**Product Requirements Document (PRD): Smart Agent Routing in TalentVibe v2.0**

### **Project Title**: Smart Agent Routing Pipeline for Resume Scoring

### **Owner**: TalentVibe Engineering

### **Status**: Draft for Implementation

### **Overview**:

Implement a modular, intelligent orchestration pipeline in TalentVibe v2.0 that dynamically decides which AI agents (e.g., Recency Agent, Depth Agent, Explainability Agent) should run on a given resume based on predefined logic. This makes resume processing faster and more cost-efficient at scale.

### **Goals**:

* Optimize resume scoring throughput by skipping unnecessary agents.
* Reduce API/model call costs.
* Keep each agent function modular and testable.
* Enable agent invocation logic to be updated/configurable in the future.
* Support parallel, scalable resume processing at batch scale.

### **User Workflow Summary**:

1. User uploads 1 JD and 100 resumes.
2. System parses all resumes using MegaParser and Llama Parse.
3. A lightweight Skill Matching Agent is always executed.
4. Based on results from skill matching, other agents may or may not be run.
5. Final output is returned per resume with a score, optional explanations, and agent traces.

### **Feature Requirements**:

#### **1. ResumeProcessingPipeline Class**

* Inputs: parsed\_resume, jd\_context
* Outputs: results dict
* Submethods:
  + run\_skill\_matching()
  + should\_run\_recency() + run\_recency\_agent()
  + should\_run\_depth\_agent() + run\_depth\_agent()
  + should\_run\_explainability() + run\_explainability\_agent()
  + get\_results() — collect all results and include skipped agents as "skipped": true

#### **2. Skill Matching Agent (Always Run)**

* Compares resume skills with JD skills.
* Returns match score, matched skills, and missing skills.

#### **3. Recency Agent**

* Runs only if:
  + Skill match ≥ 60%
  + Resume has experience section
* Extracts years used per skill and scores freshness.

#### **4. Depth & Impact Agent**

* Runs only if:
  + Resume has 2+ job entries or 3+ project entries
* Scores experience quality, impact indicators, leadership signals.

#### **5. Explainability Agent (Optional)**

* Runs only for top 25 resumes by score.
* Returns reasoning for score decisions using LLM prompt templates.

#### **6. Early Exit Logic**

* If Skill Matching < 50%, skip all other agents and return "low\_match\_cutoff": true

#### **7. Configuration Support (Optional)**

* Rules can eventually be defined in a YAML or JSON config file.
* Each agent should support a run() interface and should\_run() checker.

#### **8. Parallel Execution & Scalability**

* Resume uploads should be processed in parallel using a task queue like Celery.
* Each resume’s processing pipeline is encapsulated in its own task.
* Agents may be executed sequentially or in parallel depending on resource needs.
* Consider future upgrades where agents can run as isolated microservices.
* Resource-heavy agents (e.g., LLM-based) should support throttling or scheduling.
* Use job metadata or skill match thresholds to decide when to batch, skip, or defer work.

### **Non-Goals**:

* This PRD does not define UI changes or job queue prioritization logic.

### **Deliverables**:

* ResumeProcessingPipeline class (Python)
* Unit tests for each agent and should\_run logic
* Integration test for full resume flow

### **Milestones**:

1. Scaffold pipeline class (resume parser + stub agents)
2. Implement skill matching agent (always-on)
3. Implement conditional agents (recency, depth, explainability)
4. Integrate early exit logic
5. Add test coverage and sample resume cases

### **Notes for Cursor / VibeCoding Tool**:

* Break down each submethod and agent into isolated tasks.
* Ensure each agent is unit-testable with mocked resume input.
* Use placeholder agent logic; real LLM calls will be integrated later.
* Prioritize modularity and maintainability.
* Optimize for concurrency and scalability — each resume and agent process should be independently executable where feasible.