Resume-Matcher-AI

Alhaan

2025-05-06 19:14:03

Contents

Project Overview

No project description provided.

Codebase Statistics

- **Total files:** 18
- **Total lines of code:** 1435
- **Languages used:** python (18)
- **Generated by:** code2pdf v0.2

Directory Structure

```
resume-matcher-ai/
    ├── api_testing/
        └── cerebras_api_testing.py
        └── deepseek_api_testing.py
        └── gemini_api_testing.py
    ├── app/
        └── alt_backend.py
        └── main_backend.py
        └── temp_frontend.py
    ├── core/
        └── ats_scoring.py
        └── job_description.py
        └── job_role_matcher.py
        └── resume_feedback.py
        └── resume_skill_extractor.py
    ├── experiment/
        └── model_comparator.py
    ├── resume_generator/
        └── resume_gen_latex_1.py
        └── resume_gen_latex_2.py
        └── resume_gen_trial_1.py
        └── resume_sample_data.py
        ├── gen_resume/
    ├── resume_generator_test/
        ├── gen_resume/
    ├── services/
```

```
        └── aws_handler.py
    ├── utils/
        └── resume_parser.py
```

D:/Programming/resume-matcher-ai/api_testing/cerebras_api_testing.py

```python
import os
from cerebras.cloud.sdk import Cerebras

from dotenv import load_dotenv

load_dotenv()
# Set up the Cerebras client
client = Cerebras(
    api_key=os.getenv("CEREBRAS_API_KEY")  # Make sure the API key is set in your environment
)

# Create a simple test function to interact with the API
def test_cerebras_api():
    # Define the test prompt for resume generation
    test_prompt = [
        {
            "role": "system",
            "content": "You are an assistant for generating LaTeX-based resumes."
        },
        {
            "role": "user",
            "content": "Write me a resume generator based on job description using LaTeX code."
        }
    ]

    # Make the API call to generate the response
    stream = client.chat.completions.create(
        messages=test_prompt,
        model="llama3.1-8b",  # Use the model you need (adjust as per your actual model)
        stream=True,
        max_completion_tokens=2048,
        temperature=0.5,  # Adjust temperature if you want more or less randomness
```

```
        top_p=1
    )

    # Process and print the response stream
    print("Generating LaTeX resume:")
    for chunk in stream:
        print(chunk.choices[0].delta.content or "", end="")

# Call the function to test the API
if __name__ == "__main__":
    test_cerebras_api()
```

D:/Programming/resume-matcher-ai/api_testing/deepseek_api_testing.py

```
import os
from huggingface_hub import InferenceClient
from dotenv import load_dotenv

load_dotenv()

messages = [
    {"role": "user", "content": "What is the capital of France?"}
]

def test_provider(name, model_name, api_key=None):
    print(f"\nTesting provider: {name}")
    try:
        client = InferenceClient(provider=name, api_key=api_key)
        response = client.chat.completions.create(
            model=model_name,
            messages=messages,
            max_tokens=100
        )
        print(f"✅ {name} response: {response.choices[0].message['content']}")
    except Exception as e:
        print(f"❌ {name} failed: {e}")

if __name__ == "__main__":
    test_provider("together", "deepseek-ai/DeepSeek-R1",
    ↪ os.getenv("TOGETHER_DEEPSEEK_KEY"))
    test_provider("sambanova", "deepseek-ai/DeepSeek-R1",
    ↪ os.getenv("SAMBANOVA_DEEPSEEK_KEY"))
```

D:/Programming/resume-matcher-ai/api_testing/gemini_api_testing.py

```python
from google import genai
from dotenv import load_dotenv
import os

load_dotenv()


client = genai.Client(api_key=os.getenv('GEMINI_API_KEY'))

response = client.models.generate_content(
    model="gemini-2.0-flash",
    contents="Explain how AI works",
)

print(response.text)
```

D:/Programming/resume-matcher-ai/app/alt_backend.py

```python
import os
import sys
import uuid
import sqlite3
from flask import Flask, request, jsonify, send_file

# Ensure modules from other directories are accessible
sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), "..")))

# Import core functionalities
from utils.resume_parser import extract_text_from_pdf
from core.ats_scoring import compute_sbert_similarity
from core.resume_feedback import generate_resume_suggestions
from core.job_role_matcher import match_resume_to_job_role
from core.resume_skill_extractor import extract_resume_skills
from core.job_description import JOB_DESCRIPTIONS

# Initialize Flask app
app = Flask(__name__)

# Directory to store resumes locally
UPLOAD_FOLDER = "uploaded_resumes"
```

```
os.makedirs(UPLOAD_FOLDER, exist_ok=True)

# SQLite Database Setup
DB_FILE = "resume_metadata.db"

def init_db():
    """Initializes the SQLite database for storing resume metadata."""
    with sqlite3.connect(DB_FILE) as conn:
        cursor = conn.cursor()
        cursor.execute('''CREATE TABLE IF NOT EXISTS resumes (
                            resume_id TEXT PRIMARY KEY,
                            original_filename TEXT,
                            file_path TEXT,
                            ats_score REAL,
                            job_role TEXT,
                            skills TEXT
                        )''')
        conn.commit()

# Initialize the database
init_db()

@app.route("/upload", methods=["POST"])
def upload_resume():
    """Uploads resume, extracts text, computes ATS score, and stores metadata
    ↪ locally."""
    try:
        file = request.files["resume"]
        if not file:
            return jsonify({"error": "No file received"}), 400

        job_description = request.form.get("job_description", "").strip()

        # Extract text & skills from resume
        resume_text = extract_text_from_pdf(file)
        resume_skills = extract_resume_skills(resume_text)

        # Determine job role
        detected_job_role = match_resume_to_job_role(resume_skills)

        # If no job description provided, use a default one based on detected job
        ↪ role
        if not job_description:
```

```
            job_description = JOB_DESCRIPTIONS.get(detected_job_role, "Looking for
↪  a skilled software engineer.")
        print("🔍 Final Job Description:", job_description)  # Debug print

        # Generate unique resume ID & save locally
        resume_id = str(uuid.uuid4())
        unique_filename = f"{resume_id}.pdf"
        file_path = os.path.join(UPLOAD_FOLDER, unique_filename)
        file.save(file_path)

        # Compute ATS Score
        ats_score = compute_sbert_similarity(resume_text, job_description)
        suggestions = generate_resume_suggestions(detected_job_role, resume_skills,
↪  resume_text, ats_score)

        # Store metadata in SQLite
        with sqlite3.connect(DB_FILE) as conn:
            cursor = conn.cursor()
            cursor.execute("INSERT INTO resumes (resume_id, original_filename,
↪  file_path, ats_score, job_role, skills) VALUES (?, ?, ?, ?, ?, ?)",
                           (resume_id, file.filename, file_path, ats_score,
↪  detected_job_role, ",".join(resume_skills)))
            conn.commit()

        return jsonify({
            "message": "Upload successful!",
            "resume_id": resume_id,
            "job_role": detected_job_role,
            "skills": resume_skills,
            "ats_score": ats_score,
            "suggestions": suggestions,
            "file_path": file_path
        })

    except Exception as e:
        print(f"ERROR: {str(e)}")  # Log error
        return jsonify({"error": str(e)}), 500

@app.route("/view/<resume_id>")
def view_pdf(resume_id):
    """Fetches a PDF file stored locally using its unique resume_id."""
    try:
        with sqlite3.connect(DB_FILE) as conn:
```

```
            cursor = conn.cursor()
            cursor.execute("SELECT file_path FROM resumes WHERE resume_id = ?",
↪   (resume_id,))
            result = cursor.fetchone()

        if not result:
            return jsonify({"error": "Resume not found"}), 404

        return send_file(result[0], as_attachment=True)  # Serve the file for
        ↪  download

    except Exception as e:
        return jsonify({"error": str(e)}), 500

@app.route("/metadata", methods=["GET"])
def get_resumes():
    """Fetches all stored resume metadata from SQLite."""
    try:
        with sqlite3.connect(DB_FILE) as conn:
            cursor = conn.cursor()
            cursor.execute("SELECT resume_id, original_filename, ats_score,
↪   job_role, skills FROM resumes")
            resumes = [{"resume_id": row[0], "original_filename": row[1],
↪   "ats_score": row[2], "job_role": row[3], "skills": row[4].split(",")} for row
↪   in cursor.fetchall()]

        return jsonify({"resumes": resumes})

    except Exception as e:
        return jsonify({"error": str(e)}), 500

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

D:/Programming/resume-matcher-ai/app/main_backend.py

