AI Resume Intelligence System

Clean Project Progress Summary

1. Project Goal

Build a serious AI-powered Resume Intelligence System using Flask + PostgreSQL. Implement real NLP and ML (not rule-based shortcuts). Follow clean architecture and professional Git workflow.

2. Backend Foundation

- Built Flask backend with PostgreSQL integration. - Implemented full CRUD operations. - Designed resumes table with structured schema. - Stored raw resume_text for ML flexibility.

3. Resume Dataset Engineering

- Initially tested with 8 diverse resumes. - Built automated synthetic resume generator. - Expanded dataset to 240 resumes. - 8 domains (30 per domain): Data Scientist, ML Engineer, Backend Python Developer, Frontend React Developer, DevOps Engineer, Cloud Engineer, Cybersecurity Analyst, Full Stack Developer. - Introduced controlled cross-domain skill noise (realistic overlap).

4. NLP & Feature Engineering

- Extracted resume_text from PostgreSQL. - Built TF-IDF Vectorizer. - Vocabulary size increased from 119 $\rightarrow$ 339+ features. - Converted text into numerical vector space representation.

5. Machine Learning Evolution

Phase 1 – Weak Supervision: - Generated labels using cosine similarity threshold. - Observed class imbalance and precision-recall tradeoff. Phase 2 – True Supervised Learning: - Added domain column to resumes table. - Switched to ground-truth labeling. - Binary classification: Data Scientist vs Others.

6. Logistic Regression Training

- Performed train/test split (80/20). - Used class_weight="balanced" to handle imbalance. - Evaluated using: Accuracy Precision Recall F1 Score Confusion Matrix - Achieved realistic performance after introducing domain overlap noise.

7. Engineering Improvements

- Clean architecture separation: generate_resumes.py generate_training_data.py train_logistic_model.py - Proper Git branch hygiene. - Controlled dataset regeneration. - Designed system scalable for multi-class classification.

8. Current System Capabilities

- 240 structured resumes stored in PostgreSQL. - TF-IDF feature extraction from real database corpus. - Supervised Logistic Regression model. - Realistic evaluation metrics. - Production-ready architecture for model saving and API integration.

Next Steps (Planned)

- Save trained model and vectorizer (model.pkl). - Integrate ML predictions into Flask API. - Build React dashboard frontend. - Dockerize and deploy to cloud.