Cogni-Compliance

An End-to-End Agentic RAG System

A Full-Stack AI Chat Application for Navigating Legal & Regulatory Documents

Aman Deva

The Problem & The Vision

The Problem

Legal and compliance documents (like GDPR, CCPA, HIPAA) are dense, complex, and difficult to query. Getting a specific, accurate answer is time-consuming and requires expert knowledge.

The Vision

To create an intelligent AI assistant that allows users to ask natural language questions and receive accurate, synthesized answers grounded in the source documents.

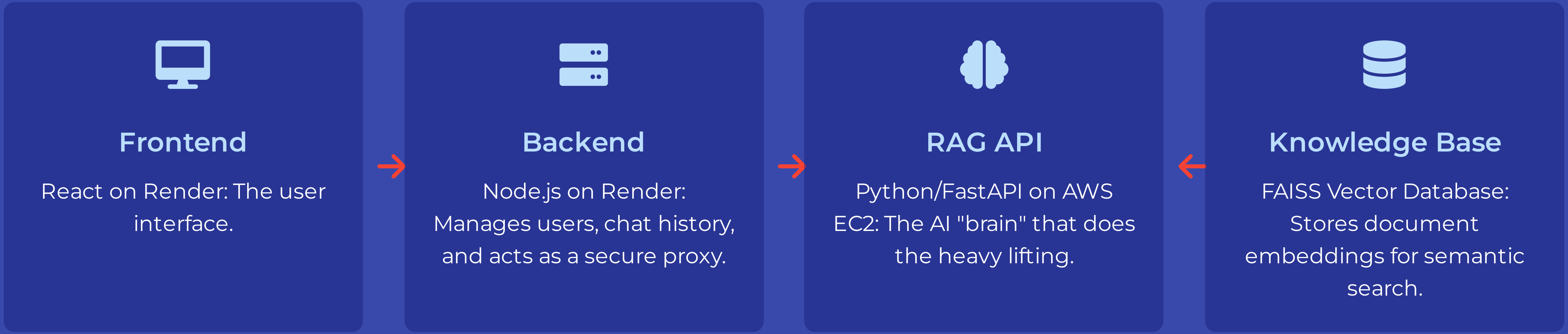
The Solution

- An advanced Agentic RAG backend for intelligent retrieval and generation.
- A modern MERN stack frontend for user authentication and an interactive chat experience.

The Architecture: From User to Answer

This project is a complete, multi-part system deployed across different cloud platforms, demonstrating a real-world microservices architecture.

(Architecture diagram representation)



Frontend

React on Render: The user interface.

Backend

Node.js on Render: Manages users, chat history, and acts as a secure proxy.

RAG API

Python/FastAPI on AWS EC2: The AI "brain" that does the heavy lifting.

Knowledge Base

FAISS Vector Database: Stores document embeddings for semantic search.

Phase 1 - Building the Knowledge Base

> *Goal: Transform raw, messy PDF documents into a clean, structured, and searchable knowledge base. The quality of this phase determines the quality of the entire system.*

1

Text Extraction

Python script using PyMuPDF library to robustly extract all text content from source PDFs: GDPR, CCPA, and HIPAA.

2

Aggressive Cleaning

Custom cleaning functions using regular expressions (re) to remove noise specific to each document, such as headers, footers, page numbers, and other artifacts.

3

Semantic Chunking

RecursiveCharacterTextSplitter from LangChain creates semantically meaningful chunks along natural paragraph breaks. Each chunk includes metadata: source document name and relevant section heading.

Outcome

A clean processed_chunks.jsonl file, where each line is a structured JSON object containing a text chunk and its metadata, ready for vectorization:

```
{
  "text": "Article 5: Principles relating to processing of personal data...",
  "metadata": {
    "source": "GDPR",
    "section": "Chapter 2 - Principles"
  }
}
```

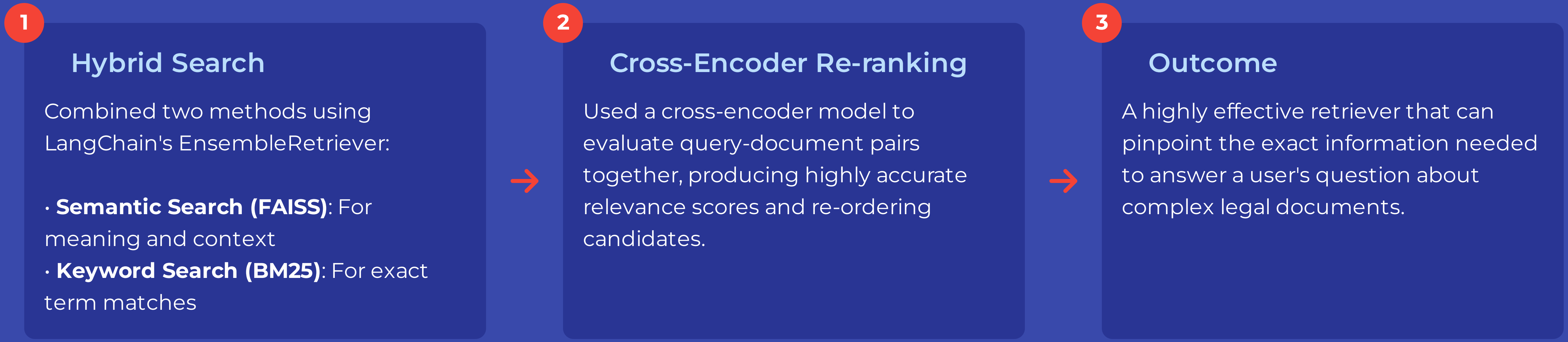
Phase 2 - Creating the 'Brain'