```
# AWS is included here
# Main backend for my project

import os
import sys
sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), "..")))
```

```
from flask import Flask, request, jsonify
import uuid
from utils.resume_parser import extract_text_from_pdf
from core.ats_scoring import compute_sbert_similarity
from core.resume_feedback import generate_resume_suggestions
from services.aws_handler import upload_to_s3, store_resume_metadata,
 ↪  get_resume_by_id, get_all_resumes
from core.job_role_matcher import match_resume_to_job_role
from core.resume_skill_extractor import extract_resume_skills
from core.job_description import JOB_DESCRIPTIONS  # Import predefined job
 ↪  descriptions


app = Flask(__name__)

@app.route("/upload", methods=["POST"])
def upload_resume():
    """Uploads resume, extracts text, computes ATS score, and stores metadata."""
    try:
        file = request.files["resume"]
        if not file:
            return jsonify({"error": "No file received"}), 400

        job_description = request.form.get("job_description", "").strip()
        if not job_description:
            detected_job_role =
 ↪  match_resume_to_job_role(extract_resume_skills(extract_text_from_pdf(file)))
            job_description = JOB_DESCRIPTIONS.get(detected_job_role, "Looking for
 ↪  a skilled software engineer.")

        original_filename = file.filename

        resume_id = str(uuid.uuid4())
        unique_filename = f"{resume_id}.pdf"


        resume_text = extract_text_from_pdf(file)
        resume_skills = extract_resume_skills(resume_text)
        detected_job_role = match_resume_to_job_role(resume_skills)

        # Add the job role and job description descripency

        # this is where we obtain our ats score from our function
```

```
        ats_score = compute_sbert_similarity(resume_text, job_description)
        suggestions = generate_resume_suggestions(detected_job_role, resume_skills,
↪   resume_text, ats_score)


        # Upload file to S3 database aws which ive integrated
        file_url = upload_to_s3(file, unique_filename)

        # Store metadata in DynamoDB
        store_resume_metadata(resume_id, original_filename, file_url, ats_score)

        return jsonify({
            "message": "Upload successful!",
            "resume_id": resume_id,
            "job_role": detected_job_role,
            "skills": resume_skills,
            "ats_score": ats_score,
            "suggestions": suggestions,
            "file_url": file_url
        })

    except Exception as e:
        print(f"ERROR: {str(e)}")  # Print full error in logs
        return jsonify({"error": str(e)}), 500

@app.route("/view/<resume_id>")
def view_pdf(resume_id):
    """Fetches a PDF from S3 using its unique resume_id and returns its URL."""
    try:
        resume_metadata = get_resume_by_id(resume_id)
        if not resume_metadata:
            return jsonify({"error": "Resume not found"}), 404

        return jsonify({"file_url": resume_metadata["file_url"]})

    except Exception as e:
        return jsonify({"error": str(e)}), 500

@app.route("/metadata", methods=["GET"])
def get_resumes():
    """Fetches all stored resume metadata from DynamoDB."""
    try:
        resumes = get_all_resumes()
```

```
        return jsonify({"resumes": resumes})

