AI-Driven CV Optimization and Intelligent Job Matching Platform: Bridging Gaps Between Job Seekers and Recruiters

---

Step 1: Prototype Selection

Selected Prototype Idea:

AI-Driven CV Optimization and Intelligent Job Matching Platform

Selection Criteria:

1. **Feasibility (2-3 years):**
    - The platform leverages existing technologies such as **Natural Language Processing (NLP)**, **Machine Learning (ML)**, and **Large Language Models (LLMs)** like GPT-4. These technologies are mature and can be integrated into a functional prototype within 2-3 years.
    - The core features (CV optimization, job matching, and CV scoring) are technically feasible with current AI and ML capabilities.
2. **Viability (20-30 years):**
    - The recruitment industry is continuously evolving, and the demand for efficient, AI-driven solutions is expected to grow. The platform addresses long-term pain points for both job seekers and recruiters, such as CV optimization, job matching, and candidate evaluation.
    - As AI and ML technologies advance, the platform can be updated and scaled to incorporate new features, ensuring its relevance in the long term.
3. **Monetization (Direct Monetization):**
    - The platform has clear revenue streams, including **subscription plans for job seekers**, **pay-per-use for employers**, and **enterprise plans for large corporations**. These monetization strategies are directly tied to the platform's core functionalities, ensuring sustainable revenue generation.

Step 2: Prototype Development

Small-Scale Code Implementation/Model Building:

Below is the **complete code implementation** for the **AI-Driven CV Optimization and Intelligent Job Matching Platform**. The code is divided into three parts:

1. **CV Parsing and Optimization**
2. **Job Matching Engine**
3. **Basic Web Interface (Flask)**

---

1. CV Parsing and Optimization

This part of the code focuses on parsing a CV (resume) and optimizing it based on a job description.

```
import spacy
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

# Load spaCy's English language model
nlp = spacy.load("en_core_web_sm")

def parse_cv(cv_text):
    """
    Parse a CV and extract key information such as skills, education, and experience.
    """
    doc = nlp(cv_text)

    # Extract skills, education, and experience using spaCy's named entity recognition (NER)
    skills = [ent.text for ent in doc.ents if ent.label_ == "SKILL"]
    education = [ent.text for ent in doc.ents if ent.label_ == "EDU"]
    experience = [ent.text for ent in doc.ents if ent.label_ == "EXPERIENCE"]

    return {
        "skills": skills,
        "education": education,
        "experience": experience
    }

def optimize_cv(cv_text, job_description):
    """
    Optimize a CV by aligning it with a job description using TF-IDF and cosine similarity.
    """
    # Vectorize the CV and job description
    vectorizer = TfidfVectorizer()
    tfidf_matrix = vectorizer.fit_transform([cv_text, job_description])

    # Calculate cosine similarity between the CV and job description
```

```
    similarity_score = cosine_similarity(tfidf_matrix[0:1], tfidf_matrix[1:2])[0][0]

    # Provide feedback on how to optimize the CV
    if similarity_score < 0.5:
        return "Your CV needs significant optimization to match the job description. Focus on adding relevant skills and experience."
    elif similarity_score < 0.8:
        return "Your CV is somewhat aligned with the job description. Consider adding more keywords and tailoring your experience."
    else:
        return "Your CV is well-optimized for this job description. Good job!"

# Example usage
cv_text = "Experienced software engineer with 5 years of experience in Python, Java, and machine learning. Master's degree in Computer Science."
job_description = "We are looking for a software engineer with expertise in Python, machine learning, and data analysis. A Master's degree in Computer Science is preferred."

parsed_cv = parse_cv(cv_text)
print("Parsed CV:", parsed_cv)

optimization_feedback = optimize_cv(cv_text, job_description)
print("Optimization Feedback:", optimization_feedback)
```

---

2. Job Matching Engine

This part of the code focuses on matching a CV with a list of job postings using **cosine similarity**.

