Resume–Role Fit Evaluator

2. Objective

Develop a microservice that:

1. Scores how well a candidate's **resume fits** a specific job role
2. Identifies **missing skills** or qualifications
3. Suggests a **detailed, step-by-step learning path** to become job-ready

The system must be:

- Fully offline
- Explainable and configurable
- Built using **FastAPI**, return structured JSON, and deployed via Docker

3. Functional Requirements

A. Input

- **Endpoint**: `POST /evaluate-fit`

```
json

{
    "resume_text": "I've worked on Django apps, SQL databases, and basic cloud deployments. Familiar with Python scripting and Flask.",
  "job_description": "We are hiring backend engineers with expertise in Node.js, MongoDB, containerized deployments (Docker), and AWS. Bonus for experience in system design."
}
```

B. Output

```
json


{
```

```
  "fit_score": 0.46,
  "verdict": "moderate_fit",
  "matched_skills": ["Python", "Cloud Basics"],
  "missing_skills": ["Node.js", "MongoDB", "Docker", "AWS",
"System Design"],
  "recommended_learning_track": [
    {
      "skill": "Node.js",
      "steps": [
        "Install Node.js and learn basic syntax",
        "Understand asynchronous programming in JS",
        "Build a REST API with Express.js",
        "Handle authentication and routing"
      ]
    },
    {
      "skill": "Docker",
      "steps": [
        "Understand containers vs virtual machines",
        "Install Docker CLI and Docker Desktop",
        "Write a Dockerfile for a simple app",
        "Build and run Docker containers locally"
      ]
    }
  ],
  "status": "success"
}
```


4. AI Model Requirements

A. Model Type

- Use **TF-IDF + Cosine Similarity** or **sentence embeddings**
- Extract skill keywords from:
    - Resume text
    - Job description
    - Skills config file

B. Learning Path Generator

- Match missing skills to detailed, **predefined track JSON**
- Example:

```
json

{
  "docker": {
    "steps": [
      "Understand containers vs virtual machines",
      "Install Docker CLI and Docker Desktop",
      "Write a Dockerfile for a simple app",
      "Build and run Docker containers locally"
    ]
  }
}
```

- Learning path must be sourced from `learning_paths.json`

5. Dataset

- Resume samples (at least 50)
- Job descriptions from real tech roles
- `skills.json`: skill keywords to normalize synonyms
- `learning_paths.json`: skill → [step 1, step 2, …]

6. Software Architecture Requirements

- Pipeline:
    1. Clean input text
    2. Extract skills
    3. Compute embedding similarity
    4. Determine missing skills
    5. Map gaps to learning steps

- `config.json` must include:
    1. Fit score cutoffs (e.g. < 0.4 = weak_fit)
    2. Skill alias map (e.g., “Amazon Web Services” = “AWS”)
    3. Max steps per skill (default 4)

7. API Endpoints

Method	Endpoint	Description
POST	`/evaluate-fit`	Resume vs job role → score + steps
GET	`/health`	Returns `{ "status": "ok" }`
GET	`/version`	Returns `{ "model_version": "1.0.0" }`

8. FastAPI Requirement

- Entire app must be built using **FastAPI**
- All inputs and outputs validated via **Pydantic**
- Hosted with **Uvicorn on port 8000**
- Swagger auto-docs available at `/docs`

9. Testing & Validation

Must test:

- Resume 90% match → `strong_fit`, no track
- Resume 50% match → `moderate_fit`, 2–3 tracks
- Resume 20% match → `weak_fit`, 4+ detailed tracks
- Resume mentions “Amazon Web Services” → normalized to `AWS`

Each response must include:

- `fit_score` (float)
- `verdict` (enum)
- `missing_skills` (string[])
- `recommended_learning_track` (detailed steps)

10. Deliverables

- Source code with:
  - `main.py`, `fit_score_engine.py`, `skill_extractor.py`
- `skills.json` and `learning_paths.json`
- Sample resumes + job descriptions
- `README.md` with API samples and Docker usage
- Health and version endpoints
- Unit tests covering all major flows

11. Timeline (6 Weeks)

Week	Tasks
Week 1	Gather resumes + job postings, define skills and tracks
Week 2	Build skill extractor and matching logic
Week 3	Train scoring engine and validate cutoffs
Week 4	Build FastAPI service and response schema
Week 5	Integrate learning path generator
Week 6	Finalize Docker + test coverage

12. Constraints

- No GPT or external APIs
- Must work fully offline
- All skill mappings and learning tracks must be configurable
- Input: only structured JSON
- Output: score, reason, gaps, and guided steps

13. Deployment Expectations

- Fully Dockerized
- Must bundle:
  - Model/vectorizer
  - `skills.json`
  - `learning_paths.json`
- Must support:

```
docker build -t resume-fit-evaluator .
docker run -p 8000:8000 resume-fit-evaluator
```