    except Exception as e:
        return jsonify({"error": str(e)}), 500

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

D:/Programming/resume-matcher-ai/app/temp_frontend.py

```
import streamlit as st
import requests
import pdfplumber
import os
from datetime import datetime

st.set_page_config(page_title="Resume ATS Checker", layout="wide")

st.markdown(
    """
    <style>
        /* Button Styling */
        .stButton>button {
            width: 100%;
            padding: 12px;
            font-size: 16px;
            font-weight: bold;
            border-radius: 8px;
            transition: 0.3s;
        }

        /* Success Message */
        .success-box {
            background-color: #1E392A;
            color: white;
            padding: 15px;
            border-radius: 8px;
            text-align: center;
            font-weight: bold;
        }

        /* Grid layout for results */
        .result-container {
```

```
    display: flex;
    justify-content: space-between;
    gap: 10px;
    margin-top: 10px;
}

.result-box {
    flex: 1;
    padding: 15px;
    border-radius: 8px;
    text-align: center;
    font-weight: bold;
}

.job-role { background-color: #4CAF50; color: white; }
.extracted-skills { background-color: #2196F3; color: white; }
.ats-score { background-color: #FF9800; color: white; }

/* Improvement Suggestions */
.improvement {
    padding: 15px;
    border-radius: 8px;
    background-color: #222;
    color: white;
    margin-top: 10px;
}

/* Resume Metadata */
.resume-metadata {
    background-color: #333;
    color: white;
    padding: 10px;
    border-radius: 8px;
}

/* Resume Preview */
.resume-preview {
    background-color: #f5f5f5;
    padding: 10px;
    border-radius: 8px;
}

/* Download Button */
```

```
        .download-link {
            background-color: #007bff;
            color: white;
            padding: 10px;
            border-radius: 8px;
            display: inline-block;
            text-decoration: none;
            font-weight: bold;
            margin-top: 10px;
        }
    </style>
    """,
    unsafe_allow_html=True,
)


def display_result_box(title, value, color):
    st.markdown(
        f'<div class="result-box" style="background-color: {color}; color: white;
        ↪  padding: 15px; border-radius: 8px; text-align: center; font-weight:
        ↪  bold;">'
        f"{title}: {value}"
        f"</div>",
        unsafe_allow_html=True,
    )

BACKEND_URL = "http://127.0.0.1:5000"  # Change this when deployed with the
↪  appropriate url address

DEFAULT_JOB_DESCRIPTION = "Looking for a Software Engineer with Python, AWS, and
↪  Machine Learning experience."



# Sidebar UI
st.sidebar.markdown(
    """
    <style>
        /* Sidebar section with border */
        .sidebar-section {
            padding: 15px;
            border: 2px solid #4CAF50;
            border-radius: 10px;
```

```
            background-color: #1E1E1E;
            text-align: center;
            margin-bottom: 15px;
        }

        /* Full-width buttons */
        .stButton>button {
            width: 100%;
            display: block;
            padding: 12px;
            border-radius: 8px;
            font-size: 18px;
            font-weight: bold;
            background-color: #007bff;
            color: white;
            border: none;
            cursor: pointer;
            transition: 0.3s;
        }

        /* Button hover effect */
        .stButton>button:hover {
            background-color: #0056b3;
            transform: scale(1.05);
        }
    </style>
    """,
    unsafe_allow_html=True,
)
st.sidebar.markdown('<div class="sidebar-section"><h2 style="color:white;">
↪  Navigation</h2></div>', unsafe_allow_html=True)

if st.sidebar.button(" Upload Resume", key="upload_page"):
    st.session_state["page"] = "upload"

if st.sidebar.button(" View Past Resumes", key="history_page"):
    st.session_state["page"] = "history"

page = st.session_state.get("page", "upload")
st.sidebar.markdown("---")

if st.sidebar.button(" Search Resumes", key="search_page"):
    st.warning("Feature Coming Soon!")
```

```
if st.sidebar.button("⚙️ Settings", key="settings_page"):
    st.warning("Settings Page Under Development!")



def extract_pdf_preview(file):
    with pdfplumber.open(file) as pdf:
        first_page = pdf.pages[0]
        return first_page.extract_text()



if page == "upload":
    st.title("📄 Resume ATS Checker")

    uploaded_file = st.file_uploader("Upload Your Resume (PDF)", type=["pdf"])

    job_description = st.text_area(
        "Enter Job Description (Leave empty for default):",
        placeholder="Paste job description here..."
    )


    analyze_clicked = st.button("🔍 Analyze Resume")

    if uploaded_file and analyze_clicked:
        if not job_description.strip():
            job_description = DEFAULT_JOB_DESCRIPTION  # Use default if empty

        with st.spinner("Uploading & Analyzing..."):
            files = {"resume": uploaded_file}
            data = {"job_description": job_description.strip()}  # Ensures a valid
↪   string is sent
            response = requests.post(f"{BACKEND_URL}/upload", files=files,
↪   data=data)

            if response.status_code == 200:
                data = response.json()
                st.markdown('<div class="success-box">✅ Resume Analysis
↪   Complete!</div>', unsafe_allow_html=True)
```

```
                col1, col2, col3 = st.columns(3)

                with col1:
                    display_result_box("Job Role", data["job_role"], "#4CAF50")  #
↪ Green

                with col2:
                    display_result_box("Extracted Skills", ",
↪ ".join(data["skills"]), "#2196F3")  # Blue

                with col3:
                    display_result_box("ATS Score", f"{data['ats_score']}%",
↪ "#FF9800")  # Orange

                st.markdown('<div class="improvement"><b>🔧 Improvement
↪ Suggestions</b>', unsafe_allow_html=True)
                for suggestion in data["suggestions"]:
                    st.markdown(f'<li style="color:white;">{suggestion}</li>',
↪ unsafe_allow_html=True)
                st.markdown("</div>", unsafe_allow_html=True)

                st.markdown('<div class="resume-metadata"><b>📄 Resume
↪ Metadata</b>', unsafe_allow_html=True)
                st.markdown(f"📄 **File Name:** {uploaded_file.name}",
↪ unsafe_allow_html=True)
                st.markdown(f"📏 **File Size:** {round(uploaded_file.size / 1024,
↪ 2)} KB", unsafe_allow_html=True)
                st.markdown(f"🕒 **Upload Time:** {datetime.now().strftime('%Y-%m-%d
↪ %H:%M:%S')}", unsafe_allow_html=True)
                st.markdown("</div>", unsafe_allow_html=True)

                # Resume Preview
                st.markdown('<div class="resume-metadata"><b>📄 Resume Preview</b>',
↪ unsafe_allow_html=True)
                pdf_text = extract_pdf_preview(uploaded_file)
                st.text_area("First Page Preview:", pdf_text, height=200)
                st.markdown("</div>", unsafe_allow_html=True)

                # Downloadable Resume Link
                # First line is when we had AWS
                # st.markdown(f'<a class="download-link" href="{data["file_url"]}"
                 ↪ download>📥 Download Resume</a>', unsafe_allow_html=True)
```

```
                st.markdown(f'<a class="download-link"
↪  href="http://127.0.0.1:5000/view/{data["resume_id"]}" download> Download
↪  Resume</a>', unsafe_allow_html=True)

                if st.button(" Generate ATS Report"):
                    st.warning("This feature is under development!")

            else:
                st.error(" Error processing the resume. Please try again.")


if page == "history":
    st.title(" View Uploaded Resumes")

    response = requests.get(f"{BACKEND_URL}/metadata")

    if response.status_code == 200:
        resumes = response.json().get("resumes", [])
        if resumes:
            for resume in resumes:
                ats_score = resume.get("ats_score", "N/A")  # Use "N/A" if missing
                with st.expander(f" {resume['original_filename']} (ATS Score:
                ↪  {ats_score}%)"):
                    st.write(f" **Resume ID:** {resume['resume_id']}")
                    # st.write(f" **Resume Link:** [View
                    ↪  Resume]({resume['file_url']})")
                    # The line above was used when we had aws established
                    st.write(f" **Resume Link:** [View
↪  Resume](http://127.0.0.1:5000/view/{resume['resume_id']})")

        else:
            st.info("No resumes found.")
    else:
        st.error(" Failed to load resumes.")
```

D:/Programming/resume-matcher-ai/core/ats_scoring.py

```
from sentence_transformers import SentenceTransformer, util

sbert_model = SentenceTransformer("all-MiniLM-L6-v2")

def compute_sbert_similarity(resume_text, job_description):
    """Computes ATS match percentage using SBERT."""
```

```
    resume_embedding = sbert_model.encode(resume_text, convert_to_tensor=True)
    job_embedding = sbert_model.encode(job_description, convert_to_tensor=True)

    similarity = util.pytorch_cos_sim(resume_embedding, job_embedding)
    # 0.0 to 1.0 --> 64
    return round(similarity.item() * 100, 2)  # Convert to percentage
```

