# PROJECT REPORT

**TURTIL CO.**

## RESUME SCORING MICROSERVICE

**PREPARED FOR:**

Raj Shanmugam
Head - Mentor Group
TURTIL GLOBAL
mentor@turtil.co

**PREPARED BY:**

Sourav Mahapatra
Team : Parallax Pioneers
+91 – 94391 70282
souravmahapatra@turtilintern.com

# ABSTRACT

As part of my internship at **Turtil Co.**, I developed a production-ready, containerized **Resume Scoring Microservice** to automate the evaluation of student resumes against distinct career goals—namely **Amazon SDE**, **ML Internship**, and **GATE ECE**. The goal was to build a smart, configurable, and offline-capable system that provides meaningful, skill-based feedback to students preparing for competitive roles and exams.

The microservice integrates **TF-IDF vectorization** with a **Logistic Regression model** to generate relevance scores, while using **Natural Language Processing (NLP)** to extract and match skills from unstructured text. It goes beyond keyword detection by implementing **skill grouping**, **alternate skill mapping**, and **learning path generation**—offering detailed insights including match scores, missing skills, grouped domains, and personalized upskilling suggestions.

Built using **FastAPI**, the system is exposed as a clean RESTful API and is fully **Dockerized** for cross-environment deployment. All goal-specific logic, skill mappings, and output behaviors are handled through modular **JSON configuration files**, enabling quick iteration and scalability. The microservice is also equipped with **robust error handling**, **unit testing**, and **cross-domain augmentation**, ensuring high accuracy and stability in real-world usage.

The model achieved **99–100% accuracy** on curated training data and met all functional, architectural, and quality benchmarks defined by Turtil's internship framework. This project reflects a scalable approach to AI-powered resume evaluation and can be extended to support additional career tracks or integrated into broader career guidance platforms.

# Table of Contents

# 1. Introduction

The Resume Scoring Microservice is a portable, offline-capable service that evaluates resumes against predefined career goals, providing a numerical score (0-1), lists of matched/missing skills, skill groups, and a learning path for skill gaps. Developed using Python, FastAPI, scikit-learn, NLTK, and Docker, it ensures flexibility and scalability. As an intern at Turtil Co., I contributed to data curation, model training, API development, skill logic, and containerization.

Key objectives:

- Train machine learning models for three career goals.
- Develop a RESTful API for resume evaluation.
- Implement NLP-driven skill matching with fuzzy logic.
- Ensure offline operation via Dockerization.

# 2. Approach

## 2.1 Data Preparation

A synthetic dataset of 612 resumes was curated (204 per goal: "Amazon SDE," "ML Internship," "GATE ECE"). Each resume was labeled as suitable (1) or unsuitable (0) based on skills defined in goals.json. Skills were sourced from industry job descriptions and GATE syllabi:

- Amazon SDE: 47 skills (e.g., Java, Data Structures, SQL).
- ML Internship: 43 skills (e.g., Python, NumPy, Scikit-learn).
- GATE ECE: 46 skills (e.g., Digital Electronics, Signals and Systems).

The dataset is balanced (50% suitable, 50% unsuitable) to prevent bias. Data cleaning involved lowercasing, removing special characters, and tokenization using NLTK. Dataset Availability: The dataset is included in the submission ZIP under data/resume_dataset.csv.

## 2.2 Model Selection and Training

Model Choice: Logistic Regression was selected over alternatives (e.g., SVM, Random Forest) for the following reasons:

- Interpretability: Logistic Regression provides clear feature importance (via coefficients), aiding skill gap analysis.
- Efficiency: Low computational cost for training and inference, ideal for offline deployment.
- Performance: Achieved 99-100% accuracy on TF-IDF vectorized data, comparable to complex models.

Other models were evaluated:

- SVM: Slower training, marginal accuracy gain (99.5%).
- Random Forest: Overfitting risk, longer inference time.

Training Process: Resumes were vectorized using TF-IDF (scikit-learn's TfidfVectorizer, max_features=5000). Separate Logistic Regression models were trained per goal with hyperparameters tuned via grid search (C=0.1, max_iter=200). Models and vectorizers were serialized as .pkl files. Test accuracies:

- Amazon SDE: 100%.
- ML Internship: 99.02%.
- GATE ECE: 99.02%.

