor implementation variants of BM25 exist, we employ the standard Okapi BM25 variant provided by the `rank_bm25` Python package [23]. In this work, BM25 serves as a strong and efficient baseline to compare with advanced methods like dense and SPARQL-based hybrid retrievers.

3.4.2. Dense Retrieval

While sparse retrieval methods such as BM25 rely on exact lexical overlap, dense retrieval encodes queries and documents into a shared high-dimensional vector space using neural embeddings so that semantically similar pairs are positioned close to one another [24]. This enables retrieval even in the absence of exact keyword matches, and dense retrieval has become a core component of RAG pipelines for tasks like question answering and semantic search [25]. Models are trained with contrastive learning, which maximizes the similarity between positive query, document pairs, and minimizes it for negatives, and some also use masked language modeling to understand context and meaning in text better [26]. Pretrained encoders, including sentence-transformers and LLM-based models, perform remarkably well on retrieval and ranking tasks as demonstrated by the MTEB leaderboard [27].

Document Chunking and Embedding Generation

In order to handle long documents and improve retrieval, each document is divided into smaller chunks (e.g., passages, paragraphs, or sliding windows of tokens) [28]. Each chunk is linearized into a textual sequence and then transformed into a dense vector embedding, which is obtained either by training an embedding model on a large text corpus or by using a pretrained language model that has learned to represent words and sentences based on their surrounding context [29],

$$\mathbf{v}_i = \text{Encoder}(d_i), \quad \mathbf{q} = \text{Encoder}(q)$$

where d_i denotes the i -th chunk of document d , $\mathbf{v}_i \in \mathbb{R}^d$ is the embedding of that chunk, and $\mathbf{q} \in \mathbb{R}^d$ is the embedding of the query.

Cosine Similarity

Once both the query and document chunks are encoded into dense vector embeddings, the semantic relevance between a query and a document chunk is measured using cosine similarity [30], which measures the cosine of the angle between the two vectors and is mathematically defined as,

$$\text{sim}(\mathbf{q}, \mathbf{v}_i) = \frac{\mathbf{q} \cdot \mathbf{v}_i}{\|\mathbf{q}\| \|\mathbf{v}_i\|}$$

where $\mathbf{v}_i \in \mathbb{R}^d$ is the embedding of that chunk, and $\mathbf{q} \in \mathbb{R}^d$ is the embedding of the query.

A value of 1 means the vectors point in the same direction, and 0 means they are completely unrelated. In the case of text embeddings, each vector represents the semantic position of a question or a document chunk in the vector space. When the question and document chunk share similar meaning or context, their vectors point in similar directions, which results in a higher cosine similarity [30].

Dense Retrieval Algorithm

The dense retrieval workflow for document chunks and a query can be summarized as,

Algorithm 1 Dense Retrieval for Document Chunks

Require: query q , collection of documents D , top- k parameter

```
1: for each document  $d \in D$  do
2:   Split  $d$  into chunks  $d_i$                                 ▷ Chunking the document
3: end for
4: Encode query  $q$  into embedding vector  $\mathbf{q}$                 ▷ Query encoding
5: for each chunk  $d_i$  in all documents do
6:   Encode chunk into embedding vector  $\mathbf{v}_i$                 ▷ Chunk encoding
7:   Compute similarity score  $\text{sim}(\mathbf{q}, \mathbf{v}_i)$         ▷ Cosine similarity
8: end for
9: Rank all chunks by similarity
10: return top- $k$  chunks
```

For efficient similarity search and faster retrieval, the vector embeddings of document chunks are stored and indexed in vector databases such as FAISS, Milvus, Pinecone, and Weaviate. These databases use approximate nearest neighbor (ANN) search to quickly and accurately retrieve the vectors that are most similar to the given query embedding \mathbf{q} . Basically, the database calculates the distance between the query vector and vectors stored to return the most similar vectors or nearest neighbors to the query vector using the similarity metrics such as cosine similarity. Thus, vector databases play a key role in supporting RAG implementations by enabling scalable, flexible, and faster matching between user queries and semantically similar document embeddings [31].

Sentence Transformer Models

Sentence Transformers [32] are widely utilized for dense retrieval. They produce semantically meaningful fixed-size embeddings for sentences, paragraphs, or documents. Built on pre-trained language models such as BERT, RoBERTa, or DistilBERT [32], these models employ contrastive training objectives (e.g., triplet loss) to map semantically similar texts closer in the embedding space. They are available in both general-purpose and domain-specific variants and support diverse tasks. The Sentence-Transformers (SBERT) Python library offers a convenient interface for generating embeddings in RAG applications.

3.4.3. Knowledge Graphs

While dense retrieval effectively maps queries and documents into a shared semantic space, Knowledge Graphs (KGs) provide a complementary approach by representing the text as entities (such as people, places, or events) and relationships explicitly across heterogeneous datasets, which enables precise querying and reasoning [33]. KGs provide an organized representation of information in the form of a directed edge-labeled graph [34] where each node (or vertex) corresponds to an *entity* which is a real-world object such as a person, place, concept, or event and each directed edge represents a *relation* linking two entities [35]. The edge label specifies the type of relation, for example `bornIn`, `locatedAt`, or `authoredBy`. KGs enable structured storage, integration, and querying of knowledge, making them particularly suitable for tasks that require explicit semantics, reasoning, and explainability. A common standard way of representing such KGs is the Resource Description Framework (RDF) [34, 36], which expresses the given data as subject-predicate-object triples as shown in Figure 3.

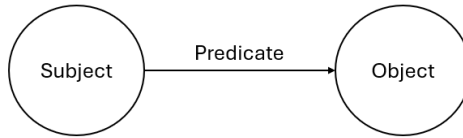


Figure 3.: Illustration of an RDF triple connecting a *subject* to an *object* via a *predicate*.

Resource Description Framework (RDF)

The RDF serves as a framework recommended by W3C for representing structured knowledge on the Semantic Web. These store information as a graph of *triples* using a subject-predicate-object model [36] where the subjects and objects are essentially nodes and the predicates are relationships as given below.

$$(s, p, o) \tag{4}$$

where,

- $s \in \mathcal{U} \cup \mathcal{B}$ is the **subject** which is either a URI/IRI or a blank node [34],
- $p \in \mathcal{U}$ is the **predicate** and always a URI/IRI representing the property or relation,

- $o \in \mathcal{U} \cup \mathcal{B} \cup \mathcal{L}$ is the **object** which can be a URI/IRI, a blank node, or a literal value.

Here,

- \mathcal{U} denotes the set of all URIs (or more generally, IRIs),
- \mathcal{B} denotes the set of blank nodes (anonymous resources),
- \mathcal{L} denotes the set of literals (e.g., strings, numbers, dates).

Internationalized Resource Identifiers (IRIs)

An Internationalized Resource Identifier (IRI) is a sequence of characters from the whole Unicode range, and it is a generalization of the Uniform Resource Identifier (URI). There is a mapping from IRIs to URIs, and hence, IRIs can be used instead of URIs appropriately to identify resources [37]. In RDF, IRIs are used to uniquely identify the following,

- **Entities**, e.g., <http://dbpedia.org/resource/Berlin>
- **Predicates or relations**, e.g., <https://dbpedia.org/ontology/country>
- **Units and unit scales**, e.g., <http://qudt.org/vocab/unit/M> for millions or <http://qudt.org/schema/qudt#unit> for general units

In addition, IRIs resemble URLs as they have the same structure with schemes like “http” and paths. IRIs are used in RDF to give each node a unique and web-accessible identity that links data directly to the internet [38].

Ontology

An ontology is a shared model that defines key concepts and their relationships within a specific domain. It provides a common vocabulary that helps align and connect data from different sources. By mapping entities and relationships in a knowledge graph to the concepts in an ontology, we can make the data easier to integrate, understand, and reason over [39]. Since we use heterogeneous tables from different data sources, we create our own ontology, which provides a structured vocabulary that is represented as predicates such as `hasColumnName` or `inYear`. This connects entities and their properties within the knowledge graph. While the table schema specifies how the data is stored, the ontology specifies what the data represents and

how different elements relate to each other. By mapping table columns to these ontology terms through our Excel file, we can generate RDF triples that organize data semantically, enabling richer querying, reasoning, and integration across datasets.

SPARQL: A Query Language for RDF

SPARQL (SPARQL Protocol And RDF Query Language) is the standard query language for RDF knowledge graphs [40]. A typical SPARQL query consists of a **SELECT** clause and a **WHERE** clause. The **WHERE** clause sets a pattern to be matched in the RDF graph, and the **SELECT** clause specifies the output variables. A question mark is used to prefix SPARQL variables such as `?person`. The **WHERE** clause can include the following,

- an **IRI** to identify a resource,
- a **literal** for a specific value,
- a **variable** to match resources or values.

The output of a SPARQL query is a result table with one column for each chosen variable in the **SELECT** clause. All the conditions in the **WHERE** clause are satisfied when a row in the table represents maps variables to RDF terms. In addition, **FILTER** constraint can be used to limit results using expressions like date ranges, string matching, and numeric comparisons [41].

Moreover, SPARQL also supports several advanced query features [42] as described below,

- **ORDER BY:** clause [41] is used to sort query results based on one or more selected variables in either ascending or descending order.
- **Aggregate Functions and GROUP BY:** SPARQL provides aggregate functions such as **COUNT**, **SUM**, **MIN**, and **MAX** [41]. These functions are used to compute numerical summaries over query results. The **GROUP BY** clause is used to divide the results into groups, and aggregates are then calculated for each group.
- **Subqueries:** Subqueries are nested SPARQL queries that appear inside a main query. They are mainly used to obtain intermediate results that cannot be easily computed using a single query.

- **OPTIONAL:** The **OPTIONAL** clause [41] allows parts of a query to be matched conditionally. If optional variables are not available, the result is still returned without eliminating the entire solution.
- **UNION:** The **UNION** operator is used to combine results from multiple alternative graph patterns into a single result set.

QLever: A SPARQL Engine

QLever (pronounced as "Clever") is a graph database that implements the standards of RDF and SPARQL. It is a high-performance SPARQL engine designed to load and query massive datasets. It is capable of indexing and efficiently querying over 100 billion RDF triples on a single standard PC or server [40]¹.

The following are the key features of QLever,

- **Better Performance** [43] by efficiently handling large intermediate and final results, which pose challenges for other engines like Blazegraph or Virtuoso,
- **Full-text search integration** [44] is enabled by search over associated literals within the RDF graph,
- **SPARQL autocompletion** [45] provides interactive support during query construction in the interface.

The architecture of QLever is designed to support high-throughput and low-latency querying. It is designed to be easy to use and is ideal for large-scale data exploration, semantic search, and interactive applications.

3.5. Evaluation Metrics

In the RAG-based Table QA system, evaluation of table retrieval and answer generation tasks requires metrics that can measure both relevance and correctness. Two main types of metrics are used in this thesis, namely, metrics for table retrieval and metrics for answer generation.

3.5.1. Table Retrieval Metrics

For table retrieval, standard IR metrics [46] such as **Recall@k** and **Precision@k** are employed to evaluate how effectively the retrieval model ranks relevant tables among the top- k results.

¹<https://github.com/ad-freiburg/qllever>

Recall@k measures the proportion of relevant tables that appear in the top- k retrieved set for a query q and is written as,

$$\text{Recall@}k(q) = \frac{|\mathcal{R}_k(q) \cap \mathcal{G}(q)|}{|\mathcal{G}(q)|} \quad (5)$$

where $\mathcal{R}_k(q)$ is the set of top- k retrieved tables for query q , and $\mathcal{G}(q)$ is the set of ground truth tables. Recall@k emphasizes coverage, which is particularly important in scenarios where missing a relevant table could lead to incorrect answers in downstream tasks.

Precision@k measures the proportion of retrieved tables among the top- k that are actually relevant and is written as,

$$\text{Precision@}k(q) = \frac{|\mathcal{R}_k(q) \cap \mathcal{G}(q)|}{|\mathcal{R}_k(q)|} \quad (6)$$

In order to evaluate the overall performance across multiple queries in the benchmark, the average Recall@k and average Precision@k are computed as follows,

$$\text{Average Recall@}k = \frac{1}{|Q|} \sum_{q \in Q} \text{Recall@}k(q) \quad (7)$$

$$\text{Average Precision@}k = \frac{1}{|Q|} \sum_{q \in Q} \text{Precision@}k(q) \quad (8)$$

Where Q denotes the set of all evaluation queries.

3.5.2. Answer Generation Metrics

For answer generation, a **weighted accuracy** metric is used. It assigns different scores depending on whether the generated answer fully matches the ground truth, partially matches it, or is incorrect. Then, the weighted score w_q is assigned as,

$$w_q = \begin{cases} 1, & \text{if } a_q \text{ is an exact match with } g_q \\ 0.5, & \text{if } a_q \text{ is a partial match with } g_q \\ 0, & \text{otherwise} \end{cases} \quad (9)$$

where a_q denotes the answer generated by the LLM and g_q is the ground truth answer for query q .

Therefore, the overall weighted accuracy across all the queries Q is computed as,

$$\text{Weighted Accuracy} = \frac{1}{|Q|} \sum_{q \in Q} w_q \quad (10)$$

Unlike the binary correct or incorrect scoring, this metric gives partial credit for answers that are partially correct or that closely match the ground truth, which is a common practice in RAG-based table QA and financial QA tasks.

4. Approach

The overall system follows a two-stage pipeline as illustrated in Figure 4, where the first stage focuses on retrieving relevant tables from the vector knowledge base, and the second stage involves generating responses using LLMs based on the retrieved data for the given user query.

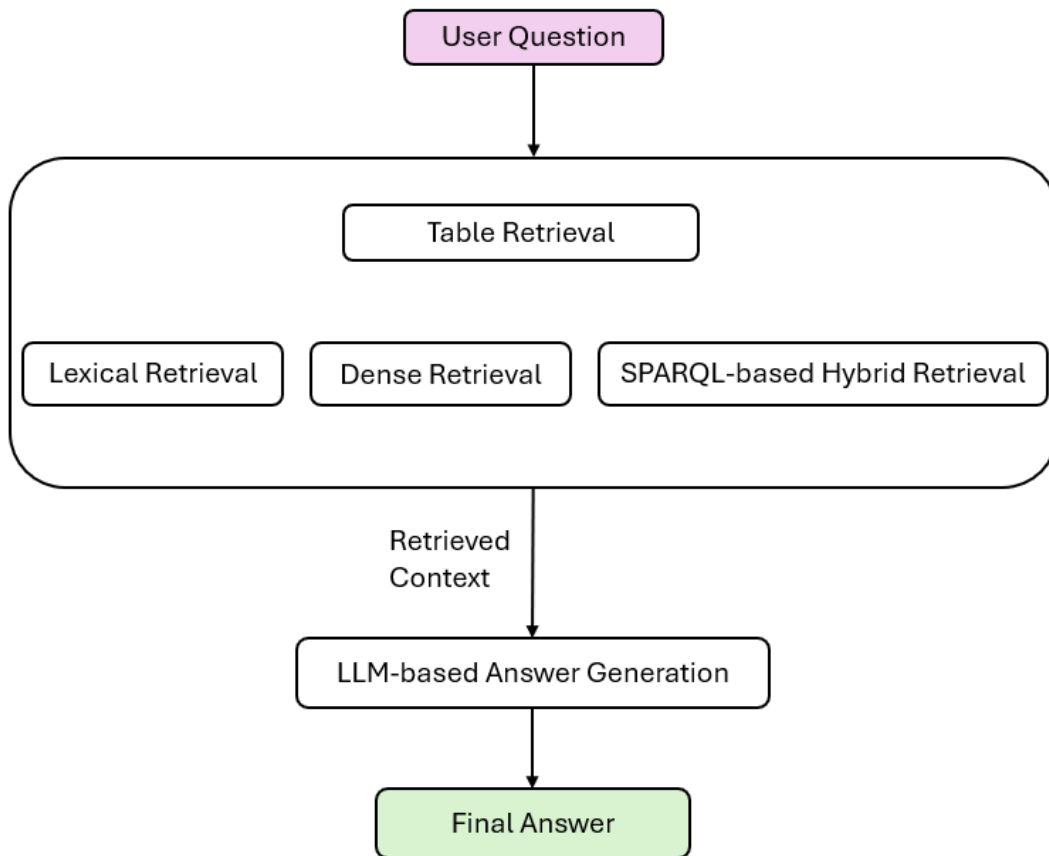


Figure 4.: Overview of our two-stage system.

Our primary focus is on the first stage, that is, table retrieval. Before this step, the large collection of tables is preprocessed into multiple formats through table

linearization and augmented with surrounding text. These processed tables are then indexed in the FAISS vector database for efficient retrieval. Lexical retrieval (BM25), semantic retrieval (embedding-based vector search), and a SPARQL-based hybrid approach that combines knowledge graph retrieval with vector-based filtering to select the most relevant RDF triples were all evaluated.

In the next answer generation stage, we look into how LLMs can effectively utilize the retrieved tables to improve their reasoning behaviors with better factual grounding, accuracy, and interpretability by also examining different prompt designs.

In the following sections, we first introduce the table linearization and context-augmentation formats of tables that are used in the retrieval experiments. Next, we go into detail about each retrieval approach, including lexical, semantic, and hybrid SPARQL-based methods. Finally, we outline the methodology for LLM-based answer generation.

4.1. Table Linearization Formats

Most of the retrieval and embedding models operate on unstructured data, whereas tables are structured data sources. Therefore, considering different assumptions about which information is most useful to the retriever, each table from the data source is converted into the following table linearization formats.

- **CSV format:** The most straightforward representation is where each row of the table, including headers, is written in comma-separated values without any explicit structural delimiters.
- **Header + First Column format:** In this representation, the column names are concatenated with the first-column values. In domains such as finance, the first column values are often useful for meaningful retrieval, whereas other columns primarily contain numerical or less descriptive data.
- **Markdown format:** The table rows including the headers are rendered in the `Markdown` syntax with pipe delimiters (`|`) between cells and `"|—|"` between the header and other rows. This format explicitly preserves the visual structure of the table.
- **HTML format:** The tables are converted into `HTML` syntax using the tags such as `<table>`, `<thead>`, and `<tbody>`. This representation allows retrieval models to distinguish headers from cell values explicitly using tags.

- **Plaintext format:** Each row is linearized into a string of `Column: Value` pairs that are separated by commas, which looks like a sentence. This format can be processed similarly to unstructured text by the models. In addition, to preserve column consistency, the empty cells and misaligned rows are padded.
- **Table Schema format:** In this format, we use an LLM (GPT-4) to generate summaries of each table instead of simply manipulating the table data. The LLM takes the table and its surrounding text, such as section headings, if available, as input to produce a brief summary that consists of the following,
 1. A description of what the table represents,
 2. The meaning of each column (including any abbreviations or symbols such as %, Δ \$,
 3. The first column values from the table, and
 4. One row from the table as an example.

This format is well-suited for embedding-based retrieval, as it represents tables as unstructured text.

- **Text format:** The table’s rows are represented in a *structured textual form* that explicitly writes the hierarchy in plain text. Here, each **category** or hierarchy header is explicitly marked with the keyword **Group:** followed by its name in the table, and all the rows that belong to that category are listed below with their associated values as described below,

```

Ungrouped:
<first column value>:
- has <column name>: <value> <unit/scale if applicable>
Group: <hierarchy category>
<first column value>:
- has <column name>: <value> <unit/scale if applicable>

```

This format clearly shows the grouping and relationships between values with an indication of hierarchy. Unlike the `plaintext` format, where the tables are flattened, and the hierarchies are potentially lost, this text format makes the hierarchy within tables explicit, which ultimately improves readability.

Account	2016	2017	2018
Net Income	615	585	595
Profit	671	654	695
Costs	562	678	706
Sales	1233	1332	1401

(a) Hierarchical Table

```
|account | 2016 | 2017 | 2018 |
|---|---|---|---|
|net Income | 615 | 585 | 595 |
|profit | 671 | 654 | 695 |
|costs | 562 | 678 | 706 |
|sales | 1233 | 1332 | 1401 |
```

(b) Markdown Format

```
account: net Income , 2016: 615 , 2017: 585 , 2018: 595
account: profit , 2016: 671 , 2017: 654 , 2018: 695
account: costs , 2016: 562 , 2017: 678 , 2018: 706
account: sales , 2016: 1233 , 2017: 1332 , 2018: 1401
```

(c) Plaintext Format

```
account 2016 2017 2018
net income profit costs sales
```

(d) Header + First Column Format

```
account,2016,2017,2018
net Income,615,585,595
profit,671,654,695
costs,562,678,706
sales,1233,1332,1401
```

(e) CSV Format

The table represents annual financial figures for a company over a span of years from 2016 to 2018. Each column in the table specifies a different financial account: the net income column shows the company's earnings after expenses and taxes, the profit column details total profit, the costs column shows total expenses, and the sales column presents total revenue from sales. The key values in the first column, which represents the years, include 2016, 2017, and 2018. Each row in the table presents data for a specific year; for example, in 2016, the company reported \$615 million in net income, \$671 million in profit, \$562 million in costs, and \$1,233 million in sales.

(f) Table Schema Format

```
<table>
<thead> <tr><th>account</th><th>2016</th><th>2017</th><th>2018</th></tr></thead>
<tbody>
<tr><td>net Income</td><td>615</td><td>585</td><td>595</td></tr>
<tr><td>profit</td><td>671</td><td>654</td><td>695</td></tr>
<tr><td>costs</td><td>562</td><td>678</td><td>706</td></tr>
<tr><td>sales</td><td>1233</td><td>1332</td><td>1401</td></tr>
</tbody>
</table>
```

(g) HTML Format

```
Ungrouped:
Net Income:
- has 2016: 615
- has 2017: 585
- has 2018: 595

Group: Net Income
Profit:
- has 2016: 671
- has 2017: 654
- has 2018: 695

Costs:
- has 2016: 562
- has 2017: 678
- has 2018: 706

Sales:
- has 2016: 1233
- has 2017: 1332
- has 2018: 1401
```

(h) Text Format

Figure 5.: Illustration of different table linearization strategies applied to a hierarchical table. The first row (highlighted in green) represents a category header, and the subsequent rows (highlighted in blue) contain data values associated with that category.