D:/Programming/resume-matcher-ai/core/job_description.py

```
# job_description.py
# Predefined job descriptions for various roles
JOB_DESCRIPTIONS = {
    "Software Engineer": "We are looking for a Software Engineer with experience in
     ↪ Python, Java, and cloud services such as AWS or GCP. The candidate should
     ↪ have strong problem-solving skills, knowledge of data structures and
     ↪ algorithms, and experience working with scalable applications.",

    "Backend Developer": "Seeking a Backend Developer proficient in Python, Django,
     ↪ Flask, and RESTful APIs. Experience with databases such as PostgreSQL and
     ↪ Redis is preferred. Understanding of system design and microservices
     ↪ architecture is a plus.",

    "Machine Learning Engineer": "Looking for a Machine Learning Engineer with
     ↪ expertise in Python, TensorFlow, PyTorch, and deep learning algorithms.
     ↪ Experience in model deployment and cloud-based AI solutions is highly
     ↪ desirable.",

    "Data Scientist": "Hiring a Data Scientist with strong skills in Python, SQL,
     ↪ and data visualization. The candidate should have experience with machine
     ↪ learning models, data analytics, and business intelligence tools like
     ↪ Tableau or Power BI.",

    "Frontend Developer": "We are seeking a Frontend Developer with expertise in
     ↪ React.js, JavaScript, and UI/UX principles. Experience with state
     ↪ management libraries like Redux and API integrations is required.",

    "Full Stack Developer": "Looking for a Full Stack Developer proficient in both
     ↪ frontend and backend development. Must have experience with React.js,
     ↪ Node.js, MongoDB, and cloud services like AWS Lambda or Firebase.",
```

```
    "DevOps Engineer": "Seeking a DevOps Engineer with experience in CI/CD
    ↪ pipelines, Docker, Kubernetes, and cloud platforms like AWS, Azure, or
    ↪ Google Cloud. Must have knowledge of monitoring tools and infrastructure
    ↪ automation.",

    "Cybersecurity Analyst": "Hiring a Cybersecurity Analyst with experience in
    ↪ threat detection, network security, and ethical hacking. Strong knowledge
    ↪ of security frameworks and compliance standards is required."
}
```

D:/Programming/resume-matcher-ai/core/job_role_matcher.py

```python
# job_role_classifier.py

def get_predefined_roles():
    """Returns a dictionary of predefined job roles and their required skills."""
    return {
        "Software Engineer": ["Python", "Java", "C++", "System Design", "AWS",
        ↪ "SQL"],
        "Data Scientist": ["Python", "Pandas", "Machine Learning", "Deep Learning",
        ↪ "TensorFlow", "SQL"],
        "ML Engineer": ["Python", "Machine Learning", "Deep Learning", "PyTorch",
        ↪ "TensorFlow", "AWS"],
        "Backend Developer": ["Python", "Django", "Flask", "PostgreSQL", "Redis",
        ↪ "Microservices"],
        "Frontend Developer": ["JavaScript", "React", "Vue.js", "CSS", "HTML",
        ↪ "TypeScript"],
        "DevOps Engineer": ["AWS", "Docker", "Kubernetes", "CI/CD", "Terraform",
        ↪ "Linux"],
        "Cybersecurity Analyst": ["Network Security", "Penetration Testing",
        ↪ "Cryptography", "Ethical Hacking"],
        "Cloud Engineer": ["AWS", "Azure", "Google Cloud", "Docker", "Kubernetes",
        ↪ "Terraform"],
        "Database Administrator": ["SQL", "PostgreSQL", "MongoDB", "Oracle",
        ↪ "Database Optimization"],
        "AI Researcher": ["Machine Learning", "Neural Networks", "Deep Learning",
        ↪ "AI Ethics"],
        "Full-Stack Developer": ["JavaScript", "React", "Node.js", "Django",
        ↪ "Flask", "SQL", "MongoDB"],
        "Embedded Systems Engineer": ["C", "C++", "Microcontrollers", "IoT",
        ↪ "RTOS"],
        "Game Developer": ["Unity", "Unreal Engine", "C#", "Game Physics",
        ↪ "Graphics Programming"],
```

```
        "Blockchain Developer": ["Solidity", "Ethereum", "Smart Contracts",
        ↪   "Cryptography"],
        "Business Intelligence Analyst": ["SQL", "Power BI", "Tableau", "Data
        ↪   Warehousing"],
    }

def match_resume_to_job_role(resume_skills):
    """Finds the closest matching job role based on skill overlap."""
    roles = get_predefined_roles()

    print(f"▯ Resume Skills for Matching: {resume_skills}")  # Debugging

    best_match = None
    highest_overlap = 0

    for role, required_skills in roles.items():
        overlap = len(set(resume_skills) & set(required_skills))
        if overlap > highest_overlap:
            highest_overlap = overlap
            best_match = role

    print(f"▯ Matched Job Role: {best_match}")  # Debugging
    return best_match
```

D:/Programming/resume-matcher-ai/core/resume_feedback.py

```
import os
import google.generativeai as genai
from core.job_role_matcher import get_predefined_roles

# Configure Gemini API
GEMINI_API_KEY = os.getenv("GEMINI_API_KEY")  # ▯ Load from environment variable
genai.configure(api_key=GEMINI_API_KEY)

def fetch_gemini_suggestions(job_role, resume_text, missing_skills):
    """Uses Gemini API to generate personalized resume improvement suggestions."""
    prompt = f"""
    You are an AI resume optimizer. The user is applying for a **{job_role}** role.
    Their resume text is as follows:

    {resume_text}

    Missing key skills: {', '.join(missing_skills)}
```

```
    Provide **three concise, actionable suggestions** to improve their resume to
↪   align better with this job.
    """

    try:
        model = genai.GenerativeModel("gemini-1.5-flash")
        response = model.generate_content(prompt)
        return response.text.split("\n")[:3]  # Return top 3 suggestions
    except Exception as e:
        print(f"Error using Gemini API: {str(e)}")
        return ["Error fetching AI-based suggestions. Try again later."]

def generate_resume_suggestions(detected_job_role, resume_skills, resume_text,
↪   ats_score):
    """Generates AI-enhanced ATS suggestions based on missing skills & resume
     ↪  format."""

    job_roles = get_predefined_roles()
    required_skills = job_roles.get(detected_job_role, [])

    # Identify missing skills
    missing_skills = [skill for skill in required_skills if skill not in
↪   resume_skills]

    # Generate improvement suggestions
    suggestions = []

    if ats_score < 50:
        suggestions.append("Your ATS score is low. Consider optimizing your resume
↪   for better keyword matching.")

    if missing_skills:
        suggestions.append(f"Consider adding these important skills: {',
↪   '.join(missing_skills[:5])}")

    if len(resume_text.split()) < 200:
        suggestions.append("Your resume is too short. Add more details about your
↪   experience and skills.")

    if "-" not in resume_text and "•" not in resume_text:  # Check if bullet points
     ↪  exist
```

```
        suggestions.append("Use bullet points to highlight your experience more
↪   clearly.")

    # Fetch AI-powered suggestions from Gemini
    ai_suggestions = fetch_gemini_suggestions(detected_job_role, resume_text,
↪   missing_skills)
    suggestions.extend(ai_suggestions)  # Add AI-generated suggestions

    return suggestions
```

D:/Programming/resume-matcher-ai/core/resume_skill_extractor.py

```
# resume_skill_extractor.py
import re

def extract_resume_skills(resume_text):
    resume_text = resume_text.lower()  # Convert all text to lowercase
    skill_keywords = ["python", "java", "c++", "aws", "sql", "machine learning",
                      "deep learning", "tensorflow", "flask", "django", "react",
                       ↪  "docker"]

    detected_skills = []
    for skill in skill_keywords:
        safe_skill = re.escape(skill)
        if re.search(rf"\b{safe_skill}\b", resume_text):
            detected_skills.append(skill.capitalize())  # Keep original casing for
↪   output

    print(f"🔍 Extracted Skills: {detected_skills}")  # Debugging
    return list(set(detected_skills))
```

