Document Analysis, Chat & Comparison Portal

🔍 A Full-Stack AI-Powered Platform for

Semantic Document Retrieval, Multi-Document Chat, and Comparison Workflows

---

📄 Project Overview

Field	Details
Project Title	Document Analysis, Chat & Comparison Portal with Advanced RAG
Author	Bhagwat Chate
Project Lead	Bhagwat Chate
Version	1.0
Status	In Development
Last Updated	July 19, 2025
GitHub Repository	github.com/bhagwat-chate/document_portal

🧰 Powered By:

- LangChain, FastAPI, Streamlit
- OpenAI / HuggingFace / BGE Embeddings
- FAISS / Chroma / Pinecone Vector Stores
- AWS Fargate, CI/CD with GitHub Actions

🧭 Vision:

To build a scalable, production-ready RAG system that enables organizations and researchers to upload, chat with, and compare documents intelligently, using state-of-the-art LLMs, multi-document retrieval, and real-time diff visualizations — all within a clean, interactive portal.

Vision Document

---

1. 🎯 Project Vision

Build a production-grade, AI-powered document intelligence portal that enables:

- Seamless document ingestion, search, and retrieval
- Context-aware chat over single or multiple documents
- Side-by-side document comparison and diff visualization
- Scalable full-stack deployment using FastAPI + Streamlit + AWS

The system leverages advanced RAG pipelines with enhancements such as reranking, query rewriting, caching, and local LLMs to ensure speed, accuracy, and affordability.

---

2. 📌 Key Features

🔍 1. Document Intelligence Core

This is the foundational layer of the system that handles document ingestion, preprocessing, and indexing for efficient retrieval.

- **Multi-format Document Ingestion**: Supports uploading PDFs, DOCX, and plain text files. You can extend to HTML, Markdown, and scanned images (OCR) in future.
- **Chunking Strategies**: Uses LangChain's *RecursiveCharacterTextSplitter*, *TokenTextSplitter*, and custom chunking logic to break documents into semantically meaningful segments while preserving context.
- **Embedding Generation**: Generates semantic vector embeddings using OpenAI (Ada-002), HuggingFace models (e.g., all-MiniLM-L6-v2), or BGE models for multilingual/bi-encoder performance.
- **Vector Storage**: Indexes the chunks into a vector database such as FAISS (local), Chroma (lightweight), or Pinecone (cloud-scale). Each chunk is stored with metadata like:
    - document_title
    - page_number
    - source_id
    - chunk_id
- **Metadata Tagging**: Enables efficient filtering and retrieval using indexed fields like tags, file type, category, and author.

🧠 2. Advanced RAG Capabilities

The core intelligence of your system lies in this layer — enabling smarter, context-rich, and accurate answer generation.

- **Query Rewriting & Context Condensation**: Implements prompt strategies like *RAG Fusion*, *ReAct*, or *Self-Ask* to enhance the quality of user questions and generate better retrieval queries.

- **Multi-Vector Retrieval**: Combines chunk retrieval across multiple documents for unified Q&A, using hybrid search strategies (dense + sparse if needed).
- **Reranking Techniques**: Adds a second stage reranking using:
  - **MMR (Maximal Marginal Relevance)** to remove redundant results
  - **Cross-encoders or LLM scoring** to improve chunk quality
- **Similarity Metric Tuning**: Supports configurable similarity algorithms for retrieval:
  - **Cosine similarity** (default for angle-based search)
  - **Euclidean / L2 distance** (for magnitude-based)
  - **Jaccard Index** (for lexical set similarity)

💬 3. Chat & Comparison Capabilities

This module allows interactive user experience with document-level Q&A and side-by-side comparisons.

- **Session Memory & History**: Supports memory-backed chat sessions using LangChain memory, Redis, or database-backed logs for continuity.
- **Single Document Q&A**: Enables focused chat over a specific document, e.g., "Summarize Section 3 of this PDF."
- **Multi-Document Chat**: Enables cross-document retrieval and Q&A — e.g., “Compare refund policy between Document A and B.”
- **Comparison Engine**:
  - **Similarity Detection**: Shows overlapping content between two documents.
  - **Difference Detection (Diff Engine)**: Highlights clauses, keywords, or paragraphs that differ across files.
  - **Side-by-Side View**: Presents source snippets in a visually comparable format.

🚀 4. Performance Optimization

Designed to ensure **fast response times** and **affordable inference** even at scale.

- **Local LLM Support**:
  - Deploy **quantized models** (e.g., QLoRA, GPTQ) via vLLM or HuggingFace for low-latency responses
  - Use **Groq**, **Mistral**, or **Mixtral** models for enterprise-grade performance
- **Asynchronous & Batched Retrieval**: Parallelizes chunk retrieval and reranking operations to reduce latency.
- **Cache-Augmented Generation (CAG)**:
  - Stores embeddings and responses of frequently asked queries
  - Avoids repeat computation and reduces cost
  - Can be implemented with Redis or SQLite-based hash map

🧰 5. Text Processing Tools

Robust document parsing is key to accurate chunking and embedding.

- **PyMuPDF (fitz)**: High-quality PDF parsing including coordinates, font, layout, and annotations.

- **PDFMiner**: For structural layout extraction and advanced PDF parsing.
- **Unstructured**: Helps break down and clean raw content into usable segments.
- **python-docx**: For parsing and cleaning .docx files with headings, tables, lists, and sections.

🧪 6. Evaluation & Testing Framework

Ensures the reliability and quality of your RAG pipeline in real-world use.