Year	Pension	Retiree medical and other
2011	574	173
2012	602	170
2013	665	169
2014	729	170
2015	785	173

(a) Normal Table

```
| year | pension | retiree medical and other |
|---|---|---|
| 2011 | 574 | 173 |
| 2012 | 602 | 170 |
| 2013 | 665 | 169 |
| 2014 | 729 | 170 |
| 2015 | 785 | 173 |
```

(b) Markdown Format

```
pension retiree medical and other
2011 2012 2013 2014 2015
```

(d) Header + First Column Format

```
year: 2011 , pension: 574 , retiree medical and other: 173
year: 2012 , pension: 602 , retiree medical and other: 170
year: 2013 , pension: 665 , retiree medical and other: 169
year: 2014 , pension: 729 , retiree medical and other: 170
year: 2015 , pension: 785 , retiree medical and other: 173
```

(c) Plaintext Format

```
year,pension,retiree medical and other
2011,574,173
2012,602,170
2013,665,169
2014,729,170
2015,785,173
```

(e) CSV Format

The table represents annual financial contributions (in millions of dollars) towards pension and retiree medical and other benefits over a span of years from 2011 to 2015, with a cumulative total for the years 2014 to 2015 also provided. Each column in the table specifies a different type of benefit contribution: the 'pension' column shows the amount contributed towards pension funds, while the 'retiree medical and other' column details the contributions towards retiree medical benefits and other similar expenses. The key values in the first column, which represents the years, include 2011, 2012, 2013, 2014, and 2015. Each row in the table presents data for a specific year or range of years; for example, in 2011, \$574 million was contributed to pensions and \$173 million to retiree medical and other benefits.

(f) Table Schema Format

```
<table>
<thead><tr><th>year</th><th>pension</th><th>retiree medical and other</th></tr></thead>
<tbody>
<tr><td>2011</td><td>574</td><td>173</td></tr>
<tr><td>2012</td><td>602</td><td>170</td></tr>
<tr><td>2013</td><td>665</td><td>169</td></tr>
<tr><td>2014</td><td>729</td><td>170</td></tr>
<tr><td>2015</td><td>785</td><td>173</td></tr>
</tbody>
</table>
```

(g) HTML Format

```
Ungrouped:
2011:
- has pension: 574
- has retiree medical and other: 173
2012:
- has pension: 602
- has retiree medical and other: 170
2013:
- has pension: 665
- has retiree medical and other: 169
2014:
- has pension: 729
- has retiree medical and other: 170
2015:
- has pension: 785
- has retiree medical and other: 173
```

(h) Text Format

Figure 6.: Illustration of different table linearization strategies applied to a normal table (no hierarchy).

The above figures 5 and 6 show examples of both normal and hierarchical tables represented using different linearization formats. In both cases, all formats handle the tables in the same way, except the text format, which explicitly represents the category header in hierarchical tables using the "Group" keyword.

4.2. Context-Augmented Table Formats

The above-discussed table linearization formats focus only on the tables, but many tables from financial and enterprise document sources appear within textual contexts such as section headings, subheadings, introductions, and notes. In order to test whether the surrounding text in documents enhances both retrieval and question-answering performance, we experimented with the following formats.

- **Combined Heading + Subheading:** The section heading and its immediate subheading (if present) are concatenated into a single descriptive sentence. This provides a brief overview of what the table discusses.
- **Combined Heading + Subheading + Table:** The section heading and subheading are concatenated with the `plaintext` [4.1] formatted table. This representation integrates both the overview and table-level information.
- **Pretext:** The paragraph or sentences that appear immediately before the table are taken as *Pretext*, which often contains an introductory description or narrative context that explains the table content.
- **Combined Heading + Subheading + Pretext:** The section heading and subheading (if present) are concatenated with the pretext, which provides both the overview and explanatory notes explaining the table content.
- **Combined Heading + Subheading + Pretext + Table:** The section heading, subheading, and pretext are concatenated with the table in `plaintext` [4.1] format, which provides the overview, explanation, and the table.
- **Combined Context:** The section heading, subheading, pretext, and posttext (text appearing immediately after the table) are concatenated to capture the full context surrounding the table, which may reference interpretations that are not contained in the table itself.
- **Combined Context + Table:** This representation is the most detailed, where heading, subheading, pretext, and posttext are all combined with the

€ billions, 2024 12/31/2023, 58.7 Add renewals, new contracts and modifications: - Cloud, 36.2 - Maintenance, 10.8 - Services and others, 5.1 Less revenue recognized in 2024: - Cloud, 17.1 - Maintenance, 11.3 - Services, 4.3 Acquisitions and divestments, 0.4 12/31/2024, 78.4
--

(a) CSV Table

Remaining Performance Obligations | Reconciliation of Remaining Performance Obligations

(b) Combined Heading + Subheading

Remaining Performance Obligations | Reconciliation of Remaining Performance Obligations | € billions: 12/31/2023, 2024: 58.7\n€ billions: Add renewals, new contracts and modifications: , 2024: \n€ billions: - Cloud , 2024: 36.2\n€ billions: - Maintenance , 2024: 10.8\n€ billions: - Services and others , 2024: 5.1\n€ billions: Less revenue recognized in 2024: , 2024: \n€ billions: - Cloud , 2024: 17.1\n€ billions: - Maintenance , 2024: 11.3\n€ billions: - Services , 2024: 4.3\n€ billions: Acquisitions and divestments , 2024: 0.4\n€ billions: 12/31/2024 , 2024: 78.4

(c) Combined Heading + Subheading + Table

Amounts of a customer contract's transaction price that are allocated to the remaining performance obligations represent contracted revenue that has not yet been recognized. They include amounts recognized as contract liabilities and amounts that are contracted but not yet due. The transaction price allocated to performance obligations that were unsatisfied or partially unsatisfied as at December 31, 2024, was \$78.4 billion (December 31, 2023: \$58.7 billion). The prior year's figure was corrected to reflect an increase of \$0.7 billion to include the transaction price from premium engagement contracts previously classified as having a termination-for-convenience-right. The transaction price thereof allocated to cloud performance obligations that were unsatisfied or partially unsatisfied (total cloud backlog) as at December 31, 2024, was \$63.3 billion (December 31, 2023: \$44.3 billion). The remaining amount mostly comprises obligations to provide software support services. The vast majority of software support contracts are contracts in the renewal phase that typically have a one-year contract term, while cloud subscription contracts typically are multiple-year contracts. The portion of remaining performance obligations related to services consists of non-cancelable revenue from contracts for projects with a predefined output and from premium engagements. Overall, approximately 40% of the total remaining performance obligations is expected to be recognized over the next 12 months following the respective balance sheet date. The transaction price allocated to remaining performance obligations varies between reporting periods, primarily due to the recognition of revenue for performance obligations outstanding at the end of the prior year and the addition of new contracts within SAP's cloud, maintenance, and services business. Other contributing factors include contract modifications, upsells, renewals, currency exchange rate fluctuations, and pricing adjustments. The revenue recognized in fiscal year 2024 for contract renewals, new contracts, and contract modifications is presented on a gross basis in the following reconciliation of outstanding performance obligations. This means that it is included as a positive entry under 'Add renewals, new contracts and modifications' and as a negative entry under 'Less revenue recognized in 2024.'

(d) Pretext

[Remaining Performance Obligations](#) | [Reconciliation of Remaining Performance Obligations](#) | Amounts of a customer contract's transaction price that are allocated to the remaining performance obligations represent contracted revenue that has not yet been recognized. They include amounts recognized as contract liabilities and amounts that are contracted but not yet due. The transaction price allocated to performance obligations that were unsatisfied or partially unsatisfied as at December 31, 2024, was \$78.4 billion (December 31, 2023: \$58.7 billion). The prior year's figure was corrected to reflect an increase of \$0.7 billion to include the transaction price from premium engagement contracts previously classified as having a termination-for-convenience-right. The transaction price thereof allocated to cloud performance obligations that were unsatisfied or partially unsatisfied (total cloud backlog) as at December 31, 2024, was \$63.3 billion (December 31, 2023: \$44.3 billion). The remaining amount mostly comprises obligations to provide software support services. The vast majority of software support contracts are contracts in the renewal phase that typically have a one-year contract term, while cloud subscription contracts typically are multiple-year contracts. The portion of remaining performance obligations related to services consists of non-cancelable revenue from contracts for projects with a predefined output and from premium engagements. Overall, approximately 40% of the total remaining performance obligations is expected to be recognized over the next 12 months following the respective balance sheet date. The transaction price allocated to remaining performance obligations varies between reporting periods, primarily due to the recognition of revenue for performance obligations outstanding at the end of the prior year and the addition of new contracts within SAP's cloud, maintenance, and services business. Other contributing factors include contract modifications, upsells, renewals, currency exchange rate fluctuations, and pricing adjustments. The revenue recognized in fiscal year 2024 for contract renewals, new contracts, and contract modifications is presented on a gross basis in the following reconciliation of outstanding performance obligations. This means that it is included as a positive entry under 'Add renewals, new contracts and modifications' and as a negative entry under 'Less revenue recognized in 2024.' | Others mainly comprises currency fluctuations for our services and maintenance contracts, price increases for our maintenance contracts, and changes to the remaining performance obligations for our software contracts. The 2024 acquisitions and divestments figure refers to the acquisition of WalkMe.

(e) Combined Heading + Subheading + Pretext

[Remaining Performance Obligations](#) | [Reconciliation of Remaining Performance Obligations](#) | Amounts of a customer contract's transaction price that are allocated to the remaining performance obligations represent contracted revenue that has not yet been recognized. They include amounts recognized as contract liabilities and amounts that are contracted but not yet due. The transaction price allocated to performance obligations that were unsatisfied or partially unsatisfied as at December 31, 2024, was \$78.4 billion (December 31, 2023: \$58.7 billion). The prior year's figure was corrected to reflect an increase of \$0.7 billion to include the transaction price from premium engagement contracts previously classified as having a termination-for-convenience-right. The transaction price thereof allocated to cloud performance obligations that were unsatisfied or partially unsatisfied (total cloud backlog) as at December 31, 2024, was \$63.3 billion (December 31, 2023: \$44.3 billion). The remaining amount mostly comprises obligations to provide software support services. The vast majority of software support contracts are contracts in the renewal phase that typically have a one-year contract term, while cloud subscription contracts typically are multiple-year contracts. The portion of remaining performance obligations related to services consists of non-cancelable revenue from contracts for projects with a predefined output and from premium engagements. Overall, approximately 40% of the total remaining performance obligations is expected to be recognized over the next 12 months following the respective balance sheet date. The transaction price allocated to remaining performance obligations varies between reporting periods, primarily due to the recognition of revenue for performance obligations outstanding at the end of the prior year and the addition of new contracts within SAP's cloud, maintenance, and services business. Other contributing factors include contract modifications, upsells, renewals, currency exchange rate fluctuations, and pricing adjustments. The revenue recognized in fiscal year 2024 for contract renewals, new contracts, and contract modifications is presented on a gross basis in the following reconciliation of outstanding performance obligations. This means that it is included as a positive entry under 'Add renewals, new contracts and modifications' and as a negative entry under 'Less revenue recognized in 2024.' | Others mainly comprises currency fluctuations for our services and maintenance contracts, price increases for our maintenance contracts, and changes to the remaining performance obligations for our software contracts. The 2024 acquisitions and divestments figure refers to the acquisition of WalkMe.

(f) Combined Context

Remaining Performance Obligations | Reconciliation of Remaining Performance Obligations | Amounts of a customer contract's transaction price that are allocated to the remaining performance obligations represent contracted revenue that has not yet been recognized. They include amounts recognized as contract liabilities and amounts that are contracted but not yet due. The transaction price allocated to performance obligations that were unsatisfied or partially unsatisfied as at December 31, 2024, was €78.4 billion (December 31, 2023: €58.7 billion). The prior year's figure was corrected to reflect an increase of €0.7 billion to include the transaction price from premium engagement contracts previously classified as having a termination-for-convenience-right. The transaction price thereof allocated to cloud performance obligations that were unsatisfied or partially unsatisfied (total cloud backlog) as at December 31, 2024, was €63.3 billion (December 31, 2023: €44.3 billion). The remaining amount mostly comprises obligations to provide software support services. The vast majority of software support contracts are contracts in the renewal phase that typically have a one-year contract term, while cloud subscription contracts typically are multiple-year contracts. The portion of remaining performance obligations related to services consists of non-cancelable revenue from contracts for projects with a predefined output and from premium engagements. Overall, approximately 40% of the total remaining performance obligations is expected to be recognized over the next 12 months following the respective balance sheet date. The transaction price allocated to remaining performance obligations varies between reporting periods, primarily due to the recognition of revenue for performance obligations outstanding at the end of the prior year and the addition of new contracts within SAP's cloud, maintenance, and services business. Other contributing factors include contract modifications, upsells, renewals, currency exchange rate fluctuations, and pricing adjustments. The revenue recognized in fiscal year 2024 for contract renewals, new contracts, and contract modifications is presented on a gross basis in the following reconciliation of outstanding performance obligations. This means that it is included as a positive entry under 'Add renewals, new contracts and modifications' and as a negative entry under 'Less revenue recognized in 2024'. | € billions: 12/31/2023 , 2024: 58.7\n€ billions: Add renewals, new contracts and modifications: , 2024: \n€ billions: - Cloud , 2024: 36.2\n€ billions: - Maintenance , 2024: 10.8\n€ billions: - Services and others , 2024: 5.1\n€ billions: Less revenue recognized in 2024: , 2024: \n€ billions: - Cloud , 2024: 17.1\n€ billions: - Maintenance , 2024: 11.3\n€ billions: - Services , 2024: 4.3\n€ billions: Acquisitions and divestments , 2024: 0.4\n€ billions: 12/31/2024 , 2024: 78.4

(g) Combined Heading + Subheading + Pretext + Table

Remaining Performance Obligations | Reconciliation of Remaining Performance Obligations | Amounts of a customer contract's transaction price that are allocated to the remaining performance obligations represent contracted revenue that has not yet been recognized. They include amounts recognized as contract liabilities and amounts that are contracted but not yet due. The transaction price allocated to performance obligations that were unsatisfied or partially unsatisfied as at December 31, 2024, was €78.4 billion (December 31, 2023: €58.7 billion). The prior year's figure was corrected to reflect an increase of €0.7 billion to include the transaction price from premium engagement contracts previously classified as having a termination-for-convenience-right. The transaction price thereof allocated to cloud performance obligations that were unsatisfied or partially unsatisfied (total cloud backlog) as at December 31, 2024, was €63.3 billion (December 31, 2023: €44.3 billion). The remaining amount mostly comprises obligations to provide software support services. The vast majority of software support contracts are contracts in the renewal phase that typically have a one-year contract term, while cloud subscription contracts typically are multiple-year contracts. The portion of remaining performance obligations related to services consists of non-cancelable revenue from contracts for projects with a predefined output and from premium engagements. Overall, approximately 40% of the total remaining performance obligations is expected to be recognized over the next 12 months following the respective balance sheet date. The transaction price allocated to remaining performance obligations varies between reporting periods, primarily due to the recognition of revenue for performance obligations outstanding at the end of the prior year and the addition of new contracts within SAP's cloud, maintenance, and services business. Other contributing factors include contract modifications, upsells, renewals, currency exchange rate fluctuations, and pricing adjustments. The revenue recognized in fiscal year 2024 for contract renewals, new contracts, and contract modifications is presented on a gross basis in the following reconciliation of outstanding performance obligations. This means that it is included as a positive entry under 'Add renewals, new contracts and modifications' and as a negative entry under 'Less revenue recognized in 2024'. | € billions: 12/31/2023 , 2024: 58.7\n€ billions: Add renewals, new contracts and modifications: , 2024: \n€ billions: - Cloud , 2024: 36.2\n€ billions: - Maintenance , 2024: 10.8\n€ billions: - Services and others , 2024: 5.1\n€ billions: Less revenue recognized in 2024: , 2024: \n€ billions: - Cloud , 2024: 17.1\n€ billions: - Maintenance , 2024: 11.3\n€ billions: - Services , 2024: 4.3\n€ billions: Acquisitions and divestments , 2024: 0.4\n€ billions: 12/31/2024 , 2024: 78.4 | Others mainly comprises currency fluctuations for our services and maintenance contracts, price increases for our maintenance contracts, and changes to the remaining performance obligations for our software contracts. The 2024 acquisitions and divestments figure refers to the acquisition of WalkMe.

(h) Combined Context + Table

Figure 7.: Illustration of different surrounding text strategies applied to the same table.

table in `plaintext` [4.1] format by providing document-level narrative with table data.

4.3. Table Retrieval

Once we have each table preprocessed into different linearized textual (Section 4.1) and context-augmented formats (Section 4.2), a textual corpus with each entry being mapped to its unique `table_id` was constructed. Now, the first and most critical component of our RAG-based Table QA pipeline is table retrieval, which retrieves the most relevant tables from a corpus of tables based on the given user queries. The retrieval outputs can be traced back to specific tables using the textual corpus. The implementation of the three retrieval methods used for evaluation is detailed in this section.

4.3.1. BM25 Retrieval Method

The BM25 (`BM250kapi`) algorithm was implemented using the `rank_bm25` Python library [23], which acts as a baseline for comparison with other retrieval methods in this work. For each table entry in the corpus, we take tables in one format, say `HTML` for indexing, and then retrieval was performed for every benchmark question from the indexed tables. This process was repeated for every format of the table in order to identify the best representation format that achieves the best retrieval performance in this method. The retrieval process is explained step by step as follows:

1. **Load Table Corpus:** The metadata file (textual corpus) is parsed to extract the desired table representations.
2. **Build BM25 Index:** The table representation for each table was tokenized to construct the BM25 index using `BM250kapi` algorithm.
3. **Query Processing:** The benchmark questions are preprocessed, like lowercasing and tokenizing, before passing them to the BM25 index.
4. **Ranking:** For each question, compute BM25 scores across all indexed tables and retrieve the top- k ($k \in \{5, 10, 20, 25, 30\}$) ranked tables.
5. **Evaluation:** The retrieved tables (mapped to their respective `table_ids`) are compared against ground truth table(s) (`table_id(s)`) present in the benchmark. Finally, we consider the retrieval step for a query to be successful when all the ground truth table(s) appear within the top-30 retrieved tables.

4.3.2. Semantic Retrieval Method

The next retrieval method employed was the semantic retrieval, which uses the dense vector representations of tables. This approach embeds both queries and tables into a shared high-dimensional space, where semantically related tables are positioned closely. The ranked relevant tables are retrieved by computing similarity scores between the query embedding and table embeddings. The quality of retrieval heavily depends on the embedding model used. Table 1 below lists the embedding models used (both open-source and proprietary API-based) and their respective output dimensions.

Model	Dimensions
text-embedding-ada-002	1536
text-embedding-3-large	32768
Qwen/Qwen3-Embedding-0.6B	32768
Salesforce/SFR-Embedding-Mistral	32768
sentence-transformers/multi-qa-mpnet-base-dot-v1	512
sentence-transformers/all-MiniLM-L12-v2	512
Snowflake/snowflake-arctic-embed-l-v2.0	8194
Alibaba-NLP/gte-base-en-v1.5	8192
intfloat/e5-base-v2	512
NovaSearch/stella_en_1.5B_v5	131072

Table 1.: Embedding models employed for vector-based retrieval, along with their output dimensions.

The following is the step-by-step vector-based retrieval process.

1. **Load and Index Corpus:** For each table in the metadata file, we take one table representation and generate embeddings, which were indexed in FAISS (Facebook AI Similarity Search) [47], a vector database that enables efficient nearest-neighbor search based on the query embedding.
2. **Query Processing:** Each query from the benchmark was processed using the same embedding model to generate a query embedding.
3. **FAISS Vector Search:** The query embedding is used by FAISS to retrieve the top- k ($k \in \{5, 10, 20, 25, 30\}$) ranked tables from the index based on semantic similarity.
4. **Evaluation:** The retrieved tables (`table_id(s)`) are compared against the ground truth `table_id(s)` present in the benchmark. Finally, we consider the

retrieval step for a query to be successful when all the ground truth table(s) appear within the top-30 retrieved tables.

The retrieval process is reiterated for each of the table formats present in the metadata file for a systematic comparison of embedding models and table formats, which helps us identify the model and format that delivers the highest retrieval accuracy.

4.4. SPARQL-based Hybrid Retrieval Method

As mentioned in the problem definition (see), we have so far discussed table retrieval. We will now discuss a retrieval method that uses a hybrid retrieval approach based on SPARQL. This approach converts tables into a knowledge graph that allows the retrieval of pertinent RDF triples (corresponding to relevant rows) rather than the retrieval of full tables, as in the case of the semantic and lexical strategies.

The tables must be converted into RDF triples before being ingested into KGs. In order to standardize this process, we built a specific ontology file for every dataset where the column names were transformed into predicates as `hasColumnName`. For example, `hasRevenue` is the predicate for the column name `Revenue`. The symbolic notations like Δ were replaced with their abbreviation as `hasDelta`. The temporal values, such as dates, years, or quarters, were encoded as `in{ColumnName}`. In the absence of table headers, the predicates were defined as `hasValue{columnNumber}`. Additionally, it should be noted that the first column values are considered as subjects, and only the second and subsequent columns were converted into predicates. When this ontology is used for triple generation, the predicate relationship across the diverse tabular datasets remains consistent and uniform. The table 2 below shows a few example rows from this ontology file.