D:/Programming/resume-matcher-ai/experiment/model_comparator.py

```
import os
import sys
sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), "..")))

import fitz
import torch
from transformers import BertTokenizer, BertModel
from sentence_transformers import SentenceTransformer, util
from utils.resume_parser import extract_text_from_pdf
```

```python
# Load Models
tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
model = BertModel.from_pretrained("bert-base-uncased")
sbert_model = SentenceTransformer("all-MiniLM-L6-v2")

def compute_bert_similarity(resume_text, job_description):
    """Computes similarity between a resume and job description using BERT
    ↪ embeddings."""

    # Limit resume text to first 500 words to fit within BERT's 512-token limit
    resume_text = " ".join(resume_text.split()[:500])

    inputs = tokenizer(resume_text, job_description, return_tensors="pt",
↪ truncation=True, max_length=512)

    with torch.no_grad():
        outputs = model(**inputs).last_hidden_state


    resume_embedding = outputs[:, 0, :]
    job_embedding = outputs[:, 1, :]

    similarity = torch.cosine_similarity(resume_embedding, job_embedding)
    score = similarity.item() * 100  # Convert to percentage

    return round(score, 2)

def compute_sbert_similarity(resume_text, job_description):
    """Computes similarity using Sentence-BERT (SBERT)."""
    resume_embedding = sbert_model.encode(resume_text, convert_to_tensor=True)
    job_embedding = sbert_model.encode(job_description, convert_to_tensor=True)

    similarity = util.pytorch_cos_sim(resume_embedding, job_embedding)
    return round(similarity.item() * 100, 2)  # Convert to percentage

if __name__ == "__main__":
    resume_path = r"./sample_resumes/Alhaan_resume.pdf"  # Use raw string
    job_description = "Looking for a software engineer with Python, AWS, and
↪ Machine Learning experience."

    # Extract text from PDF
    resume_text = extract_text_from_pdf(resume_path)
```

```
    # Compute ATS Scores
    ats_score_bert = compute_bert_similarity(resume_text, job_description)
    ats_score_sbert = compute_sbert_similarity(resume_text, job_description)

    # Calculate improvement percentage
    improvement = ((ats_score_sbert - ats_score_bert) / ats_score_bert) * 100 if
↪   ats_score_bert != 0 else 0

    # Print Comparison Results
    print("\n Match Score BERT:", ats_score_bert, "%")
    print("\n Match Score SBERT:", ats_score_sbert, "%")
    print(f"\n SBERT Improved ATS Score by {improvement:.2f}%\n")
```

D:/Programming/resume-matcher-ai/resume_generator/resume_gen_latex_1.py

```
import os
from datetime import datetime
from resume_sample_data import resume_data, jd_data  # Import the data from the
↪   external file

def generate_latex_filename():
    current_time = datetime.now()
    formatted_time = current_time.strftime("%I_%M%p")
    formatted_date = current_time.strftime("%d_%b")
    return f"JakeResume_{formatted_time}_{formatted_date}.tex"

def escape_latex_special_chars(text):
    # Escape special LaTeX characters
    special_chars = {
        '&': r'\&',
        '%': r'\%',
        '$': r'\$',
        '#': r'\#',
        '_': r'\_',
        '{': r'\{',
        '}': r'\}',
        '~': r'\textasciitilde{}',
        '^': r'\^{}',
        '\\': r'\textbackslash{}',
        '<': r'\textless{}',
```

```
        '>': r'\textgreater{}',
    }
    if isinstance(text, str):
        for char, escape in special_chars.items():
            text = text.replace(char, escape)
    return text

def generate_latex_resume():
    # Create 'gen_resume' folder if it doesn't exist
    if not os.path.exists(r'D:\Programming\resume-matcher-
    ↪ ai\resume_generator_test\gen_resume'):
        os.makedirs(r'D:\Programming\resume-matcher-
↪   ai\resume_generator_test\gen_resume')

    # Define the filename using current date and time
    filename = os.path.join(r'D:\Programming\resume-matcher-
↪   ai\resume_generator_test\gen_resume', generate_latex_filename())

    # Escape special characters in all text fields
    def process_data(data):
        if isinstance(data, str):
            return escape_latex_special_chars(data)
        elif isinstance(data, dict):
            return {k: process_data(v) for k, v in data.items()}
        elif isinstance(data, list):
            return [process_data(item) for item in data]
        return data

    resume_data_processed = process_data(resume_data)
    jd_data_processed = process_data(jd_data)

    # Start building the LaTeX content for article class
    latex_content = r"""
\documentclass[letterpaper, 11pt]{article}
\usepackage[left=1in, top=1in, right=1in, bottom=1in]{geometry}
\usepackage{enumitem}
\usepackage{hyperref}
\usepackage{titlesec}
\usepackage{fontspec}
\usepackage{microtype}
\usepackage{lmodern}
\usepackage{textcomp}
```

```
\setmainfont{Latin Modern Roman}[
    BoldFeatures={SmallCapsFont={Latin Modern Roman Caps}}
]

% For section formatting
\titleformat{\section}{\large\bfseries}{\thesection}{1em}{}

\pagestyle{empty}

\begin{document}

\begin{center}
    \textbf{\Huge \scshape """ + resume_data_processed['name'] + r"""} \\[2mm]
    """ + resume_data_processed['phone'] + r" | " + resume_data_processed['email']
 ↪  + r" | " + resume_data_processed['linkedin'] + r" | " +
 ↪  resume_data_processed['github'] + r"""
\end{center}

%------------- Summary Section -------------
\section*{Summary}
""" + resume_data_processed['summary'] + r"""

%------------- Education Section -------------
\section*{Education}
\textbf{""" + resume_data_processed['education']['degree'] + r"""} \hfill
 ↪  \textbf{""" + resume_data_processed['education']['dates'] + r"""} \\
""" + resume_data_processed['education']['institution'] + r""" \hfill GPA: """ +
 ↪  resume_data_processed['education']['gpa'] + r""" \\[2mm]
\textit{""" + resume_data_processed['education']['details'] + r"""}

%------------- Technical Skills Section -------------
\section*{Technical Skills}
\begin{tabbing}
\hspace{4cm} \= \hspace{6cm} \= \hspace{4cm} \= \kill
\textbf{Languages:} \> """ + ',
 ↪  '.join(resume_data_processed['skills']['programming_languages']) + r""" \\
\textbf{Tools:} \> """ + ', '.join(resume_data_processed['skills']['tools']) +
 ↪  r""" \\
\textbf{Soft Skills:} \> """ + ',
 ↪  '.join(resume_data_processed['skills']['soft_skills']) + r"""
\end{tabbing}

%------------- Work Experience Section -------------
```

```
\section*{Work Experience}
"""

    # Add work experience dynamically
    for experience in resume_data_processed['work_experience']:
        latex_content += f"""
\\textbf{{{experience['job_title']}}} \\hfill
↪  \\textbf{{{experience['employment_dates']}}} \\\\
\\textit{{{experience['company']}}} \\hfill \\textit{{{experience.get('location',
↪  '')}}} \\\\
{experience['brief_job_description']} \\\\
\\begin{{itemize}}[left=0pt,itemsep=0pt,parsep=0pt]"""
        for achievement in experience['key_achievements']:
            latex_content += f"\\item {achievement}\n"
        latex_content += "\\end{itemize}\n"

    latex_content += r"""

%------------ Projects Section ------------
\section*{Projects}
"""

    # Add projects dynamically
    if 'projects' in resume_data_processed:
        for project in resume_data_processed['projects']:
            latex_content += r"""
\textbf{""" + project['name'] + r"""} \hfill \textbf{""" + project['date'] + r"""}
↪  \\
\textit{Technologies:} """ + ', '.join(project['technologies']) + r""" \\
""" + project['description'] + r""" \\[2mm]
\begin{itemize}[left=0in]
"""
            for achievement in project['achievements']:
                latex_content += r"\item " + achievement + r" \n"
            latex_content += r"\end{itemize}"

    latex_content += f"""

\\section*{{Job Description}}
\\textbf{{{jd_data_processed['job_title']}}} \\hfill
↪  \\textbf{{{jd_data_processed['employment_dates']}}} \\\\
\\textit{{{jd_data_processed['company']}}} \\hfill
↪  \\textit{{{jd_data_processed.get('location', '')}}} \\\\
```

```
{jd_data_processed['brief_job_description']} \\\\
\\begin{{itemize}}[left=0pt,itemsep=0pt,parsep=0pt]"""

    for achievement in jd_data_processed['key_achievements']:
        latex_content += f"\\item {achievement}\n"

    latex_content += """\\end{itemize}
\\end{document}"""

    # Write the LaTeX content to file
    with open(filename, 'w', encoding='utf-8') as f:
        f.write(latex_content)

    print(f"LaTeX resume successfully generated and saved as: {filename}")
    print("Compile this file with XeLaTeX or PDFLaTeX to generate the PDF.")

# Generate the LaTeX resume
generate_latex_resume()
```

