SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Information Systems

Evaluating and Enhancing Location-Aware Visual Document Segmentation for Oncology Guidelines

Matteo Felipe Merz

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Information Systems

Evaluating and Enhancing Location-Aware Visual Document Segmentation for Oncology Guidelines

Evaluierung und Erweiterung positioneller visueller Dokumentensegmentierung für Onkologie-Richtlinien

Author:	Matteo Felipe Merz
Supervisor:	Prof. Dr. Florian Matthes
Advisor:	Jonas Gottal, M.Sc.; Juraj Vladika, M.Sc.
Submission Date:	22.02.2026

I confirm that this bachelor's thesis in information systems is my own work and I have documented all sources and material used.

Location, Submission Date

Author

AI Assistant Usage Disclosure

Introduction

Performing work or conducting research at the Chair of Software Engineering for Business Information Systems (sebis) at TUM often entails dynamic and multi-faceted tasks. At sebis, we promote the responsible use of *AI Assistants* in the effective and efficient completion of such work. However, in the spirit of ethical and transparent research, we require all student researchers working with sebis to disclose their usage of such assistants.

For examples of correct and incorrect AI Assistant usage, please refer to the original, unabridged version of this form, located at this link.

Use of *AI Assistants* for Research Purposes

I have used AI Assistant(s) for the purposes of my research as part of this thesis.

Yes ◯ No ◯

Explanation:

I confirm in signing below, that I have reported all usage of AI Assistants for my research, and that the report is truthful and complete.

Location, Date

Author

Acknowledgments

Abstract

Kurzfassung

Contents

1. Introduction

1.1. Problem Statement

Clinical practice guideliness (CPGs) are fundamental to the efficient and reliable treatment of various illnesses [1]. Oncology guidelines are a subgroup of these documents, revolving around the treatment of various forms of cancer [2]. CPGs not only aid doctors in deciding on the optimal treatment options but also support patients in understanding their illness. In recent years, due to advancements in technology, novel therapeutics, and personalized medicine, clinical guidelines have drastically increased in size and complexity [3]. As of 2019, the average oncology guideline published by the National Comprehensive Cancer Network (NCCN) was 198 pages long, showing an annual increase of 7.5 percent over the previous 23 years [3]. This increase of complexity forces medical personnel to invest more time in order to be able to provide optimal care for cancer patients. Especially for individual practitioners this additional strain might become unsustainable if complexity continues to increase [3].

The Aidvice project proposes to address this problem by leveraging recent advantages in artificial intelligence (AI). Specifically, the project revolves around the development of a retrieval-augmented generation (RAG) based knowledge assistant [4]. RAG is an emerging paradigm which addresses a fundamental problem of traditional large language models (LLMs) [5]. While LLMs excel at many natural language processing (NLP) tasks, they are prone to 'hallucinations' and inaccurate answers, when sought information goes beyond the model's training data [6]. This provides a major obstacle for the usage of LLMs in the medical field, where accurate and reliable answers are of the highest priority [7]. RAG mitigates these drawbacks by retrieving additional context from an external knowledge source which the LLM can take advantage of during answer generation [8, 5].

The efficiency of such a RAG system is fundamentally constrained by the quality and relevance of the context retrieved from the knowledge base [9, 10]. Therefore, the construction of the knowledge base out of the oncology guidelines is a critical aspect of the project. Additionally, as the project has a clear focus on verifiability and traceability, there is an additional requirement to provide visual source attribution with the models responses. This means that retrieved passages need to include accurate positional information, giving visual confirmation to the practitioner about the origin of the retrieved context [11].

The guideline documents are stored in the unstructured portable document format (PDF). In order to be further processed for the knowledge base, they first need to be transformed into a machine-readable structured data format through a process called document parsing (DP) [12]. The inherent structure of the guidelines poses multiple challenges for this process, such as complex tables, varying layouts and occasional formatting errors.

As LLMs are constrained by the size of their context window, it is not feasible to store the entire guideline documents as individual entries in the knowledge base [6]. Therefore, the documents need to be split up into smaller text chunks that fit into the model's context window [6]. This process is called chunking [13].

During retrieval the model identifies the most relevant passages in the knowledge base based on their similarity to the user's query [5]. If the stored text chunks are too long, important information might be lost between irrelevant details [9, 10]. On the other hand, storing too short text chunks can result in important statements being broken up into multiple chunks and losing their meaning. In order to maximize the quality of the retrieved chunks, both the chunk size as well as the chunking strategy, used to decide where to split up the oncology guidelines, need to be optimized [10].

Additionally, established implementations of popular chunking strategies do not fulfill the requirement of visual source attribution at the granularity required by the Aidvice project [14, 15, 16, 17]. Therefore there is a need for the development of a novel solution that addresses this issue.

1.2. Objectives

This study addresses three fundamental research questions regarding the data preparation for a RAG based knowledge assistant for oncology guidelines. Each of the following research questions addresses a specific aspect of the evaluation and improvement of the document segmentation process required for the construction of the knowledge base.

- **RQ1:** How are the challenges introduced by oncology guidelines reflected in established benchmarks for document parsing?
- **RQ2:** Which metrics are most useful to measure the effectiveness of document parsing and chunking methods?
- **RQ3:** How can current segmentation methods be adapted or expanded on to fulfill the requirements of a RAG based knowledge assistant with visual source attribution?

To identify the challenges posed by the oncology guidelines to the DP process, we perform a qualitative analysis identifying the characteristics of oncology guidelines that are relevant to the DP process, such as their formatting, layout and common types of structural elements. We then identify established document benchmarks and datasets which contain documents that most closely resemble these characteristics. Through this analysis, we underline the transferability of results achieved on these datasets to our application, while identifying unrepresented characteristics which require manual comparisons. This approach allows the evaluation of various DP techniques on established benchmarks, without the availability of a dedicated oncology guideline benchmark.

In order to evaluate the effectiveness of both document parsing and chunking strategies, we identify various metrics used in existing literature. We then perform a comparative analysis of the identified metrics, evaluating their suitability for our application. Based on this analysis, we select a set of metrics which are most suitable for our evaluation.

Finally, we propose a novel solution for the visual source attribution requirement of the Aidvice project. We adapt and expand on existing chunking strategies in order to provide accurate positional information for each text chunk. By introducing a universal data format for the output of the DP implementations, we enable the direct comparison of various document parsing techniques using the benchmarks and metrics identified in RQ1 and RQ2. Through this evaluation we identify promising combinations of document parsing and chunking strategies for the creation of the knowledge base of the Aidvice project, while providing a modular framework for future experiments and improvements.

2. Foundations

2.1. Oncology guideline documents

CPGs help improve patient care by giving recommendations on the optimal treatment and prevention of various diseases [18, 19]. They are developed by groups of independent multi-disciplinary experts and are based on a robust systematic review of available treatment options and knowledge gained from clinical experience [1, 18, 19]. Instead of dictating a single definitive treatment option, CPGs instead focus on aiding the decision making process, promoting treatment options with proven benefits and discouraging ineffective or harmful treatments [1, 18]. As such, they aim to improve the quality of the provided health care by encouraging the translation of research into medical practice [19]. Oncology guidelines are a subgroup of this document type, focusing on the treatment and rehabilitation options for various types of cancers [19].

In order to further improve the quality and standardization of oncology guidelines [19, 3], and therefore cancer care, several prominent organizations have emerged, which endorse and publish selected oncology guidelines. Prominent examples include the NCCN [2], the European Society for Medical Oncology (ESMO) [20], and, for oncology guidelines in the German language, the 'Arbeitsgemeinschaft der Wissenschaftlichen Medizinischen Fachgesellschaften' (AWMF) [21]. Over the last decades oncology has seen many advances in the research and treatment outcomes of many forms of cancers [3, 22]. Following these findings and advancements, the number of available treatment option has increased drastically [3]. This increase in available treatments is ultimately reflected in the increasing complexity of oncology guidelines. Kann, Johnson, Aerts, et al. [3] found that, between 1996 and 2019, the mean page count of guidelines published by the NCCN has increased from 26 to 198 pages, with the number of referenced citations per guideline also increasing from an average of 30 to 111.

In order to identify common characteristics between the layout, typography and page design of different oncology guideline documents, we perform a qualitative analysis on a selection of german and english guideline documents from multiple publishing organizations. Despite significant variability between guidelines from different publishers, several shared characteristics can be observed:

Data format: The primary data format for digital distribution of oncology guidelines is the PDF. PDF is a data format designed to enable the reliable distribution and viewing of electronic documents independent of the viewing or creating environment [23]. Particularly, these documents are born-digital PDF files, created through digital processes, instead of scanning analog documents.

Page geometry: All observed documents are provided in the standard A4 format, thereby sharing common page dimensions. While the majority of oncology guidelines are provided in a vertical orientation, both horizontal and mixed page orientations are possible. Additionally, there exist some cases where two neighboring vertical pages are contained in a single horizontal page.

Content and layout: The formatting and content of the guideline documents is heavily dependent on their target audience. 'Standard' guideline documents, addressing medical professionals, resemble typical scientific documents. They are mostly provided in a single or double-column layout, and, due to their focus on aggregating the results of previous studies, predominantly text-heavy. Additionally, they often contain complex tables which may span multiple pages, primarily to compare different treatment options against each other. While less frequent, figures, mathematical formulas and images are also occasionally included. As CPGs are often too complicated for patients to understand, some publishers provide 'patient guidelines' alongside their CPGs. These documents translate the recommendations from the CPG into a language that is understood by the general population, while leaving out scientific details that are less relevant to the patient. Compared to the CPG these documents usually incorporate more figures and visual elements while offer more variability in their typography and page designs.

Document quality: Depending on the CPG's age and publishing organization, the formatting of the document may contain significant structural errors. Observed formatting issues include overlapping text, empty pages between content, tables extending into the page margins, and invisible text on document pages.

2.2. Natural Language Processing Fundamentals

According to Hirschberg and Manning [24], NLP "employs computational techniques for the purpose of learning, understanding, and producing human language content" (p.1). The introduction of the transformer architecture by Vaswani, Shazeer, N. Parmar, et al. [25] and the subsequent development of LLMs has revolutionized the field in recent years [26]. In order to understand how LLMs process and perceive information, it is necessary to examine various fundamental concepts.

2.2.1. Tokenization

Tokenization refers to the segmentation of text into sub-word units called tokens [27]. Tokens are the fundamental text representation for most NLP tasks. With a granularity located between characters and words, tokens can retain linguistic meaning while also being able to represent arbitrary text with a relatively concise vocabulary [27]. Using tokenization any given text can essentially be represented as a list of integers, with each integer being the identifier to a specific token in the tokenizer's dictionary [28]. During training, the tokenizer creates its dictionary by finding character pairings that occur with the highest frequency in the training data [28]. Additionally, with the multitude of different techniques for modern sub-word tokenization [29, 30, 31], the same input text can lead to drastically different outputs depending on the specific tokenizer and training data. Therefore, tokenizers always need to match the NLP models they are used with.