Column Name	Predicate
pension	<code>hasPension</code>
1 - 2 years	<code>has1to2Years</code>
2013	<code>in2013</code>
12/9/2013	<code>in1292013</code>
Δ	<code>hasDelta</code>
Δ in %	<code>hasDeltaInPercent</code>

Table 2.: Example rows of ontology-mapped table column names to RDF predicates.

Year	Pension	Retiree medical and other
2011	574	173
2012	602	170
2013	665	169

(a)

```

@prefix data: <http://example.org/data#> .
@prefix ex: <http://example.org/ontology#> .
@prefix qudt: <http://qudt.org/schema/qudt/> .
@prefix unit: <http://qudt.org/vocab/unit/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

data:2011 ex:haspension [ a qudt:QuantityValue ;
  ex:note "POSITIVE" ;
  ex:signed_str "+574.00" ;
  qudt:numericValue 574.0 ;
  qudt:unit unit:UNIT ] ;
data:2011 ex:hasretireemedicalandother [ a qudt:QuantityValue ;
  ex:note "POSITIVE" ;
  ex:signed_str "+173.00" ;
  qudt:numericValue 173.0 ;
  qudt:unit unit:UNIT ] ;
data:2011 ex:label "2011" ;

data:2012 ex:haspension [ a qudt:QuantityValue ;
  ex:note "POSITIVE" ;
  ex:signed_str "+602.00" ;
  qudt:numericValue 602.0 ;
  qudt:unit unit:UNIT ] ;
data:2012 ex:hasretireemedicalandother [ a qudt:QuantityValue ;
  ex:note "POSITIVE" ;
  ex:signed_str "+170.00" ;
  qudt:numericValue 170.0 ;
  qudt:unit unit:UNIT ] ;
data:2012 ex:label "2012" ;

data:2013 ex:haspension [ a qudt:QuantityValue ;
  ex:note "POSITIVE" ;
  ex:signed_str "+665.00" ;
  qudt:numericValue 665.0 ;
  qudt:unit unit:UNIT ] ;
data:2013 ex:hasretireemedicalandother [ a qudt:QuantityValue ;
  ex:note "POSITIVE" ;
  ex:signed_str "+169.00" ;
  qudt:numericValue 169.0 ;
  qudt:unit unit:UNIT ] ;
data:2013 ex:label "2013" ;

```

Account	2016	2017
Net Income	615	585
Profit	671	654
Costs	562	678

(b)

```

@prefix data: <http://example.org/data#> .
@prefix ex: <http://example.org/ontology#> .
@prefix qudt: <http://qudt.org/schema/qudt/> .
@prefix unit: <http://qudt.org/vocab/unit/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

data:profit ex:group data:net_income ;
data:profit ex:in2016 [ a qudt:QuantityValue ;
  ex:note "POSITIVE" ;
  ex:signed_str "+671.00" ;
  qudt:numericValue 671.0 ;
  qudt:unit unit:UNIT ] ;
data:profit ex:in2017 [ a qudt:QuantityValue ;
  ex:note "POSITIVE" ;
  ex:signed_str "+654.00" ;
  qudt:numericValue 654.0 ;
  qudt:unit unit:UNIT ] ;
data:profit ex:label "Profit" ;

data:costs ex:group data:net_income ;
data:costs ex:in2016 [ a qudt:QuantityValue ;
  ex:note "POSITIVE" ;
  ex:signed_str "+562.00" ;
  qudt:numericValue 562.0 ;
  qudt:unit unit:UNIT ] ;
data:costs ex:in2017 [ a qudt:QuantityValue ;
  ex:note "POSITIVE" ;
  ex:signed_str "+678.00" ;
  qudt:numericValue 678.0 ;
  qudt:unit unit:UNIT ] ;
data:costs ex:label "Costs" ;

data:net_income a ex:Group ;
data:net_income ex:in2016 [ a qudt:QuantityValue ;
  ex:note "POSITIVE" ;
  ex:signed_str "+615.00" ;
  qudt:numericValue 615.0 ;
  qudt:unit unit:UNIT ] ;
data:net_income ex:in2017 [ a qudt:QuantityValue ;
  ex:note "POSITIVE" ;
  ex:signed_str "+585.00" ;
  qudt:numericValue 585.0 ;
  qudt:unit unit:UNIT ] ;
data:net_income ex:label "Net Income" ;

```

Figure 8.: Example of converting tables into RDF triples. (a) Normal Table to RDF conversion - The first column values are used as subjects, subsequent columns are mapped to predicates, and cell values are mapped as objects. (b) Hierarchical Table to RDF conversion - The category row (green) is written using `ex: group`, and the following rows (blue) are written as subjects, predicates, objects.

To complete the triple generation from tables with full Subject-Predicate-Object relationships for all the tables, we then take each table row and map the first column values as subjects, the predicates were taken from the ontology file, and the corresponding cell values were used as objects. Furthermore, the numerical cell values are represented with explicit units and scale by incorporating the QUDT ontology (`qudt:` and `unit:`). We also standardize unit scales such as millions, billions, and thousands using the acronyms "M", "B", and "T" respectively. Since some tables in the datasets have hierarchical relationships, we need to maintain these links using the `ex: group` predicate. Finally, all transformed tables are written into `.ttl` files and are ingested into a KG in the QLever engine. An example of table-to-RDF conversion is shown in Figure 8 above, where the table rows are transformed into Subject-Predicate-Object triples.

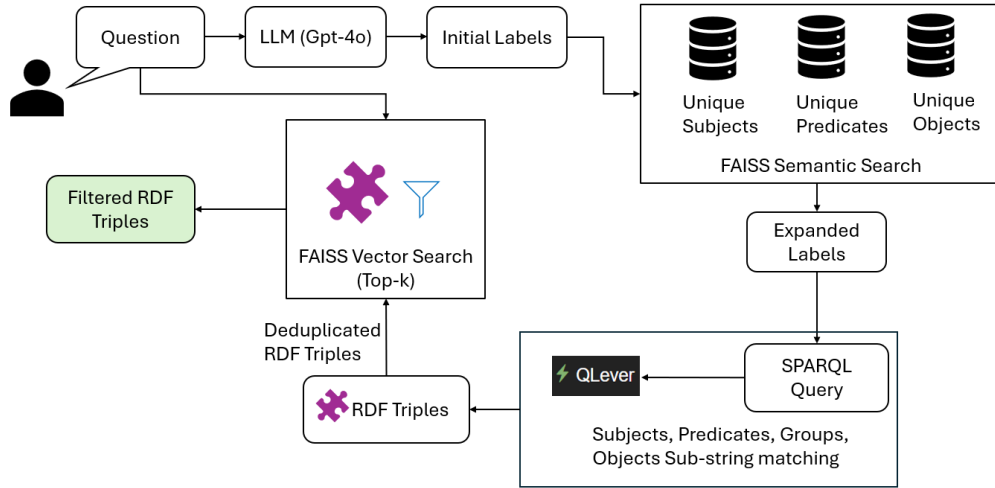


Figure 9.: Overview of the SPARQL-based hybrid retrieval pipeline combining SPARQL querying with semantic filtering.

Once the generated RDF triples are ingested into QLever as a KG, we proceed with retrieval. The retrieval pipeline, which integrates SPARQL querying with semantic filtering as shown in the above Figure 9, consists of the following steps:

- **Step 1:** For each benchmark question, an LLM (`gpt-4o`) identified labels, entities, or terms based on the user query.
- **Step 2:** We take the unique subjects, predicates, and groups stored in the knowledge graph to create three FAISS indexes. Through semantic similarity search ($k = 5$) using `text-embedding-large3` based on the query, the initial

label sets are expanded with semantically related subjects, predicates, and groups.

- **Step 3:** The expanded label set is used to retrieve relevant RDF triples from the KG using substring matching over subjects, predicates, objects, and groups by executing SPARQL queries. Each RDF triple result had a numerical value, unit, note, period, and unit scale to cover the entire cell information.
- **Step 4:** In order to limit the huge set of retrieved RDF triples to the most relevant triples based on the question, a FAISS index was created to filter triples (without duplicates) using the `text-embedding-large3` model. This filtering step gives a subset of triples, and the subset size was chosen proportionally to the size of 30 tables (approximated by the total number of cells) to compare with the top-30 retrieval approach used in other retrieval methods.
- **Step 5:** The final subset of triples is compared against the ground truth triples present for each question in the benchmark. Finally, recall was calculated as the proportion of ground truth triples present in the final subset of triples, and we consider the retrieval to be fully successful when all ground truth triples appeared within the filtered subset. The overall performance is measured using the average recall metric across all the questions.

4.5. Answer Generation

The next stage of our RAG-based table QA pipeline is answer generation, where retrieved tables or RDF triples are used as context for the large language models to generate accurate responses. In this stage, we used two state-of-the-art LLMs GPT-4, and GPT-4o due to their demonstrated strong performance on reasoning and comprehension tasks, and their capability to handle long-context inputs [1].

The models are first evaluated in a zero-shot setting where the answers for each question are generated without any context being provided. This acts as a baseline for comparison for the following evaluation. In case of table retrieval results using BM25 and embedding-based retrieval, we get the retrieved tables for each question from the top-performing format, which are then presented in different formats (`HTML`, `Markdown`, `plaintext`) as context to the LLMs, along with the question to examine which table representation best supports the LLM’s understanding of structured data. Similarly, for SPARQL-based hybrid retrieval results, the final subset of RDF triples is provided as a context along with the question to the LLM for answer generation.

The RDF triples are presented either sequentially, one by one, or grouped based on the tables they belong to, which gives additional information for the LLMs to understand this RDF representation of structured data. Additionally, we assess the impact of various prompt designs, including explicit identifiers, reordering inputs, and few-shot examples, on the answer generation of LLMs for both table-based and SPARQL-based retrieved results.

The answer generated by each model for a given question is compared against its corresponding ground truth answer present in the benchmark using an LLM (GPT-4), which acts as a judge to determine whether the generated answer and ground truth answer are an exact match, a partial match, or a no match. The outputs from the LLM judge are then used to compute a weighted accuracy score across all the questions in the benchmark. This provides a quantitative measure of each LLM’s ability to generate correct and complete answers in all the above-discussed settings, i.e., with or without context. Overall, this stage shows the direct influence of retrieval quality on the LLMs to generate factually grounded and context-aware responses. Moreover, it also provides a comprehensive comparison of which representation of tables or RDF triples enhances the LLM’s understanding of structured data and how the overall accuracy of this stage can be further boosted by using the appropriate prompt design.

5. Experiments

This chapter presents the experimental setup and results. First, we introduce the datasets used in our study, and these datasets are converted to different linearization and context-augmented formats, which are then used for evaluation using the retrieval methods introduced in the previous chapter, including lexical and semantic similarity approaches, and retrieval using a SPARQL-based approach is also implemented. Next, the answer generation is assessed by feeding the retrieved results into LLMs under different prompt designs, and the performance is measured using weighted accuracy. Finally, the best performing retrieval strategy is identified and compared against the baseline on the primary financial enterprise dataset.

5.1. Datasets and Benchmarks

For the experimental evaluation in this thesis, three datasets that contain both normal and hierarchical tables were used. The primary dataset is a private enterprise finance benchmark, and the next two datasets are finance domain and open-domain benchmarks that are publicly available.

The first is a financial domain **private enterprise dataset** that contains 397 small to medium-sized tables. We have two benchmarks under this dataset. The first benchmark’s questions have a single ground truth table (1,187 train and 1,135 test questions), whereas the second benchmark’s questions have multiple ground truth tables (96 train and 60 test questions). This set of two benchmarks reflects the evaluation complexity by having queries that have one or more relevant tables similar to real-world scenarios.

FinQA [48], the second dataset is also a financial benchmark that comprises tables, questions, and answers extracted from financial reports written by experts. The entire benchmark contains 2,788 tables and around 8,000 questions (train, dev, and test). For this work, a subset of the benchmark was used, and the selected subset contains 2,515 small to medium-sized tables and includes 3,983 train, 566 dev, and 729 test questions. The questions require multi-step numerical reasoning instead of a

simple data lookup, which is the common case in the financial datasets.

The third dataset is **WikiTableQuestions** [49]¹. It is derived from Wikipedia and covers diverse domains such as sports, politics, films, climate, and chemistry. Similar to FinQA, only a subset of the entire dataset was used. The selected subset consists of 1,116 medium to large-sized tables and includes 2,422 train, 2,086 dev, and 1,812 test questions. This dataset is used to test the generalization of retrieval and generation methods beyond the finance domain.

An overview of these datasets, including the number of tables, number of hierarchical tables, and train/dev/test splits, is presented in Table 3.

Dataset	Tables	Hierarchical Tables	Train / Dev / Test	Domain
Enterprise (Benchmark 1)	397	120	1,187 / - / 1,135	Finance
Enterprise (Benchmark 2)	397	120	96 / - / 60	Finance
FinQA	2,515	4	3,983 / 566 / 729	Finance
WikiTableQuestions	1,116	6	2,422 / 2,086 / 1,812	Open-domain

Table 3.: Summary of datasets and benchmarks used in experiments.

5.2. Table Retrieval using Linearized Table Formats

This section presents the evaluation of BM25 and dense retrieval across all four benchmarks using the table representation formats introduced in Section 4.1.

5.2.1. BM25

For the **Enterprise Dataset Benchmark 1**, there are considerable variations in the retrieval performance of top-30 tables seen across the table formats in Figure 10. The `header first column` format achieved the highest average recall@30 of **0.901** followed by `table schema` (**0.887**) which indicates that BM25 performs best when the tables are expressed in concise but informative context with high lexical overlap with the queries as the benchmark questions mainly depend on the headers and first column values of the tables. The `Markdown` format also achieved a comparatively high average recall of **0.884**, which shows that clear representations of tables with delimiters such as "|" help BM25 to perform exact token matching effectively. In contrast, the `CSV` (**0.752**) and `text` (**0.725**) formats produced only moderate results, as the CSV format has text dispersing tokens and inconsistent line breaks that dilute the term frequency. The lowest performing format is `HTML` (**0.659**), where

¹<https://github.com/ppasupat/WikiTableQuestions>

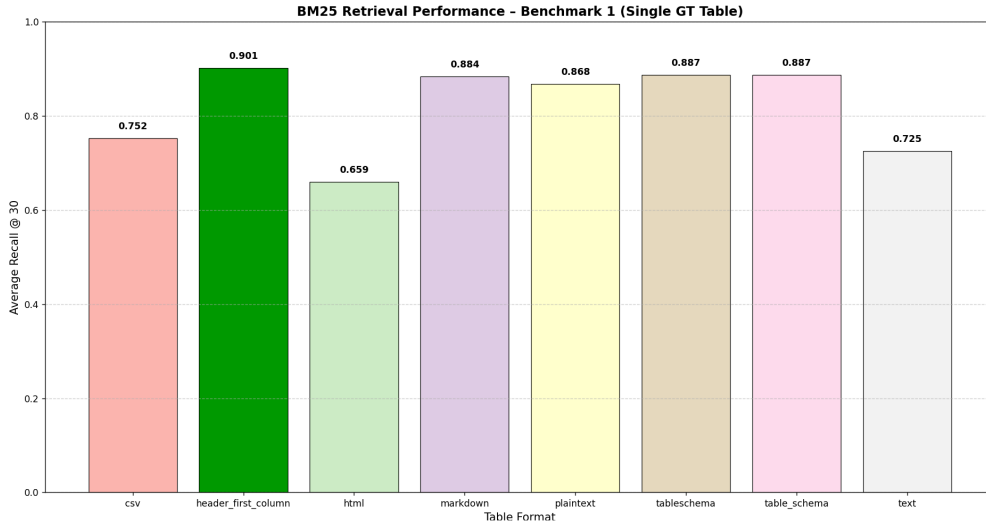


Figure 10.: BM25 retrieval performance across different table formats for Enterprise Dataset Benchmark 1.

the markup tags increase the table length, which leads to length normalization of the BM25 algorithm. Overall, these findings confirm that BM25 benefits most from simplified, linearized representations that reduce noise while preserving information.

For the **Enterprise Dataset Benchmark 2**, where each question is linked to multiple ground-truth tables, the retrieval performance of top-30 tables (Figure 11) varied across the table formats. The `table schema` format achieved the highest average recall@30 of **0.771**, followed by the `header first column` format (**0.719**), which shows that the short descriptive format with headers, first column values, and example rows helps BM25 to identify relevant tables based on the queries. Since for each question we have multiple ground truth tables, the description of what this table is helps to get all relevant tables together based on the query. The `Markdown` (**0.651**), `plaintext` (**0.637**), and `text` (**0.561**) formats provided moderate results, whereas `CSV` (**0.528**) and `HTML` (**0.469**) performed the lowest due to tokenization noise and markup interference. Overall, these findings indicate in complex multi-table scenarios that BM25 benefits most from a table format that preserves short but repetitive keyword context.

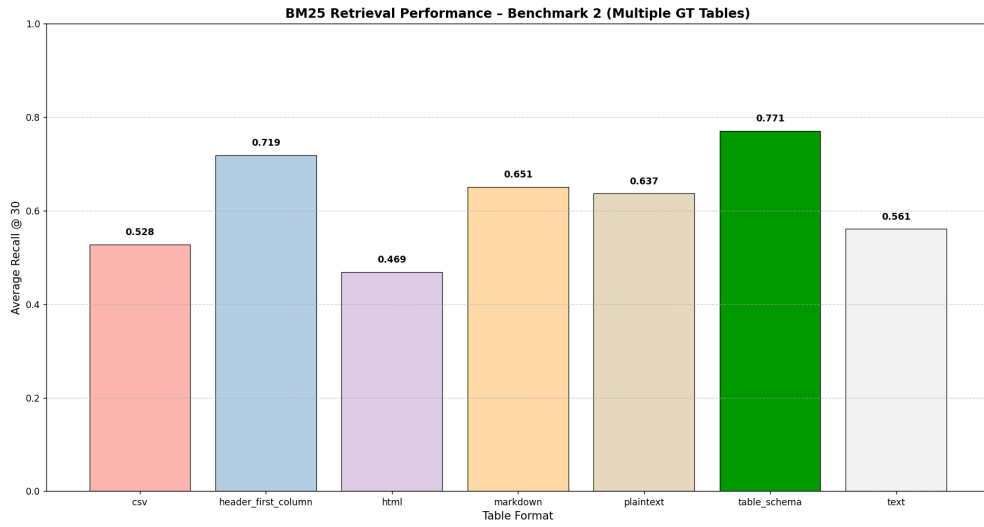


Figure 11.: BM25 retrieval performance across different table formats for Enterprise Dataset Benchmark 2.

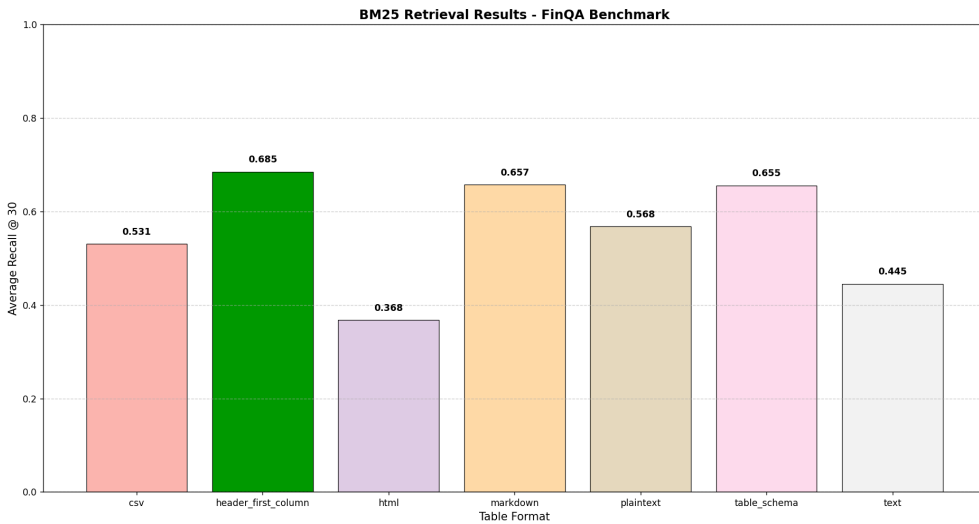


Figure 12.: BM25 retrieval performance across different table formats for FinQA Dataset Benchmark.

For the **FinQA Benchmark**, which is also a financial dataset, retrieval performance across formats follows a pattern consistent with Enterprise Benchmark 1 (Figure 12). The **header first column** representation again achieved the highest average recall@30 of **0.685**, which again shows that concise and informative formats provide BM25 with the most effective lexical cues. **Markdown (0.657)** and **table**

schema (0.655) followed closely, indicating that preserving tables in these formats remains beneficial in financial QA settings. **Plaintext (0.568)** and **CSV (0.531)** exhibited moderate effectiveness, which reflects the trade-offs of flattened representations where token dispersion or delimiters reduce retrieval performance. As seen in all benchmarks, the weakest performance was observed with **HTML (0.368)**, where markup tags inflate document length and reduce performance because of length normalization. Overall, FinQA confirms the trend that simple and context-preserving table representations consistently yield the strongest retrieval performance, whereas HTML universally hampers effectiveness across benchmarks.

