**AI Document Processing & Knowledge Retrieval Agents — Complete System**

**System Architecture**

* **Document Processing Agent**: Handles PDFs, DOCX, TXT, HTML with Textract integration
* **Knowledge Retrieval Agent**: RAG-based Q&A with semantic search
* **Legal Assistant Agent**: Contract analysis and legal document processing
* **Pipeline Manager**: Orchestrates workflows with Lambda functions

**AWS Services Integration**

* **Amazon Bedrock**: LLM operations and embeddings (Claude 3 Sonnet, Titan)
* **Amazon S3**: Document storage and management
* **Amazon OpenSearch**: Vector search and semantic indexing
* **AWS Lambda**: Serverless pipeline orchestration
* **Amazon Textract**: OCR and structured data extraction

**Features Implemented**

**1. Document Processing**

* Multi-format support (PDF, DOCX, TXT, HTML)
* Textract integration for OCR and structured data
* Chunking and embedding generation
* Metadata extraction and document indexing

**2. Knowledge Retrieval (RAG)**

* Vector search with OpenSearch
* Hybrid search (vector + text)
* Context-aware Q&A
* Related question suggestions
* Document summarization

**3. Legal Assistant**

* Contract analysis and clause extraction
* Termination conditions identification
* Payment terms extraction
* Liability and risk assessment
* Contract comparison capabilities
* Legal question answering

**4. Pipeline Orchestration**

* Lambda-based workflow management
* Batch processing capabilities
* Async pipeline execution
* Status tracking and monitoring
* Error handling and retry logic

### Project Structure

AIAgent/

├── agents/ # AI agents

│ ├── document\_processor.py # Document processing agent

│ ├── rag\_system.py # RAG system implementation

│ ├── knowledge\_agent.py # Knowledge retrieval agent

│ └── legal\_agent.py # Legal assistant agent

├── services/ # AWS service wrappers

│ ├── aws\_bedrock.py # Bedrock service wrapper

│ ├── aws\_s3.py # S3 service wrapper

│ ├── aws\_opensearch.py # OpenSearch service wrapper

│ ├── aws\_textract.py # Textract service wrapper

│ └── aws\_lambda.py # Lambda service wrapper

├── orchestration/ # Pipeline management

│ ├── lambda\_functions.py # Lambda function handlers

│ └── pipeline\_manager.py # Pipeline orchestration

├── config/ # Configuration management

│ └── config.py # Configuration settings

├── examples/ # Demo scripts and examples

│ ├── legal\_assistant\_demo.py

│ ├── knowledge\_agent\_demo.py

│ └── setup\_and\_deployment.py

├── requirements.txt # Python dependencies

└── README.md # Comprehensive documentation

### Usage Examples

#### Legal Assistant Example

from agents.legal\_agent import LegalAgent

legal\_agent = LegalAgent()

# Analyze contract

result = legal\_agent.analyze\_contract(

"contract.pdf",

contract\_type="employment"

)

# Extract termination conditions

termination = legal\_agent.extract\_termination\_conditions(

result['document\_id']

)

# Answer legal questions

answer = legal\_agent.answer\_legal\_question(

"What are the termination conditions?",

result['document\_id']

)

#### Knowledge Retrieval Example

from agents.knowledge\_agent import KnowledgeAgent

agent = KnowledgeAgent()

# Process and index document

result = agent.process\_and\_index\_document("document.pdf")

# Ask questions

answer = agent.ask\_question(

"What are the key points in this document?"

)

# Search documents

search\_results = agent.search\_documents("employee benefits")

**Key Benefits**

1. Fast contract analysis (e.g., termination clauses)
2. Semantic search across documents
3. RAG-based question answering with citations
4. Serverless scaling with Lambda
5. Multi-format document support
6. Legal-focused contract review

**Setup Instructions**

1. Install dependencies: pip install -r requirements.txt
2. Configure AWS credentials and services
3. Run setup script: python examples/setup\_and\_deployment.py
4. Test with demos: python examples/legal\_assistant\_demo.py

**Production Readiness**

* Error handling and logging
* Configuration management
* Security best practices
* Performance optimizations
* Documentation
* Demo scripts and examples
* AWS infrastructure setup

The system is ready to deploy and use for document processing, knowledge retrieval, and legal contract analysis.