Goal: Build a state-of-the-art retrieval system that can find the most relevant document chunks for any given user query.

Building the Vector Database with FAISS

Used the powerful BAAI/bge-large-en-v1.5 model to convert each text chunk into a high-dimensional vector embedding. These embeddings were stored in a FAISS (Facebook AI Similarity Search) index, creating a local, file-based vector database that is incredibly fast for semantic similarity searches.

The Advanced Retrieval Strategy



1

Hybrid Search

Combined two methods using LangChain's EnsembleRetriever:

· **Semantic Search (FAISS)**: For meaning and context
· **Keyword Search (BM25)**: For exact term matches

2

Cross-Encoder Re-ranking

Used a cross-encoder model to evaluate query-document pairs together, producing highly accurate relevance scores and re-ordering candidates.

3

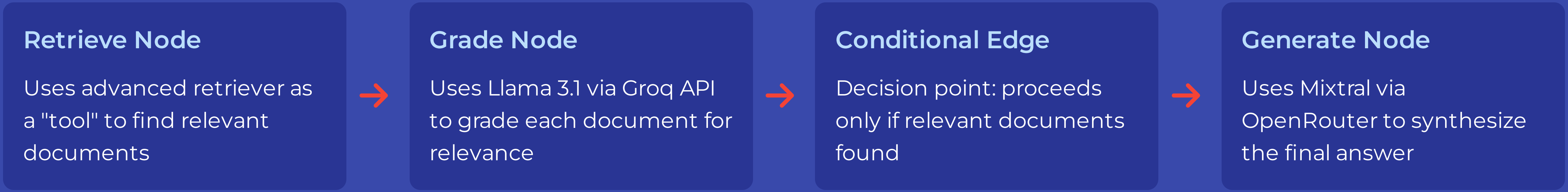
Outcome

A highly effective retriever that can pinpoint the exact information needed to answer a user's question about complex legal documents.

Phase 3 - LangGraph Orchestration & FastAPI Service

The Agentic Workflow with LangGraph

Moving beyond a simple chain to create an intelligent, decision-making agent using LangGraph's cyclical, stateful architecture.



Retrieve Node

Uses advanced retriever as a "tool" to find relevant documents

→

Grade Node

Uses Llama 3.1 via Groq API to grade each document for relevance

→

Conditional Edge

Decision point: proceeds only if relevant documents found

→

Generate Node

Uses Mixtral via OpenRouter to synthesize the final answer

The FastAPI Service

- Wrapped in a FastAPI application with a single `/ask` endpoint that accepts user questions
- Uses a lifespan manager to load models and retriever once on startup for optimal performance
- Creates a clean, scalable API interface that any frontend application can interact with

Phase 4 - Cloud Deployment

Goal: Deploy the full-stack application to the cloud, making it accessible to real users.

RAG API on AWS EC2

1. Launched a t3.large Ubuntu EC2 instance with 50 GB of storage
   Ubuntu · t3.large · 50GB
2. Configured AWS Security Group to allow public access on port 8000
3. Set up Python environment, installed dependencies, and secured API keys
4. Launched FastAPI with Uvicorn as a persistent background service
5. Configured and attached AWS Elastic IP for permanent addressing

MERN App on Render

1. Created separate branches for backend and frontend in the same repository
2. Deployed Backend (Node.js) as a Web Service
   Node.js · Express · MongoDB
3. Configured environment variables for database and API connections
4. Deployed Frontend (React) as a Static Site
   React · Redux · Tailwind CSS
5. Set up environment variables to connect frontend to the live backend

The Final Product & Key Learnings

The result is Cogni-Compliance, a polished and intelligent chat application that solves a real-world problem.

End-to-End RAG: Practical experience building the entire RAG lifecycle, from data processing to evaluation.

Full-Stack Development: Successfully built and connected a MERN stack frontend to a Python AI backend.

Cloud Deployment & DevOps: Hands-on experience deploying multi-part applications to AWS EC2 and Render.

Problem Solving: Successfully navigated numerous real-world challenges, including dependency conflicts and cloud networking issues.

Agentic AI: Moved beyond simple chains to build a more intelligent, decision-making agent with LangGraph.

✉ devaaman8@gmail.com

github.com/AmanDeva/agentic-rag

agentic-rag-frontend-p2ve.onrender.com