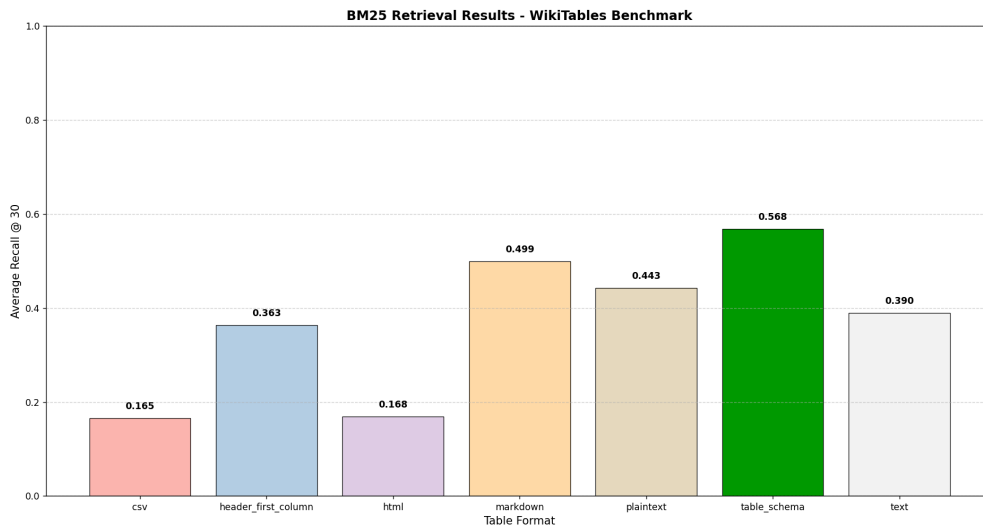


Figure 13.: BM25 retrieval performance across different table formats for WikiTableQuestions Dataset Benchmark.

For the **WikiTableQuestions dataset** (Figure 13), the BM25 performance varied considerably across the table formats. The **table schema** format, which shows that a brief summary combining headers, first column values, an example row, and a short description achieved the highest average recall@30 (top-k=30) of **0.568**. As the dataset includes medium- to long-sized tables, BM25’s length normalization impacts retrieval. Hence, this compact and informative context helps to focus on repeated and meaningful keywords that strengthen token matching. Unlike the other benchmarks, WikiTableQuestions contain questions taken not just from the table headers but also from the other table contents. Therefore, the **Markdown (0.499)** and **plaintext (0.443)** formats show moderate results as both provide linearized table content. Next is the **text** format (**0.390**), which preserves the hierarchy within the tables, followed by the **header first column** table format (**0.363**), which shows that the headers

alone without the entire table context are insufficient for this benchmark. Finally, CSV (**0.165**) and HTML (**0.168**) showed the lowest performance as delimiters and markup tags diluted the important terms and inflated the table length. Overall, these results highlight that a concise or linearized table format shows the highest recall, whereas markup representation inflates table length and dilutes token distributions, leading to degraded performance.

5.2.2. Semantic Retrieval

The semantic retrieval models leverage embeddings using semantic similarity to retrieve the relevant tables based on queries. In this section, we analyze the impact of different linearized table formats, including plaintext, Markdown, CSV, HTML, header first column, table schema, and text on the semantic retrieval method’s performance across all the benchmarks.

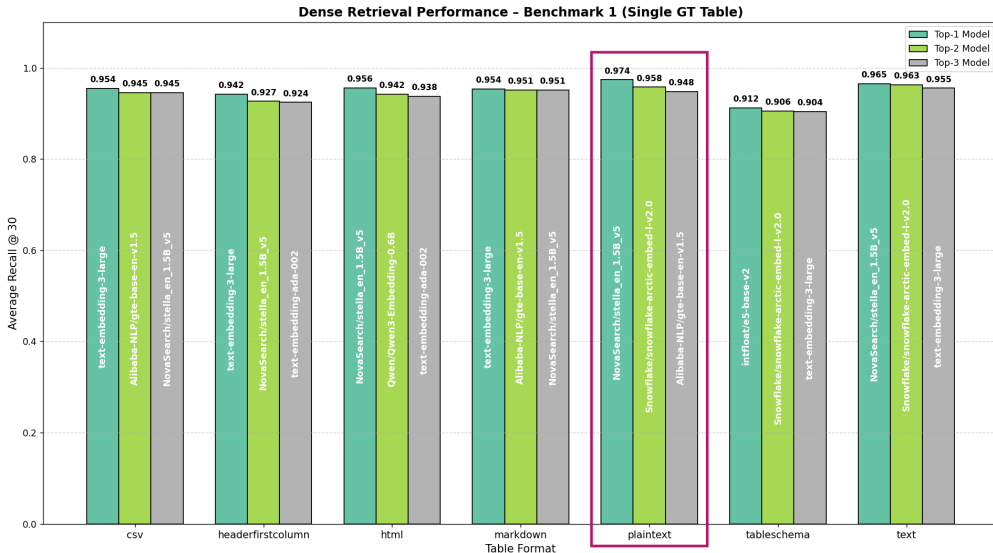


Figure 14.: Top-3 embedding models per format based on Average Recall@30 for Enterprise Dataset Benchmark 1.

For the **Enterprise Dataset Benchmark 1** (Figure 14), the retrieval performance varied across table formats. The **plaintext** format achieved the highest average recall@30 of **0.974**, followed closely by the **text** format (**0.965**), with the top model *NovaSearch/stella_en_1.5B_v5*. These formats, which resemble sentences, preserve the relation between headers and cell values, aligning well with "sentence transformer" based retrievers. The **Markdown** format also performed well (**0.954**) as the retrievers benefit from explicit separators between headers,

rows, and columns. The HTML format yielded moderate performance (**0.956** with *NovaSearch/stella_en_1.5B_v5*), then followed by the CSV format (**0.954** with *text-embedding-3-large*). The header first column format lagged slightly (best recall **0.942**) as the contextual relationships across other columns are lost. Finally, the brief table schema format yielded the weakest top recall (**0.912** with *intfloat/e5-base-v2*), indicating that concise summaries may lose the essential context for semantic retrieval. Overall, the performance differences of top models between formats were relatively small. The table formats that balance table information with sentence-like structures such as plaintext, text, and Markdown, enable embedding models to best leverage their pretrained language representations for retrieval.

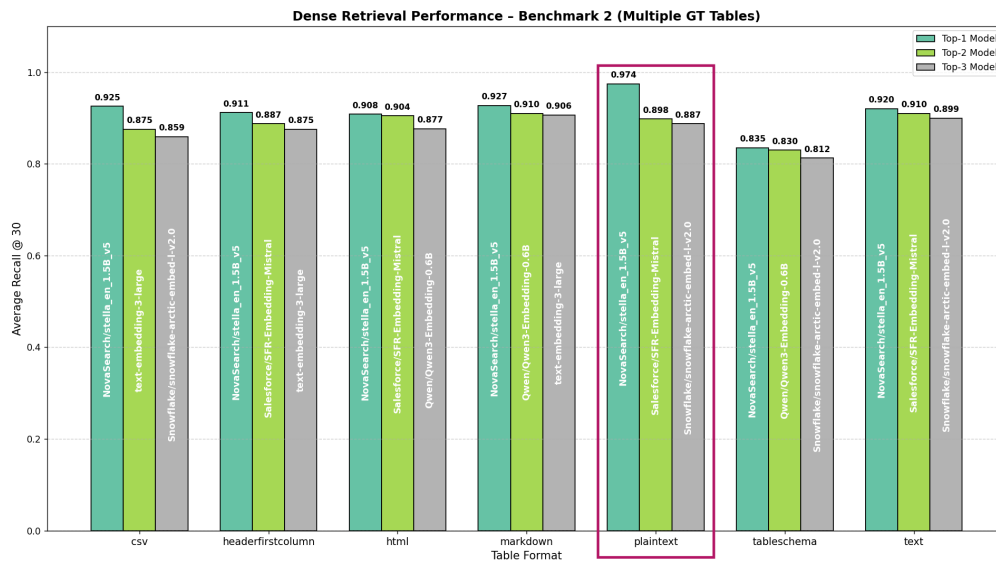


Figure 15.: Top-3 embedding models per format based on Average Recall@30 for Enterprise Dataset Benchmark 2.

For the **Enterprise Dataset Benchmark 2**, where each query is associated with multiple ground-truth tables, `plaintext` again emerged as the strongest format as shown in Figure 15. The top model, *NovaSearch/stella_en_1.5B_v5*, achieved an average recall@30 of **0.974** on the `plaintext` format, followed by similarly good performance on Markdown (**0.927**), CSV (**0.925**), and even `text` (**0.920**), which shows that dense retrievers handle most linearized formats consistently well as text-like inputs preserve contextual and semantic cues between the table elements. The `header first column` (**0.911**) and `html` (**0.908**) representations performed reasonably well as the benchmark’s questions rely on the header first column content and the sentence transformer models generalize over web-based text like HTML. The `table schema`

format again showed the lowest recall (**0.835**) as the concise summaries lacked the essential contextual cues necessary for dense retrievers to differentiate and retrieve all relevant tables. Overall, these results highlight that embedding-based retrievers benefit most from sentence-like linearized tables in multiple ground truth table scenarios. In addition, the *NovaSearch/stella_en_1.5B_v5* model consistently performed the best across various representations.

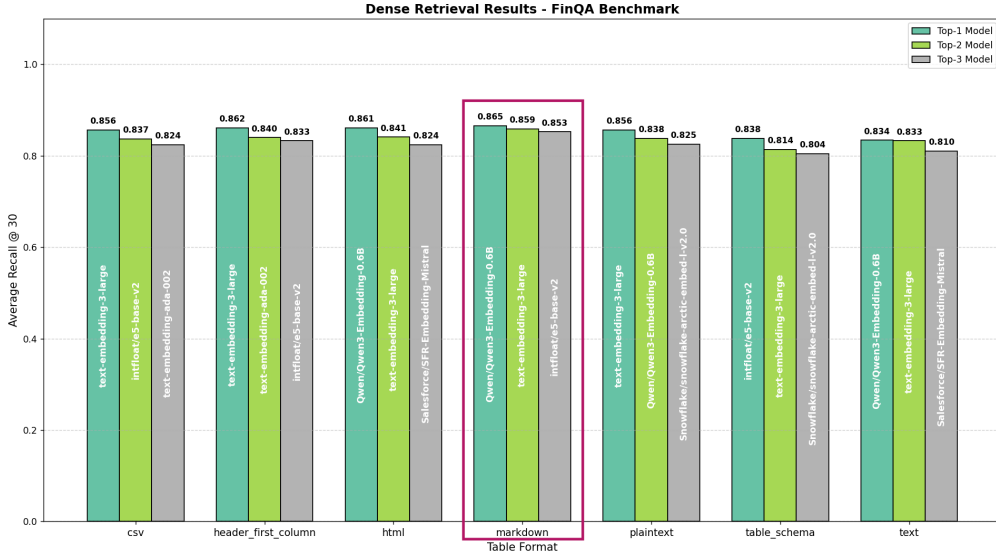


Figure 16.: Top-3 embedding models per format based on Average Recall@30 for the FinQA dataset benchmark.

For the **FinQA Benchmark** (Figure 16), dense retrieval performance again varied across table formats with some notable shifts compared to the Enterprise benchmarks. The **Markdown** format performed strongest with the *Qwen/Qwen3-Embedding-0.6B* model achieving an average recall@30 of **0.865**. The **HTML** format also performed well, where the same Qwen model reached **0.861**, which shows that dense retrievers are more robust to markup noise than the sparse method. The **header first column** and **plaintext** formats produced similarly competitive results with *text-embedding-3-large* model achieving recalls of **0.862** and **0.856** respectively. The **CSV** representation reached a comparable recall of **0.856**, although comma-separated formatting introduced tokenization noise. In contrast, the **table schema** and **text** formats lagged behind with recalls of **0.838** (using *intfloat/e5-base-v2*) and **0.834** (using *Qwen/Qwen3-Embedding-0.6B*) respectively. These flattened tables appear less effective at capturing the semantic cues required for financial QA retrieval. Overall, the dense retrievers adapt well across different representations,

with `Markdown`, `header first column`, and `HTML` achieving effective performance in this financial benchmark. Unlike `BM25`, `HTML` performs well with dense retrievers as sentence transformer models might be pretrained on web-based texts.

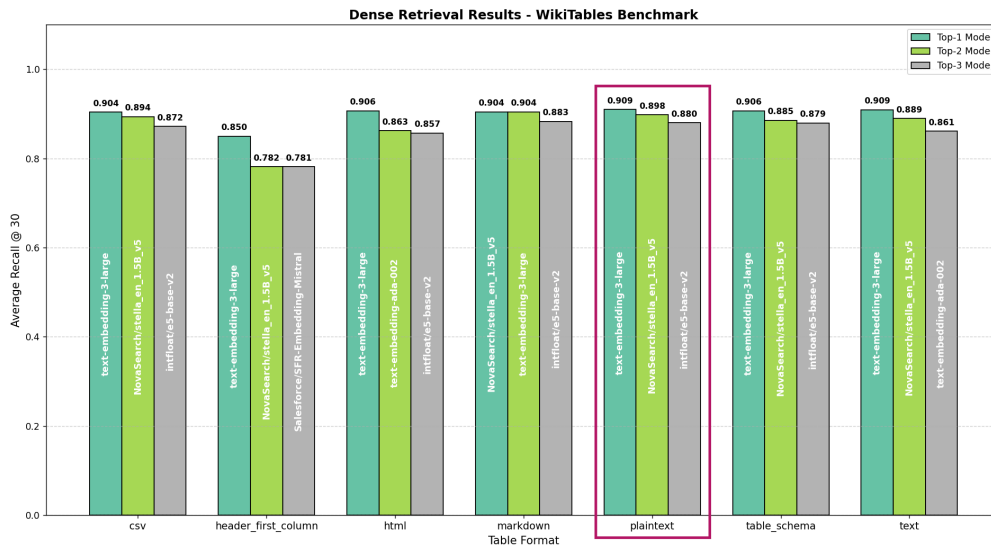


Figure 17.: Top-3 embedding models per format based on Average Recall@30 for the WikiTableQuestions dataset benchmark.

In the **WikiTableQuestions Benchmark** (Figure 17), embedding-based retrieval models achieved strong performance across most table formats, which demonstrates their ability to capture semantic similarity even when lexical overlap is limited. All the formats except the `header first column` format gave best results reaching an average recall@30 of **0.90** with the models *text-embedding-3-large* and *NovaSearch/stella_en_1.5B_v5*. The `plaintext` format achieved the highest average recall@30 of **0.909**. The `header first column` format lagged at **0.850** with *text-embedding-3-large* model, which shows that many WikiTableQuestions queries rely beyond the table headers and first column values. The top-2 models show slight differences in performance across the table formats with `Markdown` format being the best at average recall of **0.904** followed by `plaintext` (**0.892**), `CSV` (**0.894**), `text` (**0.889**), `table schema` (**0.885**), `html` (**0.863**), and `header first column` (**0.782**) respectively. Overall, these results show that dense retrievers benefit most from text-like representations or formats with explicit delimiters such as `plaintext`, and `Markdown`, whereas condensed formats like `header first column` are less effective.

5.3. Tables Retrieval using Context-Augmented Table formats

In this section, the performance of BM25 and dense retrieval is evaluated on the Enterprise and FinQA datasets, using the context-augmented table formats introduced in Section 4.2.

5.3.1. BM25

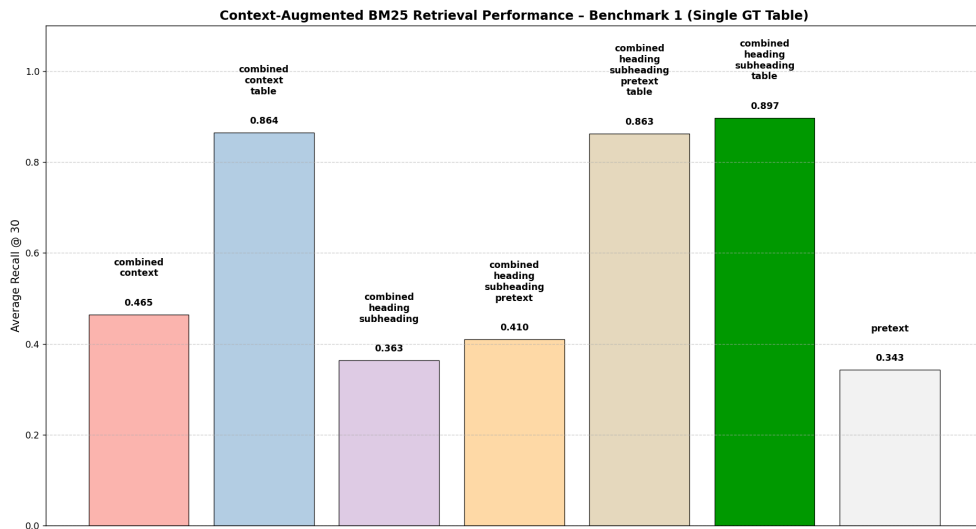


Figure 18.: BM25 retrieval performance across different context-augmented table formats for Enterprise Dataset Benchmark 1.

For the **Enterprise Dataset Benchmark 1**, the retrieval effectiveness of different context-augmented table formats was assessed using the BM25 lexical retrieval model. Figure 18 shows that **combined heading subheading table** format achieved the highest average `recall@30` of **0.897**, followed closely by the formats **combined context table** (**0.864**) and **combined heading subheading pretext table** (**0.863**). These formats include a table in its context along with the surrounding text, and tables provide exact terms, entities, and numerical values that are likely to appear in user queries, which offers lexical overlap. In addition, augmenting tables with headings or subheadings adds concise contextual cues that clarify the purpose of the table, which enhances the query–table matching. In contrast, formats without tables, such as **combined heading subheading pretext** (**0.410**), **combined heading subheading** (**0.363**), and **pretext** (**0.343**) performed substantially worse as the

words used in the text don't always match the exact words in the queries and also longer text passages can spread out the important keywords which lowers the recall because of BM25's length normalization.

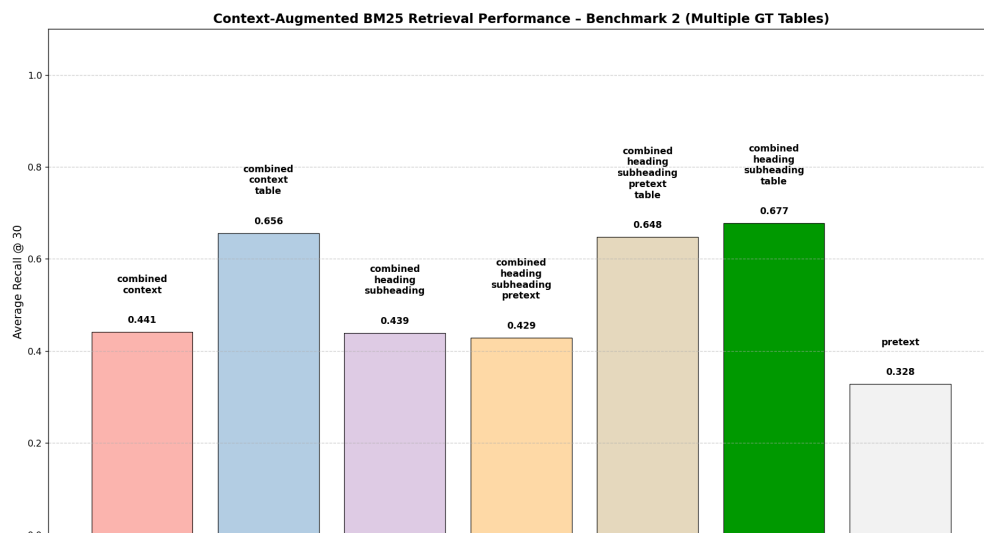


Figure 19.: BM25 retrieval performance across different context-augmented table formats for Enterprise Dataset Benchmark 2.

For the **Enterprise Dataset Benchmark 2**, Figure 19 shows the average recall@30 scores for each context-augmented table format. As shown in the above Benchmark 1, the **combined heading subheading table** achieved the highest average recall@30 of **0.677**, followed by **combined context table (0.656)** and **combined heading subheading pretext table (0.648)**. These formats perform best as they include the table in its context along with the surrounding text. The inclusion of headings and/or subheadings provides brief context that clarifies the role of table content, which helps BM25 focus on the most relevant information. The formats without tables, such as **combined heading subheading (0.439)**, **combined heading subheading pretext (0.429)**, and **pretext (0.328)** achieved noticeably lower recall as these representations rely on narrative text, which often paraphrases table content which results in limited lexical overlap with queries and inflates the length that affects the retrieval performance.

For the **FinQA Benchmark**, BM25 performance across context-augmented table formats is shown in Figure 20. The **combined context table** representation achieved the highest recall@30 of **0.737**, followed by **combined context (0.623)** and **pretext (0.570)**. Similar to the Enterprise benchmarks, the format that incorporated

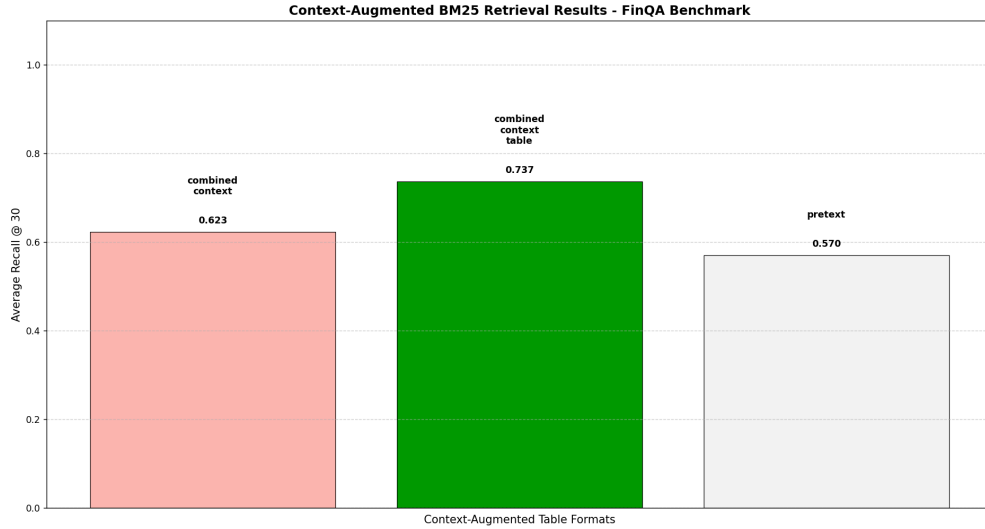


Figure 20.: BM25 retrieval performance across different context-augmented table formats for FinQA Dataset Benchmark 2.

the table content showed best results, as the presence of exact headers and cell values with domain-specific terms creates direct lexical overlap with queries. In contrast, the `pretext` format that consists only of the narrative context performs weaker because it paraphrases table content, reducing the direct term matching.