**Install Dependecies:**

pip install boto3 langchain langchain-aws langchain-community opensearch-py pydantic python-dotenv PyPDF2 python-docx tiktoken numpy sentence-transformers faiss-cpu streamlit pandas requests

**What this project is**

An AI system for end-to-end document processing, retrieval-augmented question answering, and legal contract analysis. It integrates multiple AWS services (Bedrock, S3, OpenSearch, Textract, Lambda) to extract text, generate embeddings, index/search content, and orchestrate workflows.

**Core capabilities**

* **Document processing**: Parse PDFs/DOCX/TXT/HTML, OCR via Textract, chunk text.
* **Knowledge retrieval (RAG)**: Generate embeddings, index in OpenSearch, retrieve context, answer questions with Bedrock.
* **Legal analysis**: Analyze contracts for clauses and risks using a specialized agent.
* **Orchestration**: Serverless pipeline steps via Lambda.

**Main components (where to look)**

* **Agents**
* agents/document\_processor.py: Loads files, uses Textract for OCR, chunks text.
* agents/knowledge\_agent.py: RAG pipeline—embeddings with Bedrock, vector search in OpenSearch, Q&A.
* agents/legal\_agent.py: Contract-focused analysis using Bedrock prompts and heuristics.
* agents/rag\_system.py: Shared RAG utilities (chunking, embedding, search).
* **AWS service wrappers**
* services/aws\_bedrock.py: LLM and embeddings on Bedrock.
* services/aws\_s3.py: Upload/download documents.
* services/aws\_opensearch.py: Index and vector search.
* services/aws\_textract.py: OCR/structured extraction.
* services/aws\_lambda.py: Invoke and manage Lambda.
* **Orchestration**
* orchestration/pipeline\_manager.py: High-level workflow coordination.
* orchestration/lambda\_functions.py: Lambda entrypoints/handlers.
* **Configuration**
* config/config.py: Pydantic settings loading from .env.
* config/env\_example.txt: Template of required env vars.

**Typical data flow**

1. Upload or read a document (optionally stored in S3).
2. Extract text (Textract for scans), normalize, chunk.
3. Create embeddings (Bedrock Titan), index in OpenSearch.
4. For a query, retrieve top-k chunks (vector + keyword hybrid), build prompt.
5. Generate final answer via Bedrock (e.g., Claude model), with citations/context.
6. For legal docs, run specialized prompts to extract clauses/risks/summaries.

**AWS resources used**

* **S3**: Raw and processed document storage.
* **OpenSearch**: Vector index for semantic search.
* **Textract**: OCR for scanned/complex PDFs.
* **Bedrock**: LLM for reasoning and embeddings.
* **Lambda**: Optional serverless orchestration for pipelines.

**How to try it**

* Demos in examples/:
* knowledge\_agent\_demo.py: Process a sample doc, index, Q&A, summarization.
* legal\_assistant\_demo.py: Contract analysis and pipeline walkthrough.
* setup\_and\_deployment.py: Provision AWS resources (S3, IAM, OpenSearch, Lambda) and create .env.

**Configuration essentials**

* .env keys (see config/env\_example.txt):
* AWS creds/region, Bedrock models, S3 bucket/prefixes, OpenSearch endpoint/index, Lambda function name, processing/RAG params.

**When to run locally vs AWS**

* **Local**: Rapid iteration; calls out to AWS services using your local creds.
* **AWS**: Use setup\_and\_deployment.py to create resources, then optionally package and deploy Lambda for serverless pipelines.