2.2.2. Contextual Embeddings

Contextual embeddings encode the semantical meaning of words into vectors of fixed-dimensionality. In contrast to earlier static embeddings, they take into account the meaning of a word inside a passage, such as what concept a pronoun relates to. Contextual embeddings are made possible through the emergence of the transformer concept.

Using this technique, the meaning text is transformed into a machine-understandable format, with closely related sentences being closer to each other in the vector space.

Content:

- Definition
- Types of embeddings

2.2.3. Semantic Similarity

Content:

- Definition
- Role in RAG

2.3. Vision-Language Models

LLMs are inherently confined to processing exclusively text-based data. This limitation restricts their applicability in complex, real-world scenarios, where understanding and combining data from multiple modalities is crucial [32, 33]. Vision-language model (VLM) are a class of models which respond to these limitations by combining visual and textual processing capabilities into a single architecture [32]. These models find applications involving both the comprehension and generation of multi-modal content, such as image captioning, and visual question answering [32].

2.3.1. Bounding Boxes

Bounding boxes represent the most fundamental method for annotating the position of an object within an image. A bounding box is the smallest rectangle that fully encloses the shape of the object [34]. These boxes are defined within the image's coordinate system, with its origin typically positioned at the top-left corner of the image [35]. The x-axis extends horizontally from this point, while the y-axis extends vertically. Coordinates can either be expressed in absolute pixel units or as normalized fractional values relative to the dimension's of the image. For this study, we focus exclusively on horizontal bounding boxes, which are aligned to the horizontal axis, also known as Feret Boxes [34]. There are multiple formats for representing bounding boxes, with the left-top-right-bottom (LTRB) notation, which denotes the coordinates of the top-left and bottom-right corners of the bounding box, being a prominent option [35].

2.3.2. Intersection over Union

According to the definition from Kaur and Singh [36], the intersection over union (IoU) between two bounding boxes BB_a and BB_b is defined as described in Equation 2.1. The IoU can take on any value between 0 and 1, where a value of 0 means that there is no overlap between the two bounding boxes, and a value of 1 means that the two bounding boxes are identical. In the context of object detection, IoU is commonly used to evaluate the accuracy of predicted bounding boxes against ground truth bounding boxes [36].

$$IoU(BB_a, BB_b) = \frac{\text{Area of intersection of } BB_a \text{ and } BB_b}{\text{Area of union of } BB_a \text{ and } BB_b} \tag{2.1}$$

2.4. Retrieval-Augmented Generation

While LLMs have extensive general domain knowledge due to their enormous corpora of training data, compiled from various open-domain sources [37], they struggle with tasks that require domain-specific knowledge which they did not encounter during training [6]. This can lead to 'hallucinations' and inaccuracies, as the model tries to synthesize a matching answer based of its domain-wise irrelevant training data [6, 8]. RAG addresses this limitation, extending the usage of LLMs to applications requiring extensive knowledge in a specific domain [5]. This is achieved by retrieving information from an external knowledge source comprised of application-relevant text passages, supplying additional context to the LLM during answer generation [5, 6].

2.4.1. Architecture of RAG Systems

While there are many advanced and extended versions of RAG systems, for this study we will focus on the standard Naive RAG architecture as depicted in Figure 2.1 [6]. Naive RAG is based on the original RAG architecture proposed by Lewis, Perez, Piktus, et al. [5]. Naive RAG systems consist of two modules:

Retriever: The retriever module consists of a query encoder and an external knowledge base [5]. It is responsible for retrieving relevant context from the knowledge base, based on the user's query [6]. The module is based on the bi-encoder architecture, with the query encoder q and document encoder d encoding texts into a shared embedding space [5, 38]. The knowledge base is a vector database consisting of application-specific text passages z. Each passage is stored in the database as a vector embedding $d(z)$, encoded through the document encoder d [5]. To identify the relevant passages for a query x, x is first transformed into a vector embedding $q(x)$ using the retriever's query encoder [6]. Based on the similarity scores

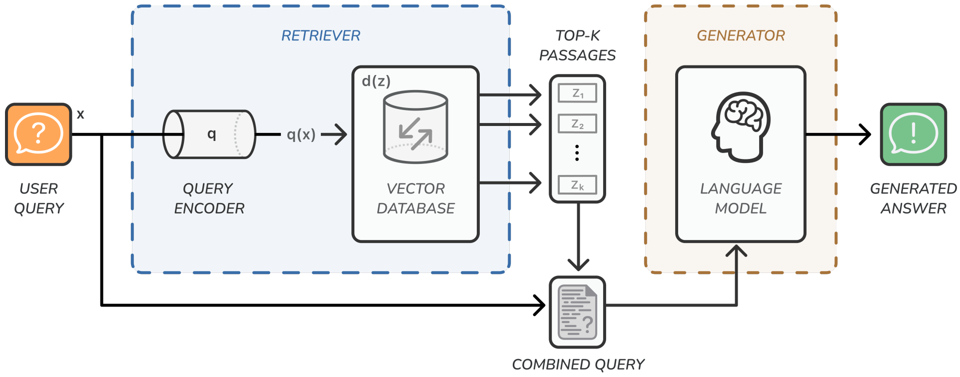

Figure 2.1.: Architecture of the Naive RAG system.

between the query embedding and the stored chunk embeddings, the top-k documents z with the highest similarity scores, are then retrieved from the database [5, 6].

Generator: The generator module is responsible for synthesizing the final answer based on the user's query and the passages retrieved by the retriever [5]. Firstly, the original query x and the retrieved passages z are combined into a single input query [6]. The LLM is then tasked with generating the final answer y, conditioned on this combined input [6, 5].

2.4.2. Indexing

In order to apply the RAG paradigm to knowledge-intensive tasks in a specific domain, the external knowledge base needs to be created from relevant data sources. This process is called Indexing [6]. Indexing begins with the preparation of the data sources into short text passages [6]. For the purpose of this study we will refer to this process as document segmentation. Document segmentation includes both DP, the conversion of unstructured documents, such as PDFs and images, into structured data [12], as well as chunking, the splitting of this data into smaller text passages called chunks [6]. Chunking is a necessary step for RAG systems, as both LLMs and encoders are limited in the number of tokens that fit into their context window [6]. Furthermore, indexing includes the encoding of these chunks into vector embeddings. Both the embeddings and the original chunks are then stored as key-value pairs in a vector database, allowing fast and frequent searches during retrieval [6].

The quality of the index construction has a crucial effect on the resulting RAG system [6]. It determines both the likelihood of retrieving relevant context as well as the quality of the generated answer. Especially chunking, which is often overlooked and seen as solely a technical requirement, has been found to be crucial for enhancing the quality of the knowledge base [6].

2.4.3. Source Attribution

Source Attribution is a mechanism that provides transparency and traceability to the output of the RAG system by linking the generated text to their source documents [11, 39]. This allows the user to verify the LLMs claims by examining the provided sources [39]. Source Attribution can be performed at different granularity levels. Document level source attribution provides citations to the entire documents that the retrieved passages are part of [11, 39]. While this approach enables the necessary verifiability, it introduces additional strain to the user, who has to find the relevant passages in the document [11]. This effect is especially critical for longer documents, such as CPGs. In order to mitigate this issue, recent research has suggested the concept of visual source attribution [11]. Visual source attribution revolves around visual confirmation for the exact location of the retrieved information [11]. This is achieved by highlighting the exact region of the retrieved text inside of the document [11]. The position of the retrieved passage is therefore immediately visible to the user, making source attribution easy and seamless.

2.5. Document Parsing

Also known as document content extraction, DP aims to convert unstructured and semi-structured documents into structured, machine readable data formats [12, 40]. During this process elements such as headings, tables, and figures are extracted from the document while preserving their structural relationships. DP is crucial for many document-related tasks, providing access to previously unavailable information sources. Especially for LLMs, where leveraging additional training data is crucial for enhancing the model's factual accuracy and knowledge grounding, DP plays an important rule [40, 41]. With the emergence of the RAG paradigm, DP has also been critical in the creation of the knowledge database, as important information is often stored inside file formats which can not directly be processed by machines [42]. While DP is used for converting a range of document formats into machine-readable content, we will focus solely on the parsing of PDF documents for the purposes of this thesis, as this is the datatype that the oncology guidelines are stored as.

Converting PDF documents is particularly challenging due to their variable formatting, lack of standardization and focus on visual characteristics [42]. The format not only includes born-digital files but also includes photographed and scanned documents. Therefore, DP systems need to be able to adapt to a wide range of different layouts, image qualities and document types, such as academic papers, invoices, or presentation slides [40, 43]. While there are many tools and implementations available for DP [42, 41, 44, 45], most of them can be categorized into either modular pipeline systems or end-to-end VLM models.

2.5.1. Modular Pipeline Systems

Modular pipeline systems employ various different modules in a sequential order to perform DP. This modular design enables the targeted optimization of individual components and flexible integration of new modules and techniques [46]. Additionally, by making use of lightweight models and integrating parallelization, pipeline systems can reach efficient parsing speeds [40]. While different formations are possible, most implementations consist of three different stages [12].

Document Layout Analysis (DLA): According to Q. Zhang, B. Wang, V. S.-J. Huang, et al. [12], DLA refers to the identification of the structural elements of a document, such as paragraphs, section headers, tables, figures, and mathematical equations, as well as their respective bounding boxes [12, 47]. There are two types of methods for performing DLA. Uni-modal methods focus purely on visual features of the document in order to identify structural elements [12, 48]. Notably, convolutional neural networks (CNN)- and transformer-based methods adapt models initially designed for object detection tasks, such as the YOLO [49] and DETR [50] families of models, to accurately identify structural elements in document images [12, 47]. Hereby, transformer-based methods excel at capturing global relationships between structural elements at the cost of computational intensity and expensive pre-training [12]. The second type of DLA methods are multi-modal methods. Additionally to the visual representations, multi-modal methods also make use of the content and position of the pages' textual elements, performing DLA using a VLM [48, 51]. This approach allows more granular classifications and the analysis of highly complex layouts [12, 48].

Content Extraction: To extract the content of the identified structural elements different recognizers are applied to the element regions based on their classifications [12, 41, 42]. For textual elements, such as paragraphs or section headings, the textual content is identified using optical character recognition (OCR). OCR engines use techniques from computer vision in order to identify and extract text from images [12, 52]. Popular OCR engines include EasyOCR [53] and the Tesseract OCR engine [54]. Additionally to extracting content using OCR, DP implementation often provide specific recognizers for additional element types [12, 41]. Most commonly this includes a specific model for table structure recognition, referring to the extraction of table content into structured file formats, such as HTML, XML or Markdown [12, 42, 41, 55]. Other options for class-specific recognizers include mathematical formula recognition and chart recognition [41, 44, 12].

Relation Integration: During relation integration the identified elements are combined into the final output format. During this stage, rule-based methods and specialized AI models may be employed, for example to filter out duplicate or unwanted elements or correct the reading order of the document [12, 41, 42]. Depending on the chosen output format, this process might lead to the loss of information, such as the loss of bounding box information for an output in Markdown format [16].

