**Intelligent Resume Skill Analyzer**

An advanced, multi-layered AI tool that provides a deep, semantic analysis of a candidate's resume against a specific job description. This project moves beyond simple keyword matching to understand context, infer potential, and deliver actionable insights for both candidates and HR professionals.

Overview

Traditional resume screeners often fail because they rely on rigid keyword matching, leading them to reject high-potential candidates who don't use the exact terminology from a job description. This tool was built to solve that problem by creating a system that thinks more like an expert recruiter.

The application accepts a resume (PDF) and a job description, and in return, it generates:

* A multi-faceted match score (Skills, Projects, Experience, Qualifications).
* A detailed skill-gap analysis showing exact, related, and missing skills.
* An AI-generated report with tailored feedback for either the candidate or a hiring manager.

**Key Features**

* Tiered Skill Matching: Doesn't just give a binary match. It identifies Exact Matches and Closely Related "Sibling" Skills, providing partial credit for conceptual alignment (e.g., understands XGBoost is related to Scikit-learn).
* Semantic Scoring for Context: Uses three different Transformer embedding models (SBERT, CodeBERT, GraphCodeBERT) to analyze the meaning of unstructured text in Project and Experience sections, rewarding demonstrated application of skills.
* Dual-Persona AI Reporting: Generates two distinct reports based on user type—constructive, coaching feedback for Candidates and an objective, analytical summary for HR.
* Project & Experience Demonstration Analysis: Intelligently distinguishes between skills a candidate simply lists versus skills they have actively demonstrated in their project or work descriptions.
* Domain-Agnostic & Customizable: The core logic is separate from the domain knowledge. The entire system can be adapted for any industry (Finance, Medical, etc.) simply by editing the skills\_ontology.json file.
* GPU Accelerated: The code automatically detects and utilizes an available GPU (via CUDA-enabled PyTorch) to dramatically speed up the analysis.

**Tech Stack**

* Backend: Python, Flask
* ML/NLP: PyTorch, Hugging Face Transformers, Sentence-Transformers, Scikit-learn
* Text Extraction: PyMuPDF (fitz), Tesseract (pytesseract) for OCR
* Generative AI: OpenAI (GPT-4o-mini)
* Frontend: HTML, Bootstrap

**Major Difficulties & How We Solved Them**

1. Inaccurate and Brittle Skill Extraction

* Problem: The initial approach of simple keyword searching was highly unreliable. It would miss key skills, incorrectly flag others (e.g., mistaking the letter "r" for the R programming language), and failed to understand the relationship between terms.
* Solution: We implemented a robust, ontology-driven system. By creating a skills\_ontology.json file to act as a central knowledge base, we could define all relevant skills and their relationships. This, combined with regex using word boundaries (\b), eliminated most false positives and allowed the system to identify skills with high precision.

2. Differentiating Conceptual Similarity

* Problem: After establishing the ontology, the matching logic was too broad. For instance, it would treat any skill in the "Cloud & MLOps" category as a valid substitute for any other, incorrectly equating knowledge of AWS with Docker.
* Solution: The breakthrough was to refine the ontology with more specific categories. We broke down the overly broad "Cloud & MLOps" group into smaller, more precise categories like Cloud Platforms and Containerization & Orchestration. This allowed our existing "sibling skill" matching code to work with much greater nuance, correctly identifying that AWS is not a substitute for Docker but that XGBoost is a valid sibling for Scikit-learn.

3. Parsing Diverse Resume Formats

* Problem: Resumes have no standard format. Our initial code could not reliably find sections like "Projects" or "Education" if the heading was slightly different (e.g., "Academic Background").
* Solution: We built a flexible, synonym-based section extractor. We created a dictionary of common synonyms for each major resume section and a dynamic function that searches for any of these synonyms to reliably find and isolate the correct block of text, regardless of the resume's specific layout.

4. Slow Performance on Local Machines

* Problem: The three Transformer embedding models are computationally heavy, and running the analysis on a CPU was unacceptably slow.
* Solution: We enabled GPU acceleration. The code was updated to automatically detect if a CUDA-enabled GPU is present (torch.cuda.is\_available()). The embed\_text function was refactored to move both the models and the data tensors to the GPU for processing, resulting in a dramatic performance increase.

5. Generic and Unprofessional AI Reports

* Problem: The initial reports generated by the LLM were generic, had an unprofessional "letter" format ("Dear Candidate..."), and included ugly markdown formatting (\*\*, ###).
* Solution: We engineered highly specific, dual-persona prompts. We created two distinct prompt templates—one for a "Career Coach" persona (for candidates) and one for a "Technical Analyst" persona (for HR). By providing the AI with a rich, structured set of data (exact matches, sibling matches, gaps, etc.) and strict instructions, we were able to generate high-quality, tailored feedback. Finally, we added a post-processing step using the markdown library to convert the AI's output into clean HTML for the frontend.

**2. Ontology Redundancy & Noise**

* **Problem:** The skills\_ontology.json had **overlapping synonyms, redundant entries** (e.g., SQL vs database vs queries), which caused inflated or incorrect missing skills.
* **Solution:**
  + Cleaned and consolidated ontology to remove duplicates and near-duplicates.
  + Kept unique, meaningful categories (e.g., "SQL" only, instead of SQL + queries + database).
  + Balanced ontology (not too short, not too redundant).

**4. Scoring Inconsistencies**

* **Problem:** Scores for the same resume–JD pair fluctuated, sometimes marking core skills as missing.
* **Solution:**
  + Identified that **embedding similarity & ontology overlap** needed consistency.
  + Standardized pipeline: first extract → normalize with ontology → compare embeddings → score.
  + Reduced reliance on GPT randomness by keeping deterministic regex/ontology for skill detection.

**5. Resume vs Project Skill Mismatch**

* **Problem:** A skill listed under *skills* section was marked missing in *projects* section (e.g., “SQL” in skills but missing in projects).
* **Solution:**
  + Split evaluation into **Resume Skills** and **Project Skills**.
  + Ensured project parsing wasn’t dropping skills due to formatting/line breaks.
  + Deduplicated overlapping skills between resume & project.

**6. Frontend & Dashboard Clarity**

* **Problem:** Dashboard initially showed **redundant & repeated skills**, looked cluttered and confusing.
* **Solution:**
  + Added cleaning logic before rendering (unique set of missing skills).
  + Improved formatting in Jinja templates (lists → clean bulleted points).
  + Separated **resume vs project scores** for better insights.