D:/Programming/resume-matcher-ai/resume_generator/resume_gen_latex_2.py

```
import os
import re
from datetime import datetime
from resume_sample_data import resume_data, jd_data  # Import the data from the
↪  external file

def generate_latex_filename():
    current_time = datetime.now()
    formatted_time = current_time.strftime("%I_%M%p")
    formatted_date = current_time.strftime("%d_%b")
    return f"JakeResume_{formatted_time}_{formatted_date}.tex"

def escape_latex_special_chars(text):
    if not isinstance(text, str):
        return text

    # First, escape backslashes (must be done first)
    text = text.replace('\\', '\\textbackslash{}')

    # Escape other special LaTeX characters
```

```
    special_chars = {
        '&': r'\&',
        '%': r'\%',
        '$': r'\$',
        '#': r'\#',
        '_': r'\_',
        '{': r'\{',
        '}': r'\}',
        '~': r'\textasciitilde{}',
        '^': r'\textasciicircum{}',
        '<': r'\textless{}',
        '>': r'\textgreater{}',
    }

    for char, escape in special_chars.items():
        text = text.replace(char, escape)

    # Additional safety check: remove any LaTeX command patterns
    text = re.sub(r'\\([a-zA-Z]+)', r'\\textbackslash{}\1', text)
    return text

def generate_latex_resume():
    # Create 'gen_resume' folder if it doesn't exist
    if not os.path.exists(r'D:\Programming\resume-matcher-
    ↪ ai\resume_generator_test\gen_resume'):
        os.makedirs(r'D:\Programming\resume-matcher-
↪ ai\resume_generator_test\gen_resume')

    # Define the filename using current date and time
    filename = os.path.join(r'D:\Programming\resume-matcher-
↪ ai\resume_generator_test\gen_resume', generate_latex_filename())

    # Escape special characters in all text fields
    def process_data(data):
        if isinstance(data, str):
            return escape_latex_special_chars(data)
        elif isinstance(data, dict):
            return {k: process_data(v) for k, v in data.items()}
        elif isinstance(data, list):
            return [process_data(item) for item in data]
        return data

    # Create deep copies to avoid modifying the original data
```

```
    import copy
    resume_data_copy = copy.deepcopy(resume_data)
    jd_data_copy = copy.deepcopy(jd_data)

    # Process the data
    resume_data_processed = process_data(resume_data_copy)
    jd_data_processed = process_data(jd_data_copy)

    # Start building the LaTeX content - using the Jake template style
    latex_content = r"""
%-------------------------
% Resume in Latex
% Based on Jake Gutierrez's template
% License: MIT
%-------------------------

\documentclass[letterpaper,11pt]{article}

\usepackage{latexsym}
\usepackage[empty]{fullpage}
\usepackage{titlesec}
\usepackage{marvosym}
\usepackage[usenames,dvipsnames]{color}
\usepackage{verbatim}
\usepackage{enumitem}
\usepackage[hidelinks]{hyperref}
\usepackage{fancyhdr}
\usepackage[english]{babel}
\usepackage{tabularx}
\usepackage{fontspec}

% Fix for XeLaTeX compatibility
\defaultfontfeatures{Ligatures=TeX}

\pagestyle{fancy}
\fancyhf{} % clear all header and footer fields
\fancyfoot{}
\renewcommand{\headrulewidth}{0pt}
\renewcommand{\footrulewidth}{0pt}

% Adjust margins
\addtolength{\oddsidemargin}{-0.5in}
\addtolength{\evensidemargin}{-0.5in}
```

```
\addtolength{\textwidth}{1in}
\addtolength{\topmargin}{-.5in}
\addtolength{\textheight}{1.0in}

\urlstyle{same}

\raggedbottom
\raggedright
\setlength{\tabcolsep}{0in}

% Sections formatting
\titleformat{\section}{
  \vspace{-4pt}\scshape\raggedright\large
}{}{0em}{}[\color{black}\titlerule \vspace{-5pt}]

%-------------------------
% Custom commands
\newcommand{\resumeItem}[1]{
  \item\small{
    {#1 \vspace{-2pt}}
  }
}

\newcommand{\resumeSubheading}[4]{
  \vspace{-2pt}\item
    \begin{tabular*}{0.97\textwidth}[t]{l@{\extracolsep{\fill}}r}
      \textbf{#1} & #2 \\
      \textit{\small#3} & \textit{\small #4} \\
    \end{tabular*}\vspace{-7pt}
}

\newcommand{\resumeSubSubheading}[2]{
    \item
    \begin{tabular*}{0.97\textwidth}{l@{\extracolsep{\fill}}r}
      \textit{\small#1} & \textit{\small #2} \\
    \end{tabular*}\vspace{-7pt}
}

\newcommand{\resumeProjectHeading}[2]{
    \item
    \begin{tabular*}{0.97\textwidth}{l@{\extracolsep{\fill}}r}
      \small#1 & #2 \\
    \end{tabular*}\vspace{-7pt}
```

```
}

\newcommand{\resumeSubItem}[1]{\resumeItem{#1}\vspace{-4pt}}

\renewcommand\labelitemii{$\vcenter{\hbox{\tiny$\bullet$}}$}

\newcommand{\resumeSubHeadingListStart}{\begin{itemize}[leftmargin=0.15in,
 ↪  label={}]}
\newcommand{\resumeSubHeadingListEnd}{\end{itemize}}
\newcommand{\resumeItemListStart}{\begin{itemize}}
\newcommand{\resumeItemListEnd}{\end{itemize}\vspace{-5pt}}

%-------------------------------------------
%%%%%%  RESUME STARTS HERE  %%%%%%%%%%%%%%%%%%%%%%%%%%%%

\begin{document}

%----------HEADING----------
\begin{center}
    \textbf{\Huge \scshape """ + resume_data_processed['name'] + r"""} \\
 ↪  \vspace{1pt}
    \small """ + resume_data_processed['phone'] + r""" $|$ \href{mailto:""" +
 ↪  resume_data_processed['email'] + r"""}{\underline{""" +
 ↪  resume_data_processed['email'] + r"""}} $|$
    \href{""" + resume_data_processed['linkedin'] + r"""}{\underline{""" +
 ↪  resume_data_processed['linkedin'].replace('https://', '') + r"""}} $|$
    \href{""" + resume_data_processed['github'] + r"""}{\underline{""" +
 ↪  resume_data_processed['github'].replace('https://', '') + r"""}}
\end{center}

%----------SUMMARY SECTION----------
\section{Summary}
""" + resume_data_processed['summary'] + r"""

%-----------EDUCATION-----------
\section{Education}
  \resumeSubHeadingListStart
    \resumeSubheading
      {""" + resume_data_processed['education']['degree'] + r"""}{""" +
 ↪  resume_data_processed['education']['dates'] + r"""}
      {""" + resume_data_processed['education']['institution'] + r"""}{GPA: """ +
 ↪  resume_data_processed['education']['gpa'] + r"""}
    % Additional education entries can be added here
```

```
  \resumeSubHeadingListEnd

%-----------EXPERIENCE-----------
\section{Experience}
  \resumeSubHeadingListStart
"""

    # Add work experience dynamically using the custom commands
    for experience in resume_data_processed['work_experience']:
        latex_content += f"""
    \\resumeSubheading
      {{{experience['job_title']}}}{{{experience['employment_dates']}}}
      {{{experience['company']}}}{{{experience.get('location', '')}}}
      \\resumeItemListStart"""

        # Add each achievement as a separate item
        for achievement in experience['key_achievements']:
            latex_content += f"""
        \\resumeItem{{{achievement}}}"""

        latex_content += """
      \\resumeItemListEnd
"""

    latex_content += r"""
  \resumeSubHeadingListEnd

%-----------PROJECTS-----------
\section{Projects}
  \resumeSubHeadingListStart
"""

    # Add projects dynamically
    if 'projects' in resume_data_processed:
        for project in resume_data_processed['projects']:
            technologies = ', '.join(project['technologies'])
            latex_content += f"""
    \\resumeProjectHeading
        {{\\textbf{{{project['name']}}} $|$
↪   \\emph{{{technologies}}}}}{{{project['date']}}}
        \\resumeItemListStart"""

            for achievement in project['achievements']:
```

```
                latex_content += f"""
            \\resumeItem{{{achievement}}}"""

            latex_content += """
        \\resumeItemListEnd
"""

    latex_content += r"""
  \resumeSubHeadingListEnd

%-----------TECHNICAL SKILLS-----------
\section{Technical Skills}
 \begin{itemize}[leftmargin=0.15in, label={}]
    \small{\item{
     \textbf{Languages}{: """ + ',
 ↪  '.join(resume_data_processed['skills']['programming_languages']) + r"""} \\
     \textbf{Tools}{: """ + ', '.join(resume_data_processed['skills']['tools']) +
 ↪  r"""} \\
     \textbf{Soft Skills}{: """ + ',
 ↪  '.join(resume_data_processed['skills']['soft_skills']) + r"""}
    }}
 \end{itemize}

% Optional Job Description Section for comparison
\section{Job Description}
  \resumeSubHeadingListStart
    \resumeSubheading
      {""" + jd_data_processed['job_title'] + r"""}{""" +
 ↪  jd_data_processed['employment_dates'] + r"""}
      {""" + jd_data_processed['company'] + r"""}{""" +
 ↪  jd_data_processed.get('location', '') + r"""}
      \resumeItemListStart"""

    for achievement in jd_data_processed['key_achievements']:
        latex_content += f"""
        \\resumeItem{{{achievement}}}"""

    latex_content += """
      \\resumeItemListEnd
  \\resumeSubHeadingListEnd

%-------------------------------------------
\\end{document}"""
```

```
    # Write the LaTeX content to file
    with open(filename, 'w', encoding='utf-8') as f:
        f.write(latex_content)

    print(f"LaTeX resume successfully generated and saved as: {filename}")
    print("Compile this file with XeLaTeX to generate the PDF.")
    print("If you encounter any issues, check your resume_sample_data.py file for
     ↪  special characters or LaTeX commands.")

# Generate the LaTeX resume
generate_latex_resume()
```