5.3.2. Semantic Retrieval

For the **Enterprise Dataset Benchmark 1**, the effectiveness of context-augmented table formats (Section 4.2) was evaluated using dense retrieval embedding models. Figure 21 presents the top-3 models for each format. Among the representations, `combined heading subheading table` achieved the highest average recall@30 of **0.960** with the *NovaSearch/stella_en_1.5B_v5* model, followed by the formats `combined heading subheading pretext table` (**0.927**) and `combined context table` (**0.926**). These formats include the table, which gives more helpful context for the semantic retrieval of relevant information based on the question. In contrast, formats without tables, such as `pretext`, `combined heading subheading`, `combined heading subheading pretext`, and `combined context` showed substantially lower recall values (0.421–0.669) with the `text-embedding-3-large` model. The formats without tables provide narrative context, which can be useful, but excessive or loosely related text may dilute signals, which limits the embedding-based retrieval effectiveness. Overall, the results show that combining table content with concise

contextual elements such as headings and/or subheadings yields the strongest retrieval performance.

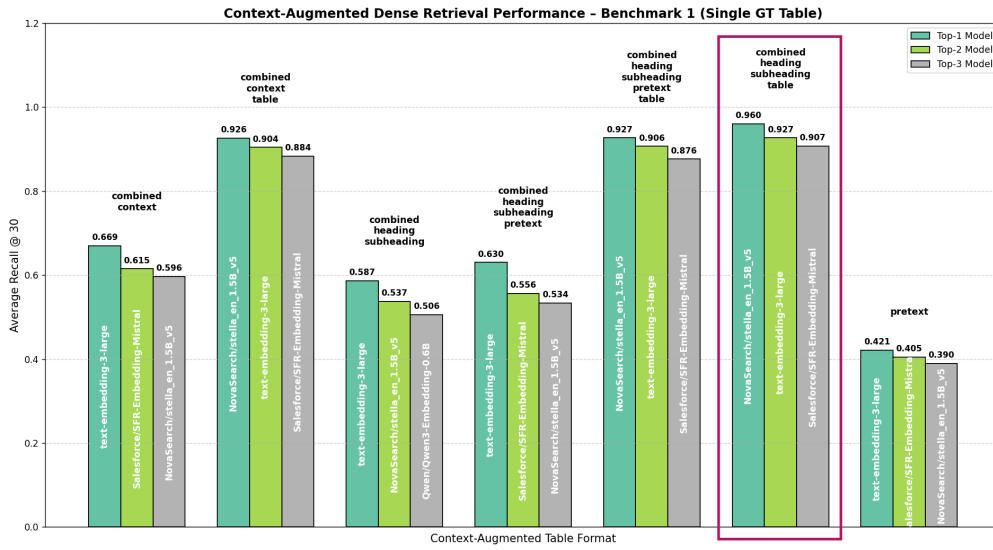


Figure 21.: Top-3 embedding models per context-augmented table format based on Average Recall@30 for Enterprise Dataset Benchmark 1.

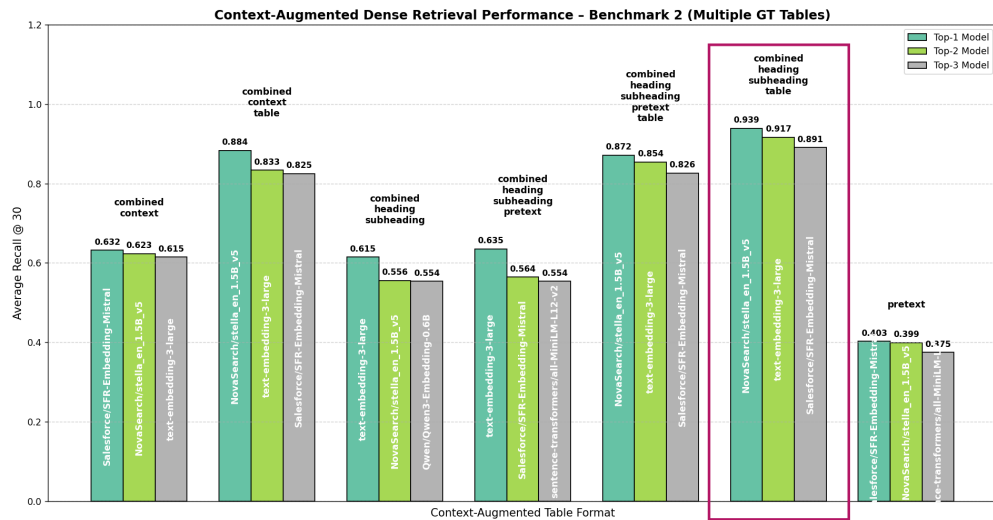


Figure 22.: Top-3 embedding models per context-augmented table format based on Average Recall@30 for Enterprise Dataset Benchmark 2.

For the **Enterprise Dataset Benchmark 2**, where answering a question requires multiple tables, the context-augmented table formats (Section 4.2) were similarly

evaluated. As illustrated in Figure 22, the best performing representation was the **combined heading subheading table**, which achieved an average recall@30 of **0.939** with the *NovaSearch/stella_en_1.5B_v5* model. The formats that included table content along with the surrounding text, such as **combined context table** and **combined heading subheading pretext table**, also performed well with average recalls of 0.884 and 0.872, respectively, using the same model. In contrast, the representations without table content, such as **pretext**, **combined heading subheading**, and **combined context**, yielded much lower recall (0.403 – 0.632), which highlights the necessity of table data for effective retrieval. Similar to benchmark 1, the surrounding text can be useful, but excessive or loosely related content often dilutes signals and lowers retrieval effectiveness. Overall, these results confirm that combining table content with carefully selected text, such as headings and/or subheadings, provides the strongest retrieval performance in scenarios requiring multiple tables.

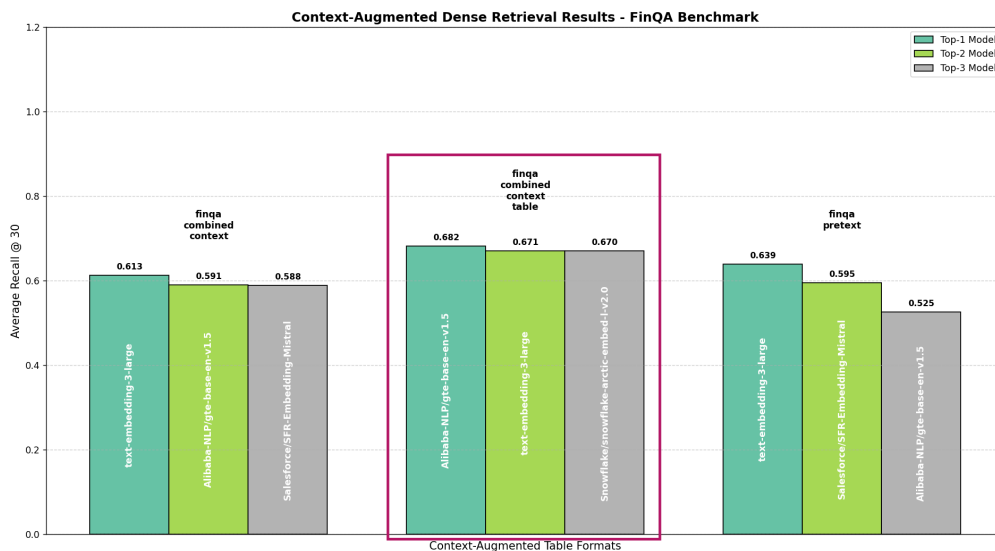


Figure 23.: Top-3 embedding models per context-augmented table format based on Average Recall@30 for FinQA Dataset Benchmark.

For the **FinQA Benchmark**, the impact of context-augmented table formats on dense retrieval performance is shown in Figure 23. Among the evaluated representations, the **combined context table** format achieved the highest average recall@30 of **0.682** with *Alibaba-NLP/gte-base-en-v1.5* model. This highlights the effectiveness of using table content with its surrounding narrative context, enabling embedding models to capture both structured and descriptive signals. The

pretext format, consisting of contextual text alone, performed moderately (**0.639** with *text-embedding-3-large*), confirming that narrative descriptions provide useful cues but remain less precise than using direct table content. Similarly, the **combined context** format without tables reached **0.613** with the same model, showing that embeddings on text alone struggle to fully capture the semantics required in financial QA.

Interestingly, FinQA reveals a contrast between the sparse and dense retrieval. For the context-augmented table formats, BM25 outperforms dense retrievers in terms of highest recall. For example, **combined context table** format achieves **0.737** with BM25 but only **0.682** under dense retrieval. At first glance, this suggests that lexical matching can be more effective than embeddings in financial QA, where queries often include highly specific numerical values, dates, or exact entity terms. However, this cannot be generalized to all financial datasets as in the **Enterprise Benchmark 1**, dense retrievers consistently outperformed BM25, even on the same context-augmented table formats. This divergence indicates that the performance of sparse versus dense retrieval in the case of the financial domain depends strongly on the dataset characteristics. FinQA questions rely heavily on precise lexical overlap with table content (e.g., numbers, units, or entities), which favors the BM25 method, whereas Enterprise Benchmark 1 questions benefit more from the semantic retrieval.

5.4. SPARQL-based Hybrid Retrieval

Benchmark	Questions Sampled	Top-K Triples Retrieved	Average Recall
Enterprise Dataset Benchmark 1	200	3,000	0.944
Enterprise Dataset Benchmark 2	96	3,000	0.933
FinQA Benchmark	200	1,000	0.873
WikiTableQuestions Benchmark	200	11,000	0.818

Table 4.: SPARQL-based hybrid retrieval results across benchmarks, showing the number of questions sampled, the top-K triples retrieved, and the achieved average recall.

For the **Enterprise Dataset Benchmark 1**, RDF triples were retrieved from the QLever endpoint using substring matching over subjects, predicates, groups, and objects. This approach yielded a large set of deduplicated triples per query. To manage computation, retrieval was restricted to the first 200 questions, which were chosen as a representative sample of the benchmark. A subsequent semantic filtering step was applied using *text-embedding-3-large* with FAISS, and the top

3,000 RDF triples were selected per question. This achieved an average recall of **0.944**, which indicates embedding-based filtering on top of SPARQL-retrieved results captured most of the relevant triples.

In the case of **Enterprise Dataset Benchmark 2**, the benchmark itself contained only 96 questions. Therefore, the entire benchmark was used for evaluation. After applying the same SPARQL retrieval and then vector-based filtering strategy (top 3,000 triples per query with FAISS), the average recall achieved was **0.933**. The results demonstrate that this method remains effective even when queries involve multiple tables.

For the **FinQA Benchmark**, a similar setup with 200 questions selected as a representative sample was followed. Since this dataset consists of small- to medium-sized tables, on top of SPARQL-retrieved triples, the vector-based filtering threshold was adjusted to the top 1,000 RDF triples per query. The average recall was **0.873**, which was lower compared to the enterprise benchmarks but still demonstrates reasonable coverage of ground-truth triples in the final subset of triples.

For the **WikiTableQuestions Benchmark**, on top of the SPARQL retrieved triples, the vector-based filtering threshold was 11,000, as this dataset has medium- to long-sized tables. The average recall of **0.818** shows that this method works effectively even on an open-domain dataset where the vocabulary varies widely.

5.4.1. Comparison with Table Retrieval Methods Using Linearized Table Formats

To further contextualize these results as shown in the following tables 5, 6, 7, and 8, SPARQL-based hybrid retrieval was compared against the best results from the table retrieval method using linearized table formats (Section 5.2). Based on the results for each benchmark, the `header first column` or the `table schema` table linearization format works best for the BM25 method, and for the **embedding-based table retrieval**, the `plaintext` or `markdown` table formats work best. To ensure a fair comparison, the same set of questions used in the SPARQL experiments (200 for Benchmark 1, 96 for Benchmark 2, 200 for WikiTableQuestions, and 200 for FinQA) was also used for the other methods.

The key metrics used for this comparison are (i) **average recall**, and (ii) **number of queries failed** (i.e., cases where no relevant table or triple was retrieved). The table retrieval methods use the entire tables, whereas the SPARQL retrieval operates at the triple level (i.e., equivalent to rows in tables and not the entire table).

Method	Average Recall	Queries Failed (0)
BM25	0.884	23
Dense Retriever	0.965	7
SPARQL	0.944	5

Table 5.: Enterprise Dataset Benchmark 1 Retrieval Comparison

Method	Average Recall	Queries Failed (0)
BM25	0.614	37
Dense Retriever	0.968	3
SPARQL	0.933	0

Table 6.: Enterprise Dataset Benchmark 2 Retrieval Comparison

Method	Average Recall	Queries Failed (0)
BM25	0.660	68
Dense Retriever	0.535	93
SPARQL	0.873	32

Table 7.: FinQA Benchmark Retrieval Comparison

Method	Average Recall	Queries Failed (0)
BM25	0.635	73
Dense Retriever	0.965	7
SPARQL	0.818	11

Table 8.: WikiTableQuestions Benchmark Retrieval Comparison

Even when operating at a more fine-grained level (triples rather than entire tables), SPARQL retrieval matched or outperformed table retrieval methods in terms of average recall. In particular, regarding the number of questions failed, that is, the questions with no matching tables or triples:

- Across all financial benchmarks, SPARQL retrieval showed fewer zero-failure cases compared to alternative methods.
- For **Enterprise Dataset Benchmark 2**, which is a relatively more complex benchmark with ground-truth triples spanning multiple tables, SPARQL re-

trieval successfully retrieved at least one relevant triple for all queries, resulting in no failures compared to both BM25 and dense retrieval methods.

- For **FinQA Benchmark**, despite the smaller size and noisier nature of the tables, SPARQL retrieval achieved the highest recall (0.873) and considerably fewer failures compared to other approaches.
- For the open-domain **WikiTableQuestions Benchmark**, even though the embedding-based retrieval performs the best in terms of average recall and queries failed, the SPARQL retrieval performed comparably well, which shows the robustness of this method.

Overall, these results highlight that **SPARQL-based hybrid retrieval remains highly effective**, with fewer outright failures than vector-based or BM25 baselines for the financial benchmarks which emphasizes that the substring matching works well in finance datasets due to consistent terminology and structured schema, but its performance is not the best in open-domain dataset where the vocabulary is diverse and query phrasing varies semantically..

5.5. Answer Generation using Table Retrieval Results

This section evaluates the impact of different retrieval methods and table representations on the performance of LLMs for question answering. We compare the zero-shot performance against the performance when the context with retrieved tables (top-k = 30) in various formats like HTML, Markdown, and plaintext, using both BM25 and dense retrieval methods, is presented to the LLM. For the zero-shot setting, the LLM temperature is set to 0.9 to allow more open-ended and independent reasoning. For retrieval-augmented settings, the temperature is set to 0.3 so that the models can focus on the given context and still allow them some flexibility to answer the follow-up questions without fully depending on the context given alone. This comparison allows an assessment of how table format influences the LLM’s ability to interpret the structured data and generate accurate answers.

On the **Enterprise Dataset Benchmark 1**, the results show a clear improvement in LLM performance when retrieved tables are incorporated in the context compared to the zero-shot setup. In the zero-shot setting, GPT-4 and GPT-4o perform relatively poorly (8.85% and 4.17%, respectively), which indicates limited capability to answer table-based questions without contextual grounding. Using the BM25 retrieval results as context, both LLMs exhibit significant gains. GPT-4 achieves its highest accuracy

Model	Zero-Shot
GPT-4	5.44%
GPT-4o	5.91%

(a) Zero-Shot

Model	HTML	Markdown	Plaintext
GPT-4	68.13%	67.88%	68.05%
GPT-4o	68.30%	67.03%	67.58%

(b) BM25 Retrieval

Model	HTML	Markdown	Plaintext
GPT-4	75.3%	74.92%	74.92%
GPT-4o	74.03%	73.19%	72.93%

(c) Dense Retrieval

Table 9.: Weighted Accuracy (%) for Enterprise Dataset Benchmark 1 across zero-shot, BM25 retrieval, and Dense retrieval for different table formats.

(38.95%) when tables are presented in HTML format, followed by Markdown (38.42%) and Plaintext (25.58%). Similarly, GPT-4o performs best with HTML (39.47%), confirming that providing the retrieved tables as context improves the model’s answer generation. In addition, the superiority of HTML can be attributed to the fact that LLMs are typically trained with extensive web data, which makes them inherently more familiar with markup languages such as HTML [16]. This format also preserves the hierarchical and relational structure of tables through tags, allowing the model to more accurately interpret tabular elements. Conversely, Plaintext yields the lowest performance as it loses crucial structural information and positional relationships, which makes it harder for models to interpret tabular data effectively. When using the embedding-based retrieval results, the answer generation performance of the LLMs further improves. The relative performance of the table formats follows a similar trend to that of BM25. The HTML format achieves the highest results, and Plaintext performs the lowest.

On the **Enterprise Dataset Benchmark 2**, where each question corresponds to multiple ground-truth tables, the results show consistent improvements from zero-shot to retrieval-augmented settings as shown in Table 10. Due to the added retrieval complexity, the overall accuracy remains lower compared to single-table benchmark scenarios. Both GPT-4 and GPT-4o benefit significantly by using retrieval

Model	Zero-Shot
GPT-4	8.85%
GPT-4o	4.17%

(a) Zero-Shot

Model	HTML	Markdown	Plaintext
GPT-4	38.95%	38.42%	25.58%
GPT-4o	39.47%	37.37%	33.72%

(b) BM25 Retrieval

Model	HTML	Markdown	Plaintext
GPT-4	50.53%	48.95%	44.74%
GPT-4o	40.00%	40.00%	40.53%

(c) Dense Retrieval

Table 10.: Weighted Accuracy (%) for Enterprise Dataset Benchmark 2 (multiple ground-truth tables per question) across zero-shot, BM25 retrieval, and Dense retrieval for different table formats.

context. The GPT-4 model increases from 8.85% (zero-shot) to 38.95% (BM25–HTML) and further to 50.53% (dense–HTML), whereas GPT-4o progresses from 4.17% to 39.47% (BM25–HTML) and 40.53% (dense–Plaintext). Across retrieval-augmented answer generation methods, the HTML format again demonstrates the strongest and most consistent performance, particularly for GPT-4, which shows its superiority in representing structured data. This advantage results from the LLMs’ pretraining on large web corpora that heavily feature markup languages like HTML and XML, which allows them to better interpret hierarchical structures and relational cues within tables. In contrast, Plaintext performs the lowest for GPT-4 due to its loss of table structure.

Similar to other benchmarks, retrieval-augmented methods significantly improve LLM performance on the FinQA benchmark, as shown in Table 11. For GPT-4, zero-shot accuracy is 6.5%, which rises to 41.07% with BM25–HTML setup and further to 45.76% under dense-HTML setup. Similarly, GPT-4o improves from 4.5% (zero-shot) to 40.52% (BM25–HTML) and 46.06% (dense–HTML). Across the table formats, HTML consistently yields the highest performance, which shows the LLM’s ability to interpret structured data presented in markup language, whereas Markdown and Plaintext formats show slightly lower performance.

Model	Zero-Shot
GPT-4	6.5%
GPT-4o	4.5%

(a) Zero-Shot

Model	HTML	Markdown	Plaintext
GPT-4	41.07%	40.66%	41.26%
GPT-4o	40.52%	39.85%	40.78%

(b) BM25 Retrieval

Model	HTML	Markdown	Plaintext
GPT-4	45.76%	45.42%	45.41%
GPT-4o	46.06%	44.59%	45.44%

(c) Dense Retrieval

Table 11.: Weighted Accuracy (%) for the FinQA Benchmark across zero-shot, BM25 retrieval, and Dense retrieval for different table formats.

Model	Zero-Shot
GPT-4	7.43%
GPT-4o	7.33%

(a) Zero-Shot

Model	HTML	Markdown	Plaintext
GPT-4	17.67%	17.24%	17.53%
GPT-4o	19.65%	19.43%	20.09%

(b) BM25 Retrieval

Model	HTML	Markdown	Plaintext
GPT-4	36.29%	34.25%	35.09%
GPT-4o	38.34%	38.6%	39.14%

(c) Dense Retrieval

Table 12.: Weighted Accuracy (%) for the WikiTableQuestions Benchmark across zero-shot, BM25 retrieval, and Dense retrieval for different table formats.

Since the WikiTableQuestions dataset contains medium to long tables, we use a top-k value of 3, as this fits the context length of the LLMs. For GPT-4, zero-shot accuracy is 7.43%, which improves to 17.67% with BM25-HTML and further to 36.29% with dense-HTML. A similar pattern is observed for the markdown and plaintext formats, where BM25 results are around 17% and dense retrieval shows comparable performance within one or two percentage points of the HTML results. For GPT-4o, zero-shot accuracy is 7.43%, which increased to 20.09% with BM25-plaintext and further to 39.14% with dense-plaintext. A similar pattern is observed for the HTML and Markdown formats in both BM25 and dense retrieval results, which were around 19% and 38%. Though the performance difference between formats is very minimal, we could see that for the open-domain dataset, the Plaintext format performs the best, whereas for the finance-domain benchmarks, converting the tables to plaintext removes structure and flattens the layout, which makes it harder for the models to infer which numbers correspond to which categories or time periods.