Systems following the modular pipeline approach also have some inherent drawbacks. Mainly, due to handling the parsing of each structural element independently of each other, pipeline systems fail to capture information about the global context of the document, leading to semantic loss [43]. Additionally, because of the sequential nature of the pipeline approach, errors from different stages propagate through the pipeline [43, 44].

2.5.2. End-to-End VLM models

Due to recent recent advancements in VLM architectures, end-to-end VLM models have emerged as a promising alternative to traditional pipeline-based approaches. Research such as General OCR Theory (GOT) have demonstrated the ability of VLMs to perform high accuracy OCR while being able to extract the content of tables, charts or mathematical formulas using a singular model [56]. Contrary to pipeline-based methods, VLM-based approaches are able to generate structured outputs directly from the input document, addressing the error propagation problem of modular pipelines [46]. Additionally, these models demonstrate advantages in understanding the structure and hierarchy of complex documents [12]. VLM-based approaches can be divided into two further subcategories:

General-Purpose VLMs: General purpose VLMs are not trained solely for document-centric tasks, but are still able to show promising results for DP, due to their large parameter count and extensive training data [44, 33]. However, these models are often either proprietary or require extensive computational resources [44]. Additionally, they often struggle with documents that follow more complex layouts or contain densely packed text blocks [44].

Domain-Specific VLMs Domain-specific VLMs are trained and optimized specifically for DP [44, 12, 43]. In recent years, there has been promising developments towards domain-specific VLMs that encapsulate DLA, content extraction and relation integration into a single model [57]. These models are able to achieve state-of-the-art performance on document parsing benchmarks, while being a fraction of the size of general-purpose VLMs [57, 44]. As VLMs are not bound to the stages of traditional pipeline systems, there has also been additional research regarding models optimized for the direct generation of content-only outputs, most notably Markdown [43]. However, this approach inherently leads to the loss of information, such as positional information for the extracted elements, which is not included in the lossy Markdown output, making this class of models unsuitable for the purposes of this research [57, 43].

Recently, there has also been research towards multi-stage VLM-based approaches [44, 46]. These models use one or more VLMs in multiple stages, aiming to encapsulate the computational efficiency of pipeline approaches with the improved accuracy and structure understanding of VLM-based methods [44]. However, especially when multiple VLMs are in use, these approaches come with a further increase in complexity and computational requirements and may show decreased performance in tasks such as reading order inference compared to single-stage VLM-based approaches [44, 57]. Current challenges regarding the development of VLM-based approaches are the risks of 'hallucinations', especially on longer documents [44, 16], as well as their high computational requirements compared to modular pipeline systems [16].

2.6. Chunking

Chunking refers to the splitting of documents into small atomic units of information called chunks [58, 6]. While the term is directly linked to the recent emergence of the RAG paradigm, the underlying task of text division is fundamentally aligned to the established concept of passages in passage-based document retrieval [59, 60].

Despite the rapid adoption of RAG, the chunking process lacks a robust scientific taxonomy. Much of the terminology associated with modern chunking strategies originates from non-scientific sources, such as technical blogs, software documentation, and community tutorials. We find that the established taxonomy of passage-based document retrieval aligns with the types of modern chunking strategies. To ensure scientific stability, we therefore adopt the terminology proposed by Callan [60]. Specifically, Callan [60] categorizes passages into three distinct types: window passages, semantic passages, and discourse passages.

2.6.1. Window Passages

Window passages are determined by splitting the content of the document into parts of a fixed length. While in passage-based retrieval, length typically referred to the number of words in a passage [60], with the advent of chunking the focus shifted towards measuring the number of tokens [58]. Modern chunking strategies have further extended this method through sliding-window approaches, which introduce a fixed overlap between neighboring chunks to preserve contextual continuity [61]. These strategies provide a simple and computationally efficient way to perform chunking [13]. However, they disregard the content of the document, which may result in chunk borders appearing inside a single word or sentence [62].

2.6.2. Semantic Passages

Semantic passages aim to enhance retrieval quality by aligning passage borders to identified subtopics of the document [62]. However, they introduce significant additional computational complexity and may vary drastically in length [13]. Strategies from this category stem from the field of Text segmentation, referring to "the task of dividing text into segments, such that each segment is topically coherent, and cutoff points indicate a change in topic" (p.1) [63]. In recent years there have also been novel strategies proposed for this task that leverage LLMs to determine semantically independent chunks [64].

2.6.3. Discourse Passages

Discourse passages are defined by the inherent structure of the document, such as sections, sentences, and paragraphs. Typically, these strategies recursively divide the document with increasing granularity until resulting chunks satisfy a specified maximum length constraint [17]. While the documents in passage-based document retrieval are simple unstructured text streams [60], modern chunking techniques often process data in structured formats such as JavaScript Object Notation (JSON) or extensible markup language (XML) [16], especially when combined with DP. Recently, specialized strategies have emerged that leverage additional metadata from these formats, such as hierarchical relationships between elements, to produce chunks that follow the structure of the document more closely [16].

2.6.4. Metadata Attachments

Additional to their textual content, chunks can also be enriched with additional metadata information [6]. This metadata can include information about the original document, such as its author, title, and publishing date. This allows the filtering of retrievable data based on document attributes, such as limiting the retrieval to documents published in a specific time frame [6]. Metadata attachments are also useful for providing source attribution. While document information provides source attribution at the document level, additional metadata such as the page number and bounding box of the chunk provides more granular grounding.

3. Methodology

3.1. Pipeline Overview

In order to be able to compare different DP implementations and chunking strategies against each other, we developed a modular document segmentation pipeline. The pipeline's architecture, illustrated in Figure 3.1, follows a two-stage process. Firstly, raw PDF documents are transformed into a structured data format. Subsequently, this data is then partitioned into metadata enriched chunks. The core principle of the pipeline's design lies in its modularity, allowing the seamless interchange of both the used DP implementation and the chunking strategy, while maintaining a unified interface for both modules.

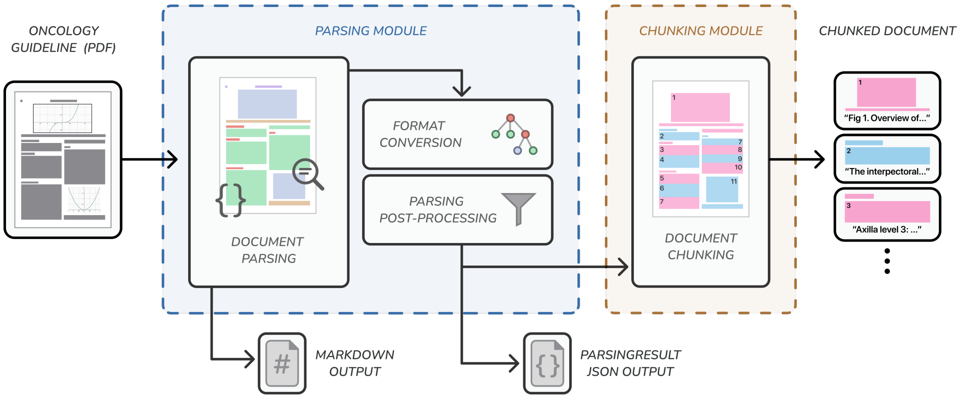

Figure 3.1.: Architecture of the document segmentation pipeline.

Parsing module: The parsing module is the first step of the pipeline. It integrates eight different DP implementations into a unified interface, normalizing their output formats into a standardized data format. Firstly, document elements and their structural information are extracted from the document using the underlying DP implementation. The normalized output then undergoes further post-processing steps, such as the filtering of unwanted element types. Finally, the result of the parsing operation is persisted to the file system as both a lossless JSON serialization and a lossy Markdown serialization for further processing and evaluating.

Chunking module: The chunking module is responsible for the splitting of the structured data into smaller chunks using one of multiple available chunking strategies. We adapt four established strategies to operate on the data format provided by the parsing module. Additionally, we propose a novel approach to the chunking paradigm, enabling traceability of the chunk's content on the token level. This allows for the determination of more accurate chunk bounding boxes, enabling high granularity, visual source attribution for downstream RAG applications.

3.2. Data Representations

A significant challenge for the evaluation and comparison of different DP implementations is the lack of standardization. As of the time of writing, every DP implementation defines their own data types, making direct comparisons very complex. In order to consolidate multiple different DP implementations into our modular pipeline and evaluate them against each other, we define our own universal data types to be used in the data processing pipeline. Before further processing, the outputs of each DP implementation is first transformed into these data types.

3.2.1. ParsingBoundingBox

ParsingBoundingBox (Figure 3.2) serves as the fundamental datatype to denote the location of an entity in the document. It expands upon a bounding box in LTRB format through the addition of a page number to support multi-page documents. The coordinates are stored as normalized fractional values of the page dimensions. Additionally, the data type includes the recursive attribute `spans`, which enables the assignment of bounding boxes of higher granularity, such as individual text lines.

```python
class ParsingBoundingBox:
  page: int
  left: float
  top: float
  right: float
  bottom: float
  spans: list[ParsingBoundingBox]
```

Figure 3.2.: Python implementation of the ParsingBoundingBox datatype.

3.2.2. ParsingResultType

A crucial problem with comparing multiple DP methods is their lack of a universal terminology for recognized element types. For example, a paragraph gets classified as `NarrativeText` by the Unstructured.io framework [45], while Docling [42] names the same category as simply `TEXT`. Additionally, some methods provide classifications, which are not provided by others. One example for this is the addition of a `ref_text` element type in the MinerU implementation [41, 65], referring to an entry in a bibliography. To address these issues, we aggregate all element categories from the evaluated DP implementations into a single collection, normalizing their categories into a unified terminology. We then provide mappings for each of the implementations to our universal categories. The full list of available ParsingResultTypes is provided in Table A.

3.2.3. ParsingResult

Inspired by the data structures of multiple DP implementations, such as Docling [16] and Google Document AI LayoutParser [66], we choose a tree structure to represent the output of the parsing module. This approach has the benefit of being able to model the structure and hierarchies of the structural elements through parent-child relationships, ensuring a lossless representation of the original document. The ParsingResult datatype (Figure 3.3) represents a node inside of this tree structure.

```python
class ParsingResult:
    id: str
    type: ParsingResultType
    content: str
    geom: list[ParsingBoundingBox]
    parent: ParsingResult | None
    children: list[ParsingResult]
    metadata: dict
    image: str
```

Figure 3.3.: Python implementation of the ParsingResult datatype.

As such, the ParsingResult contains all attributes identified during DP. Its classification and bounding box, which are identified through DLA, are stored in the `type` and `geom` fields respectively. The latter also allows multiple bounding boxes for a single structural element, to allow for more flexibility regarding the localizations returned by the DP implementation. The element's content, identified during content extraction, is stored in the `content` field. Some implementations also persist images of figures or tables to the file system during content extraction, with `image` containing their respective paths. `id` contains a document-wide unique identifier for the ParsingResult node. Lastly, `parent` and `children` model the tree structure.