### 2.3 Skill Logic

Skills are defined in goals.json (136 total) and alternate_skills.json (6 skills with 3-10 variants, e.g., "Machine Learning" includes "ML," "Statistical Learning"). Skill matching uses:

- Fuzzy Matching: FuzzyWuzzy's token-set ratio (threshold=80) to handle wording variations.
- Lemmatization: NLTK's WordNetLemmatizer to normalize skill terms.

The service identifies matched skills, missing skills, and suggests learning paths from suggestions.json. Skill Grouping: Skills are categorized (e.g., "Programming," "Electronics") in skill_groups.json for structured output.

### 2.4 API Development

A RESTful API was built using FastAPI, with endpoints:

- POST /score: Accepts student_id, goal, and resume_text; returns score, matched/missing skills, skill groups, and learning path.
- GET /health: Returns {"status": "ok"}.
- GET /version: Returns service metadata (version, supported goals).

Input validation uses Pydantic models (schema.json). Responses are JSON-compliant with <1.5s latency.

### 2.5 Configuration

The config.json file ensures enterprise-grade configurability:

- minimum_score_to_pass: Threshold (e.g., 0.6).
- model_goals_supported: Supported goals.
- log_score_details: Enables detailed logging.

The service validates config.json at startup, raising descriptive errors for invalid configurations.

### 2.6 Dockerization

The service is containerized using python:3.10-slim, including dependencies, models, and JSON files. Key Dockerfile commands:

- Building: docker build -t resume-scoring-service.
- Running: docker run -p 8000:8000 resume-scoring-service

Port 8000 is exposed, and the application is launched via Uvicorn. No external dependencies ensure offline capability.

# 3. Implementation Details

### 3.1 Project Structure

The project follows a modular design:

- app/main.py: FastAPI server and endpoint definitions.
- app/scorer.py: Model loading, scoring, and skill matching logic.
- model/: Pickled models and vectorizers.

- data/: JSON configs and dataset.
- tests/test_score.py: Unit tests using pytest.

**Table 1: Key Project Files**

| File/Folder | Description |
|---|---|
| main.py | FastAPI server |
| scorer.py | Scoring and skill logic |
| model/ | Pickled models/vectorizers |
| data/goals.json | Skill definitions |
| data/alternate_skills.json | Skill variants |
| config.json | Runtime configurations |
| Dockerfile | Container build |
| tests/test_score.py | Unit tests |

### 3.2 Enterprise-Level Code Practices

- No Hardcoding: All thresholds, file paths, and goals are configurable via config.json.
- Modularity: Separated concerns (API, scoring, skill matching).
- Error Handling: Graceful handling of invalid inputs, missing files, and unsupported goals.
- Logging: Configurable logging for debugging and auditing.

# 4. Testing

### 4.1 Unit Tests

Unit tests in test_score.py (using pytest) cover:

- Valid inputs: Correct score, skills, and learning path.
- Edge cases: Empty resumes, invalid goals, malformed JSON.
- Configuration errors: Invalid config.json.

Test coverage: >90% (verified via pytest-cov).

### 4.2 Validation

Scenarios tested:

- High-scoring resume: Correct high score, accurate skill insights.
- Low-scoring resume: Low score, identified missing skills.

- Edge cases: Appropriate error messages (e.g., "Resume text cannot be empty").

Response time: <1.5s. JSON structure validated using JSON Schema.

**Table 2: Test Scenarios**

| Scenario | Outcome |
|---|---|
| High-Scoring Resume | High score, correct skills |
| Low-Scoring Resume | Low score, missing skills |
| Empty Resume | Error: "Resume text cannot be empty" |
| Invalid Goal | Error: "Unsupported goal" |
| Malformed JSON | Error: Invalid input |

# 5. Challenges and Solutions

- Challenge: Realistic resume data curation. Solution: Researched job descriptions and GATE syllabus for diverse, relevant samples.
- Challenge: Skill matching accuracy due to wording variations. Solution: Implemented fuzzy matching and alternate skill names.
- Challenge: Model underperformance. Solution: Tuned hyperparameters and increased dataset quality.

# 6. Conclusion

The Resume Scoring Microservice meets all objectives:

- 612-sample dataset across three goals.
- 136 skills, 6 with alternate names.
- 99-100% model accuracies.
- Enterprise-grade code with configurability and robust testing.

Future enhancements include:

- Expanding the dataset with real resumes.
- Advanced NLP (e.g., BERT for contextual skill matching).
- Supporting additional career goals.