Overall, the experiments demonstrate that GPT-4 achieves the highest performance across all the financial benchmarks, closely followed by GPT-4o, and for the open-domain benchmark, GPT-4o shows the best results. In addition, the results indicate that the largest performance gains occur when moving from zero-shot to retrieval-augmented pipelines, with dense retrieval providing additional improvements in question answering tasks. Among the table representations, HTML performs best likely because LLMs are pretrained on web-based corpora and are therefore good at interpreting HTML structure compared to other formats.

5.5.1. Prompt Settings

LLMs possess fundamental capabilities to interpret structured data, but their performance remains imperfect. Hence, selecting appropriate input representations and prompt settings can influence their effectiveness when we work with tabular data. Since dense retrieval outputs in HTML and Markdown formats consistently yielded the highest performance across all the evaluated datasets, these representations were taken for analyzing the impact of different prompt designs. Experiments were conducted on the first 200 questions of three single-table benchmarks, whereas the entire Enterprise Dataset Benchmark 2 (multiple-table) was used for evaluation, as its size is small enough to allow complete assessment.

The following six distinct prompt designs were examined,

Prompt Setting 1: The first prompt setting follows the design used in the LLM answer generation experiments of the previous section 5.5. It presents the instructions followed by the context and question without any explicit separators, and is considered the baseline for comparison with the other prompt settings.

Prompt Setting 2: Similar to Setting 1, but the context and question are presented *before* the instructions.

Prompt Setting 3: Incorporates output format as part of the instructions. The context and question are written with explicit separators *after* the instructions for the LLM clearly distinguish between context and query as follows:

```
=== CONTEXT START ===
{context}
=== CONTEXT END ===

=== QUESTION START ===
{question}
=== QUESTION END ===
```

Prompt Setting 4: Similar to Setting 3, but the context and question are placed *before* the instructions.

Prompt Setting 5: Extends Setting 3 by including two examples within the instructions, followed by the table context and question.

Prompt Setting 6: Same as Setting 5, but the table context and question are placed *before* the instructions.

This systematic evaluation of prompt designs allows the assessment of how context placement, explicit delimiters, and examples influence LLMs' ability to understand and interpret tabular data. In the following results sections, we use Setting 1 as the baseline. All increases or decreases reported for other prompt settings are direct percentage changes with respect to Setting 1.

Enterprise Dataset Benchmark 1 Results

Model	Setting 1	Setting 2	Setting 3	Setting 4	Setting 5	Setting 6
gpt-4	75.30	74.50	75.00	70.25	73.50	65.75
gpt-4o	74.03	76.50	70.25	71.25	71.00	66.00

(a) HTML format.

Model	Setting 1	Setting 2	Setting 3	Setting 4	Setting 5	Setting 6
gpt-4	74.92	76.00	73.25	72.00	70.25	67.00
gpt-4o	73.19	78.25	73.25	70.25	72.50	67.50

(b) Markdown format.

Table 13.: Performance (in %) under different prompt settings for the Enterprise Dataset Benchmark 1.

For the HTML format, GPT-4 performs best with the reference prompt (Setting 1), while the other prompt settings result in slight to moderate decreases. The largest performance drop occurs in Setting 6 (-9.55%). Reordering the context and question (Setting 2) produces a small decrease (-0.80%), whereas prompts with explicit identifiers (Settings 3 and 4) and few-shot examples (Setting 5) lead to moderate declines (-0.30%, -5.05%, -1.80%) which indicates that GPT-4 benefits from the instruction first design in single table scenarios. Alternatively, GPT-4o experiences its highest gain (+2.47%) in Setting 2, which shows that the context and question provided before the instruction can enhance performance. The other prompt settings generally reduce accuracy (-3.78%, -2.78%, -3.03%, -8.03%) compared to the reference prompt setting 1.

For the Markdown format, both models show notable improvements when the context and question are presented before the instructions (Setting 2). GPT-4 gains approximately +1.08% compared to the reference setting, and GPT-4o achieves its highest performance in this configuration (+5.06%). The other prompt designs including few-shot examples (Setting 5, 6) or explicit identifiers (Settings 3, 4) produced decrease in performance for GPT-4 model whereas GPT-4o results with minor gain in setting 3 (+0.06%) and performance drops for other settings which indicates that these settings do not consistently improve reasoning for Markdown tables.

Overall, these results suggest that providing the context and question before

instructions is the most effective modification for both models in Markdown format, whereas GPT-4 generally prefers the instruction first reference prompt in HTML format. Explicit identifiers and few-shot examples provide limited benefit in this single-table benchmark, which highlights that simpler prompt structures are often sufficient for accurately reasoning over tables.

Enterprise Dataset Benchmark 2 Results

Model	Setting 1	Setting 2	Setting 3	Setting 4	Setting 5	Setting 6
gpt-4	46.94	41.15	40.62	41.67	41.67	43.23
gpt-4o	34.18	46.35	53.12	51.04	39.58	48.96

(a) HTML format.

Model	Setting 1	Setting 2	Setting 3	Setting 4	Setting 5	Setting 6
gpt-4	44.39	52.60	42.19	46.35	43.23	43.23
gpt-4o	36.73	42.19	50.52	56.25	39.06	45.83

(b) Markdown format.

Table 14.: Performance (in %) under different prompt settings for the Enterprise Dataset Benchmark 2.

For the HTML format, GPT-4 shows a decrease in performance across all prompt settings relative to Setting 1, with the largest loss in Setting 3 (-6.32%). Settings incorporating explicit identifiers and few-shot examples (Settings 3, 4, 5, 6) also result in moderate declines, indicating that GPT-4 performs best with the instruction-first reference design (Setting 1) in multi-table scenarios. In contrast, GPT-4o experiences substantial gains with prompts that include explicit identifiers, achieving +18.94% and +16.86% in Settings 3 and 4, respectively. The context and question given without identifiers before instructions (Setting 2) provides a +12.17% improvement, whereas prompts with few-shot examples (Settings 5 and 6) lead to a gain of +5.40% and +14.78%, respectively. These results suggest that GPT-4o benefits significantly from few-shot examples and providing context, question before instructions in complex multi-table scenarios.

For the Markdown format, GPT-4 achieves its highest improvement in Setting 2 (+8.21%), which indicates that providing the context and question first can help this model. The other settings with explicit identifiers or few-shot examples produce

mixed results with minor gains or losses. GPT-4o shows a performance increase in all settings. The largest gain is seen in Setting 4 (+19.52%), and few-shot examples (Setting 5) provide only modest improvements, which highlights that providing the context and question first majorly benefits this model.

Overall, these results demonstrate that prompt effectiveness is highly model- and format-dependent in multi-table benchmarks. Explicit identifiers and providing the context and question first can significantly enhance GPT-4o performance, whereas GPT-4 generally performs best with the instruction first reference prompt (Setting 2), and few-shot examples offer limited benefit in this model.

FinQA Dataset Benchmark Results

Model	Setting 1	Setting 2	Setting 3	Setting 4	Setting 5	Setting 6
gpt-4	45.75	45.50	41.50	42.93	39.00	41.00
gpt-4o	48.25	51.25	46.00	43.18	44.25	41.75

(a) HTML format.

Model	Setting 1	Setting 2	Setting 3	Setting 4	Setting 5	Setting 6
gpt-4	46.75	46.00	44.00	43.25	39.00	45.25
gpt-4o	43.25	49.25	44.00	43.00	41.00	39.50

(b) Markdown format.

Table 15.: Performance (in %) under different prompt settings for the FinQA Benchmark.

For the HTML format, the two LLMs show different responses to the prompt settings relative to the reference prompt design (Setting 1). For GPT-4, Setting 2, where the context and question are placed before instructions, yields very minimal effect on performance. However, GPT-4o gains around +3.00%, which suggests that context-question reordering benefits this model. Prompts incorporating few-shot examples (Settings 5 and 6) or explicit separators for context and question (Settings 3 and 4) generally reduce performance for both models, which indicates that these modified settings do not reliably improve reasoning over HTML tables. Overall, GPT-4 performs best with the instruction-first reference prompt design, while GPT-4o benefits modestly from reordering the context and question in the reference prompt.

For the Markdown format, the pattern is similar to the one observed in the HTML

format but more pronounced. GPT-4 shows a slight decline with Setting 2 and larger drops with few-shot examples or explicit separators, but the drop is more prominent in Setting 5. In contrast, GPT-4o achieves a +6.00% improvement in Setting 2, but also suffers performance losses with few-shot examples or explicit separators, which was most significant in Setting 6 with a drop of -3.75%. These results reinforce that reference prompts in instruction first design for GPT-4 and the same prompt in context, question first design for GPT-4o are more effective for Markdown tables, whereas adding examples or separators does not consistently enhance performance.

Overall, across both HTML and Markdown formats, the GPT-4 performs best with the reference prompt with instruction-first design, whereas GPT-4o benefits from providing the context and question first. In both models, prompts that include few-shot examples or explicit context, question separators do not consistently enhance performance, which indicates that simpler, well-structured prompts are generally more effective for reasoning over tables in the FinQA dataset.

WikiTableQuestions Benchmark Results

Model	Setting 1	Setting 2	Setting 3	Setting 4	Setting 5	Setting 6
gpt-4	16.75	23.50	52.00	54.75	41.00	41.00
gpt-4o	21.50	31.50	46.75	53.50	44.50	44.50

(a) HTML format.

Model	Setting 1	Setting 2	Setting 3	Setting 4	Setting 5	Setting 6
gpt-4	27.00	25.00	52.25	52.75	40.75	45.00
gpt-4o	33.25	31.75	49.00	53.25	43.25	47.00

(b) Markdown format.

Table 16.: Performance (in %) under different prompt settings for the WikiTableQuestions Benchmark.

For the HTML format, GPT-4 exhibits substantial gains when we move from the reference prompt (Setting 1) to other Settings 2, 3, and 4 with increases of approximately +6.75%, +35.25%, and +38.00%, respectively. The largest improvements are observed in Settings 3 and 4, which incorporate explicit separators for context and question, suggesting that providing clear structural cues is highly helpful for GPT-4 on WikiTableQuestions. GPT-4o shows similar trends with moderate gains in Setting

2 (+10.00%) and substantial increases in Settings 3 and 4 (+25.25% and +32.00% compared to Setting 1), which again indicates that explicit separation of context and question from instructions significantly enhances performance. Setting 5, including few-shot examples, offers some improvement compared to Setting 1 for both models (+24.25% for GPT-4 and +23.00% for GPT-4o), but these gains are smaller than those achieved by prompts using explicit separators.

For the Markdown format, both models display a slightly different pattern. GPT-4 shows a slight decline in Setting 2 (-2.00%) compared to Setting 1, indicating that providing context and question first alone does not improve performance. Substantial gains are observed in Settings 3 and 4 (+25.25% and +25.75%), which confirms that explicit separators are effective in guiding the model, while Settings 5 and 6, with few-shot examples, provide only modest improvement (+13.75% and +18.00%). Similarly, GPT-4o experiences a small decrease in Setting 2 (-1.50%) and a moderate gain in Setting 3 (+15.75%), with comparatively less pronounced benefit from few-shot examples (+10.00% and +13.75%) in Settings 5 and 6. The highest performance improvement is observed in Setting 4 (+20.00%).

Overall, both models benefit most from prompts that clearly separate context and question, demonstrating that structural clarity is crucial for LLM reasoning on the WikiTableQuestions benchmark. Reordering context and question before instructions alone provides limited benefit, and few-shot examples improve performance modestly but are not as effective as explicit identifiers.

5.6. Answer Generation using SPARQL-based Retrieval Results

To evaluate the effectiveness of using RDF triples as context for answer generation, SPARQL queries were used to retrieve the RDF triples for each question, and the triples were presented to LLMs in three different ways:

- Approach 1:** Sequential presentation in a simple `Subject | Predicate | Object` format, one by one.
- Approach 2:** Triples presented sequentially but grouped by the tables they belong to.
- Approach 3:** Triples grouped by both table and subject, so that all triples with the same subject appear together under each table.

Benchmark	Approach 1 (%)		Approach 2 (%)		Approach 3 (%)	
	GPT-4	GPT-4o	GPT-4	GPT-4o	GPT-4	GPT-4o
Enterprise Benchmark 2	10.42	18.75	11.98	18.75	13.54	16.15
Enterprise Benchmark 1 (200 QA)	41.00	44.25	32.50	48.50	35.25	51
FinQA Benchmark (200 QA)	24.25	22.75	12.00	23.50	22.25	19.75
WikiTableQuestions Benchmark (200 QA)	16.25	13.25	17.25	13.25	16.75	14.25

Table 17.: Answer generation performance (% weighted accuracy) using SPARQL-retrieved RDF triples as context in three presentation approaches: (1) sequential triples, (2) table-grouped triples, and (3) table- and subject-grouped triples.

Enterprise Benchmark 1 (200 QA Sample)

Using Approach 1, GPT-4 and GPT-4o achieved 41% and 44.25% respectively. The performance is relatively low compared to the answer generation using entire tables as context, as the sequential presentation separates triples that belong to the same table, which loses their relational context. With Approach 2, GPT-4o showed a notable gain to 48.5%, indicating that table-level grouping helps preserve the relational context, whereas GPT-4 performance declined to 32.5%. The approach 3 further improved entity-level comprehension, as we can see that GPT-4 achieved 35.25% and GPT-4o reached 51%, confirming that grouping triples by both table and subject helps the GPT-4o model better integrate all related facts about each entity in this single-table scenario.

Enterprise Benchmark 2 (Multi-Table)

For Approach 1, GPT-4 scored 10.42% and GPT-4o achieved 18.75%, which highlights the difficulty of reasoning across relevant sequential triples spanning multiple tables. Approach 2 provided minor gains for GPT-4 (11.98%) while GPT-4o remained unchanged (18.75%), showing that table-level grouping alone is insufficient in multi-table scenarios. With Approach 3, GPT-4 improved to 13.54% and GPT-4o scored 16.15%, which indicates that even subject-level grouping offers limited benefit when the task requires integrating information from multiple relevant tables.

FinQA Benchmark (200 QA Sample)

Using sequential triples (Approach 1), GPT-4 and GPT-4o achieved 24.25% and 22.75% respectively. Grouping by table (Approach 2) slightly improved GPT-4o performance to 23.5% but lowered GPT-4 performance to 12%, whereas grouping by table and subject (Approach 3) gave similar results to sequential presentation but with a

slight decrease to 22.25% in GPT-4 and 19.75% in GPT-4o. Sequential triples are straightforward for LLMs to identify the relevant triples in this single-table scenario.

WikiTableQuestions Benchmark (200 QA Sample)

In this Benchmark, we use the 3000 top-k RDF triples as context instead of 11000 due to the context length constraint. Using sequential triples (Approach 1), GPT-4 and GPT-4o achieved 16.25% and 13.25%, respectively. In Approach 2, grouping by table slightly improved the performance of GPT-4o to (17.25%), but GPT-4 performance stayed the same at 13.25%. Approach 3 of grouping by table and subject gave similar results to sequential presentation, where there was a very slight increase in performance to 16.75% in GPT-4 and 14.25% in GPT-4o. Similar to Enterprise Benchmark 1, approach 2 helped the LLMs identify the relevant triples for answer generation in this single-table scenario.

5.6.1. Prompt Settings

To systematically investigate how different prompt designs influence reasoning over RDF triples, we conducted a series of experiments using the GPT-4 and GPT-4o models across all the benchmarks. Each model was evaluated under the above-discussed three approaches of presenting the RDF triples with prompt designs with a varied arrangement of RDF triples, questions, and instructions. The answer generation results presented in Table 17 are considered as the baseline for comparison.

The prompt designs used for the evaluation are defined as follows,

Prompt Setting 1: Using the same baseline prompt, but with the RDF triples context and question placed *before* the instructions.

Prompt Setting 2: The prompt incorporates output format as part of the instructions, and the RDF Triples context and question are written with explicit separators *after* the instructions.

Prompt Setting 3: Similar to Setting 2, but the RDF triples context and question are presented *before* the instructions.

Prompt Setting 4: Builds on Setting 2 by including three examples within the instructions, and the RDF triples context and question follow the instructions.

Prompt Setting 5: Builds on Setting 4 with the RDF triples context and question being placed *before* the instructions.

Approach 1 Results

Benchmark	Model	Baseline	Prompt 1	Prompt 2	Prompt 3	Prompt 4	Prompt 5
Enterprise Benchmark 1	GPT-4	41.00	19.75	58.00	55.50	59.75	53
	GPT-4o	44.25	63.25	68.50	70.75	64.75	71
Enterprise Benchmark 2	GPT-4	10.42	5.21	23.44	29.69	18.75	22.4
	GPT-4o	18.75	30.21	38.54	42.71	32.81	40.1
FinQA (200 QA)	GPT-4	24.25	18.25	21.75	27.75	23	23.25
	GPT-4o	22.75	25.75	27.75	31.00	26.75	26.5
WikiTableQuestions (200 QA)	GPT-4	16.25	13.5	21	25.75	25	19.5
	GPT-4o	13.25	17.5	23.5	29.75	23.5	25

Table 18.: Answer generation performance (% weighted accuracy) for Approach 1 using SPARQL-retrieved RDF triples as context across different prompt settings.

For the **Enterprise Benchmark 1**, baseline performance using RDF triples in sequential presentation was 41.00% for GPT-4 and 44.25% for GPT-4o. The different prompt designs had a strong positive impact on both models. GPT-4 achieved its highest score of 59.75% under Prompt 4 which marks an 18.75% improvement from the baseline. GPT-4o reached 71.00% under Prompt 5 gaining over 26% compared to the baseline. The prompts 2 and 3 also yielded substantial gains for both models, showing that placing triples before instructions and including explicit separators significantly enhance the answer generation. These findings indicate that both models, especially GPT-4o, benefit from including few-shot examples, as this guides the LLMs to interpret the relevant data from the RDF triples context based on the question.

For the **Enterprise Benchmark 2** (multi-table), baseline performance was 10.42% for GPT-4 and 18.75% for GPT-4o. GPT-4 improved steadily across the prompt settings reaching 32.29% with Prompt 5 which is a gain of nearly 22% whereas GPT-4o model peaked at 42.71% under Prompt 3 improving by 24% from the baseline. These results show that in multi-table benchmarks, prompts incorporating explicit identifiers and placing context, question before instructions help maintain relational consistency across tables, which guides the models to keep track of the entities and relationships that align across multiple tables. However, few-shot examples add limited benefit as the examples given were for the single-table reasoning scenarios.

For the **FinQA benchmark**, baseline performance was 24.25% for GPT-4 and 22.75% for GPT-4o. The prompt designs had a modest but notable impact. GPT-4

benefited most from Prompt 3 (+3.5%) and Prompt 2 gave a smaller gain (+1.5%). GPT-4o showed consistent improvements across Prompt 1 (+3%), Prompt 2 (+5%), and Prompt 3 (+8.25%). For the Prompts 4 and 5, GPT-4o showed an increase in performance, but GPT-4 performance degraded. These results indicate that prompts with explicit separators and few-shot examples enhance reasoning, and that GPT-4o is generally more responsive to prompt variations than GPT-4.

For the **WikiTableQuestions benchmark**, the trends were similar to the FinQA benchmark. The baseline performance was 16.25% for GPT-4 and 13.25% for GPT-4o. For the GPT-4 model, the prompt designs except Prompt 1 had improvement from the baseline, whereas the GPT-4o model shows improvement across all prompt designs. Both GPT-4 and GPT-4o performed the best in prompt setting 3 with explicit separators for RDF Triples context and questions, which are being placed before instructions. The GPT-4 and GPT-4o models showed an improvement of 9.5% and 16.5%.

Approach 2 Results

Benchmark	Model	Baseline	Prompt 1	Prompt 2	Prompt 3	Prompt 4	Prompt 5
Enterprise Benchmark 1	GPT-4	32.50	21.75	57.25	62.00	62.00	56.00
	GPT-4o	48.50	61.50	68.75	69.75	67.75	72.00
Enterprise Benchmark 2	GPT-4	11.98	4.69	29.69	25.52	27.08	32.29
	GPT-4o	18.75	36.98	39.06	41.15	40.62	40.10
FinQA (200 QA)	GPT-4	12.00	15.75	23.00	26.25	24.75	22.50
	GPT-4o	23.50	13.75	32.00	31.75	31.75	28.25
WikiTableQuestions (200 QA)	GPT-4	17.25	14.25	23.5	23.75	25	20.25
	GPT-4o	13.25	14.5	28.25	30.75	16.5	22.25

Table 19.: Answer generation performance (% weighted accuracy) for Approach 2 using SPARQL-retrieved RDF triples as context across different prompt settings.

For the **Enterprise Benchmark 1**, baseline performance was 32.5% for GPT-4 and 48.5% for GPT-4o. The prompt designs had a substantial impact. GPT-4 improved most with Prompts 3 and 4 (+29.5%, reaching 62%), whereas Prompts 2 and 5 provided moderate gains, and Prompt 1 decreased performance compared to the baseline. On the other hand, GPT-4o showed consistent improvements across all prompts, achieving the highest accuracy with Prompt 5 (+23.5%, reaching 72%). These results indicate that prompts with explicit separators and few-shot examples greatly enhance LLM reasoning in the single-table finance QA benchmark.

For the **Enterprise Benchmark 2**, which is a multi-table QA benchmark, baseline

performance was 11.98% for GPT-4 and 18.75% for GPT-4o. The prompt designs led to steady improvements. GPT-4 rose to 32.29% under Prompt 5 (+20.31%), while GPT-4o peaked at 41.15% with Prompts 3 (+22.4%). This demonstrates that prompts with explicit separators for context, question, and placing them before instructions are effective for multi-table reasoning. The overall accuracy remains lower than single-table benchmarks due to the increased relational complexity.

