CV Sorting using LLMs

Objective: Design and develop a capstone project on "CV Sorting using LLMs", to enable automated ranking of candidate resumes for specific job requirements using large language models.

This guide outlines preferred open-source models, an implementation roadmap, and rigorous evaluation strategies.

Required Models

- **Resume Matcher**: An open-source, free tool leveraging LLMs to compare and rank resumes with job descriptions.
- **Llama/LlamaIndex**: Versatile open-source frameworks suitable for semantic candidate-job matching.
- **LangChain**: Powerful orchestrator for prompt-based matching, score aggregation, and structured output parsing.
- **pyresparser / open-resume**: Utilized for efficient information extraction from PDF/Word resume formats.
- **Smaller LLMs via Ollama (e.g., Gemma, Llama 3 8B)**: Ideal for privacy-focused and local deployment scenarios.

Step by Step Instructions

1. Project Setup

- Select and configure one or more open-source LLMs (e.g., via Ollama, LlamaIndex, or Resume Matcher).
- Set up the development environment with necessary Python libraries (e.g., Transformers, LangChain, pyresparser).
- Prepare infrastructure for handling resume uploads and storage (e.g., file directory or simple database).

2. Resume and Job Description Parsing

- Implement parsing pipelines to convert uploaded candidate resumes (PDF/Word) into structured data objects (using pyresparser or open-resume).
- Extract job requirements, skills, responsibilities, and keywords from job descriptions using LLM-based prompt engineering to ensure contextual understanding and normalization.

3. Candidate-Job Matching

- For each resume, utilize the LLM to assess the alignment between candidate experience, skills, and job-specific criteria.
- Develop a prompt templating system that:
    - Takes as input the structured resume data and the parsed job description.
    - Asks the LLM to score the match for each requirement and generate an overall fit score with an explanation.

- Collect results into a ranked list with detailed per-candidate feedback.

4. Intelligent Ranking and User Interface

- Aggregate match scores to determine candidate order, and display results in an intuitive dashboard or exportable file.
- Allow users (e.g., recruiters) to inspect match explanations and supporting evidence for transparency.
- Support interactive query refinement where users can adjust job criteria or filter by required skills and re-run sorting in real time.

Evaluation Criteria

1. Report

- **Scope**: High-level goals, model selection process, pipeline architecture, and design rationale.
- **Analysis**: Examination of ranking accuracy, alignment with recruiter expectations, and ability to surface top candidates for diverse roles.
- **Metrics**: Use accuracy, recall, precision, and mean average precision to quantify sorting performance. Analyze comparison of different models, template quality, and information extraction robustness.

2. Presentation

- **Slides**: Concisely outline background, technical approach, the stepwise pipeline, and experimental results.
- **Demo**: Show a live or video demonstration: upload sample resumes and a job post, trigger sorting, and analyze the ranked output.
- **Highlights**: Emphasize improvements over keyword-based ATS, semantic understanding, and interpretability of LLM scores.

3. Project Code Submission

- **Repository**: Well-documented codebase including:
    - Source code for the parsing, matching, and ranking pipeline.
    - Infrastructure scripts for environment setup and deployment (e.g., Docker, environment.yml).
    - Sample datasets (candidate resumes, job descriptions) and example results.
    - README with installation and usage instructions, documentation on custom prompts, and troubleshooting.