TabRAG: Tabular Document Retrieval via Structured Language Representations

Jacob Si[1]*, Mike Qu[2]*, Michelle Lee[1], Yingzhen Li[1]
Imperial College London[1]
Columbia University[2]
{y.si23, chi.m.lee24, yingzhen.li}@imperial.ac.uk
{zq2234}@columbia.edu

Abstract

Ingesting data for Retrieval-Augmented Generation (RAG) involves either fine-tuning the embedding model directly on the target corpus or parsing documents for embedding model encoding. The former, while accurate, incurs high computational hardware requirements, while the latter suffers from suboptimal performance when extracting tabular data. In this work, we address the latter by presenting TabRAG, a parsing-based RAG pipeline designed to tackle table-heavy documents via structured language representations. TabRAG outperforms existing popular parsing-based methods for generation and retrieval. Code is available at: https://github.com/jacobyhsi/TabRAG.

1 Introduction

The latest retrieval-augmented generation (RAG) pipelines focus on fine-tuning language models to output high-quality embeddings [7, 28, 27]. However, these approaches often require expensive computational hardware to extend to new, unseen data. On the other hand, traditional RAG [13, 8] relies on parsing the document to obtain text [15, 6]. While straightforward, the most common parsing techniques [19, 9, 20, 26, 24] often struggle to tackle large, complex tables.

Since the newest embedding models in RAG systems are built upon foundation language models, processing and designing an effective data representation is critical for the embedding model to work effectively [29]. In this work, we propose TabRAG, an end-to-end parsing-based RAG pipeline for tabular documents via structured language representations. TabRAG first segments the data via a layout detector [14, 11] that allows the pipeline to focus on smaller data chunks. Next, we engineer prompts for a vision language model (VLM) to parse various data types [5, 21]. Specifically, for tables, we focus on extracting the values along with their corresponding column and row names into a structured representation. This is followed by a large language model (LLM) that generates natural language descriptions from the structured representation, obtaining a coherent and sequential output that can be effectively encoded by the embedding model [29, 23]. Extensive experiments demonstrate that TabRAG outperforms existing methods in both generation and retrieval across various benchmarks. Our architecture is shown in Figure 1.

2 Method

While retrieval-augmented generation (RAG) performs well on standard text corpora, extending it to tabular documents is not as straightforward. Unlike plain text, which follows a natural sequence and benefits from a wide range of pretrained language embeddings [3, 8, 13], tables convey meaning

*Equal Contribution.

AI for Tabular Data Workshop at EurIPS 2025.

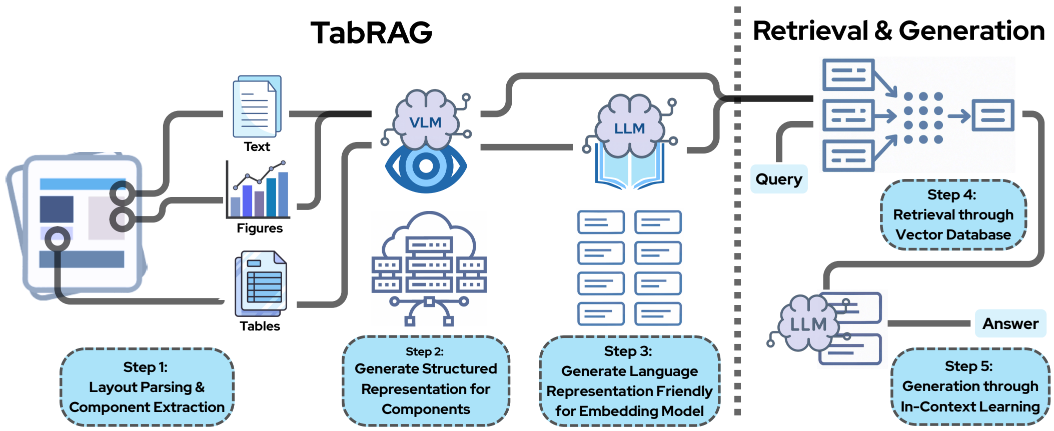

Figure 1: The TABRAG Architecture, a parsing-based RAG pipeline designed specifically for tables. First, a layout detection model is applied to segment various components from the documents. Specifically, the tables are then passed into a vision language model, which extracts cell values along with their corresponding column and row names in a structured representation. Lastly, the structured representation is inputted into a language model that generates natural language descriptions.

through a two-dimensional layout, where each cell's interpretation depends on both its row and column context [2, 22]. Naive text extraction [19, 9, 20] can often disrupt this structure, as it is unable to identify layout-related cues such as merged cells, hierarchical headers, implicit headers, and other spatial relationships. To address these complexities, we propose an alternative approach for parsing-based RAG frameworks, specifically designed to enhance generation and retrieval performance on tabular documents. Our model configurations, prompts, and outputs can be found in Appendix A.1, A.2, and A.3.

