# Building the Legal Assistant with RAG – Data Processing & Storage Pipeline

In this section, we describe the data ingestion and preparation pipeline we implemented for our Legal Assistant project. The goal of this pipeline is to transform legal documents (laws, codes, regulations, etc.) into structured, searchable, and semantically indexed data stored in MongoDB Atlas with vector indexing. This enables Retrieval-Augmented Generation (RAG) for precise and context-aware legal question answering.

## Step 1 – Document Loading with OCR Fallback

Legal documents come in different formats: some are digital PDFs (easy to parse), while others are scanned images (harder to extract). We need a reliable way to extract text from any kind of legal PDF.  
  
Method:  
- Try PDFPlumber for digital PDFs.  
- If extraction fails or returns empty text → switch to OCR (Tesseract).  
- Save each page with its metadata.  
  
Outcome: Every legal document, whether digital or scanned, is transformed into machine-readable text.

## Step 2 – Splitting Text into Chunks

Large legal texts are too long for embeddings and retrieval models. Splitting ensures:  
- Each chunk is small enough for vectorization.  
- Overlap preserves context across boundaries.  
  
Method:  
- Use RecursiveCharacterTextSplitter with:  
 • Chunk size = 1000 characters  
 • Overlap = 150 characters  
  
Outcome: Large legal documents are split into semantically meaningful chunks, ensuring efficient retrieval.

## Step 3 – Embedding & Keyword Extraction

To support semantic search, we must convert each chunk into a vector representation. Keywords provide additional filtering.  
  
Method:  
- Generate embeddings with OpenAIEmbeddings.  
- Extract keywords with YAKE.  
  
Stored Data:  
- Raw text content  
- Embedding vector (1536 dimensions)  
- Extracted keywords  
- Metadata  
  
Outcome: Each chunk is semantically indexed and enriched with keywords for hybrid search (vector + keyword).

## Step 4 – Saving to MongoDB Atlas

MongoDB Atlas provides scalable storage with support for structured metadata and vector indexing.  
  
Method:  
- Validate required fields (titre, pays, categorie, langue, date\_publication, url\_source).  
- Avoid duplicates (check url\_source).  
- Insert chunks in batches.  
  
Outcome: Each legal document is saved as multiple entries (one per chunk), ready for semantic retrieval.

## Step 5 – Vector Index Creation

Vector search enables semantic retrieval: finding text passages that are meaning-related to a user query, not just keyword matches.  
  
MongoDB Atlas Index:  
- knnVector field on vecteur\_embedding (1536 dimensions, cosine similarity).  
- Other fields indexed as strings or dates for filtering.  
  
Outcome: We can now query MongoDB Atlas using both vector similarity (semantic search) and metadata filters.

## Pipeline Overview Diagram

┌─────────────────────────┐  
 │ PDF Legal Document │  
 └───────────┬─────────────┘  
 │  
 ┌──────────────▼──────────────┐  
 │ Step 1: Load Document │  
 │ (PDFPlumber → OCR Fallback)│  
 └──────────────┬──────────────┘  
 │  
 ┌──────────────▼──────────────┐  
 │ Step 2: Split into Chunks │  
 │ (1000 chars, 150 overlap) │  
 └──────────────┬──────────────┘  
 │  
 ┌──────────────▼──────────────┐  
 │ Step 3: Embedding + Keywords│  
 │ (OpenAI + YAKE) │  
 └──────────────┬──────────────┘  
 │  
 ┌──────────────▼──────────────┐  
 │ Step 4: Save to MongoDB │  
 │ (Validation + Batching) │  
 └──────────────┬──────────────┘  
 │  
 ┌──────────────▼──────────────┐  
 │ Step 5: Vector Index │  
 │ (1536-dim cosine search) │  
 └─────────────────────────────┘