The root node of the ParsingResult tree structure contains additional metadata about the parsing process, such as the elapsed parsing time, the used DP implementation, or the path to the parsed PDF document, in the `metadata` field. Traversing the tree from the root node in a depth-first manner, iterates through the elements in reading order.

3.2.4. ChunkingResult

The ChunkingResult (3.4a) is the final output of the document segmentation pipeline. It provides a wrapper around the list of generated chunks, adding a `metadata` field for information about the document and the document segmentation process. Hereby, the ChunkingResult incorporates both information about the chunking process, such as the chosen chunking strategy, as well as the metadata from the root node of the preceding ParsingResult tree.

```python
class ChunkingResult:
    chunks: list[Chunk]
    metadata: dict
```

(a)

```python
class Chunk:
    id: str
    content: str
    metadata: dict
    geom: list[ParsingBoundingBox]
```

(b)

Figure 3.4.: Python implementation of the ChunkingResult (a) and Chunk (b) data types.

3.2.5. Chunk

The Chunk (3.4b) represents a singular passage used for the creation of the knowledge base in downstream RAG applications. In addition to the textual content of the chunk, which is stored in `content`, the datatype contains the ParsingBoundingBoxes required for visual source attribution. Lastly, the metadata field contains additional information about the chunk, such as its token length.

3.3. Parsing Module

[(TODO: rewrite)] The parsing module processes an incoming PDF document in a multi-step process. Firstly, the document is passed to the internal DP implementation using the `_parse` function. Through the `_transform` function, the raw output is then converted into the ParsingResult format. Additionally, the `_get_md` function extracts the Markdown representation of the document from the raw output of the internal parser. Finally, additional metadata about the parsing process, such as the duration of the internal parsing operation, are added to the root ParsingResult. After performing additional post-processing steps, both the Markdown as well as the lossless serialization of the ParsingResult in JSON output are then persisted to the file system. As the raw types returned by the internal parser differ based on the specific implementation, each of the abstract functions mentioned above need to be implemented by the specific parser class. An overview of the abstract functions to be implemented can be found in Table 3.3.

3.3.1. Unstructured.io

Unstructured.io is a prominent provider for DP, offering both a cloud-based API as well as an open-source library. For our study, we will focus on the open-source library version of Unstructured.io [45, 55]. While the developers themselves explicitly highlight that the open-source library is not suited for large-scale production environments [55], its inclusion within

Function	Input	Output
`_parse`	PDF document	Custom data format
`_transform`	Custom data format	ParsingResult
`_get_md`	Custom data format	Markdown string

Table 3.1.: Abstract functions of the Parsing Module to be fulfilled by the implementation.

the documentation of popular RAG frameworks, such as Langchain [14] and LlamaIndex [15] make it a popular choice for a first point of contact with DP. Therefore, we will regard the open-source library as a baseline for the compared implementations. Unstructured.io follows a modular pipeline approach. Specifically, the implementation uses YOLOX, a uni-modal vision transformer, to perform DLA [55, 67]. The library also includes a specialized model for table structure recognition [55].

3.3.2. Docling

Docling, which was developed by IBM in 2022, is a one of the most popular available open-source DP libraries [42, 16]. Docling particularly stands out from other DP implementations through its permissive MIT license. To achieve this, Docling relies primarily on custom models instead of using third-party software, which are often not as permissive [42]. Docling offers two different approaches for DP:

Parsing pipeline: Docling's processing processing pipeline consists of three components: a PDF backend called DoclingParse, an internal model pipeline containing multiple AI models, and a post-processing stage [16, 42]. Firstly, the PDF backend extracts useful information from the document using both contained programmatic information as well as OCR techniques. This includes bounding boxes for every text element inside the document. The internal model pipeline then performs both the DLA as well as content extraction steps. Hereby, Docling provides their own models for table structure recognition with the TableFormer model [68] as well as for DLA with their Heron model [47]. Heron is derived from RT-DETR [69], a uni-modal vision transformer, and retrained on DocLayNet [70], Docling's own dataset for DLA. During DLA, identified bounding boxes are compared and intersected with bounding boxes retrieved from the PDF backend in order to provide more accurate localization [16]. Using TableFormer, Docling is the only evaluated open-source system, that provides individual content and bounding boxes for table cells. During post-processing the recognized elements are then combined into the DoclingDocument datatype [16].

Granite Docling: Granite Docling is an end-to-end VLM for DP. It belongs to the group of domain-specific VLM models, specifically build for document understanding and conversion [71]. The model is very compact, consisting of around 258 million parameters [72, 71]. With this model, Docling proposes the DocTags data format, a structured format designed for representing both text and structure of the document through XML-style tags [72].

3.3.3. MinerU

Another popular choice for open-source on-device DP is the MinerU framework. Similar to Docling, MinerU also offers both a pipeline as well as a VLM-based approach for DP [41, 44, 65].

Parsing pipeline: MinerU extends the traditional processing pipeline through a pre- and postprocessing stage. In the preprocessing stage unprocessable files are filtered out and metadata about the document is extracted using the PyMuPDF library [41, 73]. This metadata includes the language of the document, the document's page dimensions and the identification of scanned documents [41]. The pipeline then uses models from the DP model library PDF-Extract-Kit for DLA and content extraction [41, 65, 74, 75]. For content extraction, special models for formula and table recognition are employed by the pipeline. The model used for DLA is a fine-tuned version of LayoutLMv3, a multi-modal model [41, 51]. During the final postprocessing stage, overlapping elements are cleaned up, unneeded elements are filtered out and the reading order of the document elements is inferred using a segmentation algorithm [41].

VLM: With MinerU2.5, the implementation's offerings were expanded by a multi-stage VLM-based DP approach. This approach employs a 1.2 billion parameter VLM to perform DP in a two-stage approach [44]. Firstly, the model is used to perform DLA on the document, identifying elements and their reading order. In the second stage, the same model is applied again on individual image crops of the page element and is tasked to extract the content from the crop [44].

3.3.4. Gemini 2.5 Flash

Gemini 2.5 Flash is a closed-source proprietary model developed by Google with strong multi-modal capabilities across text, vision and audio [76]. While Google offers a more capable model in the form of Gemini 2.5 Pro, we follow the sentiment from Niu, Z. Liu, Gu, et al. [44], that DP tasks "typically exhibit relatively low dependency on large-scale language models" (p.7) and both models similar results on various image understanding benchmarks [76], instead opting to rely on the cheaper, faster Gemini 2.5 Flash model for our study. Gemini 2.5 Flash belongs to the group of general-purpose VLMs and, due to its closed-source nature, is only accessible through an application programming interface (API). The Gemini family of models received additional training in order to provide improved accuracy on object detection and image segmentation tasks [76, 77]. We follow the documentation provided by Google on harnessing Gemini's image understanding capabilities [77] to formulate a prompt, that takes advantage of this additional training for the DP task. The full prompt is available in Listing A.1.

3.3.5. LlamaParse

LlamaParse is a cloud-based paid DP service from the makers of LlamaIndex, a popular framework for building RAG systems and workflows [15, 78]. While there is no official information on the architecture used for the DP system behind LlamaParse, its marketing as a "GenAI-native document parser" [78] as well as the option to provide custom prompts to the service suggests that at least some of its functionality stems from a VLM.

3.3.6. Google Document AI LayoutParser

LayoutParser from Google Document AI is another cloud-based paid provider of DP services [66]. Contrary to other services such as Google Document AI's Enterprise Document OCR [79], LayoutParser has a strong focus on identifying the relationships between different page elements. As such, LayoutParser can recognize the level of section headers, infer the hierarchy between different elements and extract the content from individual table cells. LayoutParser follows a multi-stage pipeline approach to perform DP, but, as LayoutParser is a proprietary system, its exact architecture is unknown.

3.4. Post-Processing Methods

During post-processing various rule-based methods are applied to the ParsingResult. The post-processing stage employs three steps:

Element Filtering: Most documents contain textual information that does not belong to the main content of the document, such as page numbers and repeating page headers or footers. Some DP implementations, such as MinerU [41], already remove these elements in their own post-processing stages. In order to facilitate a fair comparison and remove unneeded information before the chunking stage, we remove any element that belongs to non-main content element types and textual elements with empty content. Specifically we remove all elements from the following types: `[REFERENCE_LIST, REFERENCE_ITEM, PAGE_FOOTER, PAGE_HEADER, FORM_AREA, WATERMARK]`.

Hierarchy Inference: In order to represent the document's hierarchy as a tree structure, the relationships between the different elements need to be established. However, many of the evaluated DP implementations do not return their output in a tree structure directly and instead provide a list of document elements in reading order. Other implementations, such as Docling [42], contain some hierarchy such as the relationships between tables and their constituent table cells while missing the relationships between section headings and the content belonging to their section. In order to

3.5. Chunking Module

The chunking module provides an interface to perform chunking on the ParsingResult tree representation of the PDF document. We propose a novel solution aimed at increasing the traceability of the chunk content to its constituent ParsingResults. Prominent implementations of chunking strategies, such as the ones found in the RAG frameworks LlamaIndex [15] and Langchain [14], treat the chunking process as a transformation of a single string, typically the Markdown representation of the document, into multiple smaller strings. Following this approach, resulting chunks lose their direct connection to the underlying structural elements, complicating source attribution.

Domain-specific implementations, such as Docling's hybrid and hierarchical chunkers [42, 16], improve upon this by including a list of elements associated with the resulting chunk. Additionally, they take advantage of the document's hierarchy by including relevant section headers in the chunk text. However, these implementations do not distinguish between partially and fully included elements. This results in chunk bounding boxes which always contain the entire element, regardless of how much of its text is actually included in the text of the chunk. Furthermore, while the contents of the section headers are included in the chunk, their corresponding bounding boxes are not retained.

Our solution addresses these limitations by enabling traceability at the highest relevant granularity for RAG chunking. While in traditional text segmentation, characters are the smallest instance that a text can be broken up at, the same can not be said about chunking for AI models. AI models are not directly constrained by the character length of their input and are instead limited by the amount of tokens that can fit into their context window. Therefore, we propose that the smallest instance that a text needs to be broken up into in the context of RAG is a token. To enable the tracing of every token inside the chunk to a token inside an element, we introduce the RichToken. A RichToken (Figure 3.5) is an object that wraps around a single token and its original text, the id of the element that the token belongs to, and its corresponding index inside that element. When the module performs the chunking operation on a ParsingResult, the specific chunking implementation processes the tree and returns the content of each chunk as a list of RichTokens. The module then creates a Chunk object using the RichTokens. By aggregating the content of the RichToken, the module can reconstruct the content of the entire Chunk, while maintaining the relationships to the constructing ParsingResults. Since the module is aware of the specific tokens that are included in the Chunk, we can determine tighter chunk bounding boxes using the span-level bounding boxes added to the elements in the DP post-processing.

```python
class RichToken:
  element_id: str
  token_idx: int
  token: int
  text: str
```

Figure 3.5.: Python implementation of the RichToken datatype.

