Objective

The objective of this project is to design and develop an AI-assisted Resume Analyzer and Job Recommendation System that helps evaluate a candidate's resume against job requirements in an accurate, explainable, and practical manner.

The system aims to automatically process resumes provided in PDF or text format, extract relevant technical skills, and compare them with job descriptions to determine how well a candidate fits a particular role. By combining rule-based skill matching with AI-based semantic analysis, the system is able to assess both exact skill matches and contextual similarity between the resume and job description.

In addition to resume analysis, the project focuses on recommending suitable job roles based on predefined skill requirements and providing clear improvement suggestions by identifying missing or weak skills. The overall goal is to simulate a real-world recruitment and Applicant Tracking System (ATS) workflow while maintaining transparency, reliability, and usability for students and job seekers.

This project ultimately aims to assist users in understanding their current skill alignment, improving their resumes, and making informed career decisions.

Functionality

The system works in multiple well-defined stages:

A. Resume Input

User can:

Upload a PDF resume, or

Paste resume text manually

User also provides a job description

B. Resume Processing

If PDF is uploaded:

Text is extracted using PyPDF2

If text is pasted:

Text is used directly

C. Skill Extraction

Skills are detected using regex-based NLP

Supports:

Single-word skills (Python, SQL)

Multi-word skills (Machine Learning, REST API)

The system also stores sentence snippets where skills appear (explainability)

D. Resume vs Job Analysis

Compares extracted resume skills with job skills

Generates:

Rule-based match score (exact skill matching)

AI semantic score (meaning-based similarity using embeddings, if available)

Identifies:

Matched skills

Missing skills

E. Job Recommendation

Uses a predefined job database

Each job has:

Must-have skills

Good-to-have skills

Scoring logic:

Must-have skills → 70%

Good-to-have skills → 30%

Returns top 3 suitable job roles

F. Resume Improvement Suggestions

Missing must-have skills → HIGH priority

Missing bonus skills → MEDIUM priority

Helps the candidate understand what to learn next

G. Fault-Tolerant AI Design

If AI API quota is unavailable:

System gracefully falls back to rule-based logic

Application continues working without crashing

Product Folder Structure

AI_Resume_Analyzer/

│

├── app.py

├── jobs.py

├── requirements.txt

│

├── static/

│ └── index.html

│

├── venv/

│ └── (virtual environment files)

│

└── README.md

app.py

The main backend file that initializes the Flask application and contains all core logic including resume analysis, skill extraction, job matching, AI semantic similarity, and API routes.

jobs.py

Stores structured job role data, including must-have and good-to-have skills, which is used by the recommendation engine for scoring and role matching.

static/

A Flask-specific folder used to store frontend assets that are directly served to the browser.

static/index.html

Provides the user interface of the application, allowing users to upload resumes, enter job descriptions, trigger analysis, and view results using HTML, CSS, and JavaScript.

venv/

A virtual environment directory that isolates project-specific Python packages from the system environment to avoid dependency conflicts.

README.md

Contains a high-level overview of the project, setup instructions, feature descriptions, and future enhancement ideas.

