Project Report: Smart Resume Screener (AI-Powered Recruitment Assistant)

Name: [Your Name]

Date: November 21, 2025

Challenge: Applied AI Build Challenge

1. Title & Objective

Project Title: Smart Resume Screener: An Automated Candidate Analysis Tool

Objective: The primary objective of this project is to solve the "resume fatigue" problem faced by recruiters and HR professionals. Manually screening hundreds of resumes against a job description is time-consuming and prone to human error or bias.

This project builds a functional AI workflow that automatically parses PDF resumes, compares them against a specific Job Description (JD), and generates a structured "Match Report." The tool aims to provide a percentage match score, identify missing critical skills, and generate custom interview questions, thereby streamlining the initial screening phase of recruitment.

2. Tools & Frameworks Used

- **Core Language:** Python 3.10+
- **User Interface (Frontend):** Streamlit (for rapid web app deployment).
- **LLM Orchestration:** LangChain (for managing prompts and model interaction).
- **AI Model:** OpenAI `gpt-4o-mini` (chosen for high speed and low cost) or Groq `Llama-3` (for low latency).
- **Document Processing:** `PyPDF2` (for extracting raw text from PDF files).
- **Environment:** VS Code, Python Virtual Environment (`venv`).

3. Approach / Workflow Summary

The application follows a Retrieval-Augmented Generation (RAG) style workflow, though simplified for single-document context.

1. **Input:** The user uploads a candidate's Resume (PDF) and pastes the Job Description (Text) into the web interface.
2. **Extraction:** The system extracts raw text from the PDF using `PyPDF2` .
3. **Context Construction:** The extracted resume text and the job description are combined into a robust "System Prompt."
4. **Analysis:** This prompt is sent to the LLM with instructions to act as a Senior Technical Recruiter.
5. **Output:** The LLM returns a structured analysis (Match Score, Missing Keywords, Assessment), which is rendered neatly in the UI.

Workflow Diagram:

User Uploads PDF + JD → Text Extraction (PyPDF2) → Prompt Engineering (Context Window) → LLM Inference → Structured JSON/Text Output → Streamlit Display

4. Key Implementation Steps

A. Setup and PDF Extraction

The first technical challenge was converting binary PDF data into a string format the LLM could understand. I used `PyPDF2` to iterate through pages and concatenate text.

```
# Snippet: Extracting text from PDF
def get_pdf_text(pdf_docs):
    text = ""
    pdf_reader = PdfReader(pdf_docs)
    for page in pdf_reader.pages:
        text += page.extract_text()
    return text
```

B. Prompt Engineering

The quality of the output relied heavily on the system prompt. I used a "Role-Playing" prompting technique to ensure the AI adopted the persona of a strict recruiter. I explicitly requested the output in a specific structure to make it easy to read.

Prompt Strategy: "Act as a skilled ATS (Applicant Tracking System). Evaluate the resume based *only* on the provided Job Description. Assign a match percentage from 0 to 100. List missing keywords."

C. Integrating the LLM

I integrated the model using LangChain's `ChatOpenAI` wrapper. This abstraction allows the model to be easily swapped (e.g., from OpenAI to a local Llama model) without rewriting the core logic.

D. Building the UI with Streamlit

Streamlit was used to make the tool interactive. I used `st.file_uploader` for the resume and `st.text_area` for the Job Description. A "Submit" button triggers the analysis pipeline.

5. Results & Observations

- **Functional Success:** The application successfully accepts PDFs, reads them, and outputs a coherent analysis within 3–5 seconds.
- **Accuracy:** The `gpt-4o-mini` model demonstrated a strong ability to map semantic meaning. For example, if the JD asked for "Version Control" and the resume said "Git," the AI correctly identified it as a match, whereas keyword-based (Ctrl+F) searching would fail.
- **Error Handling:** Early tests showed that complex resume layouts (two columns) sometimes scrambled text extraction. The prompts were adjusted to be more resilient to unstructured text.

6. Learnings & Future Improvements

Key Learnings (Reflection)

1. **Data Cleaning is Crucial:** The raw text extracted from PDFs often contains headers, footers, and page numbers that can confuse the model.
2. **Prompt Sensitivity:** Small changes in the prompt (e.g., asking for "critique" vs. "analysis") radically changed the tone of the output.
3. **Latency vs. Cost:** Using smaller models (`gpt-4o-mini`) provided a perfect balance for this task. Larger models like `gpt-4` were overkill and slower.

Future Improvements

- **Batch Processing:** Allow uploading 50 resumes at once and outputting a CSV ranking them from highest to lowest match.
- **Resume Chat:** Add a chat feature where the recruiter can ask specific questions to the resume (e.g., "Does this candidate have leadership experience?").
- **Local Privacy:** Migrate to a purely local model (using Ollama) to ensure no candidate data ever leaves the secure local environment, addressing privacy concerns.