2.1 Layout Detection

The first stage of the TABRAG framework, layout detection, identifies and segments visually coherent regions to preserve the document's structural organization. Given an input page image I, we apply a layout detection model f_{layout} [14], which is trained on large-scale document understanding datasets such as PubLayNet [30], to identify bounding boxes and labels corresponding to components like text blocks, tables, figures, and titles. Formally, $f_{\text{layout}}(I)$ outputs a list of detected layout components c_i, each of which is represented as a tuple (b_i, x_i), where b_i denotes the bounding box coordinates, and x_i is the cropped image representation of that component.

To retain contextual cues, we normalize and group components to preserve spatial and hierarchical relations such as figure-caption links and title-table associations. This allows for explicit localized reasoning and addresses several key limitations of page-level methods. When an entire page is represented as a single visual embedding, global attention often blurs fine-grained dependencies that are especially crucial in tables, such as column alignments, header associations, and cell-to-cell relations, erasing these fine-grained dependencies. While locality-aware methods exist, they often rely on loss-based regularization to encourage locality [1, 11]. In contrast, our method models these dependencies explicitly through region-level segmentation, ensuring that spatial and logical relationships are preserved in the representation itself. This allows TABRAG to establish a more faithful and structurally grounded foundation for downstream multimodal processing.

2.2 Vision Language Model

Once the document is decomposed into localized regions, each segment is then semantically interpreted using a vision language model f_{VLM}. We translate the visual structure of each component into a concise, structured, yet information-dense summary of each region. More formally, for each component c_i, we generate a JSON representation $s_i = f_{\text{VLM}}(c_i)$ that captures table structure, hierarchical relations, and the value along with its corresponding row and column names.

Rather than merely transcribing text like an OCR model would, the VLM provides structured understanding of each visual element. It captures cues such as alignment, cell grouping, and

Algorithm 1 TABRAG Framework

Input: Document page I, query q
Output: Grounded response y
$\mathcal{C} \leftarrow f_{\text{layout}}(I) = \{(b_i, x_i)\}_{i=1}^{N}$ {Bounding boxes and cropped regions corresponding to layout components}
for all $(b_i, x_i) \in \mathcal{C}$ **and** Document page I **do**
  $d_i \leftarrow f_{\text{VLM}}(x_i)$ {Extract structured semantic representation d_i from region image}
  $r_i \leftarrow f_{\text{LLM}}(d_i)$ {Reformulate into linguistically natural, embedding-friendly rationale}
end for
$\mathcal{R} \leftarrow \{r_i\}_{i=1}^{N}$ {Corpus of region-level textual rationales}
$\mathcal{E} \leftarrow f_{\text{emb}}(\mathcal{R})$ {Encode rationales into dense embeddings}
$\mathcal{R}_q \leftarrow \text{RetrieveTopK}(q, \mathcal{E}, k)$ {Retrieve top-k relevant rationales}
$y \leftarrow f_{\text{LLM}}(q, \mathcal{R}_q)$ {Generate grounded answer using retrieved evidence}

Figure 2: Algorithmic overview of the TABRAG framework.

formatting style, corresponding to how information is organized and interpreted by humans. This process will pay special attention to irregularities in alignment, merged cells, and other potential edge cases. This semantic enrichment transforms raw visual segments into coherent, context-aware textual representations that can later be indexed and reasoned over by a language model.

In the case where the layout model fails, either due to the model's lack of generalization or ambiguous visual boundaries, we introduce a fallback mechanism where the vision-language model f_{VLM} is also applied to the full document image I. This full-page inference provides a coarse global description that captures overall content even when region extraction is unreliable. During retrieval, summaries produced under this fallback mode are treated equivalently to region-level descriptions, allowing the system to maintain continuity and prevent failure propagation across the pipeline.

2.3 Large Language Model

Structured representations such as JSONs are syntactically well defined but poorly aligned with the token distributions of pretrained foundational embedding models, which are primarily optimized for natural text. This mismatch can degrade embedding quality when used for retrieval or semantic similarity tasks [10, 4].

To address this, we use a large language model f_{LLM} to reformulate each structured region into a linguistically natural description that preserves the same factual content while conforming to the embedding model's training distribution. By producing embedding-friendly textual descriptions, we effectively bridge structured and unstructured modalities, enabling retrieval models to operate in a unified semantic space. The resulting corpus of rationales $\{r_i\}_{i=1}^{N}$ serves as the input to the embedding model f_{emb}, which encodes them into dense vectors for retrieval and downstream reasoning.

3 Experiments

3.1 Setup

TABRAG employs Qwen2.5-VL-32B-Instruct as the vision-language model for region-level semantic extraction, Qwen3-14B (No Think) as the large language model for structured-to-text rationale generation, and Qwen3-Embedding-8B as the embedding backbone for retrieval. All models are used without task-specific fine-tuning. Further experiment details are provided in Appendix B. We leave ablations on model selection and the contribution of individual TABRAG modules for future work.