CODE

app.py

```
from flask import Flask, send_from_directory, request, jsonify

import re

import math

from PyPDF2 import PdfReader

from jobs import JOBS

from openai import OpenAI


# ========================
```

```
# APP INIT
# ========================
app = Flask(__name__, static_folder="static")
client = OpenAI()


# ========================
# SKILLS CONFIG
# ========================
SKILLS = [
    "python", "flask", "django", "sql", "postgres", "postgresql", "mysql",
    "mongodb", "javascript", "react", "node", "aws", "docker",
    "kubernetes", "html", "css", "git",
    "machine learning", "deep learning",
    "pandas", "numpy", "tensorflow", "pytorch",
    "rest api", "rest", "api",
    "unit testing", "pytest"
]


SKILL_PATTERNS = {
    skill: re.compile(r"\b" + re.escape(skill) + r"\b", re.IGNORECASE)
    for skill in SKILLS
}
```

```
# ========================
# UTILITY FUNCTIONS
# ========================
def split_to_sentences(text):
    parts = re.split(r"[.?!]\s*", text)
    return [p.strip() for p in parts if p.strip()]

def find_skills_with_snippets(text):
    results = {}
    sentences = split_to_sentences(text)
    lower_text = text.lower()

    for skill, pattern in SKILL_PATTERNS.items():
        if not pattern.search(lower_text):
            continue

        snippets = []
        for s in sentences:
            if pattern.search(s):
                snippets.append(s)

        if not snippets:
            snippets = [lower_text[:120]]
```

```
        results[skill] = {
            "count": len(snippets),
            "snippets": snippets
        }

    return results

def extract_text_from_pdf(pdf_file):
    reader = PdfReader(pdf_file)
    text = ""
    for page in reader.pages:
        page_text = page.extract_text()
        if page_text:
            text += page_text + "\n"
    return text

# ---------- AI SEMANTIC SIMILARITY ----------
def semantic_similarity(text1, text2):
    emb1 = client.embeddings.create(
        model="text-embedding-3-small",
        input=text1
    ).data[0].embedding
```

```
    emb2 = client.embeddings.create(
        model="text-embedding-3-small",
        input=text2
    ).data[0].embedding

    dot = sum(a * b for a, b in zip(emb1, emb2))
    norm1 = math.sqrt(sum(a * a for a in emb1))
    norm2 = math.sqrt(sum(b * b for b in emb2))

    return dot / (norm1 * norm2)

# ========================
# ROUTES
# ========================
@app.route("/")
def index():
    return send_from_directory("static", "index.html")

# ------------------------
# RESUME vs JOB ANALYSIS
# ------------------------
@app.route("/analyze", methods=["POST"])
```

```
def analyze():
    data = request.get_json() or {}
    resume_text = data.get("resume", "")
    job_text = data.get("job", "")

    resume_info = find_skills_with_snippets(resume_text)
    job_info = find_skills_with_snippets(job_text)

    resume_skills = list(resume_info.keys())
    job_skills = list(job_info.keys())

    matched = [s for s in job_skills if s in resume_skills]
    missing = [s for s in job_skills if s not in resume_skills]

    rule_score = int((len(matched) / len(job_skills)) * 100) if job_skills else 0
    ai_score = int(semantic_similarity(resume_text, job_text) * 100)

    matched_details = {s: resume_info[s] for s in matched}

    return jsonify({
        "rule_based_score": rule_score,
        "ai_semantic_score": ai_score,
        "matched_skills": matched,
```

```
        "missing_skills": missing,
        "matched_details": matched_details
    })

# ------------------------
# ANALYZE WITH PDF
# ------------------------
@app.route("/analyze-file", methods=["POST"])
def analyze_file():
    job_text = request.form.get("job", "")

    if "resume_file" in request.files and request.files["resume_file"].filename:
        resume_text = extract_text_from_pdf(request.files["resume_file"])
    else:
        resume_text = request.form.get("resume_text", "")

    resume_info = find_skills_with_snippets(resume_text)
    job_info = find_skills_with_snippets(job_text)

    resume_skills = list(resume_info.keys())
    job_skills = list(job_info.keys())

    matched = [s for s in job_skills if s in resume_skills]
```

```
    missing = [s for s in job_skills if s not in resume_skills]

    rule_score = int((len(matched) / len(job_skills)) * 100) if job_skills else 0
    ai_score = int(semantic_similarity(resume_text, job_text) * 100)

    matched_details = {s: resume_info[s] for s in matched}

    return jsonify({
        "rule_based_score": rule_score,
        "ai_semantic_score": ai_score,
        "matched_skills": matched,
        "missing_skills": missing,
        "matched_details": matched_details
    })

# ------------------------
# JOB RECOMMENDATION (PDF)
# ------------------------
@app.route("/recommend-jobs-file", methods=["POST"])
def recommend_jobs_file():
    if "resume_file" in request.files and request.files["resume_file"].filename:
        resume_text = extract_text_from_pdf(request.files["resume_file"])
    else:
```

```
    resume_text = request.form.get("resume_text", "")

resume_info = find_skills_with_snippets(resume_text)
resume_skills = list(resume_info.keys())

recommendations = []

for job in JOBS:
    must = job["must_have"]
    bonus = job["good_to_have"]

    matched_must = [s for s in must if s in resume_skills]
    matched_bonus = [s for s in bonus if s in resume_skills]
    missing_must = [s for s in must if s not in resume_skills]

    must_score = (len(matched_must) / len(must)) * 70 if must else 70
    bonus_score = (len(matched_bonus) / len(bonus)) * 30 if bonus else 0
    score = int(must_score + bonus_score)

    recommendations.append({
        "job_title": job["title"],
        "score": score,
        "matched_must_have": matched_must,
```

```
            "matched_good_to_have": matched_bonus,
            "missing_must_have": missing_must
        })

    recommendations.sort(key=lambda x: x["score"], reverse=True)
    return jsonify(recommendations[:3])

# ========================
# RUN SERVER
# ========================
if __name__ == "__main__":
    app.run(debug=True, port=5000)
```