How the RichTokens are grouped together into chunks is to be decided by the specific chunking strategy. The module provides a `_tokenize` function, which transforms a given ParsingResult node into RichTokens using the `all-MiniLM-L6-v2` sentence transformer as a tokenizer. To prevent distinct document elements from merging into a single text block during chunk creation, `\n` delimiters are appended to the content of the ParsingResult nodes before the chunking process begins. These delimiters act as textual representations of the breakpoints between document elements. Additionally, to avoid creating chunks, which are too big for the embedding module's context window, a limit N for the maximum amount of tokens per chunk can be set on the chunking module. We provide implementations for four different chunking strategies.

3.5.1. Fixed-Size Chunking

Fixed-size chunking segments the document into chunks of the chunking module's maximum chunk length N with an overlap of O tokens. It provides a simple and computationally efficient way to perform chunking, while producing chunks of a constant size [13]. However, fixed-size chunking has no regard for the content of the document, which may result in chunk borders inside a single word or sentence [62].

The strategy traverses the ParsingResult tree in reading order (e.g., depth-first), creating a queue of the document's RichTokens in the progress. When the queue reaches a length larger then N, the first N tokens inside the queue are combined into a chunk while $N - O$ tokens are removed from the queue. This sliding window approach, leaving O tokens inside the queue after the chunk is created, aims to maintain some contextual continuity between the chunks, leading to improved recall during the retrieval phase [13, 62].

3.5.2. Recursive Character Chunking

Recursive character chunking encapsulates a similar approach to fixed-size chunking. However, instead of naively setting the border of a chunk at a fixed token count, recursive character chunking utilizes an hierarchical list of delimiters (e.g., paragraphs, sentences, words) to define chunk boundaries [62, 17, 14]. When the RichToken queue exceeds the maximum chunk length N, the strategy splits the tokens using the highest priority delimiter. If there still exists a split which is larger than N, the process recurses on the oversized split with the next delimiter in the list. This 'coarse-to-fine' approach preserves logical groupings while avoiding unnecessary fragmentation [62]. Similar to fixed-size chunking, recursive character chunking also incorporates sliding window chunking with an overlap of O tokens between adjacent chunks. The delimiters used in this implementation are adapted from LangChain's `RecursiveCharacterTextSplitter` [14, 17], with punctuation added to better identify sentence endings, as suggested by B. Smith and Troynikov [58]. This results in the following delimiters: `["\n\n", "\n", ".", "!", "?", "␣", ""]`.

3.5.3. Breakpoint-based Semantic Chunking

Breakpoint-based semantic chunking separates the document at the sentence level, inserting breakpoints in between sentences to denote chunk borders [13]. Instead of relying on

For this strategy, the semantic distances between the embeddings of every adjacent sentence pair are calculated, with a high distance indicating a topical shift between the sentences. If the distance is larger than the Q-th percentile of all distances, a breakpoint is inserted.

3.5.4. Hierarchical Chunking

3.6. Evaluation Framework

3.6.1. Document Layout Analysis Evaluation

The goal of the DLA evaluation is to assess the correctness of the bounding boxes and type labels produced by the parsing module [80]. While there are multiple datasets available for this task [81, 70, 40], we will use the PubLayNet dataset [82] for our evaluation. While many datasets focus on evaluating DLA on a range of different document types such as forms, invoices or handwritten documents, PubLayNet consists solely of medical scientific articles [40, 82]. This format closely resembles the format of the oncology guideline documents which makes it a suitable choice for this evaluation. Compromised of over 360.000 automatically annotated document pages collected from PubMed Central Open Access (PMCOA), PubLayNet is one of the largest datasets for DLA. As the dataset in its entirety is no longer publicly available and far too large for the purposes of this thesis, we will use `publaynet-mini`, a small subset of 500 pages of the original dataset for this evaluation [83]. As seen in Table 3.2, the subset contains around 5000 ground truth annotations for elements from 5 different classes.

Category	**Annotations**
Text	3,676
Title	1,000
List	73
Table	128
Figure	172
Total	**5,049**

Table 3.2.: Distribution of ground truth annotations across the different element types contained in the `publaynet-mini` subset of the PubLayNet dataset.

To assess the performance of different predictors on object detection tasks such as DLA, average precision (AP) is the most commonly used metric [84]. Previous evaluations of DLA models on the PubLayNet dataset also use a version of this metric [85]. However, AP relies on the predictor's confidence values, indicating how confident the predictor is about a predicted bounding box and class label. As most of the DP implementations provide 'hard-predictions', which do not contain any confidence values, the AP is not a viable metric for the purposes of this thesis [86, 87].

For this reason, we will compare the implementations based on their achieved F1 score. Similarly to AP, this metric takes into account two important measures for object detectors: precision and recall [88]. According to Padilla, Passos, Dias, et al. [88], "Precision is the ability of a model to identify only relevant objects. [...] Recall is the ability of a model to find all relevant cases [...]" (p. 9). In order to calculate their values, firstly the detected bounding boxes (DTBBs) are classified into true positives (TPs) and false positives (FPs). A DTBB is classified as a TP if there exists a ground truth bounding box (GTBB) from the same class, so that their IoU is greater than a given threshold. One GTBB can not be matched to multiple DTBBs. If there does not exist a GTBB that fulfils these criterions, the DTBB is classified as a FP. Any GTBBs which were not matched to a DTBB are classified as false negatives (FNs). Following the definition from Padilla, Passos, Dias, et al. [88] for a model that, on a dataset with G GTBBs, outputs N DTBBs, out of which $S, (S \leq N)$ are TPs, precision and recall can be formulated as shown in Equation 3.1 and Equation 3.2.

$$\mathrm{Pr} = \frac{\sum_{n=1}^{S} \mathrm{TP}_n}{\sum_{n=1}^{S} \mathrm{TP}_n + \sum_{n=1}^{N-S} \mathrm{FP}_n} = \frac{\sum_{n=1}^{S} \mathrm{TP}_n}{\mathit{all\ detections}} \tag{3.1}$$

$$\mathrm{Re} = \frac{\sum_{n=1}^{S} \mathrm{TP}_n}{\sum_{n=1}^{S} \mathrm{TP}_n + \sum_{n=1}^{G-S} \mathrm{FN}_n} = \frac{\sum_{n=1}^{S} \mathrm{TP}_n}{\mathit{all\ ground\ truths}} \tag{3.2}$$

The F1 score is the weighted harmonic mean between precision and recall and is calculated as defined in Equation 3.3 [88]. The F1 score is calculated for a single class at a set IoU threshold. Selecting a higher threshold will lead to a stricter metric as predictions need to be more precise to be counted as a TP [88]. A F1 score calculated at an IoU threshold T% is commonly referred to as F1@T [89].

$$F_1 = 2\frac{\mathrm{Pr} \cdot \mathrm{Rc}}{\mathrm{Pr} + \mathrm{Rc}} \tag{3.3}$$

For scenarios with multiple classes, such as the PubLayNet dataset, the Macro F1 score can be used to assess the overall performance of the predictor [90]. The Macro F1 score is the mean of the single class F1 scores. For a dataset with M different classes, the calculation of the Macro F1 score is described in Equation 3.4. Hereby, $N_{:j}$ and $G_{:j}$ denote the DTBBs and GTBBs belonging to elements of class j [90].

$$F_{1_{\mathrm{Macro}}}(N, G) = \frac{1}{M} \sum_{j=1}^{M} F_1(N_{:j}, G_{:j}) \tag{3.4}$$

The DP implementations will be evaluated on both their single-class and Macro F1 scores. Specifically, their (Macro) F1@50 and F1@50:95 will be compared against each other. F1@50:95 refers to the mean of the F1 values calculated at 10 evenly spaced IoU thresholds between 0.5 and 1.0 and is inspired by the primary challenge metric found in the MS COCO dataset [91]. This rewards implementations, which provide more accurate bounding boxes [91]. F1@50 is chosen, as a threshold of 50% is one of the most commonly used threshold values for metrics in object detection [88]. To calculate these metrics, the `faster-coco-eval` package is used to determine the recall and precision values at the IoU thresholds [92].

3.6.2. Content Parsing Evaluation

OmniDocBench Content:

- Data Creation

- Evaluation Modes (End2End *to* Evaluate Markdown output (Content))
- Usages in other papers
- State of the art evaluations (if I find any, most for all kinds of documents)
- Types of Documents (English and Chinese, different kinds: Scientific Paper. . .) all single page
- Subset for the thesis: English Scientific Papers

3.6.3. Chunking Evaluation

Chroma Evaluation

4. Results

	text	title	list	table	figure	all
unstructured_io	0.8123	0.8032	0.1269	0.9337	0.6138	0.6216
docling	0.8687	0.8775	**0.8022**	0.9530	0.5951	**0.8063**
docling_granite	<u>0.6737</u>	<u>0.6309</u>	0.6329	0.9001	0.1811	0.5296
mineru_pipeline	0.8735	0.9558	0.4771	0.9784	0.6534	0.6528
mineru_vlm	**0.9119**	0.8822	0.5702	0.9796	0.2508	0.5941
llamaparse	0.7711	0.6370	<u>0.0000</u>	<u>0.6831</u>	<u>0.0000</u>	<u>0.4110</u>
document_ai	0.7830	**0.9789**	<u>0.0000</u>	**0.9911**	**0.9848**	0.7393
gemini	0.8242	0.7619	0.1271	0.8725	0.6530	0.6153

(a) F1@50

	text	title	list	table	figure	all
unstructured_io	0.7583	0.6029	0.0682	0.8707	0.4987	0.5095
docling	**0.8206**	0.6311	**0.7406**	0.9143	0.4952	**0.6650**
docling_granite	<u>0.6241</u>	0.4302	0.5694	0.8497	0.1608	0.4382
mineru_pipeline	0.8097	0.6170	0.4331	0.9407	0.5436	0.5342
mineru_vlm	0.8032	0.4461	0.4906	0.9409	0.2034	0.4338
llamaparse	0.7240	<u>0.3057</u>	<u>0.0000</u>	<u>0.6594</u>	<u>0.0000</u>	<u>0.3073</u>
document_ai	0.7136	**0.6595**	<u>0.0000</u>	**0.9669**	**0.9524**	0.6015
gemini	0.7347	0.5002	0.0760	0.7636	0.5822	0.4890

(b) F1@50:95

Figure 4.1.: F1 scores of the evaluated DP implementations on the PubLayNet dataset. (a) contains the F1@50 scores, (b) contains the F1@50:95 scores. Scores are reported per element type. The column `all` reports the respective Macro F1 score for each implementation. Highest values are bolded and smallest values are underlined. Higher values are preferred.