Datasets. We evaluate TABRAG across multiple document question-answering benchmarks that vary in layout complexity and domain coverage, including TAT-DQA [31], MP-DocVQA [25], WikiTableQuestions [17], and SPIQA [18]. To ensure consistency across modalities, all multi-page documents are preprocessed into single-page units (e.g., `file_p0.pdf`, `file_p1.pdf`) and converted into high-resolution PNG images for methods requiring visual input.

Baselines. We compare our TABRAG against existing parsing frameworks: PyMuPDF, PyTesseract, and Qwen2.5-VL-32B-Instruct, which respectively represent text-based parsing, OCR-based

Table 1: Overall generation performance in accuracy (%) and L3Score (%) for SPIQA.

Models	Accuracy (%) ↑			L3Score (%) ↑
	TAT-DQA	MP-DocVQA	WikiTQ	SPIQA
PyMuPDF	66.83±.000	N/A	59.49±.004	56.14±.003
PyTesseract	62.01±.002	59.73±.002	49.32 ±.005	55.60 ±.003
Qwen2.5-VL-32B-Instruct	63.54±.052	83.59±.650	67.32 ±.243	52.28 ±.107
TabRAG	**92.44**±.006	**86.26**±.005	**69.08**±.008	**60.63**±.009

Table 2: Overall retrieval performance in mean reciprocal rank (MRR@10).

Models	MRR@10 ↑			
	TAT-DQA	MP-DocVQA	WikiTQ	SPIQA
PyMuPDF	75.60±.355	N/A	**86.74±.276**	63.59±.348
PyTesseract	75.95±.354	83.77±.321	82.23±.307	65.72±.341
Qwen2.5-VL-32B-Instruct	74.97±.359	84.33±.319	85.93±.279	**65.86±.351**
TabRAG	**77.86±.339**	**84.98**±.316	86.27±.281	64.86±.339

extraction, and direct multimodal reasoning. For PyMuPDF and PyTesseract, we use the default text extraction and OCR pipelines to reconstruct page-level content, while for Qwen2.5-VL we prompt the model directly on full document images to obtain answer predictions.

Benchmarks. Evaluation metrics include exact-match accuracy and LLMLogScore (L3Score) for generation, as well as mean reciprocal rank (MRR@10) for retrieval.

3.2 Results

Generation. We benchmark TabRAG against baselines on generation tasks to study their effectiveness in generating the answer given the query and the corresponding document processed by various parsing methods. We assume that the corresponding document is the ground truth document that corresponds to the query. In an in-context learning [5] setting, we prompt an LLM (Qwen3-8B [29]) to use the information from the documents to answer the question. We observe in Table 1 that across all benchmarks, TabRAG exhibits consistent and substantial gains over baselines. By aligning visual, structural, and textual cues before generation, TabRAG enables the language model to reason over documents in a manner that preserves both local tabular relationships and global contextual meaning.

Retrieval. We evaluate TabRAG's ability to retrieve the most relevant documents corresponding to each query. Given a query, the model searches over the indexed document representations generated by different parsing methods and ranks them based on semantic similarity. We report the mean reciprocal rank at 10 (MRR@10) as the primary evaluation metric. As shown in Table 2, TabRAG achieves competitive retrieval accuracy compared to existing baselines.

Computational Efficiency. In terms of compute, our method is comparable to directly using a VLM. The computational duration required for the layout detection and LLM is negligible. Since inference time scales by the number of input tokens, processing chunks of data (TabRAG) is comparable to directly processing the whole page via the VLM.

4 Conclusion

In this work, we present TabRAG, a tabular document retrieval framework via structured language representations. Motivated by modern embeddings that are built upon foundational language models, we design a representation that effectively captures meaningful structural and semantic information from tabular documents. Our experimental results on multiple tabular-document QA datasets demonstrate TabRAG's prowess in generation tasks while remaining competitive in retrieval. TabRAG, coupled with state-of-the-art retrievers, can help bridge the gap between table understanding and document-level comprehension.

Acknowledgements

We sincerely thank Professor Marek Rei and Zijing Ou for their helpful discussions and feedback.

References

[1] Srikar Appalaraju, Bhavan Jasani, Bhargava Urala Kota, Yusheng Xie, and R. Manmatha. Docformer: End-to-end transformer for document understanding, 2021.

[2] Sercan O. Arik and Tomas Pfister. Tabnet: Attentive interpretable tabular learning, 2020.

[3] Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. Self-rag: Learning to retrieve, generate, and critique through self-reflection, 2023.

[4] Junwei Bao, Duyu Tang, Nan Duan, Zhao Yan, Yuanhua Lv, Ming Zhou, and Tiejun Zhao. Table-to-text: Describing table region with natural language, 2018.

[5] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

[6] Harrison Chase and the LangChain Contributors. Langchain, 2022.

