

Detailed Technical Report on the Extraction Layer and Its Role in the Side-by-Side PDF Viewer

1. Introduction

The extraction layer is the core system responsible for transforming raw PDF documents into structured, searchable, and visually referenceable components. In a side-by-side PDF viewer—especially one integrated with an AI assistant, semantic search, and form-field grounding—the extraction layer is the foundation that enables precise highlighting, text alignment, semantic retrieval, and interactive navigation. Without this layer, the viewer would function merely as a passive PDF display with no intelligent behavior.

2. Objectives of the Extraction Layer

The extraction layer is designed to accomplish several highly technical objectives:

- Structure raw PDFs into machine-readable text and metadata.
- Extract layout geometry including bounding boxes for visual mappings.
- Normalize coordinates into a universal reference system.
- Segment the document into semantic chunks.
- Generate embeddings that enable semantic retrieval.
- Persist all structured data across the data lake, SQL DB, and vector DB.

These objectives allow the side-by-side viewer to provide instant highlighting, search-to-location navigation, and AI-grounded explanations linked to exact positions within the PDF.

3. Full System Workflow of the Extraction Layer

The extraction layer processes PDFs through the following stages:

- 3.1 PDF Ingestion PDFs are uploaded and stored in the data lake. Each document is assigned a unique document ID, and upload triggers an extraction job.
- 3.2 Parsing and Layout Extraction Using libraries such as PyMuPDF or pdf.js, the system extracts:
 - Raw text
 - Line blocks and paragraph blocks
 - Bounding boxes (x0, y0, x1, y1)
 - Page dimensions
 - Structural markers (columns, headings, bullet points)
- 3.3 Bounding Box Normalization Since PDF coordinates differ from browser pixel coordinates, bounding boxes are normalized so that they can be mapped accurately in a responsive viewer. Normalization ensures that (0,0) always means top-left and all values remain in the range [0,1].
- 3.4 Semantic Chunking Chunks are derived by grouping text into meaningful units such as paragraphs, procedure steps, bullet items, or logical sentences. This improves retrieval accuracy and ensures the viewer highlights entire conceptual units.
- 3.5 Embedding Generation Each chunk is encoded using an embedding model (OpenAI or HuggingFace) to produce high-dimensional vectors optimized for semantic similarity. These embeddings enable RAG-based search, allowing users to query concepts instead of keywords.
- 3.6 Metadata Packaging All extracted information—chunk ID, page number, bounding box, text, embedding—is compiled into metadata structures and stored across the data lake, SQL DB, and vector DB.
- 3.7 Persistence Structured outputs are stored as:
 - Parquet files in the data lake (for scalable analytics)
 - SQL rows (for fast metadata lookup)
 - Vector DB entries (for semantic retrieval)

4. How the Side-by-Side Viewer Uses Extraction Metadata

The viewer makes extensive use of extraction outputs in the following ways:

- 4.1 PDF Highlighting The bounding boxes allow the UI to draw highlight rectangles directly onto the PDF using absolute positioning.
- 4.2 Scroll-to-Chunk Navigation Each chunk includes a page number and bounding box, allowing seamless navigation when:
 - A user clicks a search result.
 - A form field gains focus.
 - The assistant cites a source.
- 4.3 Semantic Search Integration The viewer interacts with the vector DB to retrieve chunks that match a user's query. The metadata then allows the viewer to highlight precisely where the relevant content appears.
- 4.4 Form Field Grounding Every form field can be linked to an exact location in the PDF through extraction metadata. When a user taps a field, the viewer scrolls instantly to the correct location.
- 4.5 AI Response Grounding The assistant can quote and point to exact lines in the PDF because extraction metadata connects semantic concepts to visual locations.

5. Efficiency Enhancements Provided by the Extraction Layer

The extraction layer makes the entire system efficient by:

- Eliminating real-time parsing requirements.
- Reducing viewer-side processing to simple coordinate mapping.
- Providing precomputed metadata for immediate load.
- Supporting parallelized extraction for large documents.
- Enabling caching via Redis for rapid metadata retrieval.
- Allowing incremental updates using hashing-based change detection.
- Batch-processing embeddings to reduce cost and latency.

6. Storage Outputs and Architecture Integration

The extraction layer outputs are stored in a multi-tier architecture:

- Data Lake (S3/Azure/GCS): Raw PDFs, Parquet chunk metadata, embedding backups.
- SQL Database: Chunk references, normalized bounding boxes, page indices.
- Vector Database: Embeddings with metadata for semantic similarity queries.

This hybrid storage strategy ensures scalability, fast lookup, and semantic precision.

7. End-to-End Lifecycle in the Viewer

The real-time side-by-side viewer flow depends entirely on extraction outputs. The lifecycle works as follows:

1. Viewer loads metadata from SQL DB (instant fetch).
2. Viewer renders PDF pages lazily.
3. User interacts with search, form fields, or chat.
4. Viewer uses chunk metadata to find page and bounding box.
5. Viewer scrolls to the exact location and overlays highlight.
6. Assistant responses point back to the original PDF text using metadata.

This allows the system to act as an intelligent assistant rather than a simple document viewer.

8. Conclusion

The extraction layer is the backbone of the side-by-side PDF viewer. It transforms static PDF content into actionable, searchable, navigable, and AI-aware structures. Through precise layout extraction, chunking, embedding, and metadata organization, it enables powerful capabilities including semantic search, real-time highlighting, and AI-generated form guidance. Every interactive feature in the viewer—from scroll-to-text to form grounding—relies on the extraction layer's outputs.