CODE

jobs.py

```
JOBS = [
    {
        "id": 1,
        "title": "Backend Developer",
        "must_have": [
```

```
      "python",
      "flask",
      "sql"
    ],
    "good_to_have": [
      "docker",
      "git",
      "rest api"
    ]
  },
  {
    "id": 2,
    "title": "Data Analyst",
    "must_have": [
      "python",
      "pandas",
      "numpy"
    ],
    "good_to_have": [
      "sql",
      "machine learning"
    ]
  },
```

```
{
  "id": 3,
  "title": "DevOps Engineer",
  "must_have": [
    "docker",
    "aws"
  ],
  "good_to_have": [
    "kubernetes",
    "git",
    "python"
  ]
},
{
  "id": 4,
  "title": "Frontend Developer",
  "must_have": [
    "javascript",
    "html",
    "css"
  ],
  "good_to_have": [
    "react",
```

```
        "git"
      ]
    },
    {
      "id": 5,
      "title": "Full Stack Developer",
      "must_have": [
        "javascript",
        "python",
        "sql"
      ],
      "good_to_have": [
        "react",
        "flask",
        "docker"
      ]
    }
]
```


CODE

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>AI Resume Analyzer</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      background: #f4f6f8;
      padding: 30px;
    }
    h1 {
      text-align: center;
    }
    .container {
      max-width: 800px;
      margin: auto;
      background: white;
      padding: 25px;
      border-radius: 8px;
      box-shadow: 0 0 10px rgba(0,0,0,0.1);
```

```
    }
    textarea, input[type="file"] {
      width: 100%;
      margin-top: 8px;
      margin-bottom: 15px;
      padding: 10px;
    }
    button {
      padding: 10px 16px;
      margin-right: 10px;
      cursor: pointer;
    }
    pre {
      background: #eee;
      padding: 15px;
      white-space: pre-wrap;
      border-radius: 6px;
      margin-top: 15px;
    }
  </style>
</head>
<body>
```

```
<h1>AI Resume Analyzer & Job Recommendation</h1>

<div class="container">

<h3>Upload Resume (PDF)</h3>

<input type="file" id="resumeFile" accept=".pdf">

<h3>OR Paste Resume Text</h3>

<textarea id="resumeInput" rows="6" placeholder="Paste resume text here..."></textarea>

<h3>Paste Job Description</h3>

<textarea id="jobInput" rows="6" placeholder="Paste job description here..."></textarea>

<button id="analyzeBtn">Analyze Resume</button>

<button id="recommendBtn">Recommend Jobs</button>

<h3>Analysis Result</h3>

<pre id="resultBox">---</pre>

<h3>Job Recommendations</h3>

<pre id="recommendBox">---</pre>
```

```
  </div>

<script>
const analyzeBtn = document.getElementById("analyzeBtn");
const recommendBtn = document.getElementById("recommendBtn");
const resumeFile = document.getElementById("resumeFile");
const resumeInput = document.getElementById("resumeInput");
const jobInput = document.getElementById("jobInput");
const resultBox = document.getElementById("resultBox");
const recommendBox = document.getElementById("recommendBox");

// ---------- ANALYZE BUTTON ----------
analyzeBtn.addEventListener("click", async () => {
  resultBox.textContent = "Analyzing...";

  try {
    let response;