[7] Manuel Faysse, Hugues Sibille, Tony Wu, Bilel Omrani, Gautier Viaud, Céline Hudelot, and Pierre Colombo. Colpali: Efficient document retrieval with vision language models, 2025.

[8] Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. Realm: Retrieval-augmented language model pre-training, 2020.

[9] Samuel Hoffstaetter and the pytesseract Contributors. A python wrapper for google tesseract. Version 0.3.13, 2025.

[10] Junjie Huang, Wanjun Zhong, Qian Liu, Ming Gong, Daxin Jiang, and Nan Duan. Mixed-modality representation learning and pre-training for joint table-and-text retrieval in openqa, 2022.

[11] Yupan Huang, Tengchao Lv, Lei Cui, Yutong Lu, and Furu Wei. Layoutlmv3: Pre-training for document ai with unified text and image masking, 2022.

[12] Anthony Kay. Tesseract: an open-source optical character recognition engine. *Linux J.*, 2007(159):2, July 2007.

[13] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks, 2021.

[14] Junlong Li, Yiheng Xu, Tengchao Lv, Lei Cui, Cha Zhang, and Furu Wei. Dit: Self-supervised pre-training for document image transformer, 2022.

[15] Jerry Liu and the LlamaIndex Contributors. Llamaindex, 2022.

[16] Minesh Mathew, Dimosthenis Karatzas, and CV Jawahar. Docvqa: A dataset for vqa on document images. In *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, pages 2200–2209, 2021.

[17] Panupong Pasupat and Percy Liang. Compositional semantic parsing on semi-structured tables, 2015.

[18] Shraman Pramanick, Rama Chellappa, and Subhashini Venugopalan. Spiqa: A dataset for multimodal question answering on scientific papers, 2025.

[19] PyMuPDF Development Team. Pymupdf. Version 1.26.5, 2025.

[20] Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025.

[21] Pranab Sahoo, Ayush Kumar Singh, Sriparna Saha, Vinija Jain, Samrat Mondal, and Aman Chadha. A systematic survey of prompt engineering in large language models: Techniques and applications, 2025.

[22] Jacob Si, Wendy Yusi Cheng, Michael Cooper, and Rahul G. Krishnan. Interpretabnet: Distilling predictive signals from tabular data by salient feature interpretation, 2024.

[23] Saba Sturua, Isabelle Mohr, Mohammad Kalim Akram, Michael Günther, Bo Wang, Markus Krimmel, Feng Wang, Georgios Mastrapas, Andreas Koukounas, Nan Wang, and Han Xiao. jina-embeddings-v3: Multilingual embeddings with task lora, 2024.

[24] Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, Pouya Tafti, Léonard Hussenot, Pier Giuseppe Sessa, Aakanksha Chowdhery, Adam Roberts, Aditya Barua, Alex Botev, Alex Castro-Ros, Ambrose Slone, Amélie Héliou, Andrea Tacchetti, Anna Bulanova, Antonia Paterson, Beth Tsai, Bobak Shahriari, Charline Le Lan, Christopher A. Choquette-Choo, Clément Crepy, Daniel Cer, Daphne Ippolito, David Reid, Elena Buchatskaya, Eric Ni, Eric Noland, Geng Yan, George Tucker, George-Christian Muraru, Grigory Rozhdestvenskiy, Henryk Michalewski, Ian Tenney, Ivan Grishchenko, Jacob Austin, James Keeling, Jane Labanowski, Jean-Baptiste Lespiau, Jeff Stanway, Jenny Brennan, Jeremy Chen, Johan Ferret, Justin Chiu, Justin Mao-Jones, Katherine Lee, Kathy Yu, Katie Millican, Lars Lowe Sjoesund, Lisa Lee, Lucas Dixon, Machel Reid, Maciej Mikuła, Mateo Wirth, Michael Sharman, Nikolai Chinaev, Nithum Thain, Olivier Bachem, Oscar Chang, Oscar Wahltinez, Paige Bailey, Paul Michel, Petko Yotov, Rahma Chaabouni, Ramona Comanescu, Reena Jana, Rohan Anil, Ross McIlroy, Ruibo Liu, Ryan Mullins, Samuel L Smith, Sebastian Borgeaud, Sertan Girgin, Sholto Douglas, Shree Pandya, Siamak Shakeri, Soham De, Ted Klimenko, Tom Hennigan, Vlad Feinberg, Wojciech Stokowiec, Yu hui Chen, Zafarali Ahmed, Zhitao Gong, Tris Warkentin, Ludovic Peran, Minh Giang, Clément Farabet, Oriol Vinyals, Jeff Dean, Koray Kavukcuoglu, Demis Hassabis, Zoubin Ghahramani, Douglas Eck, Joelle Barral, Fernando Pereira, Eli Collins, Armand Joulin, Noah Fiedel, Evan Senter, Alek Andreev, and Kathleen Kenealy. Gemma: Open models based on gemini research and technology, 2024.