D:/Programming/resume-matcher-ai/resume_generator/resume_gen_trial_1.py

```
import os
from datetime import datetime
from reportlab.platypus import SimpleDocTemplate, Paragraph
from reportlab.lib.styles import getSampleStyleSheet
from reportlab.lib.pagesizes import letter
from resume_sample_data import resume_data, jd_data  # Import the data from the
 ↪  external file

# Function to generate PDF filename based on current date and time
def generate_filename():
    # Get current date and time
    current_time = datetime.now()

    # Format the time and date
    formatted_time = current_time.strftime("%I_%M%p")  # Hour_MinuteAM/PM
    formatted_date = current_time.strftime("%d_%b")    # Day_Month (e.g., 4_Apr)

    # Return the file name in the format you specified
    return f"{formatted_time}_{formatted_date}.pdf"

# Generate PDF
def generate_pdf():
    # Create 'gen_resume' folder if it doesn't exist
    if not os.path.exists(r'D:\Programming\resume-matcher-
     ↪  ai\resume_generator_test\gen_resume'):
        os.makedirs(r'D:\Programming\resume-matcher-
 ↪  ai\resume_generator_test\gen_resume')
```

```
    # Define the filename using current date and time
    filename = os.path.join(r'D:\Programming\resume-matcher-
↪   ai\resume_generator_test\gen_resume', generate_filename())

    # Create the PDF document
    doc = SimpleDocTemplate(filename, pagesize=letter)

    # Get the style sheet for the document
    styles = getSampleStyleSheet()
    style = styles['Normal']

    # Creating sections for the resume
    content = []

    # Contact Information
    contact_info = f"Name: {resume_data['name']}\nAddress:
↪   {resume_data['address']}\nPhone: {resume_data['phone']}\nEmail:
↪   {resume_data['email']}\nLinkedIn: {resume_data['linkedin']}\nGitHub:
↪   {resume_data['github']}"
    content.append(Paragraph(contact_info, style))

    # Education
    education = f"Education: {resume_data['education']}"
    content.append(Paragraph(education, styles['Heading2']))

    # Work Experience
    work_experience = 'Work Experience:\n'
    for experience in resume_data['work_experience']:
        work_experience += f"{experience['job_title']}, {experience['company']},
↪   {experience['employment_dates']}\n"
        work_experience += f"{experience['brief_job_description']}\n"
        work_experience += 'Key Achievements:\n'
        for achievement in experience['key_achievements']:
            work_experience += f"- {achievement}\n"
        work_experience += '\n'
    content.append(Paragraph(work_experience, styles['Heading2']))

    # Skills
    skills = 'Skills:\n'
    skills += f"Programming Languages: {',
↪   '.join(resume_data['skills']['programming_languages'])}\n"
    skills += f"Tools: {', '.join(resume_data['skills']['tools'])}\n"
```

```
    skills += f"Soft Skills: {', '.join(resume_data['skills']['soft_skills'])}"
    content.append(Paragraph(skills, styles['Heading2']))

    # Summary
    summary = f"Summary: {resume_data['summary']}"
    content.append(Paragraph(summary, styles['Heading2']))

    # Job Description
    jd = 'Job Description:\n'
    jd += f"{jd_data['job_title']}, {jd_data['company']},
↪   {jd_data['employment_dates']}\n"
    jd += f"{jd_data['brief_job_description']}\n"
    jd += 'Key Achievements:\n'
    for achievement in jd_data['key_achievements']:
        jd += f"- {achievement}\n"
    content.append(Paragraph(jd, styles['Heading2']))

    # Building PDF document
    doc.build(content)

    print(f"Resume successfully generated and saved as: {filename}")

# Generate the PDF
generate_pdf()
```