For the **FinQA benchmark**, baseline performance was 12% for GPT-4 and 23.5% for GPT-4o. The prompt designs influenced the answer generation notably. GPT-4 improved most with Prompt 3 (26.25%) while Prompts 1, 2, 4, and 5 gave only smaller gains. GPT-4o benefited most from Prompt 2 (32%), with Prompts 3 and 4 showed improvement of 31.75% and Prompt 5 gave moderate improvement of 28.25% compared to the baseline. These results indicate that prompts with explicit identifiers and placing the context, question first are most effective as they help the LLMs interpret and reason over RDF triples. The prompt with few-shot examples provides only limited additional benefit in this single-table benchmark.

For the **WikiTableQuestions benchmark**, baseline performance was 17.25% for GPT-4 and 13.25% for GPT-4o. The prompt designs except Prompt 1 showed improvement for GPT-4 model, whereas all the prompts showed an increase in answer generation performance for GPT-4o model. GPT-4 performed best at 25% with Prompt 4, with few-shot examples, and GPT-4o performed best at 30.75% with Prompt 3, which provided context, question first with explicit separators. In addition, providing the context and question before instructions was not helpful for GPT-4 model, whereas it improved performance in GPT-4o as seen in Prompts 1, 3, 5 compared to the same prompts (2, 4) with context, question following instructions.

Approach 3 Results

Benchmark	Model	Baseline	Prompt 1	Prompt 2	Prompt 3	Prompt 4	Prompt 5
Enterprise Benchmark 1	GPT-4	35.25	19	58.25	61.00	62.25	56.25
	GPT-4o	51	65.25	61.00	68.25	65.25	73.00
Enterprise Benchmark 2	GPT-4	13.54	7.29	20.31	31.25	27.60	32.29
	GPT-4o	16.15	29.17	19.27	39.06	39.58	35.94
FinQA (200 QA)	GPT-4	22.25	23.25	19.25	26.75	22.50	18.50
	GPT-4o	19.75	25.25	21.25	33.00	28.25	29.75
WikiTableQuestions (200 QA)	GPT-4	16.75	13.25	23.75	20	23.75	18.75
	GPT-4o	14.25	16.5	26	28.75	17.25	24

Table 20.: Answer generation performance (% weighted accuracy) for Approach 3 using SPARQL-retrieved RDF triples as context across different prompt settings.

For the **Enterprise Benchmark 1**, baseline performance was 35.25% for GPT-4 and 51% for GPT-4o. Then, GPT-4 performance improved across all prompts except prompt 1 and reached the highest score of 62.25% under Prompt 4 (+27% compared to the baseline). In contrast, GPT-4o performance improved across all the prompts and peaked at 73% with Prompt 5 (+22 points compared to the baseline). These results indicate that grouping triples by table and subject (Approach 3) and using prompts with few-shot examples and explicit separators enhance LLMs’ reasoning by clearly presenting related facts together and guiding the model in interpreting the facts.

For the **Enterprise Benchmark 2** (multi-table), baseline performance was 13.54% for GPT-4 and 16.15% for GPT-4o. GPT-4 steadily improved across prompts except Prompt 1. It reaches 32.29% under Prompt 5, whereas GPT-4o attained its peak of 39.06% under Prompt 3. These gains show that organization of triples grouped by tables and subjects helps LLMs integrate information across multiple tables, while prompts with context, questions presented first, and explicit identifiers further aid reasoning even in complex multi-table scenarios.

For the **FinQA benchmark** (200 QA sample), baseline performance was 22.25% for GPT-4 and 19.75% for GPT-4o. The prompt designs had a noticeable effect. GPT-4 improved most with Prompt 3 reaching 26.75% which is +4.5% compared to baseline. Though Prompts 1 and 4 gave smaller gains, Prompts 2 and 5 decreased performance. GPT-4o showed more consistent improvements across all prompts with Prompt 1 (+5%), Prompt 2 (+1.5%), and Prompt 3 (+13.25%) providing a substantial increase, and Prompts 4 and 5 yield moderate gains of around 8–9% compared to the baseline. These results indicate that explicit instructions with providing context, and questions first, significantly enhance LLM reasoning. Prompts with few-shot examples contributed positively for GPT-4o compared to the baseline.

For the **WikiTableQuestions benchmark**, baseline performance was 16.75% and 14.25% for GPT-4 and GPT-4o, respectively. The GPT-4 model showed the highest performance with Prompt 2 (23.75%), which had explicit separators for context, question, and Prompt 4 (23.75%), which had few-shot examples. Similarly, GPT-4o model performed its best with Prompt 3 (28.75%), which presents the context and question with explicit separators first. These results show that presenting the context and question before the instruction did not significantly improve performance for GPT-4, whereas it proved highly beneficial for GPT-4o.

Final Insights

Overall, these findings indicate that the optimal triple presentation and prompt design depend on the table structure of the dataset. For single-table benchmarks where all relevant information resides within one table, grouping triples by table and subject (finance benchmarks) and grouping by table (open-domain benchmark) enables the model to clearly associate related facts about the same entity, which reduces confusion and improves reasoning consistency. Adding a few-shot examples or explicit identifiers for context and question further strengthens this effect by guiding the model on how to interpret and connect triples to answer the question. On the other hand, for multi-table benchmarks where reasoning requires integrating facts across multiple tables, the simpler sequential presentation (Approach 1) performs better, which highlights that adding few-shot examples in such cases can cause the model to focus too narrowly on local relationships within each table, which hinders cross-table reasoning. Prompts where context and question are provided first with explicit identifiers improve reasoning performance in these scenarios.

5.7. Discussion

The experiments presented in Chapter 5 provide a comprehensive evaluation of how table representation, retrieval strategies, and dataset characteristics influence the performance of table-based question answering systems. The analysis reveals that the effectiveness of a retrieval pipeline depends not only on the retrieval model itself but also on the quality of table preprocessing, the inclusion of relevant contextual information, and the degree of semantic alignment between the table content and user queries. In short, both structural design and semantic representation critically shape retrieval effectiveness.

5.7.1. Effect of Table Representation on Retrieval Performance

The experiments on **table-based linearization formats** revealed distinct patterns in retrieval behavior between sparse and dense methods. For the **BM25** sparse retriever, results showed high sensitivity to tokenization noise and document (table) length. Concise representations such as *header first column* and *table schema* consistently achieved the strongest performance across all benchmarks as these formats maximize lexical overlap with queries while minimizing the context length. In contrast, verbose formats such as HTML and CSV led to degraded performance due to

inflated table length, which adversely affected BM25’s term-frequency scoring due to length normalization.

By contrast, **dense retrieval** models used semantic embeddings to capture contextual similarity and consistently outperformed BM25 across all benchmarks. **Plaintext** and **Markdown** formats yielded the highest average Recall@30 as they align well with how the sentence transformers process text data. This allows embedding models to better capture semantic relationships between headers, cells, and query terms. The formats that performed best for sparse retrieval (table schema, header first column) performed poorly in dense retrievers due to their lack of contextual information, whereas the HTML format, which was well handled in dense retrievers, was likely to be pre-trained using web-based data.

Dataset characteristics further influenced these trends. **WikiTableQuestions** is an open-domain dataset that has lower lexical overlap due to its vocabulary diversity, exhibiting the largest performance improvement when moving from sparse to dense retrieval compared to the other finance-domain datasets. This demonstrates the benefits of semantic similarity in handling diverse vocabulary and natural language queries. Alternatively, **FinQA** and **Enterprise** datasets, which contain finance domain-specific terminology and repetitive financial terms, had less benefit from embeddings due to their high lexical consistency between queries and tables. These results collectively suggest that sparse retrievers excel with concise representation of tables, whereas dense retrievers are superior for information-dense text-like table formats, and the performance boost from sparse to dense retrieval method is more pronounced in datasets with open-domain diversity.

5.7.2. Impact of Context Augmented Table Formats

Following the analysis of table linearized formats on retrievers, experiments were conducted to evaluate the impact of **context augmented table formats** on the retrieval performance. As shown in Section 5.3, table formats enriched with surrounding textual context, such as section headings and subheadings, consistently achieved higher average Recall@30 for both BM25 and dense retrieval. The inclusion of section headings provided concise semantic signals that clarified the scope or purpose of a table (for example, distinguishing between “revenue by region” and “revenue by product line”).

However, excessive or loosely related context, like a pretext, negatively affected retrieval. For BM25, irrelevant narrative text introduced vocabulary mismatch and diluted key term cues, which reduced the retrieval effectiveness. Dense retrievers

were more robust to noise but still exhibited slight declines in recall when excessive non-relevant context was included. These findings suggest that well-curated and semantically relevant context enhances retrieval, whereas inclusion of loosely related surrounding text can introduce unnecessary noise.

5.7.3. SPARQL-based Hybrid Retrieval

Beyond full-table retrieval, the **SPARQL-based hybrid retrieval** introduced in Section 5.4 enabled retrieval of RDF triples (equivalent to table rows). This approach proved to be a robust retrieval method that allows for row-level reasoning and achieves average recall comparable to table-level retrieval methods. Although it operates at a finer granularity, the number of questions that completely failed to retrieve relevant triples was lower than in full-table retrieval in finance datasets. The main advantage of this method lies in its ability to filter and prioritize relevant triples, which is particularly beneficial in a multi-table benchmark where the ground truth triples are distributed across multiple tables.

Nevertheless, several challenges emerged, such as the conversion of large collections of tables into RDF triples, which introduced computational overhead. While variability in schema design across datasets initially affected triple consistency, we mitigated this issue by mapping predicates to a unified ontology, ensuring that the RDF triples were aligned across tables. Furthermore, the final number of retrieved triples (k) required careful tuning of the k value to ensure comparability with full-table retrieval methods. Despite these challenges, the hybrid SPARQL-based method demonstrated substantial improvements in retrieval, particularly in multi-table scenarios.

5.7.4. Table-based Answer Generation and Prompt Design

The answer generation experiments showed that table-level retrieval augmentation is key to improving LLM reasoning. Across all the evaluated benchmarks, the zero-shot performance of LLMs remained minimal (below 10% weighted accuracy), whereas the inclusion of retrieved tables in the context led to substantial performance gains. Among the evaluated table presentation formats for LLMs, tables represented in **HTML** consistently yielded the highest accuracies, followed by the **Markdown** formatted tables with GPT-4 and GPT-4o, which demonstrates that web-based table representations can support the LLMs’ ability to understand structured data.

LLMs can interpret tables, but the performance is highly sensitive to the prompt design used [16]. Prompts that are simple and well-ordered by placing the *context*

and *question* before instructions outperform complex prompt designs. The impact of using few-shot examples depends on how well they align with the task at hand. For single-table benchmarks like FinQA, which require logical and mathematical reasoning, generic examples offer little improvement compared to the baseline. In Enterprise benchmarks, HTML examples can reduce performance due to slight structural differences in the tables when preprocessed in HTML format. The Markdown examples closely match the table preprocessed Markdown format and hence, improve reasoning by showing how to extract information from the context provided. These results indicate that including few-shot examples alone is not enough; their format, structure, and task relevance are key to enhancing LLM performance [50].

These experiments also revealed clear format-dependent trends, that is, with well-structured prompts, presenting tables in context as Markdown outperformed HTML in single-table financial benchmarks using GPT-4o, as this known format for LLMs made relational patterns easier to interpret. For open-domain and multi-table benchmarks, HTML remained more effective with GPT-4 and GPT-4o, respectively, which is likely due to the models' pretraining with web-based tabular structures.

5.7.5. RDF Triple-Based Answer Generation and Prompt Design

The experiments using SPARQL-based hybrid retrieval pipeline evaluated how effectively LLMs could interpret RDF triples formatted as `Subject | Predicate | Object` rather than using entire tables as context. In single-table benchmarks, all required information is contained within one table. Therefore, grouping triples by table and by subject helps the model view all facts for each entity together, which reduces confusion and aids LLM reasoning. Also, adding explicit identifiers and a few-shot examples further guides the model to interpret triples correctly and reason consistently, since this way of representing structured data as context is relatively uncommon.

In multi-table benchmarks, the relevant information is distributed across several tables. Hence, presenting triples sequentially is generally more effective because excessive grouping can cause the model to focus too much on individual tables and overlook cross-table relationships. In addition, including explicit identifiers and placing context, the question first guides reasoning, which allows models, particularly GPT-4o, to integrate information across tables without being overwhelmed by additional grouping.

5.7.6. Summary

Overall, these findings highlight several key insights. First, the retrieval performance is strongly influenced by the table representation format, retrieval model, and dataset complexity. Sparse lexical methods perform best with compact and concise table representations, whereas dense retrieval approaches perform best with information-dense text-like table representations. The SPARQL-based hybrid approach works well for fine-grained row-level retrieval using RDF triples, particularly in multi-table scenarios. For LLM-based reasoning, careful prompt design and the table formats used for context are essential because simple and clear prompts often outperform more elaborate designs. These results emphasize the value of using retrieval pipelines in advancing the effectiveness and robustness of table-based question answering systems.

5.8. Evaluation Against Existing Method

We compare our proposed dense retrieval method and SPARQL-based hybrid retrieval method with two existing baseline approaches.

The first baseline is the method presented in *A Method for Parsing and Vectorization of Semi-Structured Data Used in Retrieval-Augmented Generation* [7], which is used for the retrieval of tables and text using dense embeddings. Although the problem setting is similar to our approach, the paper has several limitations. It uses a chunking strategy that splits tables together with surrounding text and stores them in a single vector index. In addition, the paper shows results on only a small number of English and Chinese examples and does not provide an extensive quantitative evaluation. Therefore, it does not serve as a strong or fully comparable baseline. Despite these limitations, we include this comparison to show how our table chunking and indexing strategy improves retrieval performance on enterprise finance benchmarks. The Appendix A.1 provides the complete comparison and results.

The second baseline is *TableRAG: A Retrieval Augmented Generation Framework for Heterogeneous Document Reasoning* [13], which represents the closest and strongest baseline in terms of methodology and evaluation setting. TableRAG answers questions by combining information from both text and tables. The system consists of two phases, namely, an **offline phase** that extracts tables and text, stores tables in a relational database (MySQL), and builds a FAISS vector index of chunked text and markdown-formatted tables. Then, the **online** phase performs iterative reasoning to answer queries. Initially, the entire question is used to do

retrieval, and the BAAI/bge-m3 retrieves top-30 results, which are then re-ranked using BAAI/bge-reranker-v2-m3, and the top 1 table is selected. Then this table and question are sent to an LLM as a prompt, and it decides whether further query decomposition is needed. If yes, the same retrieval and re-ranking is repeated for the sub queries, and the generated intermediate answers are combined to form a final answer.

While this describes the approach presented in the publicly released code, there were some differences observed in the paper. In the paper, table retrieval is performed using SQL queries to first retrieve the top-30 tables. These tables are then re-ranked, and the top-5 are selected. From these, the top-3 tables are finally used for answer generation. But, the code indexes both `Markdown` formatted tables and textual content together in a single FAISS index in addition to the SQL database. Furthermore, the original code uses the ground truth table identifier or table name along with the question during the initial retrieval step, which was modified in our experiments to use only the question in order to ensure a fair and realistic evaluation setting.

Despite these differences, this paper is selected because it is the closest baseline to our work in terms of problem setting and methodology. Both the paper and our approach focus on table retrieval, use dense embeddings with FAISS, and retrieve top-30 tables before further processing. The paper evaluates on the WikiTableQuestions dataset [49], which is also one of our benchmarks. Another important reason is that the paper includes experiments on both single and multiple ground truth table scenarios, which closely align with the benchmarks we used. Finally, the paper provides a complete pipeline that includes both the retrieval and answer generation stages, making it a strong and relevant baseline for evaluating our system.

However, since we strictly focus only on table retrieval, only tables are stored in both the SQL database and the FAISS index. As our open-source datasets do not contain accompanying text documents, we do not use the text-table hybrid reasoning as described in the paper. The results presented in the following Table 21 compare the paper’s retrieval performance with our proposed semantic vector-based retrieval and our SPARQL-based hybrid retrieval, which shows how each method performs across different benchmarks.

A key reason for our method to achieve an improvement of roughly 7–10% over the paper’s retrieval performance is the way we select models and formats for each dataset. Instead of using a single embedding model for all datasets (as done in the paper), we choose the best-performing model and table format for each dataset individually.

Benchmark	Paper (Baseline)	Proposed Dense Retriever	Proposed Hybrid Approach
Enterprise Dataset Benchmark 1	86.0	96.0	89.2
Enterprise Dataset Benchmark 2	83.3	94.1	96.5
FinQA	77.0	84.0	80.7
WikiTableQuestions	88.0	99.0	86.1

Table 21.: Performance results (Average Recall@30 in %) between baseline and our proposed retrieval approaches.

For all datasets except FinQA, we use plaintext tables instead of Markdown. The plaintext representation does not have formatting artifacts such as pipes or separators, which reduces noise and produces higher quality embeddings, especially for models like `text-embedding-3-large` and `NovaSearch/stella_en_1.5B_v5` (sentence-transformer) models. Furthermore, these models are known to perform better than the model used in the paper according to the MTEB leaderboard [27].

In contrast, our SPARQL-based hybrid method is conceptually different from both the paper and our proposed vector-based approaches. The SPARQL-based hybrid method explicitly extracts entities and predicates from the question and expands them via FAISS, and then retrieves all RDF triples that contain them. For questions that require multiple tables, as in Enterprise Dataset Benchmark 2, this becomes a natural advantage. The goal is to retrieve all tables that contain the required entities rather than retrieving a relevant single table. Our SPARQL-based hybrid approach enables this by directly retrieving the relevant RDF triples, which capture entity-level matches across tables. This is the reason for this method to outperform both the paper and our vector methods on this benchmark. However, when we increase the top-k for vector retrieval, the gap would likely be narrowed further.

Even though the TableRAG paper and our work share several similarities, such as using (top-30) FAISS for retrieval (dense embeddings), and evaluating on both single-and multi-table scenarios, the two approaches are not directly comparable on a strict 1:1 basis. This is mainly because the core RAG logic and the way tables are handled are fundamentally different. The paper stores the tables separately in a SQL database and together with text in an FAISS index. In contrast, our work focuses purely on table retrieval, and therefore, we convert all tables into `plaintext` format and index them directly using FAISS. Additionally, the open-source datasets used in our thesis do not contain additional document text, so we use tables in both SQL and FAISS logic in the paper, which eventually leads to performance differences.

Next, we evaluate the answer generation stage, where we observe clear differences in performance compared to the TableRAG paper as seen in the table 22. Our

Benchmark	Paper (Baseline)	Proposed Dense Retrieval + LLM	Proposed Hybrid Retrieval + LLM
Enterprise Dataset Benchmark 1	30.5	61.0	50.5
Enterprise Dataset Benchmark 2	8.33	52.5	44.92
FinQA	3.0	50.5	21.0
WikiTableQuestions	22.5	59.0	39.0

Table 22.: Performance comparison (weighted accuracy in %) between the baseline and proposed answer generation approaches using **GPT-4o**.

method achieves substantial gains across all datasets, which shows that providing a larger set of retrieval contexts can significantly improve the final answer quality. These differences become clearer when looking at how each system is designed. In the TableRAG paper, the question is broken down into sub-queries for which retrieval is performed repeatedly, and at each step, the top-1 table after re-ranking is used to generate intermediate answers. Then, finally, combined into the final response. In contrast, our approach directly provides the LLM with the top-30 retrieved tables in a single prompt. This allows the model to reason over the entire context at once, and the improvements we see mainly come from this difference in methodology; therefore, the comparison is not fully fair. Additionally, we use the best prompts and table or RDF triples representations for each dataset based on the results from our experiments.

Overall, while our approach has some methodological similarities with TableRAG, the differences in table preprocessing and storage, retrieval strategy selection, and how tables are presented to the LLM show that this is not a strict 1:1 comparison. One important takeaway is that table preprocessing and choosing the right embedding model for the right table representation format have a major impact on retrieval performance. If future embedding models become stronger on web-native formats like `Markdown` and `HTML`, then using those formats may become beneficial. But currently, with sentence-transformer models, `plaintext` remains the most effective representation for tables in our datasets.

6. Conclusion

This thesis explored the challenges of table-based question answering, focusing on improving retrieval and reasoning over structured financial tabular data and extended to an open-domain dataset. It was demonstrated through extensive experiments that table representation, retrieval model, and dataset characteristics play critical roles in retrieval system performance. Sparse lexical methods such as BM25 perform best with concise formats such as `table schema` and `header first column`, whereas embedding-based retrieval excels with text-like formats such as `Plaintext` and `Markdown` as they can capture the semantic relationships that lexical methods often miss. Context augmented table formats with relevant headings also improve performance, moving from lexical to dense retrieval, and excessive or loosely related text can introduce noise, particularly for sparse methods.

On the other hand, the SPARQL-based hybrid retrieval approach, which represents tables as RDF triples, emerged as a robust method for fine-grained retrieval. This enables row-level reasoning and prioritizes relevant triples. Though this method is a little complex as it works at row-level instead of table-level, it still performs on par with or better than both dense retrieval and lexical methods, particularly in multi-table scenarios.

Moreover, the evaluation against existing methods confirmed the robustness of our dense retrieval and SPARQL-based hybrid retrieval methods, with average Recall@30 improving substantially across the single- and multi-table enterprise finance benchmarks. Additionally, LLM-based answer generation experiments highlighted the necessity of retrieval augmentation, as providing retrieved tables or RDF triples along with carefully designed prompts led to significant accuracy gains compared to the zero-shot performance, which was very minimal. When providing RDF triples as context, which is an unfamiliar form of structured data input to LLM, grouping RDF triples by table and subject, and using prompts with explicit identifiers or few-shot examples helped the LLMs understand them better and hence, improved reasoning.