[25] Rubèn Tito, Dimosthenis Karatzas, and Ernest Valveny. Hierarchical multimodal transformers for multi-page docvqa, 2023.

[26] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023.

[27] Zhepei Wei, Wei-Lin Chen, and Yu Meng. Instructrag: Instructing retrieval-augmented generation via self-synthesized rationales, 2025.

[28] Shi Yu, Chaoyue Tang, Bokai Xu, Junbo Cui, Junhao Ran, Yukun Yan, Zhenghao Liu, Shuo Wang, Xu Han, Zhiyuan Liu, and Maosong Sun. Visrag: Vision-based retrieval-augmented generation on multi-modality documents, 2025.

[29] Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. Qwen3 embedding: Advancing text embedding and reranking through foundation models, 2025.

[30] Xu Zhong, Jianbin Tang, and Antonio Jimeno Yepes. Publaynet: largest dataset ever for document layout analysis, 2019.

[31] Fengbin Zhu, Wenqiang Lei, Fuli Feng, Chao Wang, Haozhou Zhang, and Tat-Seng Chua. Towards complex document understanding by discrete reasoning. In *Proceedings of the 30th ACM International Conference on Multimedia*, MM '22, page 4857–4866. ACM, October 2022.

Appendix

Contents

A Implementation

The following delineates the foundation of our experiments:

- Codebase: Python & PyTorch
- CPU: AMD EPYC 7443P
- GPU: NVIDIA A100 80GB PCIe / NVIDIA A6000 48GB

A.1 Model Configuration

The model configurations used are shown in Table 3. The hyperparameters for the remaining baselines [19, 9] are their default configurations. Qwen3-14B (No Think) and Qwen2.5-VL-32B-Instruct are used for TabRAG, while Qwen3-8B (No Think) is used for evaluating generation performance.

Table 3: Model Configurations.

Model	**Temperature**	**Max Tokens**
Qwen3-14B (No Think)	1.0	8192
Qwen2.5-VL-32B-Instruct	1.0	16384
Qwen3-8B (No Think)	1.0	8192

A.2 Prompts

TabRag

TabRAG VLM Prompt (Table)

```
You are a precise information extraction engine. Output ONLY a JSON array of objects, each with:
{"row": <string>, "column": <string>, "value": <string|null>}.
No markdown, explanations, or text before/after the JSON.

Task: Extract every visible cell in the attached table image into JSON triples.

Each table cell must be represented as:
{
"row": string,         // the row label (e.g. "Revenue", "2024", "Row 1" if unnamed)
"column": string,      // column header text; if multi-level, join levels with " -> "
"value": string|null   // exact text as seen in the table (keep symbols and brackets)
}

Rules:
- Output ONLY a JSON array: [ {row, column, value}, ... ].
- Order: top-to-bottom, left-to-right.
- Preserve all text formatting exactly as shown:
- Keep parentheses, minus signs, commas, currency symbols, and percent signs.
- Do NOT normalize numbers or remove punctuation.
- Multi-line text: join with a single space.
- Multi-level headers: join with " -> " (e.g. "2024 -> Revenue").
- If a row header spans multiple rows, repeat its label for each affected row.
- Use null only for empty or blank cells.

---

**Example 1: Two-level header**

Input:
| Item          | 2024                | 2023                |
|--------------|--------------------|--------------------|
|              | Revenue | Profit    | Revenue | Profit    |
| Sales        | 1,234   | 400       | 1,200   | 350       |
| Net Income   | (56)    | 80        | -40     | 70        |

Output:
[
{"row": "Sales", "column": "2024 -> Revenue", "value": "1,234"},
{"row": "Sales", "column": "2024 -> Profit", "value": "400"},
{"row": "Sales", "column": "2023 -> Revenue", "value": "1,200"},
{"row": "Sales", "column": "2023 -> Profit", "value": "350"},
{"row": "Net Income", "column": "2024 -> Revenue", "value": "(56)"},
{"row": "Net Income", "column": "2024 -> Profit", "value": "80"},
{"row": "Net Income", "column": "2023 -> Revenue", "value": "-40"},
{"row": "Net Income", "column": "2023 -> Profit", "value": "70"}
]

---

**Example 2: Three-level header**

Input:
| Metric    | 2024                                  | 2023                                  |
|-----------|--------------------------------------|--------------------------------------|
|           | Q1                 | Q2               | Q1                 | Q2               |
|           | Revenue  | Profit  | Revenue  | Profit | Revenue  | Profit  | Revenue  | Profit |
| Product A | 500      | 120     | 600      | 150    | 450      | 100     | 550      | 140    |
| Product B | (50)     | 80      | (30)     | 100    | -20      | 60      | 10       | 90     |

Output:
[
{"row": "Product A", "column": "2024 -> Q1 -> Revenue", "value": "500"},
{"row": "Product A", "column": "2024 -> Q1 -> Profit", "value": "120"},
{"row": "Product A", "column": "2024 -> Q2 -> Revenue", "value": "600"},
{"row": "Product A", "column": "2024 -> Q2 -> Profit", "value": "150"},
{"row": "Product A", "column": "2023 -> Q1 -> Revenue", "value": "450"},
{"row": "Product A", "column": "2023 -> Q1 -> Profit", "value": "100"},
{"row": "Product A", "column": "2023 -> Q2 -> Revenue", "value": "550"},
{"row": "Product A", "column": "2023 -> Q2 -> Profit", "value": "140"},
{"row": "Product B", "column": "2024 -> Q1 -> Revenue", "value": "(50)"},
{"row": "Product B", "column": "2024 -> Q1 -> Profit", "value": "80"},
{"row": "Product B", "column": "2024 -> Q2 -> Revenue", "value": "(30)"},
{"row": "Product B", "column": "2024 -> Q2 -> Profit", "value": "100"},
{"row": "Product B", "column": "2023 -> Q1 -> Revenue", "value": "-20"},
{"row": "Product B", "column": "2023 -> Q1 -> Profit", "value": "60"},
{"row": "Product B", "column": "2023 -> Q2 -> Revenue", "value": "10"},
{"row": "Product B", "column": "2023 -> Q2 -> Profit", "value": "90"}
]

---
```