- **RAG Evaluation Metrics**:
  - *Precision@k*, *Recall@k* for retrieval
  - *Faithfulness*, *Relevance*, *Factuality* for generation
- **Manual Feedback Loop**: Allows users to provide thumbs-up/down or text-based feedback for continuous learning.
- **Final Workflow Testing**: End-to-end validation of flow: **Upload → Chunk → Ask → Compare → View Sources** with logging and visual audit trails.

🌐 7. Frontend & Backend Development

Provides a modern, responsive UI with a robust backend API layer.

- **Streamlit UI**:
  - Responsive file upload, chat window, comparison tool, and citation trace
  - Interactive widgets to choose documents, highlight text, or view source chunks
- **Gradio (Optional)**: For building lightweight demos
- **FastAPI Backend**:
  - Exposes APIs for document processing, chunk retrieval, and LLM responses
  - Supports async execution, error handling, logging, and versioned APIs

---

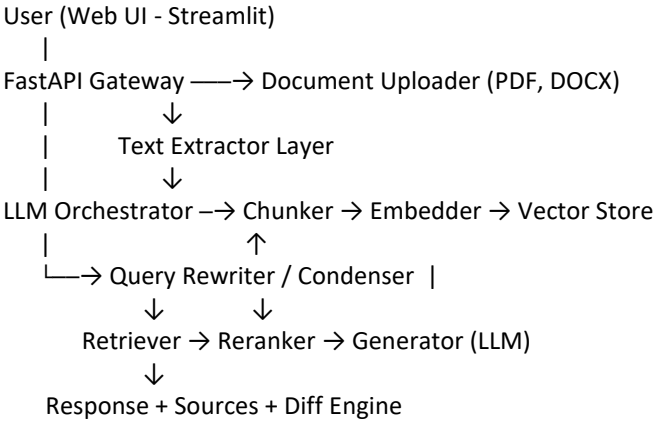
⚙️ 8. LLMOps and DevOps

Supports enterprise-readiness, version control, and scalable deployment.

- **CI/CD via GitHub Actions**:
  - Automatically test, lint, and deploy code
  - Unit test each microservice (chunker, retriever, generator, etc.)
- **AWS Fargate + S3**:
  - Deploy scalable containers without managing servers
  - Store documents and outputs securely in S3 buckets
- **Environment Configurations**:
  - .env and config.yml driven setup
  - Model and vector store configurations switchable via CLI or UI

---

3. 🧱 System Architecture (High-Level)



---

4. 📐 Phased Execution Plan

Phase	Milestone
Phase 1	Build ingestion, chunking, embedding & retrieval
Phase 2	Enable single doc chat, RAG enhancements, reranker
Phase 3	Implement multi-doc chat, comparison, caching
Phase 4	Add frontend (Streamlit), FastAPI integration
Phase 5	AWS Deployment + CI/CD + End-to-End Testing

---

5. 📊 Metrics for Evaluation

- **Retrieval Precision@k**
- **LLM Answer Accuracy** (human-reviewed)
- **Latency per query**
- **Cache hit rate (CAG)**
- **Comparison diff accuracy** (manual validation)

---

6. 🧠 Tech Stack

Layer	Tools
Embedding	HuggingFace, OpenAI, BGE
Vector DB	FAISS (local), Pinecone (cloud)
LLMs	GPT-4, Mistral, LLaMA, Groq, vLLM
Backend	FastAPI, LangChain
Frontend	Streamlit, Gradio
DevOps	GitHub Actions, AWS Fargate, S3
Parsing	PyMuPDF, Unstructured, PDFMiner

---

7. 🔒 Future Enhancements

- 🔁 Auto-retraining with user feedback (RAG refinement)
- 🌐 Real-time web document ingestion (news, URLs)
- 🧠 Agent-based document digests and summaries
- 🗂️ Document tagging & classification
- 📄 Export answers and comparisons as PDF reports

---

8. 🚧 Risks & Mitigations

Risk	Mitigation
High latency on large docs	Use caching + reranking + async
Inaccurate comparisons	Enforce structured JSON output & eval
Cost escalation (OpenAI)	Switch to vLLM or local LLM
Failures in multi-doc retrieval	Implement fallback retriever logic

---

9. 💡 Strategic Goal

This project will be a flagship showcase of your GenAI capabilities, combining LangChain, advanced RAG, multi-modal chat, and AWS deployment into one unified, FAANG-grade platform. It is ideal for:

- Enterprise document workflows
- Contract and policy comparison
- Research knowledge assistants
- Startup MVPs

---

10. 📌 Summary

The **Document Analysis, Chat & Comparison Portal with Advanced RAG** is a comprehensive, full-stack Generative AI platform designed to address real-world document intelligence challenges. By integrating advanced RAG pipelines with multi-document retrieval, chat, and comparison capabilities, this system aims to empower users with seamless semantic search, structured Q&A, and insightful document comparison — all through an intuitive and scalable interface.

Key highlights include:

- ☑️ Multi-format document ingestion and semantic indexing
- 💬 Intelligent chat over single or multiple documents with memory and reranking
- 🔍 Side-by-side document comparison with similarity and difference detection
- 🚀 Performance optimization using local LLMs, async processing, and caching
- 🧪 Rigorous evaluation metrics and feedback loops
- 🌐 Production-ready deployment with CI/CD, FastAPI backend, and Streamlit UI

This project represents a **FAANG-grade GenAI implementation**, serving as a platform for enterprise use cases, research environments, and AI-powered knowledge systems. With scalable infrastructure, modular

architecture, and robust LLMOps practices, it stands as a future-proof solution in the evolving landscape of AI-driven document analysis.