AI Document Summarizer & Question-Answering System

RAG Pipeline with Pinecone, Groq LLM, FastAPI, and Next.js

Priyanshu Kumar Singh

Abstract

This project implements a complete Retrieval-Augmented Generation (RAG) system designed to ingest PDF documents, index them using vector embeddings, retrieve relevant information through semantic search, and generate accurate summaries or answers using Groq's Llama 3.3 model. The system includes a FastAPI backend, modular document-processing pipeline, Pinecone vector database, and a modern Next.js frontend for user interaction. The solution is production-ready and suitable for enterprise document intelligence tasks.

1 Introduction

Modern Large Language Models struggle with long documents due to context limitations. Retrieval-Augmented Generation (RAG) solves this problem by combining semantic search with a generative model.

This project builds an end-to-end pipeline that:

- Extracts text from PDFs
- Splits documents into high-quality chunks
- Generates embeddings for semantic search
- Stores and retrieves vectors from Pinecone
- Uses Groq Llama 3.3 for fast Q/A and summarization
- Provides a clean UI through a Next.js frontend

The system is modular, fast, scalable, and suitable for real-world use.

2 System Architecture

2.1 Pipeline Overview

The complete workflow:

1. **Document Loader** – Extracts text from PDF files.
2. **Text Splitter** – Recursive splitting (1500 chars, 200 overlap).
3. **Embedding Generator** – HuggingFace MiniLM (384-dim).
4. **Vector Store** – Pinecone Serverless Index.
5. **Retriever** – Top-k semantic similarity search.
6. **LLM Generation** – Groq Llama 3.3 answers queries.
7. **Summarizer** – TL;DR, medium, and detailed summaries.

2.2 Backend Technologies

- FastAPI (Python)
- Pinecone Vector DB
- Groq Llama 3.3
- HuggingFace MiniLM Embeddings
- LangChain for vector retrieval utilities
- DVC for pipeline tracking

3 Backend Implementation

3.1 FastAPI Endpoints

The backend exposes three main routes:

- **POST /upload** – Upload PDFs, extract text, chunk, and index in Pinecone.
- **GET /ask?query=** – Query the RAG system for answers.
- **POST /summarize** – Generate short, medium, or detailed summaries.

3.2 Pipeline Modules

Each stage is implemented as an independent component:

- **stage_01** – Document loader (single/multiple PDFs)
- **stage_02** – Text splitter
- **stage_03** – Vector indexing (MiniLM + Pinecone)
- **stage_04** – RAG inference
- **stage_05** – Summary and output formatter

4 Frontend Implementation

4.1 Technology Stack

- Next.js 14 (App Router)
- TailwindCSS + shadcn/ui
- Framer Motion animations
- Sonner for toast notifications

4.2 Frontend Pages

1. Upload Page Drag-and-drop PDF uploader with:

- Multi-PDF upload
- Real-time progress
- Error + success toasts

2. Chat Page RAG-based chat interface:

- Two-pane layout (history + chat)
- Message bubbles
- Auto-scroll and typing indicator
- Citation display

3. **Summaries Page** Generates summaries through:

- Textarea input
- Mode selector (short/medium/detailed)

5 DVC Pipeline

DVC tracks four stages:

1. PDF loading
2. Text chunking
3. Embedding + vector indexing
4. RAG inference test

Running the pipeline:

```
dvc repro
```

This ensures reproducibility and modularity.

6 Results

6.1 RAG Accuracy

- High retrieval accuracy due to overlapping chunks
- Fast inference (Groq)
- Clean and coherent responses

6.2 Frontend Performance

- Smooth UX, responsive design
- Real-time chat experience
- Fast uploads and status tracking

7 Conclusion

This project successfully demonstrates a complete full-stack RAG system combining text extraction, semantic search, LLM reasoning, and modern UI. The architecture is modular, scalable, and suitable for enterprise document intelligence applications.

8 Future Enhancements

- User-specific Pinecone namespaces
- Chat session persistence
- Document auto-summarization after upload
- OCR support for scanned PDFs
- Deployment using Docker + cloud hosting

9 Author

Priyanshu Kumar Singh
GitHub: github.com/Priyanshu1303d