AI Data Prep Pipeline

Why This Matters

Every organization has knowledge scattered across dozens of formats. Engineering specs live in Confluence. Project updates are in Jira tickets. Architecture decisions are in Google Docs. API documentation is in Markdown. Configuration is in YAML and JSON. Presentations from last quarter's planning are in PowerPoint. That critical diagram explaining the system is embedded in a PDF someone shared on Slack two years ago.

When you want to build an AI system that can answer questions about your organization's knowledge, the first problem is not embeddings or retrieval or prompt engineering. The first problem is getting all that data into a format the AI can actually use.

This project builds a universal document processing pipeline that ingests files in any format, extracts content as clean Markdown, handles embedded images (extracting them or generating AI descriptions), and stores everything with rich metadata for search. The output can feed a vector database for RAG, or serve as a corpus for agentic search without embeddings.

The Product Vision

Users point the pipeline at a directory, a Confluence space, a Jira project, or a cloud storage bucket. The system crawls all documents, identifies their formats, and processes each one:

- Office documents (doc, docx, ppt, pptx, xls, xlsx) → Markdown with structure preserved
- PDFs → Markdown with text extraction, tables as Markdown tables, images extracted or described
- Confluence/Notion pages → Markdown with links resolved
- Jira tickets → Markdown with metadata (status, assignee, labels) preserved
- Code files → Markdown with syntax highlighting markers
- JSON/YAML → Markdown code blocks with appropriate language tags
- Images → AI-generated descriptions stored as Markdown with image references

For embedded images that cannot be extracted (screenshots in PDFs, diagrams in presentations), the pipeline uses a vision model to generate descriptions. The description captures what the image shows, any text visible in it, and why it might be relevant.

The output is a directory of Markdown files with consistent structure, plus a metadata index. This can be loaded into any vector database (Pinecone, Weaviate, Qdrant, ChromaDB) or used directly with agentic search tools.

Agentic Architecture

Format Detection Agent identifies file types beyond just extensions. A .doc file might actually be RTF. A PDF might be scanned images vs. native text. The agent uses magic bytes, structure analysis, and content sampling to determine the true format and select the appropriate processor.

Document Processor Agents are specialized per format:
- **Office Agent**: Uses python-docx, python-pptx, openpyxl to extract content while preserving structure (headings, lists, tables)
- **PDF Agent**: Uses PyMuPDF or pdfplumber for text, Camelot or tabula for tables, extracts embedded images
- **Web Content Agent**: Handles Confluence API responses, HTML pages, Notion exports
- **Structured Data Agent**: Parses JSON, YAML, XML into readable Markdown representations

Image Processing Agent handles visual content:
- Extracts embedded images and saves them with meaningful filenames
- For images that cannot be extracted (embedded in complex formats), captures screenshots
- Sends images to a vision model (GPT-4V, Claude Vision) for description generation
- Stores descriptions as alt-text in the Markdown

Metadata Extraction Agent captures document context:
- Title, author, creation date, last modified
- Source system (Confluence space, Jira project, file path)
- Document type and detected topics
- Relationships to other documents (links, references)

Chunking Agent splits documents for retrieval:
- Respects document structure (don't split mid-paragraph, keep headings with content)
- Handles different chunk sizes for different use cases
- Preserves metadata through chunking

Indexing Agent loads processed documents into the target system:
- Vector database with embeddings for semantic search
- Full-text index for keyword search
- Hybrid search combining both approaches

Implementation Hints

Start with the easy formats. Plain text, Markdown, and JSON are trivial. Get those working perfectly before tackling PDFs and Office documents, which have endless edge cases.

Use existing libraries aggressively. Do not write your own PDF parser. PyMuPDF, pdfplumber, and pdf2image are battle-tested. python-docx and python-pptx handle Office formats well. The value is in orchestration and AI integration, not reinventing extraction.

Handle failures gracefully. Some documents will fail to process. Corrupted files, password-protected PDFs, proprietary formats. Log these clearly, skip them, and continue. A pipeline that processes 95% of documents is useful; one that crashes on edge cases is not.

Image descriptions need context. Telling a vision model "describe this image" produces generic output. Tell it "this image is from a technical document about AWS architecture, describe what it shows in that context" and you get useful descriptions.