Method	Parsing mean	Parsing std	Transformation mean	Transformation std
docling	2.1146	1.3147	0.0017	0.0024
docling_granite	16.7379	63.5661	0.0017	0.0034
document_ai	**1.6445**	**0.4928**	1.6354	0.2689
gemini	12.5033	11.1618	0.0013	0.0025
llamaparse	37.0956	41.6822	0.7354	0.1782
mineru_pipeline	15.4215	20.2052	0.0020	0.0059
mineru_vlm	42.0182	35.4030	0.0017	0.0012
unstructured_io	4.9745	5.5285	**0.0008**	**0.0005**

Table 4.1.: Mean and standard deviation of the DP implementation's parsing and transformation times per page. Times are reported in seconds. Fastest times are bolded and slowest times are underlined. Lower values are preferred.

	text_block_Edit_dist	display_formula_CDM	table_TEDS	table_TEDS_structure_only	reading_order_Edit_dist	overall
unstructured_io	0.0850	**0.0000**	0.0000	0.0000	**0.1160**	30.5000
docling	0.0780	**0.0000**	66.3290	85.2590	0.0790	52.8430
docling_granite	**0.1360**	**0.0000**	64.2110	70.0420	0.0890	50.2037
mineru_pipeline	0.0440	**0.0000**	80.2160	90.1320	0.0390	58.6053
mineru_vlm	0.0250	**0.0000**	**86.0940**	**92.8200**	0.0060	**61.1980**
gemini	0.0450	**0.0000**	65.3680	72.6080	0.0410	53.6227
document_ai	0.0480	**0.0000**	0.0000	0.0000	0.0380	31.7333

Table 4.2.: Results of the DP implementation on the OmniDocBench benchmark.

5. Discussion

6. Conclusion

- How to improve tables
- How to include figures
- Make bounding boxes more granular *to* token level instead of line level

A. General Addenda

Listing A.1: Prompt to apply the Gemini 2.5 Flash model to DP tasks. Gemini models are trained to output coordinates from 0 to 1000, with the origin at the left-top corner of the image. Additionally, they are trained to provide bounding boxes as tuples in the (y_0, x_0, y_1, x_1) format. In order to maximize the accuracy of detected bounding boxes, `box_2d`, the key used in Google's official documentation, is used to denote the bounding box tuples in the output JSON.

```
<system_role>
You are an expert Document Layout Analysis AI. Your goal is to perfectly transcribe
    and segment PDF documents into structured data.
</system_role>

<task_description>
Analyze the provided document image. Identify every layout element, its bounding box,
     its category, and its textual content.
</task_description>

<categories>
Classify each element into exactly one of these categories:
section_header, text, formula, list_item, ref_item, table, image, caption,
    page_header, page_footer, watermark

Rules for Categorization:
- Use "section_header" for titles and headings. Infer hierarchy based on content and
    font size/boldness.
- Use "image" for charts, diagrams, or photos.
- Use "unknown" if the element is ambiguous.
</categories>

<bounding_boxes>
1. **Format:** [y0, x0, y1, x1] (Top-Left to Bottom-Right). You MUST provide the
    coordinates in this exact order.
2. **Success conditions:**
- The bounding box MUST enclose the entire layout element while minimizing
    unnecessary white space.
- If a character belongs to the content ALL of its pixels MUST BE CONTAINED inside
    the bounding box.
3. **Page Index:** The current page is "page_number": {*}.
</bounding_boxes>
```

```
<extraction_rules>
- **Text Fidelity:** Extract text EXACTLY as it appears. Do NOT fix spelling or
    grammar. You MAY use any formatting that is available for a standard Markdown
    document.
- **Character Escaping:** You MUST escape any special characters that can break the
    final JSON output. Also you must escape any quotation marks.
- **Reading Order:** Sort elements by natural human reading order.
- **Special Formatting:**
    - image: Content must be an empty string "".
    - formula: Content must be LaTeX.
    - table: Content must be a Markdown table representation. TABLE CONTENT MUST NOT
        BREAK THE JSON FORMAT!
    - list_item, ref_item: Content MUST be a valid Markdown list. You MUST replace
        alternative bullet point symbols with "-". Ordered lists must start with their
         numbering followed by ".".
    - section_header: You MUST NOT use Markdown header formatting. You MUST add a "
        heading_level" field (int). Infer the level by checking the content for any
        numbering and analyzing the font size and styling of the header.
</extraction_rules>

<output_schema>
Do not return any additional text with the result.
Return a SINGLE JSON object with this exact structure:
{
  "layout_elements": [
    {
      "category": "string␣(from␣list)",
      "heading_level": integer (include only for headers),
      "content": "string",
      "bbox": {
        "page_number": integer,
        "box_2d": bounding_box (list[integer]) (SINGLE bounding box)
      }
    }
  ]
}
YOU MUST ENSURE THAT YOUR OUTPUT IS A VALID JSON OBJECT!
</output_schema>
```

Classification	Description
ROOT	The top-level node containing the entire document structure
TEXTS	
TITLE	The specific main title of the document
PARAGRAPH	Standard body text content
SECTION_HEADER	Section headings or subheaders within the text body
FOOTNOTE	Explanatory notes usually placed at the bottom of a page/text
LISTS	
LIST	A container node for a list of items
LIST_ITEM	An individual item within a list
REFERENCE_LIST	A container node for a list of reference items
REFERENCE_ITEM	An individual item within a reference list
FIGURES AND TABLES	
CAPTION	Descriptive text immediately accompanying a table or figure
FIGURE	Graphical elements, diagrams, or pictures
TABLE	A container node for tabular data
DOC_INDEX	A tabular node containing the TOC
TABLE_ROW	A horizontal row within a table
TABLE_CELL	An individual cell containing data within a table row
MISCELLANEOUS	
PAGE_FOOTER	Repeating page footer (page numbers, copyright, etc.)
KEY_VALUE	A specific key-value pair
PAGE_HEADER	Repeating header found at the top of pages (e.g., journal name)
KEY_VALUE_AREA	A distinct region grouped by key-value pairs (e.g., article info)
FORM_AREA	A region indicating form content (e.g., text-fields)
FORMULA	A mathematical formula
WATERMARK	A watermark from the publishing organization
FALLBACK	
UNKNOWN	Parser cannot determine the element type
MISSING	Parser returns a classification for which no mapping exists

Table A.1.: Complete list of classifications permitted to be returned by a DP implementation. Each implementation provides a mapping from their native output classifications to the standard set defined here. Some ParsingResultTypes may only be returned from a subset of these implementations.

List of Figures

List of Tables

Acronyms

AI artificial intelligence. 1, 7, 13, 14

AP average precision. 16

CNN convolutional neural networks. 7

DLA document layout analysis. 6, 7, 12, 13, 16

DP document parsing. 1–3, 6, 7, 9–14, 16–19, 23

DTBB detected bounding box. 16, 17

FN false negative. 16

FP false positive. 16

GTBB ground truth bounding box. 16, 17

IoU intersection over union. 5, 16, 17

LLM large language model. 1, 2, 4, 6

NCCN National Comprehensive Cancer Network. 1

NLP natural language processing. 1, 4, 5

OCR optical character recognition. 7, 13

PDF portable document format. 1, 6, 9, 12, 13

RAG retrieval-augmented generation. 1, 2, 4, 6, 9, 12–14

TP true positive. 16, 17

VLM vision-language model. 7, 13

Bibliography

[1] E. Steinberg, S. Greenfield, D. M. Wolman, M. Mancher, and R. Graham. *Clinical practice guidelines we can trust*. national academies press, 2011.

[2] National Comprehensive Cancer Network. *About Clinical Practice Guidelines*. NCCN. URL: https://www.nccn.org/guidelines/guidelines-process/about-nccn-clinical-practice-guidelines (visited on 01/29/2026).

[3] B. H. Kann, S. B. Johnson, H. J. Aerts, R. H. Mak, and P. L. Nguyen. "Changes in length and complexity of clinical practice guidelines in oncology, 1996-2019". In: *JAMA Network Open* 3.3 (2020), e200841–e200841.

[4] Chair of Software Engineering for Business Information Systems, TUM School of Computation, Information and Technology, Technichal University of Munich. *AI-Based Knowledge Assistant for Cancer Care (Aidvice)*. 2025. URL: https://www.cs.cit.tum.de/sebis/research/natural-language-processing/ai-based-knowledge-assistant-for-cancer-care-aidvice/ (visited on 01/28/2026).

[5] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, and D. Kiela. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. 2021. arXiv: 2005.11401 [cs.CL]. URL: https://arxiv.org/abs/2005.11401.

[6] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, M. Wang, and H. Wang. *Retrieval-Augmented Generation for Large Language Models: A Survey*. 2024. arXiv: 2312.10997 [cs.CL]. URL: https://arxiv.org/abs/2312.10997.

[7] S. Wang, F. Zhao, D. Bu, and et al. "LINS: A general medical Q&A framework for enhancing the quality and credibility of LLM-generated responses". In: *Nature Communications* 16.1 (Oct. 2025), p. 9076. DOI: 10.1038/s41467-025-64142-2. URL: https://doi.org/10.1038/s41467-025-64142-2.

[8] J. Hladěna, K. Šteflovič, P. Čech, K. Štekerová, and A. Žváčková. "The Effect of Chunk Size on the RAG Performance". In: *Software Engineering: Emerging Trends and Practices in System Development*. Ed. by R. Silhavy and P. Silhavy. Cham: Springer Nature Switzerland, 2025, pp. 317–326. ISBN: 978-3-032-00712-4.

[9] T. Chen, H. Wang, S. Chen, W. Yu, K. Ma, X. Zhao, H. Zhang, and D. Yu. *Dense X Retrieval: What Retrieval Granularity Should We Use?* 2024. arXiv: 2312.06648 [cs.CL]. URL: https://arxiv.org/abs/2312.06648.

[10] L. Müller, J. Holstein, S. Bause, G. Satzger, and N. Kühl. *Data Quality Challenges in Retrieval-Augmented Generation*. 2025. arXiv: 2510.00552 [cs.AI]. URL: https://arxiv.org/abs/2510.00552.

[11] X. Ma, S. Zhuang, B. Koopman, G. Zuccon, W. Chen, and J. Lin. *VISA: Retrieval Augmented Generation with Visual Source Attribution*. 2024. arXiv: 2412.14457 [cs.IR]. URL: https://arxiv.org/abs/2412.14457.

[12] Q. Zhang, B. Wang, V. S.-J. Huang, J. Zhang, Z. Wang, H. Liang, C. He, and W. Zhang. *Document Parsing Unveiled: Techniques, Challenges, and Prospects for Structured Information Extraction*. 2025. arXiv: 2410.21169 [cs.MM]. URL: https://arxiv.org/abs/2410.21169.

[13] R. Qu, R. Tu, and F. Bao. "Is semantic chunking worth the computational cost?" In: *Findings of the Association for Computational Linguistics: NAACL 2025*. 2025, pp. 2155–2177.

[14] H. Chase. *LangChain*. https://github.com/langchain-ai/langchain. Oct. 2022. URL: https://github.com/langchain-ai/langchain.

[15] J. Liu. *LlamaIndex*. Nov. 2022. DOI: 10.5281/zenodo.1234. URL: https://github.com/jerryjliu/llama_index.