TabRAG VLM Prompt Cont. (Table)

```
**Example 3: Mixed 1-row, 2-row, and 3-row headers**

Input:
| Category | 2024                                  | 2023              | Growth % | Notes    |
|----------|---------------------------------------|-------------------|----------|----------|
|          | Q1                 | Q2               | Revenue | Profit |          |          |
|          | Revenue  | Profit | Revenue | Profit |         |        |          |          |
| Sales    | 1,000    | 300    | 900     | 250    | 1,700   | 550    | 12%      | N/A      |
| Cost     | (200)    | (50)   | -180    | -40    | (380)   | (90)   | N/A      | Adjusted |

Output:
[
{"row": "Sales", "column": "2024 -> Q1 -> Revenue", "value": "1,000"},
{"row": "Sales", "column": "2024 -> Q1 -> Profit", "value": "300"},
{"row": "Sales", "column": "2024 -> Q2 -> Revenue", "value": "900"},
{"row": "Sales", "column": "2024 -> Q2 -> Profit", "value": "250"},
{"row": "Sales", "column": "2023 -> Revenue", "value": "1,700"},
{"row": "Sales", "column": "2023 -> Profit", "value": "550"},
{"row": "Sales", "column": "Growth %", "value": "12%"},
{"row": "Sales", "column": "Notes", "value": "N/A"},
{"row": "Cost", "column": "2024 -> Q1 -> Revenue", "value": "(200)"},
{"row": "Cost", "column": "2024 -> Q1 -> Profit", "value": "(50)"},
{"row": "Cost", "column": "2024 -> Q2 -> Revenue", "value": "-180"},
{"row": "Cost", "column": "2024 -> Q2 -> Profit", "value": "-40"},
{"row": "Cost", "column": "2023 -> Revenue", "value": "(380)"},
{"row": "Cost", "column": "2023 -> Profit", "value": "(90)"},
{"row": "Cost", "column": "Growth %", "value": "N/A"},
{"row": "Cost", "column": "Notes", "value": "Adjusted"}
]

---

Now, extract all visible cells from the attached table image and output only the JSON array of
{row, column, value} objects using the " -> " separator for multi-level headers, keeping all cell
values exactly as written in the table. ENSURING THAT ALL EXTRACTED VALUES ARE ACCURATE IS THE MOST
IMPORTANT! DO NOT OUTPUT ANYTHING ELSE.
```

TabRAG VLM Prompt (Text)

```
Please extract and output the **visible text** in the image exactly **as it appears**, without
rephrasing, summarizing, or skipping any content. Preserve original formatting such as line breaks,
punctuation, and capitalization. This includes any small footnotes or embedded labels. DO NOT OUTPUT
ANYTHING ELSE!
```

TabRAG VLM Prompt (Title)

```
Please extract and output the **title text** from the image exactly **as displayed**, preserving
capitalization and formatting. Do not interpret or rewrite. Output the title as it appears visually.
DO NOT OUTPUT ANYTHING ELSE!
```

TabRAG VLM Prompt (Figure)

```
Please interpret the figure and describe it in detail. Your output should include:
1. Descriptions of individual data points if visible,
2. Descriptions of trend lines, axes, and labels,
3. Explanations of any color or shape encodings, and
4. Any other notable features (e.g., anomalies, clustering, outliers).
Be precise and avoid speculation. Ensure your interpretation **accurately matches the figure** and
corresponds to what is visually present. DO NOT OUTPUT ANYTHING ELSE!
```