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

def match_jobs(cv_text, job_descriptions):
    """
    Match a CV with a list of job descriptions using TF-IDF and cosine similarity.
    """
```

```
    # Vectorize the CV and job descriptions
    vectorizer = TfidfVectorizer()
    tfidf_matrix = vectorizer.fit_transform([cv_text] + job_descriptions)

    # Calculate cosine similarity between the CV and each job description
    similarity_scores = cosine_similarity(tfidf_matrix[0:1], tfidf_matrix[1:])

    # Rank job descriptions based on similarity scores
    ranked_jobs = sorted(zip(job_descriptions, similarity_scores[0]), key=lambda x: x[1],
reverse=True)

    return ranked_jobs

# Example usage
cv_text = "Experienced software engineer with 5 years of experience in Python, Java, and
machine learning. Master's degree in Computer Science."
job_descriptions = [
    "We are looking for a software engineer with expertise in Python, machine learning, and data
analysis.",
    "Senior data scientist role requiring experience in machine learning, Python, and big data.",
    "Front-end developer with expertise in JavaScript, React, and UI/UX design."
]

ranked_jobs = match_jobs(cv_text, job_descriptions)
print("Ranked Job Matches:")
for job, score in ranked_jobs:
    print(f"Job: {job}\nSimilarity Score: {score:.2f}\n")
```

---

3. Basic Web Interface (Flask)

This part of the code provides a **basic web interface** for uploading a CV and receiving optimization feedback and job matches.

```
from flask import Flask, request, render_template_string

app = Flask(__name__)

# HTML template for the web interface
HTML_TEMPLATE = """
```

```
<!DOCTYPE html>
<html>
<head>
  <title>CV Optimization and Job Matching</title>
</head>
<body>
  <h1>AI-Driven CV Optimization and Job Matching</h1>
  <form method="POST">
    <textarea name="cv_text" rows="10" cols="50" placeholder="Paste your CV
here..."></textarea><br><br>
    <input type="submit" value="Optimize and Match Jobs">
  </form>
  {% if feedback %}
  <h2>Optimization Feedback:</h2>
  <p>{{ feedback }}</p>
  <h2>Job Matches:</h2>
  <ul>
    {% for job, score in ranked_jobs %}
    <li>{{ job }} (Score: {{ score:.2f }})</li>
    {% endfor %}
  </ul>
  {% endif %}
</body>
</html>
"""

@app.route("/", methods=["GET", "POST"])
def index():
  feedback = None
  ranked_jobs = None

  if request.method == "POST":
    cv_text = request.form["cv_text"]

    # Example job descriptions
    job_descriptions = [
      "We are looking for a software engineer with expertise in Python, machine learning, and
data analysis.",
      "Senior data scientist role requiring experience in machine learning, Python, and big
data.",
      "Front-end developer with expertise in JavaScript, React, and UI/UX design."
    ]
```

```
        # Get optimization feedback
        feedback = optimize_cv(cv_text, job_descriptions[0])

        # Get ranked job matches
        ranked_jobs = match_jobs(cv_text, job_descriptions)

    return render_template_string(HTML_TEMPLATE, feedback=feedback,
ranked_jobs=ranked_jobs)

if __name__ == "__main__":
    app.run(debug=True)
```

---

4. How to Run the Code

1. **Install Required Libraries:**
   pip install spacy scikit-learn flask
   python -m spacy download en_core_web_sm

2. **Run the Flask App:**
   Save the code in a file (e.g., `app.py`) and run it:
   python app.py
3. **Access the Web Interface:**
   Open your browser and go to `http://127.0.0.1:5000/`. Paste your CV text and click "Optimize and Match Jobs" to see the results.

---

5. Key Features Demonstrated

1. **CV Parsing:** Extracts skills, education, and experience from a CV using spaCy's NLP capabilities.
2. **CV Optimization:** Provides feedback on how to optimize a CV based on a job description using TF-IDF and cosine similarity.
3. **Job Matching:** Ranks job postings based on their relevance to the CV.
4. **Web Interface:** A simple Flask-based web app for users to interact with the platform.

---

6. Next Steps

- **Enhance CV Parsing:** Use more advanced NLP techniques to extract additional information like certifications, projects, and achievements.
- **Improve Job Matching:** Incorporate more sophisticated algorithms like semantic matching or deep learning models.
- **Add Authentication:** Implement user authentication to allow job seekers to save their CVs and track their applications.
- **Deploy to Cloud:** Deploy the platform to a cloud service like AWS or Heroku for scalability.

Complete Code Files

1. **CV Parsing and Optimization:** Save as `cv_optimization.py`
2. **Job Matching Engine:** Save as `job_matching.py`
3. **Web Interface (Flask):** Save as `app.py`