[16] N. Livathinos, C. Auer, M. Lysak, A. Nassar, M. Dolfi, P. Vagenas, C. B. Ramis, M. Omenetti, K. Dinkla, Y. Kim, S. Gupta, R. T. de Lima, V. Weber, L. Morin, I. Meijer, V. Kuropiatnyk, and P. W. J. Staar. *Docling: An Efficient Open-Source Toolkit for AI-driven Document Conversion*. 2025. arXiv: 2501.17887 [cs.CL]. URL: https://arxiv.org/abs/2501.17887.

[17] LangChain Inc. *LangChain Docs: Splitting recursively*. URL: https://docs.langchain.com/oss/python/integrations/splitters/recursive_text_splitter (visited on 01/27/2026).

[18] E. Guerra-Farfan, Y. Garcia-Sanchez, M. Jornet-Gibert, J. H. Nuñez, M. Balaguer-Castro, and K. Madden. "Clinical practice guidelines: The good, the bad, and the ugly". In: *Injury* 54 (2023). AOTrauma Europe Supplement: Clinical Research: Lessons Learned-Looking Ahead, S26–S29. ISSN: 0020-1383. DOI: https://doi.org/10.1016/j.injury.2022.01.047. URL: https://www.sciencedirect.com/science/article/pii/S0020138322000778.

[19] N. L. Stout, D. Santa Mina, K. D. Lyons, K. Robb, and J. K. Silver. "A systematic review of rehabilitation and exercise recommendations in oncology guidelines". In: *CA: a cancer journal for clinicians* 71.2 (2021), pp. 149–175.

[20] European Society for Medical Oncology. *European Society for Medical Oncology (ESMO)*. URL: https://www.esmo.org/ (visited on 02/04/2026).

[21] Arbeitsgemeinschaft der Wissenschaftlichen Medizinischen Fachgesellschaften. *Arbeitsgemeinschaft der Wissenschaftlichen Medizinischen Fachgesellschaften (AWMF)*. URL: https://www.awmf.org (visited on 02/04/2026).

[22] S. Banerjee, C. M. Booth, E. Bruera, M. W. Buechler, A. Drilon, T. J. Fry, I. M. Ghobrial, L. Gianni, R. K. Jain, G. Kroemer, et al. "Two decades of advances in clinical oncology—lessons learned and future directions". In: *Nature Reviews Clinical Oncology* 21.11 (2024), pp. 771–780.

[23] International Organization for Standardization. *ISO 32000-2:2020: Document management – Portable document format – Part 2: PDF 2.0*. Internation Organization for Standardization, 2020.

[24] J. Hirschberg and C. D. Manning. "Advances in natural language processing". In: *Science* 349.6245 (2015), pp. 261–266.

[25] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[26] B. Min, H. Ross, E. Sulem, A. P. B. Veyseh, T. H. Nguyen, O. Sainz, E. Agirre, I. Heintz, and D. Roth. "Recent Advances in Natural Language Processing via Large Pre-trained Language Models: A Survey". In: *ACM Comput. Surv.* 56.2 (Sept. 2023). ISSN: 0360-0300. DOI: `10.1145/3605943`. URL: `https://doi.org/10.1145/3605943`.

[27] X. Song, A. Salcianu, Y. Song, D. Dopson, and D. Zhou. "Fast wordpiece tokenization". In: *Proceedings of the 2021 conference on empirical methods in natural language processing*. 2021, pp. 2089–2103.

[28] S. J. Mielke, Z. Alyafeai, E. Salesky, C. Raffel, M. Dey, M. Gallé, A. Raja, C. Si, W. Y. Lee, B. Sagot, and S. Tan. *Between words and characters: A Brief History of Open-Vocabulary Modeling and Tokenization in NLP*. 2021. arXiv: `2112.10508 [cs.CL]`. URL: `https://arxiv.org/abs/2112.10508`.

[29] M. Schuster and K. Nakajima. "Japanese and Korean voice search". In: *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2012, pp. 5149–5152. DOI: `10.1109/ICASSP.2012.6289079`.

[30] T. Kudo and J. Richardson. "SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing". In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Ed. by E. Blanco and W. Lu. Brussels, Belgium: Association for Computational Linguistics, Nov. 2018, pp. 66–71. DOI: `10.18653/v1/D18-2012`. URL: `https://aclanthology.org/D18-2012/`.

[31] T. Kudo. "Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates". In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by I. Gurevych and Y. Miyao. Melbourne, Australia: Association for Computational Linguistics, July 2018, pp. 66–75. DOI: `10.18653/v1/P18-1007`. URL: `https://aclanthology.org/P18-1007/`.

[32] A. Ghosh, A. Acharya, S. Saha, V. Jain, and A. Chadha. *Exploring the Frontier of Vision-Language Models: A Survey of Current Methodologies and Future Directions*. 2025. arXiv: `2404.07214 [cs.CV]`. URL: `https://arxiv.org/abs/2404.07214`.

[33] S. Bai, K. Chen, X. Liu, J. Wang, W. Ge, S. Song, K. Dang, P. Wang, S. Wang, J. Tang, et al. "Qwen2. 5-vl technical report". In: *arXiv preprint arXiv:2502.13923* (2025).

[34] L. d. F. D. Costa and R. M. Cesar Jr. *Shape analysis and classification: theory and practice*. CRC Press, Inc., 2000.

[35] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola. *Dive into Deep Learning*. 2023. arXiv: 2106.11342 [cs.LG]. URL: https://arxiv.org/abs/2106.11342.

[36] R. Kaur and S. Singh. "A comprehensive review of object detection with deep learning". In: *Digital Signal Processing* 132 (2023), p. 103812. ISSN: 1051-2004. DOI: https://doi.org/10.1016/j.dsp.2022.103812. URL: https://www.sciencedirect.com/science/article/pii/S1051200422004298.

[37] N. Aksoy, Z. A. Güven, and M. O. Ünalır. "Understanding the Impact of Dataset Characteristics on RAG based Multi-hop QA Performance". In: (2025).

[38] J. Ni, C. Qu, J. Lu, Z. Dai, G. H. Abrego, J. Ma, V. Zhao, Y. Luan, K. Hall, M.-W. Chang, et al. "Large dual encoders are generalizable retrievers". In: *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. 2022, pp. 9844–9855.

[39] T. Gao, H. Yen, J. Yu, and D. Chen. "Enabling Large Language Models to Generate Text with Citations". In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Ed. by H. Bouamor, J. Pino, and K. Bali. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 6465–6488. DOI: 10.18653/v1/2023.emnlp-main.398. URL: https://aclanthology.org/2023.emnlp-main.398/.

[40] L. Ouyang, Y. Qu, H. Zhou, J. Zhu, R. Zhang, Q. Lin, B. Wang, Z. Zhao, M. Jiang, X. Zhao, J. Shi, F. Wu, P. Chu, M. Liu, Z. Li, C. Xu, B. Zhang, B. Shi, Z. Tu, and C. He. *OmniDocBench: Benchmarking Diverse PDF Document Parsing with Comprehensive Annotations*. 2024. arXiv: 2412.07626 [cs.CV]. URL: https://arxiv.org/abs/2412.07626.

[41] B. Wang, C. Xu, X. Zhao, L. Ouyang, F. Wu, Z. Zhao, R. Xu, K. Liu, Y. Qu, F. Shang, B. Zhang, L. Wei, Z. Sui, W. Li, B. Shi, Y. Qiao, D. Lin, and C. He. *MinerU: An Open-Source Solution for Precise Document Content Extraction*. 2024. arXiv: 2409.18839 [cs.CV]. URL: https://arxiv.org/abs/2409.18839.

[42] D. S. Team. *Docling Technical Report*. Tech. rep. Version 1.0.0. Aug. 2024. DOI: 10.48550/arXiv.2408.09869. eprint: 2408.09869. URL: https://arxiv.org/abs/2408.09869.

[43] H. Xing, F. Gao, Q. Zheng, Z. Zhu, Z. Shao, and M. Yan. "Intelligent Document Parsing: Towards End-to-end Document Parsing via Decoupled Content Parsing and Layout Grounding". In: *Findings of the Association for Computational Linguistics: EMNLP 2025*. Ed. by C. Christodoulopoulos, T. Chakraborty, C. Rose, and V. Peng. Suzhou, China: Association for Computational Linguistics, Nov. 2025, pp. 19987–19998. ISBN: 979-8-89176-335-7. DOI: 10.18653/v1/2025.findings-emnlp.1088. URL: https://aclanthology.org/2025.findings-emnlp.1088/.

[44] J. Niu, Z. Liu, Z. Gu, B. Wang, L. Ouyang, Z. Zhao, T. Chu, T. He, F. Wu, Q. Zhang, Z. Jin, G. Liang, R. Zhang, W. Zhang, Y. Qu, Z. Ren, Y. Sun, Y. Zheng, D. Ma, Z. Tang, B. Niu, Z. Miao, H. Dong, S. Qian, J. Zhang, J. Chen, F. Wang, X. Zhao, L. Wei, W. Li, S. Wang, R. Xu, Y. Cao, L. Chen, Q. Wu, H. Gu, L. Lu, K. Wang, D. Lin, G. Shen, X. Zhou, L. Zhang, Y. Zang, X. Dong, J. Wang, B. Zhang, L. Bai, P. Chu, W. Li, J. Wu, L. Wu, Z. Li, G. Wang, Z. Tu, C. Xu, K. Chen, Y. Qiao, B. Zhou, D. Lin, W. Zhang, and C. He.

MinerU2.5: A Decoupled Vision-Language Model for Efficient High-Resolution Document Parsing. 2025. arXiv: 2509.22186 [cs.CV]. URL: https://arxiv.org/abs/2509.22186.

[45] Unstructured.io Team. *Unstructured.io: Open-Source Pre-Processing Tools for Unstructured Data*. URL: https://unstructured.io (visited on 01/20/2026).

[46] Z. Li, Y. Liu, Q. Liu, Z. Ma, Z. Zhang, S. Zhang, Z. Guo, J. Zhang, X. Wang, and X. Bai. *MonkeyOCR: Document Parsing with a Structure-Recognition-Relation Triplet Paradigm*. 2025. arXiv: 2506.05218 [cs.CV]. URL: https://arxiv.org/abs/2506.05218.

[47] N. Livathinos, C. Auer, A. Nassar, R. T. de Lima, M. Lysak, B. Ebouky, C. Berrospi, M. Dolfi, P. Vagenas, M. Omenetti, K. Dinkla, Y. Kim, V. Weber, L. Morin, I. Meijer, V. Kuropiatnyk, T. Strohmeyer, A. S. Gurbuz, and P. W. J. Staar. *Advanced Layout Analysis Models for Docling*. 2025. arXiv: 2509.11720 [cs.CV]. URL: https://arxiv.org/abs/2509.11720.

[48] T. Sun, C. Cui, Y. Du, and Y. Liu. *PP-DocLayout: A Unified Document Layout Detection Model to Accelerate Large-Scale Data Construction*. 2025. arXiv: 2503.17213 [cs.CV]. URL: https://arxiv.org/abs/2503.17213.