    // If PDF uploaded -> use /analyze-file
    if (resumeFile.files.length > 0) {
      const formData = new FormData();
      formData.append("resume_file", resumeFile.files[0]);
      formData.append("job", jobInput.value);
```

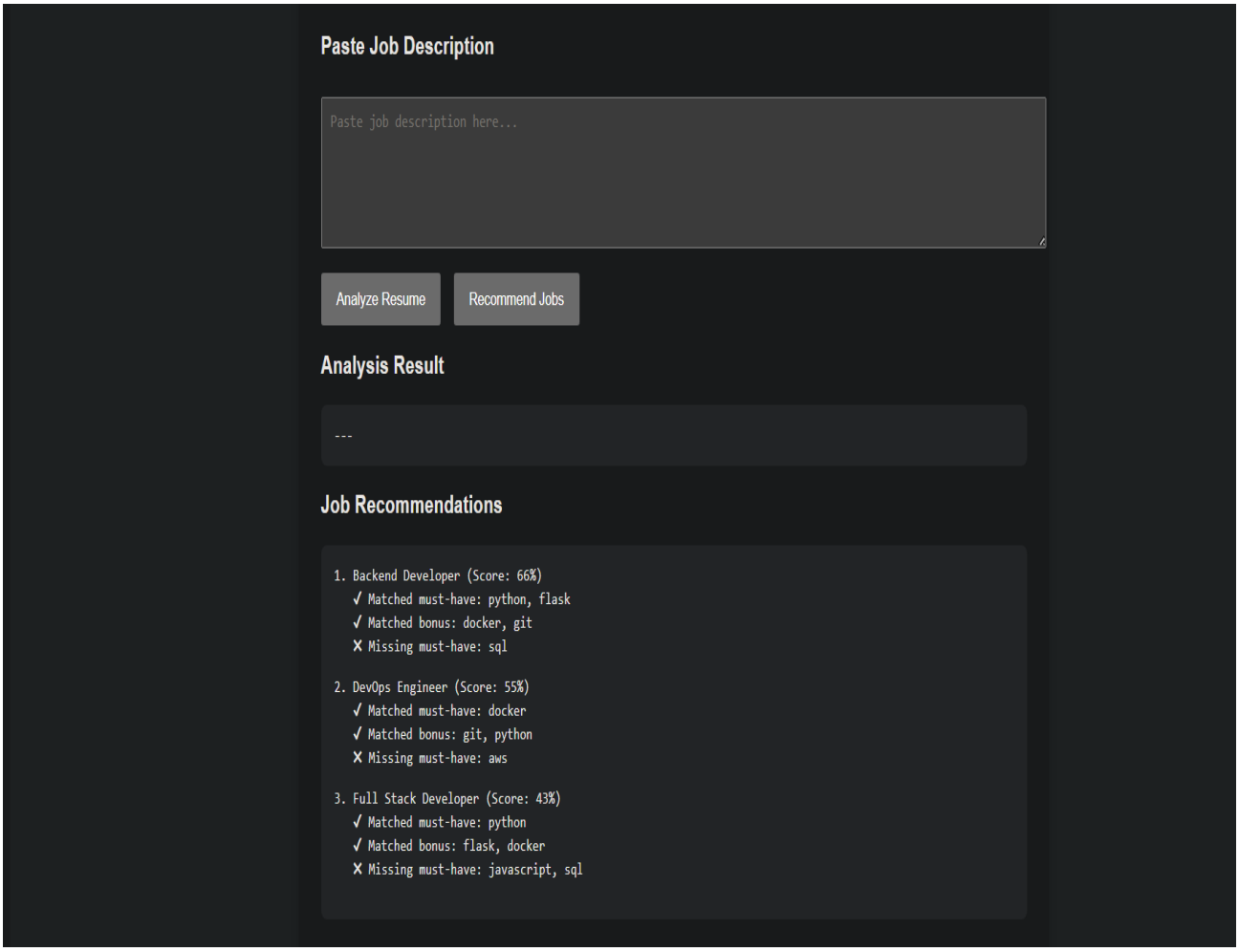
```
  response = await fetch("/analyze-file", {
    method: "POST",
    body: formData
  });
}
// Else -> use /analyze (JSON)
else {
  response = await fetch("/analyze", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({
      resume: resumeInput.value,
      job: jobInput.value
    })
  });
}

const data = await response.json();

resultBox.textContent =
  `Rule-based score: ${data.rule_based_score}%\n` +
  `AI semantic score: ${data.ai_semantic_score}%\n\n` +
```

```
    `Matched skills: ${data.matched_skills.join(", ") || "None"}\n` +
    `Missing skills: ${data.missing_skills.join(", ") || "None"}`;

  } catch (err) {
    resultBox.textContent = "Error: " + err.message;
  }
});