Preserve provenance. Every chunk should trace back to its source document, page number, and section. When the AI answers a question, users need to verify the source. Broken provenance makes the system untrustworthy.

Test with real messy data. Corporate documents are full of surprises: embedded fonts that break extraction, tables that span pages, images with no alt text, mixed languages, OCR errors in scanned PDFs. Your pipeline needs to handle all of this.

Datasets for Validation

****Your Own Data**** is the best test case. Gather documents from a real project: meeting notes, specs, tickets, presentations. Process them and try to answer questions you actually had during the project. This reveals gaps that synthetic data misses.

****Open Source Project Documentation**** provides diverse, publicly available content:
- [Kubernetes docs](https://github.com/kubernetes/website) - Markdown with embedded diagrams, YAML examples
- [AWS documentation](https://github.com/awsdocs) - Multiple repos with different formats and structures
- [Apache project wikis](https://cwiki.apache.org/) - Confluence-style content with rich formatting

****arXiv Papers**** (arxiv.org) offer PDFs with complex layouts:
- Two-column formats, equations, figures with captions
- Good for testing PDF extraction quality
- Ground truth available (LaTeX source for many papers)

****CommonCrawl**** (commoncrawl.org) provides web content at scale:
- HTML pages with embedded images, tables, mixed content
- Tests the web content extraction pipeline
- Useful for stress-testing at volume

****Kaggle Document Datasets****:
- [RVL-CDIP](https://www.kaggle.com/datasets/pdavpoojan/rvl-cdip-dataset) - 400k

document images across 16 categories
- [DocVQA](https://www.docvqa.org/) - Documents with question-answer pairs for evaluation

Government Open Data sites publish documents in multiple formats:
- data.gov - PDFs, spreadsheets, reports
- [UK National Archives](https://www.nationalarchives.gov.uk/) - Historical documents in various formats

Practical approach: Start with a small set of your own documents to debug the pipeline. Add open source documentation for format diversity. Use arXiv PDFs to stress-test complex layouts. Evaluate search quality with DocVQA-style questions.

Evaluation Approach

Extraction Quality: For formats where you have ground truth (LaTeX source for PDFs, original Markdown for HTML), measure character-level accuracy of extraction. Target >95% for clean documents, >80% for complex layouts.

Image Description Quality: Have humans rate whether AI-generated descriptions are accurate and useful. Also test whether the descriptions enable finding the image via search ("show me the architecture diagram" should find the image described as containing an architecture diagram).

Search Relevance: Create a set of questions and manually identify which documents should be retrieved. Measure precision@k and recall@k. Compare vector search, keyword search, and hybrid approaches.

End-to-End QA: Ask questions that require synthesizing information across documents. Measure whether the system retrieves the right chunks and whether an LLM can answer correctly given those chunks.

Interesting Questions

How do you handle versioning? The same document may exist in multiple versions across different systems. Should the pipeline deduplicate, keep all versions, or track version history?

What is the right chunking strategy? Fixed-size chunks lose context. Semantic chunks (by section) vary wildly in size. Overlapping chunks duplicate content. Is there an optimal approach, or does it depend on the use case?

How do you handle private/sensitive content? Documents may contain PII, credentials, or confidential information. Should the pipeline detect and redact these, or is that out of scope?

Can the pipeline learn from corrections? If a user reports that an image description is wrong, can that feedback improve future descriptions of similar images?

How do you keep the index fresh? Documents change. New documents are created. How does the pipeline detect changes and update the index incrementally without reprocessing everything?

Broader Value

Data preparation is the unglamorous foundation of every AI project. Teams spend weeks writing custom scripts to process their specific document formats, then discover edge cases, then rewrite, then discover more edge cases. A robust, general-purpose pipeline saves that effort.

The multimodal aspect is increasingly important. As vision models improve, the ability to understand diagrams, screenshots, and visual content becomes a differentiator. A system that can only search text misses half the information in most organizations.

This project also teaches practical engineering skills: handling diverse file formats, integrating multiple APIs and libraries, building reliable data pipelines, and designing for failure. These skills transfer directly to industry work.