D:/Programming/resume-matcher-ai/resume_generator/resume_sample_data.py

```
# resume_sample_data.py

# Resume Data
resume_data = {
    'name': 'John Doe',
    'address': '123 Main St, City, State, 12345',
    'phone': '(555) 555-5555',
    'email': 'john.doe@email.com',
    'linkedin': 'linkedin.com/in/johndoe',
    'github': 'github.com/johndoe',
    'education': {
        'degree': "Bachelor's Degree in Computer Science",
        'institution': 'University of California',
        'dates': '2015',
```

```
        'gpa': '3.8',
        'details': 'Relevant coursework: Data Structures, Algorithms, Software
        ↪  Engineering'
    },
    'work_experience': [
        {
            'job_title': 'Software Engineer',
            'company': 'Google',
            'employment_dates': '2018-2020',
            'brief_job_description': 'Developed and deployed multiple software
            ↪  applications using Java and Python.',
            'key_achievements': [
                'Improved code quality by 30%',
                'Reduced deployment time by 50%',
                'Increased user engagement by 25%',
            ]
        },
        {
            'job_title': 'Junior Software Engineer',
            'company': 'Tech Innovators',
            'employment_dates': '2016-2018',
            'brief_job_description': 'Worked on developing user-facing features and
            ↪  backend services.',
            'key_achievements': [
                'Optimized codebase, resulting in a 20% increase in application
                ↪  speed.',
                'Led a project that improved UI/UX design for a major client.',
            ]
        }
    ],
    'projects': [
        {
            'name': 'Personal Portfolio Website',
            'technologies': ['HTML', 'CSS', 'JavaScript', 'Bootstrap'],
            'date': '2019',
            'achievements': [
                'Designed and implemented a responsive portfolio website to
                ↪  showcase projects',
                'Integrated contact form functionality using JavaScript'
            ]
        },
        {
            'name': 'Inventory Management System',
```

```
            'technologies': ['Python', 'Django', 'PostgreSQL', 'Docker'],
            'date': '2020',
            'achievements': [
                'Built a full-stack inventory management application with user
                ↪  authentication',
                'Implemented RESTful API endpoints for CRUD operations',
                'Deployed the application using Docker containers'
            ]
        }
    ],
    'skills': {
        'programming_languages': ['Java', 'Python', 'C++', 'JavaScript'],
        'tools': ['Git', 'Jenkins', 'Docker', 'Kubernetes'],
        'soft_skills': ['Communication', 'Teamwork', 'Problem-solving',
        ↪  'Leadership']
    },
    'summary': 'Highly motivated and experienced software engineer with a strong
    ↪  background in computer science and software development. Proficient in
    ↪  multiple programming languages and tools, with a proven track record of
    ↪  delivering high-quality software applications.'
}

# Job Description Data
jd_data = {
    'job_title': 'Software Engineer',
    'company': 'Google',
    'employment_dates': '2018-2020',
    'brief_job_description': 'Developed and deployed multiple software applications
    ↪  using Java and Python.',
    'key_achievements': [
        'Improved code quality by 30%',
        'Reduced deployment time by 50%',
        'Increased user engagement by 25%',
    ]
}
```

D:/Programming/resume-matcher-ai/services/aws_handler.py

```
import boto3
import os
from decimal import Decimal  # Import Decimal for DynamoDB

# AWS Credentials & Config
```

```
AWS_ACCESS_KEY = os.getenv("AWS_ACCESS_KEY_ID")
AWS_SECRET_KEY = os.getenv("AWS_SECRET_ACCESS_KEY")
AWS_REGION = "us-east-1"
S3_BUCKET = "resume-parser-bucket-12345"  # Replace with your bucket name
DYNAMODB_TABLE = "ResumeMetadata"  # Replace with your table name

# Connect to AWS S3
s3 = boto3.client(
    "s3",
    aws_access_key_id=AWS_ACCESS_KEY,
    aws_secret_access_key=AWS_SECRET_KEY,
    region_name=AWS_REGION
)

# Connect to DynamoDB
dynamodb = boto3.resource(
    "dynamodb",
    aws_access_key_id=AWS_ACCESS_KEY,
    aws_secret_access_key=AWS_SECRET_KEY,
    region_name=AWS_REGION
)
table = dynamodb.Table(DYNAMODB_TABLE)

def upload_to_s3(file, unique_filename):
    """Uploads a file to S3 and returns its URL."""
    file.seek(0)
    s3.upload_fileobj(
        file,
        S3_BUCKET,
        unique_filename,
        ExtraArgs={"ContentType": "application/pdf"}
    )
    return f"https://{S3_BUCKET}.s3.amazonaws.com/{unique_filename}"

def store_resume_metadata(resume_id, original_filename, file_url, ats_score):
    """Stores resume metadata in DynamoDB."""
    table.put_item(Item={
        "resume_id": resume_id,
        "original_filename": original_filename,
        "file_url": file_url,
        "ats_score": Decimal(str(ats_score))
    })
```

```
def get_resume_by_id(resume_id):
    """Fetches a resume's metadata from DynamoDB."""
    response = table.get_item(Key={"resume_id": resume_id})
    return response.get("Item")

def get_all_resumes():
    """Fetches all stored resume metadata from DynamoDB."""
    response = table.scan()
    return response.get("Items", [])
```

D:/Programming/resume-matcher-ai/utils/resume_parser.py

```
import fitz  # PyMuPDF

def extract_text_from_pdf(pdf_input):
    """Extracts text from a PDF file."""
    if isinstance(pdf_input, str):  # If input is a file path
        doc = fitz.open(pdf_input)
    else:  # If input is a file object
        pdf_input.seek(0)
        doc = fitz.open(stream=pdf_input.read(), filetype="pdf")
    text = "\n".join(page.get_text("text") for page in doc)
    return text.strip() if text else "No text extracted"


# Text value --> new_text_value -->
```