TabRAG VLM Prompt (Page)

```
Please parse everything in the attached image and output the parsed contents only without anything
else.
```

TabRAG LLM Prompt (Table)

```
You receive one JSON object containing "cells", each cell having:
{"row": <string>, "column": <string>, "value": <string|null>}.
These cells were extracted from a table using the " -> " convention for multi-level headers.

You must output one natural-language sentence per cell, each on a new line.

Your natural language description MUST include the cell's value, and provide a description of what
the cell describes.
Put all of this information in context, use your discretion, and produce a succinct, reasonable
description.
You **MUST NOT** use table terminology (e.g. the value for row A column B is C) in your response.

EXAMPLE:
Input:
[
{"row": "Sales", "column": "2024 -> Q1 -> Revenue", "value": "1,000"},
{"row": "Sales", "column": "2024 -> Q1 -> Profit", "value": "300"},
{"row": "Sales", "column": "2024 -> Q2 -> Revenue", "value": "900"},
{"row": "Sales", "column": "2024 -> Q2 -> Profit", "value": "250"},
{"row": "Sales", "column": "2023 -> Revenue", "value": "1,700"},
{"row": "Sales", "column": "2023 -> Profit", "value": "550"},
{"row": "Sales", "column": "Growth %", "value": "12%"},
{"row": "Sales", "column": "Notes", "value": "N/A"},
{"row": "Cost", "column": "2024 -> Q1 -> Revenue", "value": "(200)"},
{"row": "Cost", "column": "2024 -> Q1 -> Profit", "value": "(50)"},
{"row": "Cost", "column": "2024 -> Q2 -> Revenue", "value": "-180"},
{"row": "Cost", "column": "2024 -> Q2 -> Profit", "value": "-40"},
{"row": "Cost", "column": "2023 -> Revenue", "value": "(380)"},
{"row": "Cost", "column": "2023 -> Profit", "value": "(90)"},
{"row": "Cost", "column": "Growth %", "value": "N/A"},
{"row": "Cost", "column": "Notes", "value": "Adjusted"}
]

OUTPUT:
In Q1 of 2024, the Sales Revenue is 1,000.
In Q1 of 2024, the Sales Profit is 300.
In Q2 of 2024, the Sales Revenue is 900.
In Q2 of 2024, the Sales Profit is 250.
In 2023, the Sales Revenue is 1,700.
In 2023, the Sales Profit is 550.
The Sales Growth % is 12%.
The Sales Notes are N/A.
In Q1 of 2024, the Cost Revenue is (200).
In Q1 of 2024, the Cost Profit is (50).
In Q2 of 2024, the Cost Revenue is -180.
In Q2 of 2024, the Cost Profit is -40.
In 2023, the Cost Revenue is (380).
In 2023, the Cost Profit is (90).
The Cost Growth % is N/A.
The Cost Notes are Adjusted.
}

You MUST do this for the full JSON object. Do not omit, or skip any cell.
```

Baseline: Qwen2.5-VL-32B-Instruct

Baseline: Qwen2.5-VL-32B-Instruct Prompt

```
Please parse everything in the attached image and output the parsed contents only without anything
else.
```

A.3 Outputs

TabRag

TabRAG VLM Output (Table)


````
```json
[
    {"row": "Non-current assets", "column": "2019 $ million", "value": "196.9"},
    {"row": "Non-current assets", "column": "2018 $ million", "value": "184.6"},
    {"row": "Americas", "column": "2019 $ million", "value": "7.4"},
    {"row": "Asia Pacific", "column": "2019 $ million", "value": "11.5"},
    {"row": "Europe, Middle East and Africa", "column": "2019 $ million", "value": "215.8"},
    {"row": "Non-current assets", "column": "2018 $ million", "value": "4.4"},
    {"row": "Americas", "column": "2018 $ million", "value": "5.1"},
    {"row": "Asia Pacific", "column": "2018 $ million", "value": "194.1"}
]
```
````


TabRAG LLM Output (Table)

```
In 2019, the Non-current assets amount is 196.9 $ million.
In 2018, the Non-current assets amount is 184.6 $ million.
In 2019, the Americas amount is 7.4 $ million.
In 2019, the Asia Pacific amount is 11.5 $ million.
In 2019, the Europe, Middle East and Africa amount is 215.8 $ million.
In 2018, the Non-current assets amount is 4.4 $ million.
In 2018, the Americas amount is 5.1 $ million.
In 2018, the Asia Pacific amount is 194.1 $ million.
```

B Experiments

B.1 Datasets

Prior to constructing the ragstores, the datasets undergo preprocessing. Each file in the dataset can contain multiple pages; thus, for consistency, we ensure that each file is separated as individual files containing a single page each i.e. `file_p0.pdf`, `file_p1.pdf` etc. For methods that require image files as inputs, we convert data modalities such as PDFs and HTML files into high-resolution PNG files with 288 DPI.

