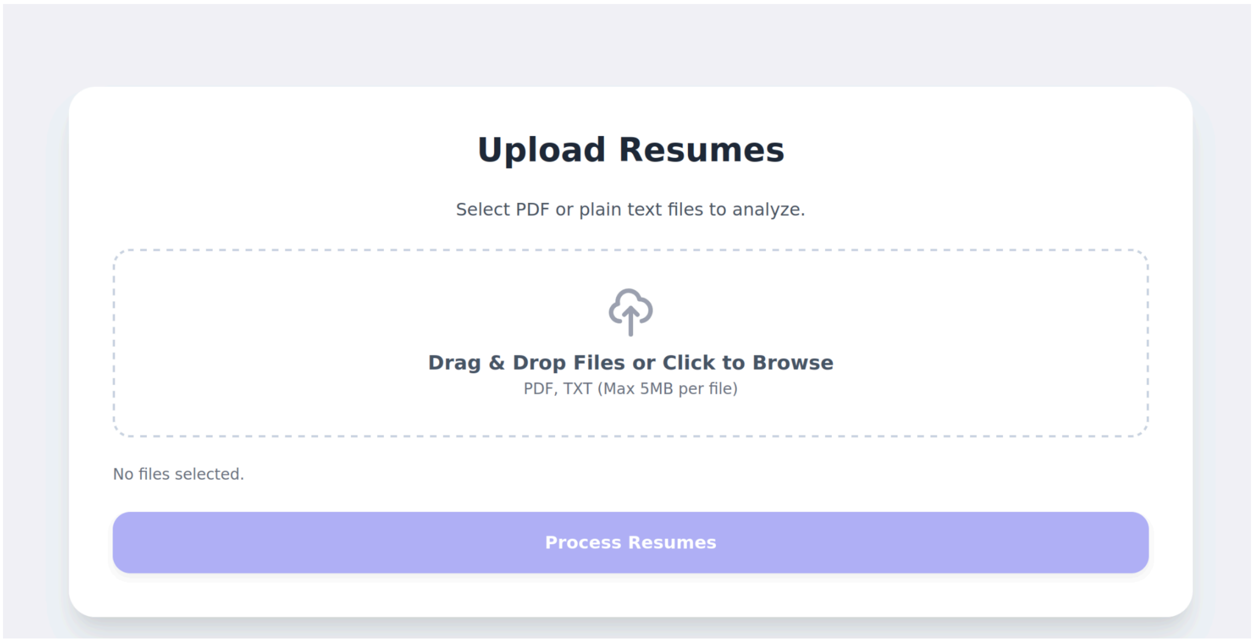
AI Resume Analyzer


Upload Resumes
Select PDF or plain text files to analyze.
Drag & Drop Files or Click to Browse
PDF, TXT (Max 5MB per file)
No files selected.
Process Resumes

Overview

This document outlines the design and development requirements for a Resume Analyzer Chatbot focused on job matching. The core system will ingest and index multiple resume files (PDF/plain text), accept a job description, and leverage AI to identify the most relevant candidates from the indexed pool. It will provide insights and answer queries in a conversational manner. Use of free-tier cloud services and open-source libraries is emphasized to build a robust and cost-effective solution. **Candidates may use an open-source resume dataset (e.g., public academic or community datasets).** A web-based user interface is optional and can be implemented as a bonus feature.

Core Functionality (Required)

1. Resume Ingestion and Processing

- **Index multiple resumes**: Ingest and process multiple resume files (PDF and plain text).
- **Parse content**: Extract text from resumes.
- **Chunk text**: Split parsed text into meaningful chunks (100–500 tokens with overlap).
- **Generate embeddings**: Produce vector embeddings for each chunk using a free-tier model.

    - *Recommended:* Google Gemini (free tier, e.g., 1.5 Flash)
    - *Other:* Hugging Face all-MiniLM-L6-v2

2. Vector Database Integration

- **Store embeddings**: Save generated embeddings in a vector database.
- **Efficient retrieval**: Implement top-K similarity search for queries.
    - *Recommended:* Aiven PostgreSQL with pgvector (free tier)
    - *Other:* Pinecone (free tier)

3. Retrieval-Augmented Generation (RAG) Chatbot

- **Job description input**: Accept a text description of the job role.
- **Retrieve relevant chunks**: Perform vector search to find top resume chunks.
- **LLM-based matching**: Use an LLM to generate conversational answers about candidate fit, citing retrieved content.
- **Conversational interface**: Support follow-up questions for deeper insights.
    - *Recommended LLM:* Google Gemini (free tier)
    - *Other:* Open-source LLM (e.g., LLaMA) running locally

Optional Bonus Features

A. Web User Interface

- **Frontend (Bonus)**: Implement a simple web UI for file upload, job description entry, and chat interaction.
    - *Tech Stack:* Next.js with TypeScript
    - *Hosting:* Vercel (Hobby/Free Tier)

B. SQL-Based Metadata Search

- **Extract metadata**: Tag resumes with structured metadata (skills, titles, experience).
- **Metadata storage**: Save tags in a relational database (e.g., PostgreSQL alongside vector store).
- **Metadata API**: Expose an endpoint for SQL queries (e.g., `SELECT * FROM resume_metadata WHERE skills @> ARRAY['TypeScript'] AND years_experience >= 5`).

Technical Stack & Free-Tier Considerations

Candidates should choose and justify one option per component, demonstrating mitigation of free-tier limits.

- **Backend**: Python (FastAPI/Flask) or TypeScript (Next.js API routes)
    - *Hosting:* Render (Python) or Vercel (Next.js)
- **Object Storage**: Cloudflare R2 or Vercel Blob
- **Orchestration & Processing**: LangChain for loading, splitting, embedding, RAG orchestration, and metadata extraction

---

Deliverables

Required

1. **Backend System**:
    - Batch resume parsing & chunking
    - Embedding generation & storage
    - Vector similarity search for job queries
    - RAG-based response generation
2. **Public GitHub Repository**:
    - README with setup, usage, free-tier notes
    - Clean, modular, documented code

Optional (Bonus)

- **Deployed Web Application**:
    - File upload UI, job description input, chat interface
    - Live URL (e.g., Vercel)
- **Metadata Search API**:
    - Endpoint for structured SQL queries
- **Demo Walkthrough**:
    - Short video or write-up showcasing features

Notes for Candidates: Focus first on delivering the core ingestion, retrieval, and RAG functionality. If time permits, implement the optional web UI and metadata search for extra credit.