Technical Specification Document

Project: Resume AI Optimizer – ATS & Job Match Analysis System

Group: QuantumLoops

Course: AML 3303 – Term Project

1. Introduction

Modern employers rely heavily on Applicant Tracking Systems (ATS) to screen resumes long before a human recruiter sees them. As a result, strong candidates are often filtered out simply because their resume does not reflect the specific language or priorities of a given job posting.

The purpose of this project is to build a system that helps job seekers align their resume to the expectations of ATS filters and hiring managers. The system compares a user's resume with a target job description and returns a match score, missing skills, and practical recommendations for improvement.

This technical specification outlines the full design of the Resume AI Optimizer, including its architecture, data design, processing logic, and implementation plan.

2. User Flow

The system is designed around a simple but effective interaction model:

1. The user lands on the web application and sees two main inputs:
    - a text box for the job description
    - a file-upload section for the resume
2. The user copies a job posting from LinkedIn or Indeed and pastes it into the text field.
3. The user uploads a PDF or DOCX version of their resume. The system validates the file type and stores it in an S3 bucket.
4. The backend extracts skills from the job description and compares them to the parsed resume using both keyword matching and semantic similarity.
5. A local LLM provides qualitative feedback, giving the user specific suggestions for strengthening their resume.

6. The user can re-upload multiple versions of their resume to see how their score improves.

This flow intentionally mirrors the way job applicants naturally refine their resumes during a job search.

3. System Architecture

The system consists of four main components:

3.1 Streamlit Frontend

- Handles user input (job description + resume upload)
- Displays match score and missing keywords
- Visualizes results using simple charts
- Maintains session history to track improvements

3.2 FastAPI Backend

- Provides API endpoints for resume analysis
- Manages communication between frontend, storage, NLP modules, and LLM
- Validates inputs and handles errors cleanly

3.3 Storage Layer

- **AWS S3** holds uploaded resume files
- **MongoDB** stores structured analysis results, allowing the system to track multiple submissions per user

3.4 AI/NLP Engine

- Text preprocessing using spaCy
- Keyword extraction using either spaCy or KeyBERT
- Resume parsing using pdfplumber or PyMuPDF
- Semantic similarity using Sentence Transformers
- Local LLM (Llama or Mistral) for higher-level feedback

This architecture ensures clear separation of concerns while remaining lightweight and deployable on a student environment.

4. Data Model

All analysis results are stored in the MongoDB collection `resume_reviews`.

A typical record contains:

```json
{
  "_id": "UUID",
  "timestamp": "2025-01-20T19:45:00Z",

  "s3_file_url": "https://s3.amazonaws.com/resume-ai/uploads/resume_123.pdf",

  "job_description": {
    "raw_text": "We are looking for a Python Engineer...",
    "extracted_keywords": ["Python", "Django", "AWS"]
  },

  "parsed_resume": {
    "skills": ["Python", "Flask"],
    "experience_years": 3,
    "education": "BS Computer Science"
  },

  "analysis_result": {
    "match_score": 65,
    "missing_keywords": ["Django", "AWS"],
    "feedback_summary": "Strong coding foundation, but lacks cloud experience required for this role."
  }
}
```

5. Methodology

5.1 Job Description Processing

The job description is cleaned and tokenized to isolate the elements that matter most. This includes:

- lowercasing
- stopword removal
- lemmatization
- extracting technical keywords (tools, frameworks, certifications)
- identifying experience requirements (“3+ years”, “senior”, “entry level”)

The goal is to reduce noise and extract the expectations that truly influence whether a resume passes an ATS filter.

5.2 Resume Parsing

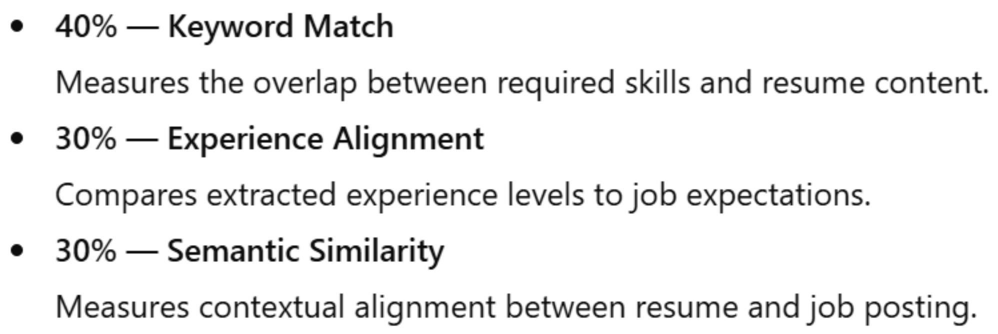
Because resumes have inconsistent layouts, the system uses libraries that handle multi-column and stylized text effectively. Extracted elements include:

- skills section
- experience bullet points
- employment dates
- education
- measurable achievements

If extraction fails, the system alerts the user rather than producing unreliable results.

5.3 Scoring Model

The match score is based on a weighted formula:



- **40% — Keyword Match**

  Measures the overlap between required skills and resume content.
- **30% — Experience Alignment**

  Compares extracted experience levels to job expectations.
- **30% — Semantic Similarity**

  Measures contextual alignment between resume and job posting.

This structure balances exact-match criteria with semantic understanding.

5.4 LLM Feedback

The local LLM provides critique and suggestions contextualized to the job. Example improvements:

- “Include metrics in your bullet points”
- “Mention frameworks relevant to this posting, such as Django”
- “Improve formatting for ATS readability”

The LLM is intentionally restricted to avoid hallucination.

6. Implementation Plan

Week 1 — Core Setup

- Create FastAPI service
- Integrate S3 uploads and validate file inputs
- Set up MongoDB collections
- Build initial Streamlit interface

Week 2 — AI and Scoring Logic

- Implement JD processing pipeline
- Build resume parser and text extractor
- Develop scoring algorithm
- Integrate semantic similarity and LLM feedback

Week 3 — Refinement

- Add UI enhancements (charts, progress bars)
- Improve error handling for bad files or missing fields
- Conduct user testing
- Finalize presentation and documentation

Tasks are structured to allow iterative testing and incremental refinement.

7. Risks & Mitigation

Risk	Impact	Mitigation
Slow LLM performance	High	Use quantized models and cache repeated analyses
PDF parsing failures	High	Use PyMuPDF and fallback to Tika if needed
Excess hallucination	Medium	Restrict prompts to “resume improvement only”
Storage growth	Low	Add S3 lifecycle rules to delete old files

8. Technology Stack

- **Python 3.10**
- **Streamlit** for the UI
- **FastAPI** for backend services
- **AWS S3** for file storage
- **MongoDB** for structured metadata
- **spaCy / KeyBERT** for keyword extraction
- **Sentence Transformers** for semantic similarity
- **Llama/Mistral (quantized)** for feedback generation
- **pdfplumber / PyMuPDF** for resume parsing

This stack balances accuracy, speed, and ease of deployment.

9. Conclusion

The Resume AI Optimizer provides job applicants with targeted, data-driven insight to improve their resumes for a specific job posting. The system combines NLP, semantic search, and LLM-based feedback into a simple, intuitive workflow.

The design prioritises reliability, transparency, and user control, ensuring that the final recommendations are both practical and grounded in the content of the job posting.