Table 4: Dataset Overview.

Dataset	Document Type	# Pages	# QA Pairs
TAT-DQA [31]	Financial	312	1640
MP-DocVQA [25]	Multi-domain	500	515
WikiTQ [17]	Web Tables	243	511
SPIQA [18]	ArXiV PDF	1090	300

TAT-DQA [31]. TAT-DQA comprises of PDF document pages that include both semi-structured tables and unstructured textual content, each paired with corresponding question–answer sets. The documents are drawn from financial reports and feature a strong presence of numerical information. The dataset consists of separated individual PDF pages thus, there is no designated "group" of pages available for retrieval experiments. Hence, we first sort the document pages then split these pages into groups of 25. This yields 12 folders which we use to construct ragstores for retrieval.

MP-DocVQA [25]. Document Visual Question Answering (DocVQA) datasets answer questions from document images. MP-DocVQA extends DocVQA [16] to the multi-page scenario where DocVQA contains images such as typed or handwritten text, layout, diagrams, figures, tabular structures, and signatures. The dataset consists a total of 46K questions, 6K documents, and 48K pages (Images). We filter the dataset by selecting documents that contains tables (via layout detection), followed by documents with a high number of questions to pages ratio.

SPIQA [18]. SPIQA (Scientific Paper Image Question Answering) is a question-answering dataset specifically designed to interpret complex figures and tables within the context of scientific research articles across various domains of computer science. SPIQA comprises 270K questions divided into training, validation, and three different evaluation splits. We utilize the training split, and filter the dataset by selecting documents that contains tables (via layout detection), followed by documents with a high number of questions to pages ratio.

WikiTableQuestions [17]. WikiTableQuestions aim to answer complex questions on semi-structured HTML tables using question-answer pairs as supervision. The dataset consists of question-answer pairs corresponding to an accompanying table. The questions include a variety of operations such as comparison, superlatives, aggregation, and arithmetic.

B.2 Baselines

TABRAG's performance is evaluated in comparison to popular parsing methods. This includes PyMuPDF [19], PyTesseract [9] and a vision language model, Qwen2.5-VL-32B-Instruct [20].

PyMuPDF [19]. PyMuPDF is a Python library for data extraction and analysis of PDF documents. It interfaces with the MuPDF rendering engine, which parses the PDF's internal object structure—decoding vector drawing commands, text placements, and embedded resources—to reconstruct the page layout.

PyTesseract [9]. PyTesseract is a Python wrapper for Google's Tesseract OCR engine [12], used for extracting text from images and scanned documents. It performs a raster-based analysis of the input image—segmenting it into connected components, detecting character shapes, and matching them against trained language models—to recognize and reconstruct textual content from pixel patterns.

Qwen2.5-VL-32B-Instruct [20]. Qwen2.5-VL-32B-Instruct is a large-scale vision-language model developed by Alibaba Cloud. It integrates visual and textual understanding through a unified multimodal transformer architecture capable of processing both images and natural language. The model is trained on large-scale multimodal corpora, enabling it to interpret complex document layouts, tables, and figures while aligning visual cues with textual semantics. Its instruction-tuned variant enhances performance on document understanding and question answering tasks by following structured prompts and adapting to diverse multimodal reasoning scenarios.

B.3 Benchmarks

Accuracy [28, 7]. Accuracy is computed by evaluating the model's ability to produce correct answers to the ground-truth questions. For each document, the corresponding docstore and question are passed to the language model to generate an answer. The generated response is normalized and compared against the normalized ground-truth answers. A prediction is counted as correct if all ground-truth answers appear in the model's response. The final accuracy is calculated as the ratio of correctly answered questions to the total number of evaluated questions.

LLMLogScore (L3Score) [18]. LLMLogScore (L3Score) incorporates the confidence of LLMs for assessing the equivalence of answers with the ground-truths based on the log-likelihood token probabilities. For each question, the corresponding document page is identified, and its vector store is loaded to retrieve the most relevant context using the question embedding. The retrieved context and question are then passed to the language model to generate a candidate answer. A separate judging model evaluates the candidate against the ground-truth answer by computing a log-likelihood–based similarity score, reflecting both correctness and semantic alignment. The final L3Score is reported as the average score across all evaluated questions.

Mean Reciprocal Rank [28]. Mean Reciprocal Rank (MRR@10) evaluates the ranking quality of the retrieved document components relative to the ground-truth evidence. For each question, the corresponding document's vector store is loaded, and the question embedding is used to retrieve the top-10 most relevant components. The reciprocal rank is computed based on the first retrieved document containing the ground-truth evidence. The final MRR@10 score is obtained by averaging the reciprocal ranks across all evaluated questions, as defined by

$$\text{MRR} = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{\text{rank}_i}, \tag{1}$$

where $|Q|$ is the total number of questions and rank_i denotes the rank position of the first relevant retrieved component for the i-th query.