In conclusion, robust Table QA requires approaches that bridge relevant table information retrieval with LLM reasoning. The proposed dense and SPARQL-based

hybrid retrieval achieves this by ensuring both interpretability and high performance across single- and multi-table datasets. This work lays the groundwork for future enhancements such as domain-specific fine-tuning of retrieval models, improved table linearization that preserves structure, and further improvement of LLM reasoning over RDF triples to advance Table QA.

Bibliography

- [1] X. Wang, M. Costa, J. Kovaceva, S. Wang, and F. C. Pereira, “Plugging schema graph into multi-table QA: A human-guided framework for reducing LLM reliance,” *CoRR*, vol. abs/2506.04427, 2025. [Online]. Available: <https://doi.org/10.48550/arXiv.2506.04427>
- [2] M. Trabelsi, Z. Chen, S. Zhang, B. D. Davison, and J. Hefflin, “Strubert: Structure-aware BERT for table search and matching,” in *WWW*. ACM, 2022, pp. 442–451. [Online]. Available: <https://doi.org/10.1145/3485447.3511972>
- [3] J. Herzig, T. Müller, S. Krichene, and J. M. Eisenschlos, “Open domain question answering over tables via dense retrieval,” 2021. [Online]. Available: <https://arxiv.org/abs/2103.12011>
- [4] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, Q. Guo, M. Wang, and H. Wang, “Retrieval-augmented generation for large language models: A survey,” *CoRR*, vol. abs/2312.10997, 2023. [Online]. Available: <https://doi.org/10.48550/arXiv.2312.10997>
- [5] Z. Cheng, H. Dong, Z. Wang, R. Jia, J. Guo, Y. Gao, S. Han, J. Lou, and D. Zhang, “Hitab: A hierarchical table dataset for question answering and natural language generation,” in *ACL (1)*. Association for Computational Linguistics, 2022, pp. 1094–1110. [Online]. Available: <https://doi.org/10.18653/v1/2022.acl-long.78>
- [6] Z. Sepasdar, S. Gautam, C. Midoglu, M. A. Riegler, and P. Halvorsen, “Enhancing structured-data retrieval with graphrag: Soccer data case study,” *CoRR*, vol. abs/2409.17580, 2024. [Online]. Available: <https://doi.org/10.48550/arXiv.2409.17580>
- [7] H. Yang, J. Guo, J. Qi, J. Xie, S. Zhang, S. Yang, N. Li, and M. Xu, “A method for parsing and vectorization of semi-structured data used in retrieval augmented generation,” *CoRR*, vol. abs/2405.03989, 2024. [Online]. Available: <https://doi.org/10.48550/arXiv.2405.03989>

- [8] C. Zhang and Q. Chen, “HD-RAG: retrieval-augmented generation for hybrid documents containing text and hierarchical tables,” *CoRR*, vol. abs/2504.09554, 2025. [Online]. Available: <https://doi.org/10.48550/arXiv.2504.09554>
- [9] A. S. Sundar and L. Heck, “ctbl: Augmenting large language models for conversational tables,” *CoRR*, vol. abs/2303.12024, 2023. [Online]. Available: <https://doi.org/10.48550/arXiv.2303.12024>
- [10] N. Jin, D. Li, J. Chen, J. Siebert, and Q. Chen, “Enhancing open-domain table question answering via syntax- and structure-aware dense retrieval,” in *IJCNLP (2)*. Association for Computational Linguistics, 2023, pp. 157–165. [Online]. Available: <https://doi.org/10.18653/v1/2023.ijcnlp-short.18>
- [11] X. Wang, J. Chi, Z. Tai, T. S. T. Kwok, M. Li, Z. Li, H. He, Y. Hua, P. Lu, S. Wang, Y. Wu, J. Huang, J. Tian, and L. Zhou, “Finsage: A multi-aspect RAG system for financial filings question answering,” *CoRR*, vol. abs/2504.14493, 2025. [Online]. Available: <https://doi.org/10.48550/arXiv.2504.14493>
- [12] P. B. Chen, Y. Zhang, and D. Roth, “Is table retrieval a solved problem? exploring join-aware multi-table retrieval,” in *ACL (1)*. Association for Computational Linguistics, 2024, pp. 2687–2699. [Online]. Available: <https://doi.org/10.18653/v1/2024.acl-long.148>
- [13] X. Yu, P. Jian, and C. Chen, “Tablerag: A retrieval augmented generation framework for heterogeneous document reasoning,” *CoRR*, vol. abs/2506.10380, 2025. [Online]. Available: <https://doi.org/10.48550/arXiv.2506.10380>
- [14] A. Biswal, L. Patel, S. Jha, A. Kamsetty, S. Liu, J. E. Gonzalez, C. Guestrin, and M. Zaharia, “Text2sql is not enough: Unifying AI and databases with TAG,” *CoRR*, vol. abs/2408.14717, 2024. [Online]. Available: <https://doi.org/10.48550/arXiv.2408.14717>
- [15] Z. Guo, X. Ren, L. Xu, J. Zhang, and C. Huang, “Rag-anything: All-in-one RAG framework,” *CoRR*, vol. abs/2510.12323, 2025. [Online]. Available: <https://doi.org/10.48550/arXiv.2510.12323>
- [16] Y. Sui, M. Zhou, M. Zhou, S. Han, and D. Zhang, “Table meets LLM: can large language models understand structured table data? A benchmark and empirical study,” in *WSDM*. ACM, 2024, pp. 645–654. [Online]. Available: <https://doi.org/10.1145/3616855.3635752>

- [17] C. D. Manning, P. Raghavan, and H. Schütze, “An introduction to information retrieval,” 2009. [Online]. Available: <https://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf>
- [18] A. Leventidis, M. P. Christensen, M. Lissandrini, L. D. Rocco, K. Hose, and R. J. Miller, “A large scale test corpus for semantic table search,” in *SIGIR*. ACM, 2024, pp. 1142–1151. [Online]. Available: <https://doi.org/10.1145/3626772.3657877>
- [19] P. Li, Y. He, C. Yan, Y. Wang, and S. Chaudhuri, “Auto-tables: Synthesizing multi-step transformations to relationalize tables without using examples,” *CoRR*, vol. abs/2307.14565, 2023. [Online]. Available: <https://doi.org/10.48550/arXiv.2307.14565>
- [20] A. Aggarwal, “Rag made simple: A beginner’s journey to smarter ai applications,” 2025. [Online]. Available: <https://medium.com/@arushiagg04/rag-made-simple-a-beginners-journey-to-smarter-ai-applications-8f68354d97f7>
- [21] S. Mehta, “Understanding okapi bm25: A guide to modern information retrieval,” 2024. [Online]. Available: <https://adasci.org/understanding-okapi-bm25-a-guide-to-modern-information-retrieval/>
- [22] E. Park, “Understanding okapi bm25 — document ranking algorithm,” 2025. [Online]. Available: https://medium.com/@readwith_emma/understanding-okapi-bm25-document-ranking-algorithm-70d81adab001
- [23] D. Brown, “Rank-bm25: A two line search engine,” 2022. [Online]. Available: <https://pypi.org/project/rank-bm25/>
- [24] V. Karpukhin, B. Oguz, S. Min, P. Lewis, L. Wu, S. Edunov, D. Chen, and W. Yih, “Dense passage retrieval for open-domain question answering,” in *EMNLP (1)*. Association for Computational Linguistics, 2020, pp. 6769–6781. [Online]. Available: <https://doi.org/10.18653/v1/2020.emnlp-main.550>
- [25] J. Moshcovitis and T. Ramdas, “The dense fog of rag: navigating dense retrieval’s blind spots,” 2024. [Online]. Available: <https://chamomile.ai/challenges-dense-retrieval/>
- [26] Y. Li, Z. Liu, C. Xiong, and Z. Liu, “More robust dense retrieval with contrastive dual learning,” in *ICTIR*. ACM, 2021, pp. 287–296. [Online]. Available: <https://doi.org/10.1145/3471158.3472245>

- [27] K. C. Enevoldsen, I. Chung, I. Kerboua, M. Kardos, A. Mathur, D. Stap, J. Gala, W. Siblini, D. Krzeminski, G. I. Winata, S. Sturua, S. Utpala, M. Ciancone, M. Schaeffer, D. Misra, S. Dhakal, J. Rystrøm, R. Solomatin, Ö. V. Çagatan, A. Kundu, and et al., “MMTEB: massive multilingual text embedding benchmark,” in *ICLR*. OpenReview.net, 2025. [Online]. Available: <https://openreview.net/forum?id=zl3pfz4VCV>
- [28] K. Dong, D. Deik, Y. Q. Lee, H. Zhang, X. Li, C. Zhang, and Y. Liu, “Multi-view content-aware indexing for long document retrieval,” *CoRR*, vol. abs/2404.15103, 2024. [Online]. Available: <https://doi.org/10.48550/arXiv.2404.15103>
- [29] D. Bergmann and C. Stryker, “What is vector embedding?” [Online]. Available: <https://www.ibm.com/think/topics/vector-embedding>
- [30] S. Porter, “Understanding cosine similarity and word embeddings,” 2023. [Online]. Available: <https://spencerporter2.medium.com/understanding-cosine-similarity-and-word-embeddings-dbf19362a3c>
- [31] J. Holdsworth and M. Kosinski, “What is a vector database?” [Online]. Available: <https://www.ibm.com/think/topics/vector-database>
- [32] N. Reimers and I. Gurevych, “Sentence-bert: Sentence embeddings using siamese bert-networks,” in *EMNLP/IJCNLP (1)*. Association for Computational Linguistics, 2019, pp. 3980–3990. [Online]. Available: <https://doi.org/10.18653/v1/D19-1410>
- [33] The Alan Turing Institute, “Knowledge graphs.” [Online]. Available: <https://www.turing.ac.uk/research/interest-groups/knowledge-graphs>
- [34] A. Hogan, E. Blomqvist, M. Cochez, C. d’Amato, G. de Melo, C. Gutierrez, J. E. L. Gayo, S. Kirrane, S. Neumaier, A. Polleres, R. Navigli, A. N. Ngomo, S. M. Rashid, A. Rula, L. Schmelzeisen, J. F. Sequeda, S. Staab, and A. Zimmermann, “Knowledge graphs,” *CoRR*, vol. abs/2003.02320, 2020. [Online]. Available: <http://dx.doi.org/10.1145/3447772>
- [35] J. Stegeman, “What is a knowledge graph?” 2024. [Online]. Available: <https://neo4j.com/blog/knowledge-graph/what-is-knowledge-graph/>
- [36] A. Martin and N. Reichmann, “What is a knowledge graph?” 2024. [Online]. Available: <https://www.esri.com/arcgis-blog/products/arcgis-enterprise/data-management/what-is-a-knowledge-graph>

- [37] M. Dürst, “Internationalized resource identifiers (iris),” 2011. [Online]. Available: <https://www.w3.org/International/O-URL-and-ident.html>
- [38] R. O. S. Technologies, “What is an iri? (what does iri mean?).” [Online]. Available: <https://www.oxfordsemantic.tech/faqs/what-is-an-iri-what-does-iri-mean>
- [39] Graph.Build, “Graph model ontology,” 2025. [Online]. Available: <https://graph.build/resources/ontology>
- [40] H. Bast, J. Kalmbach, T. Klumpp, and C. Korzen, “Knowledge graphs.” University of Freiburg, 2023. [Online]. Available: https://ad-publications.cs.uni-freiburg.de/CHAPTER_knowledge_graphs_BKKK_2023.pdf
- [41] S. Harris, A. Seaborne, and E. Prud’hommeaux, “Sparql 1.1 query language,” 2013. [Online]. Available: <https://www.w3.org/TR/sparql11-query/>
- [42] K. Kjernsmo and A. Passant, “Sparql new features and rationale,” 2011. [Online]. Available: https://www.w3.org/2009/sparql/docs/features/#Introduction_features
- [43] H. Bast, J. Kalmbach, R. Textor-Falconi, and C. Ullinger, “Sparqlescope: A generic benchmark for the comprehensive and concise performance evaluation of SPARQL engines,” in *ISWC (2)*, ser. Lecture Notes in Computer Science, vol. 16141. Springer, 2025, pp. 22–40. [Online]. Available: https://doi.org/10.1007/978-3-032-09530-5_2
- [44] H. Bast and B. Buchhold, “Qlever: A query engine for efficient sparql+text search,” in *CIKM*. ACM, 2017, pp. 647–656. [Online]. Available: <https://doi.org/10.1145/3132847.3132921>
- [45] H. Bast, J. Kalmbach, T. Klumpp, F. Kramer, and N. Schnelle, “Efficient and effective SPARQL autocompletion on very large knowledge graphs,” in *CIKM*. ACM, 2022, pp. 2893–2902. [Online]. Available: <https://doi.org/10.1145/3511808.3557093>
- [46] Zilliz, “Information retrieval metrics,” 2024. [Online]. Available: https://medium.com/@zilliz_learn/information-retrieval-metrics-0b50ffc5873b
- [47] M. Douze, A. Guzhva, C. Deng, J. Johnson, G. Szilvasy, P. Mazaré, M. Lomeli, L. Hosseini, and H. Jégou, “The faiss library,” *CoRR*, vol. abs/2401.08281, 2024. [Online]. Available: <https://doi.org/10.48550/arXiv.2401.08281>

- [48] Z. Chen, W. Chen, C. Smiley, S. Shah, I. Borova, D. Langdon, R. Moussa, M. Beane, T. K. Huang, B. R. Routledge, and W. Y. Wang, “Finqa: A dataset of numerical reasoning over financial data,” in *EMNLP (1)*. Association for Computational Linguistics, 2021, pp. 3697–3711. [Online]. Available: <https://doi.org/10.18653/v1/2021.emnlp-main.300>
- [49] P. Pasupat and P. Liang, “Compositional semantic parsing on semi-structured tables,” in *ACL (1)*. The Association for Computer Linguistics, 2015, pp. 1470–1480. [Online]. Available: <https://doi.org/10.3115/v1/p15-1142>
- [50] D. A. Rojo-Echeburúa, “Few-shot prompting: Examples, theory, use cases,” 2024. [Online]. Available: <https://www.datacamp.com/tutorial/few-shot-prompting>

A. Appendix

A.1. Evaluation Against Additional Existing Method

In order to evaluate the effectiveness of the proposed retrieval methods, a comparison against the approach introduced in the paper *A Method for Parsing and Vectorization of Semi-Structured Data Used in Retrieval-Augmented Generation* [7] was conducted. The goal of this comparison was to assess how well the proposed system retrieves tabular data for question answering tasks across both single- and multi-table enterprise finance benchmarks.

The method from the paper (baseline) employed a two-stage process, namely, *parsing* and *vectorization*. In this setup, documents in PDF, HTML, and Word formats were first converted to the `.docx` format and then segmented into textual and tabular chunks. Each chunk was embedded using the `text-embedding-ada-002` model and stored in Pinecone for retrieval. This paper was chosen for comparison because it addresses table retrieval using dense embeddings and a vector index, which is closely related to the goal of our work. At the same time, the paper is a preprint and reports results on only three English and two Chinese examples without extensive quantitative evaluation or comparison with other approaches. Therefore, this comparison should be seen as an approximate baseline used to understand how our method performs rather than as a strict performance reference.

Following the paper’s strategy, our finance enterprise dataset was segmented into 2047 chunks. Each document was divided into small text and table fragments, which were stored in the Pinecone index. In contrast, our chunking strategy is a little different, where we create two indices. The first uses tables as boundaries and combines each table with its surrounding context, like the section heading, subheading, and the pretext and the posttext into a single chunk. The second index includes only the tables in plain-text format. As a result, the dataset produces 397 FAISS indices, which are larger and more meaningful chunks comparatively. The following Figure 24 shows how a table is chunked in the paper’s method and in our proposed method.

€ billions,2024
1/1/2024,5.0
"Increases resulting from billing and invoices becoming due",13.2
"Decreases resulting from satisfaction of performance obligations",-12.3
Other,0.2
12/31/2024,6.1

(a) Original Table in CSV Format.

Contract Liabilities\n\n€ billions 1/1/2024 Increases resulting from billing and invoices becoming due Decreases resulting from satisfaction of performance obligations Other1 12/31/2024\n\n1 Other includes, for example, the impact of foreign currency translation and business combinations.\n\nThe amount of revenue recognized in the reporting period that was included in the contract liability balance at the beginning of the reporting period was €4.7 billion (December 31, 2023: €4.5 billion).

(b) Baseline Method Chunk.

Contract Liabilities | The following table presents the activities impacting contract liabilities balances during the year ended December 31, 2024: | € billions: 1/1/2024 , 2024: 5.0\n€ billions: Increases resulting from billing and invoices becoming due , 2024: 13.2\n€ billions: Decreases resulting from satisfaction of performance obligations , 2024: -12.3\n€ billions: Other , 2024: 0.2\n€ billions: 12/31/2024 , 2024: 6.1 | Other includes, for example, the impact of foreign currency translation and business combinations. The amount of revenue recognized in the reporting period that was included in the contract liability balance at the beginning of the reporting period was â€4.7 billion (December 31, 2023: â€4.5 billion)

(c) Proposed Method Chunk (Table + Text).

€ billions: 1/1/2024 , 2024: 5.0\n€ billions: Increases resulting from billing and invoices becoming due , 2024: 13.2\n€ billions: Decreases resulting from satisfaction of performance obligations , 2024: -12.3\n€ billions: Other , 2024: 0.2\n€ billions: 12/31/2024 , 2024: 6.1

(d) Proposed Method Chunk (Plaintext Table).

Figure 24.: Comparison of chunking strategies between the baseline and proposed method.

We compare the following three retrieval strategies from our proposed framework with the method from the paper as a baseline described above:

1. **Semantic Retrieval (Plaintext)** — uses the top-performing embedding model (NovaSearch/stella_en_1.5B_v5) with tables in plaintext format for vector search (top-k=30).
2. **Semantic Retrieval (Full Context)** — uses the top-performing embed-

ding model (NovaSearch/stella_en_1.5B_v5) with full context (tables with surrounding text) for vector search (top-k=30).

3. **SPARQL-based Hybrid Retrieval** — combines SPARQL-based retrieval of RDF triples with FAISS-based filtering (top-k=3000).

For evaluation, a subset of 100 questions from the **Enterprise Benchmark 1** and the full **Enterprise Benchmark 2** dataset with 60 questions were used for comparison.

Method	Average Recall
Original Paper (Baseline)	0.630
Proposed Dense Retriever (Plaintext)	0.96
Proposed Dense Retriever (Full Context)	0.94
Proposed Hybrid Approach	0.892

Table 23.: Comparison results for Enterprise Benchmark 1 (100 QA) between baseline and proposed retrieval approaches.

For **Enterprise Benchmark 1** as shown in the Table 23, the average Recall increases significantly from 0.630 in the baseline to 0.96 with our *NovaSearch* model using only *Tables* in plaintext approach, 0.94 with *NovaSearch* model using *Full Context* approach, and 0.89 with the SPARQL-based hybrid method. The baseline achieves lower recall mainly because of its chunking strategy, where it splits documents into many small and disconnected text fragments by processing tables as plaintext or HTML, and sometimes large tables are split into multiple smaller chunks, which loses the table structure and the semantic relationship between tables and text. In contrast, our chunking method preserves the table structure and semantic relationship by embedding each table together with its surrounding explanatory text (Table + Text) or by focusing purely on the tabular data in a consistent **Plaintext** format, as shown in the Figure 24.

The table in plaintext format performs best (0.96) because of a number of reasons. Primarily, the benchmark questions are based on the tables and indices, with only tables having no unrelated text, which reduces noise during embedding and keeps the essential information needed for retrieval. Second, the embedding model used in our proposed method is a significantly better retriever, as shown in the MTEB leaderboard [27] than the one used in the baseline study, as the sentence transformer *NovaSearch* model is trained for sentence and passage level embeddings more effectively.

In addition, the SPARQL-based hybrid method also performs strongly (0.89) because it uses entity and relation extraction using LLM for accurate matching of table contents with the query. Then, the substring-based retrieval effectively identifies relevant triples. However, since this approach operates at the triple level instead of the table level and the filtering of top 3000 triples using FAISS can sometimes omit relevant triples required to fully answer a query, which lowers the average recall compared to the *NovaSearch* method.

Method	Average Recall
Original Paper (Baseline)	0.633
Proposed Dense Retriever (Plaintext)	0.941
Proposed Dense Retriever (Full Context)	0.933
Proposed Hybrid Approach	0.965

Table 24.: Comparison results for Enterprise Benchmark 2 between baseline and proposed retrieval approaches.

In Enterprise Benchmark 2, as shown in the Table 24, our proposed approaches again outperform the baseline. The *NovaSearch* model with *Plaintext* tables and *Full Context* achieves recall scores of 0.941 and 0.933, respectively. The SPARQL-based hybrid method achieves a notably higher average recall of 0.965. The improvement in our vector-based methods is mainly because of our chunking strategy and model used. The surrounding text, along with section headings, is captured for each table, which is missing for the chunks in the baseline method. In contrast, the SPARQL-based approach relies on substring matching and focuses directly on the table content. This proves highly effective for this benchmark, where most questions are derived from the tables. However, the baseline method from the original paper was not optimized for multi-table retrieval scenarios, which could also be a reason for its relatively lower performance.

Although the baseline method shows reasonable performance on both single- and multi-table financial benchmarks, it treats tables as unstructured text, which loses table structure and semantic relationships between tables and their associated text. Its chunking strategy often splits large tables into smaller pieces, which makes it difficult to retrieve full, relevant information. In contrast, our proposed methods preserve table structure and related text within each chunk, and the use of a stronger embedding model results in more accurate and reliable retrieval across both benchmarks. Finally, the baseline study evaluates only a small number of English and Chinese examples, which makes it difficult to draw strong conclusions.