[49] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi. "You Only Look Once: Unified, Real-Time Object Detection". In: *CoRR* abs/1506.02640 (2015). arXiv: 1506.02640. URL: http://arxiv.org/abs/1506.02640.

[50] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko. "End-to-End Object Detection with Transformers". In: *CoRR* abs/2005.12872 (2020). arXiv: 2005.12872. URL: https://arxiv.org/abs/2005.12872.

[51] Y. Huang, T. Lv, L. Cui, Y. Lu, and F. Wei. *LayoutLMv3: Pre-training for Document AI with Unified Text and Image Masking*. 2022. arXiv: 2204.08387 [cs.CL]. URL: https://arxiv.org/abs/2204.08387.

[52] N. Islam, Z. Islam, and N. Noor. "A survey on optical character recognition system". In: *arXiv preprint arXiv:1710.05703* (2017).

[53] R. Kittinaradorn and JaidedAI. *EasyOCR*. https://github.com/JaidedAI/EasyOCR. 2020.

[54] R. Smith. "An overview of the Tesseract OCR engine". In: *Ninth international conference on document analysis and recognition (ICDAR 2007)*. Vol. 2. IEEE. 2007, pp. 629–633.

[55] Unstructured.io Team. *Unstructured.io: Documentation for the open-source library*. URL: https://docs.unstructured.io/open-source/introduction/overview (visited on 01/28/2026).

[56] H. Wei, C. Liu, J. Chen, J. Wang, L. Kong, Y. Xu, Z. Ge, L. Zhao, J. Sun, Y. Peng, C. Han, and X. Zhang. *General OCR Theory: Towards OCR-2.0 via a Unified End-to-end Model*. 2024. arXiv: 2409.01704 [cs.CV]. URL: https://arxiv.org/abs/2409.01704.

[57] Y. Li, G. Yang, H. Liu, B. Wang, and C. Zhang. *dots.ocr: Multilingual Document Layout Parsing in a Single Vision-Language Model*. 2025. arXiv: 2512.02498 [cs.CV]. URL: https://arxiv.org/abs/2512.02498.

[58] B. Smith and A. Troynikov. *Evaluating Chunking Strategies for Retrieval*. Tech. rep. Chroma, June 2024. URL: https://research.trychroma.com/evaluating-chunking.

[59] K. Kise, M. Junker, A. Dengel, and K. Matsumoto. "Passage-Based Document Retrieval as a Tool for Text Mining with User's Information Needs". In: *Discovery Science*. Ed. by K. P. Jantke and A. Shinohara. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 155–169. ISBN: 978-3-540-45650-6.

[60] J. P. Callan. "Passage-level evidence in document retrieval". In: *SIGIR'94: Proceedings of the Seventeenth Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval, organised by Dublin City University*. Springer. 1994, pp. 302–310.

[61] LlamaIndex. *NodeParser Modules*. URL: https://developers.llamaindex.ai/python/framework/module_guides/loading/node_parsers/modules/ (visited on 01/27/2026).

[62] S. Jaiswal, P. Bisht, K. Kansara, and M. S. Datta. "Comparison of Chunking Techniques Across Diverse Document Types in NLP Retrieval Tasks". In: *2025 International Conference on Responsible, Generative and Explainable AI (ResGenXAI)*. 2025, pp. 1–6. DOI: 10.1109/ResgenXAI64788.2025.11344045.

[63] O. Koshorek, A. Cohen, N. Mor, M. Rotman, and J. Berant. "Text segmentation as a supervised learning task". In: *arXiv preprint arXiv:1803.09337* (2018).

[64] A. V. Duarte, J. D. Marques, M. Graça, M. Freire, L. Li, and A. L. Oliveira. "Lumberchunker: Long-form narrative document segmentation". In: *Findings of the Association for Computational Linguistics: EMNLP 2024*. 2024, pp. 6473–6486.

[65] C. He, W. Li, Z. Jin, C. Xu, B. Wang, and D. Lin. "Opendatalab: Empowering general artificial intelligence with open datasets". In: *arXiv preprint arXiv:2407.13773* (2024).

[66] Google Cloud. *Document AI: Process documents with Gemini layout parser*. URL: https://docs.cloud.google.com/document-ai/docs/layout-parse-chunk (visited on 02/06/2026).

[67] Z. Ge, S. Liu, F. Wang, Z. Li, and J. Sun. *YOLOX: Exceeding YOLO Series in 2021*. 2021. arXiv: 2107.08430 [cs.CV]. URL: https://arxiv.org/abs/2107.08430.

[68] A. Nassar, N. Livathinos, M. Lysak, and P. Staar. "TableFormer: Table Structure Understanding With Transformers". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2022, pp. 4614–4623. DOI: https://doi.org/10.1109/CVPR52688.2022.00457.

[69] Y. Zhao, W. Lv, S. Xu, J. Wei, G. Wang, Q. Dang, Y. Liu, and J. Chen. *DETRs Beat YOLOs on Real-time Object Detection*. 2024. arXiv: 2304.08069 [cs.CV]. URL: https://arxiv.org/abs/2304.08069.

[70] B. Pfitzmann, C. Auer, M. Dolfi, A. S. Nassar, and P. W. J. Staar. "DocLayNet: A Large Human-Annotated Dataset for Document-Layout Analysis". In: (2022). DOI: 10.1145/3534678.353904. URL: https://arxiv.org/abs/2206.01062.

[71] IBM Research. *Granite Docling Documentation*. IBM. URL: https://www.ibm.com/granite/docs/models/docling (visited on 02/03/2026).

[72] A. Nassar, A. Marafioti, M. Omenetti, M. Lysak, N. Livathinos, C. Auer, L. Morin, R. T. de Lima, Y. Kim, A. S. Gurbuz, M. Dolfi, M. Farré, and P. W. J. Staar. *SmolDocling: An ultra-compact vision-language model for end-to-end multi-modal document conversion*. 2025. arXiv: 2503.11576 [cs.CV]. URL: https://arxiv.org/abs/2503.11576.

[73] J. X. M. Artifex Software Inc. *PyMuPDF*. Version 1.26.7. 2025. URL: https://github.com/pymupdf/PyMuPDF.

[74] Z. Zhao, H. Kang, B. Wang, and C. He. *DocLayout-YOLO: Enhancing Document Layout Analysis through Diverse Synthetic Data and Global-to-Local Adaptive Perception*. 2024. arXiv: 2410.12628 [cs.CV]. URL: https://arxiv.org/abs/2410.12628.

[75] B. Wang, Z. Gu, C. Xu, B. Zhang, B. Shi, and C. He. *UniMERNet: A Universal Network for Real-World Mathematical Expression Recognition*. 2024. arXiv: 2404.15254 [cs.CV].

[76] G. Comanici, E. Bieber, M. Schaekermann, et al. *Gemini 2.5: Pushing the Frontier with Advanced Reasoning, Multimodality, Long Context, and Next Generation Agentic Capabilities*. 2025. arXiv: 2507.06261 [cs.CL]. URL: https://arxiv.org/abs/2507.06261.

[77] Google. *Image Understanding*. Google AI for Developers. URL: https://ai.google.dev/gemini-api/docs/image-understanding (visited on 02/03/2026).

[78] LlamaIndex. *LlamaParse: GenAI-native document parsing platform*. 2024. URL: https://www.llamaindex.ai/llamaparse (visited on 02/03/2026).

[79] Google Cloud. *Document AI: Enterprise Document OCR*. URL: https://docs.cloud.google.com/document-ai/docs/enterprise-document-ocr (visited on 02/06/2026).

[80] A. Antonacopoulos, S. Pletschacher, D. Bridson, and C. Papadopoulos. "ICDAR 2009 Page Segmentation Competition". In: *2009 10th International Conference on Document Analysis and Recognition*. 2009, pp. 1370–1374. DOI: 10.1109/ICDAR.2009.275.

[81] M. Li, Y. Xu, L. Cui, S. Huang, F. Wei, Z. Li, and M. Zhou. *DocBank: A Benchmark Dataset for Document Layout Analysis*. 2020. arXiv: 2006.01038 [cs.CL]. URL: https://arxiv.org/abs/2006.01038.

[82] X. Zhong, J. Tang, and A. J. Yepes. "PubLayNet: largest dataset ever for document layout analysis". In: *2019 International Conference on Document Analysis and Recognition (ICDAR)*. IEEE. Sept. 2019, pp. 1015–1022. DOI: 10.1109/ICDAR.2019.00166.

[83] K. Benkirane. *publaynet-mini*. https://huggingface.co/datasets/kenza-ily/publaynet-mini. 2029.

[84] Z. Zou, K. Chen, Z. Shi, Y. Guo, and J. Ye. *Object Detection in 20 Years: A Survey*. 2023. arXiv: 1905.05055 [cs.CV]. URL: https://arxiv.org/abs/1905.05055.

[85] A. J. Yepes, X. Zhong, and D. Burdick. *ICDAR 2021 Competition on Scientific Literature Parsing*. 2021. arXiv: 2106.14616 [cs.IR]. URL: https://arxiv.org/abs/2106.14616.

[86] K. Oksuz, B. C. Cam, S. Kalkan, and E. Akbas. *One Metric to Measure them All: Localisation Recall Precision (LRP) for Evaluating Visual Detection Tasks*. 2021. arXiv: 2011.10772 [cs.CV]. URL: https://arxiv.org/abs/2011.10772.

[87] I. Karmanov, A. S. Deshmukh, L. Voegtle, P. Fischer, K. Chumachenko, T. Roman, J. Seppänen, J. Parmar, J. Jennings, A. Tao, et al. “\'Eclair–Extracting Content and Layout with Integrated Reading Order for Documents”. In: *arXiv preprint arXiv:2502.04223* (2025).

[88] R. Padilla, W. L. Passos, T. L. B. Dias, S. L. Netto, and E. A. B. da Silva. “A Comparative Analysis of Object Detection Metrics with a Companion Open-Source Toolkit”. In: *Electronics* 10.3 (2021). ISSN: 2079-9292. DOI: 10.3390/electronics10030279. URL: https://www.mdpi.com/2079-9292/10/3/279.

[89] A. Avetisyan, C. Xie, H. Howard-Jenkins, T.-Y. Yang, S. Aroudj, S. Patra, F. Zhang, D. Frost, L. Holland, C. Orme, et al. “Scenescript: Reconstructing scenes with an autoregressive structured language model”. In: *European Conference on Computer Vision*. Springer. 2024, pp. 247–263.

[90] Z. C. Lipton, C. Elkan, and B. Narayanaswamy. *Thresholding Classifiers to Maximize F1 Score*. 2014. arXiv: 1402.1892 [stat.ML]. URL: https://arxiv.org/abs/1402.1892.

[91] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. “Microsoft coco: Common objects in context”. In: *European conference on computer vision*. Springer. 2014, pp. 740–755.

[92] MiXaiLL76. “Faster-COCO-Eval: Faster and Enhanced COCO Evaluation Library”. In: (2024).