// ---------- RECOMMEND JOBS BUTTON ----------
recommendBtn.addEventListener("click", async () => {
  recommendBox.textContent = "Finding best jobs...";

  try {
    let response;

    if (resumeFile.files.length > 0) {
      const formData = new FormData();
      formData.append("resume_file", resumeFile.files[0]);

      response = await fetch("/recommend-jobs-file", {
        method: "POST",
        body: formData
      });
```

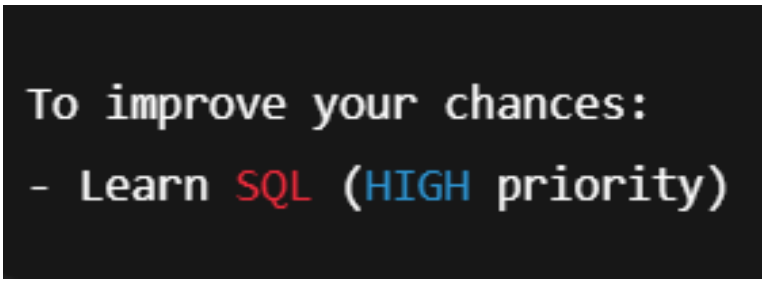
```
  } else {
    recommendBox.textContent = "Please upload a resume PDF for job recommendations.";
    return;
  }

  const jobs = await response.json();

  if (!jobs || jobs.length === 0) {
    recommendBox.textContent = "No suitable jobs found.";
    return;
  }

  let out = "";
  jobs.forEach((job, idx) => {
    out += `${idx + 1}. ${job.job_title} (Score: ${job.score}%)\n`;
    out += `   ✓ Matched must-have: ${job.matched_must_have.join(", ") || "None"}\n`;
    out += `   ✓ Matched bonus: ${job.matched_good_to_have.join(", ") || "None"}\n`;
    out += `   ✗ Missing must-have: ${job.missing_must_have.join(", ") || "None"}\n\n`;
  });
```

```
    recommendBox.textContent = out;

  } catch (err) {
    recommendBox.textContent = "Error: " + err.message;
  }
});
</script>

</body>
</html>
```

OUTPUT


Paste Job Description
Paste job description here...
Analyze Resume
Recommend Jobs
Analysis Result

Job Recommendations
1. Backend Developer (Score: 66%)
✓ Matched must-have: python, flask
✓ Matched bonus: docker, git
✗ Missing must-have: sql
2. DevOps Engineer (Score: 55%)
✓ Matched must-have: docker
✓ Matched bonus: git, python
✗ Missing must-have: aws
3. Full Stack Developer (Score: 43%)
✓ Matched must-have: python
✓ Matched bonus: flask, docker
✗ Missing must-have: javascript, sql


To improve your chances:
- Learn SQL (HIGH priority)

Tech Stack Used

The AI Resume Analyzer and Job Recommendation System is developed using the following technologies:

Python is used as the primary programming language for implementing backend logic due to its simplicity and strong support for text processing and AI integration.

Flask is used as a lightweight backend web framework to create RESTful APIs and handle communication between the frontend and backend.

HTML, CSS, and JavaScript are used to build the frontend interface, allowing users to upload resumes, enter job descriptions, and view analysis results dynamically.

PyPDF2 is used to extract text from uploaded PDF resumes, enabling support for real-world resume formats.

Regular Expressions (Regex) are used for rule-based skill extraction from resume and job text, ensuring transparent and explainable matching.

OpenAI Embeddings API is integrated to perform semantic similarity analysis between resumes and job descriptions, improving contextual matching beyond exact keywords.

Virtual Environment (venv) and requirements.txt are used for dependency management and environment isolation.

Visual Studio Code is used as the development environment, and the project is tested on the Windows operating system.

Conclusion

The AI Resume Analyzer and Job Recommendation System successfully demonstrates how modern resume screening and job matching can be automated using a combination of rule-based techniques and AI-assisted analysis. The system effectively processes resumes in PDF or text format, extracts relevant skills, compares them with job requirements, and provides meaningful insights to users.

By integrating transparent rule-based skill matching with semantic similarity analysis, the project ensures both accuracy and explainability in resume evaluation. The job recommendation and resume improvement features further enhance the usefulness of the system by guiding users toward suitable roles and identifying skill gaps.

Overall, the project simulates real-world recruitment and Applicant Tracking System (ATS) workflows while maintaining simplicity, reliability, and scalability. It serves as a strong foundation for further enhancements such as real-time job portal integration, advanced AI models, and personalized career guidance, making it a practical and valuable tool for students and job seekers.