AI-Powered Job-Candidate Matching System

Technical Report

Course: Advanced AI/ML Project
Team: Group B
Date: January 2026

---

1. Introduction & Motivation

1.1 Problem Statement

The recruitment industry faces a significant challenge: efficiently matching job seekers with suitable positions from an ever-growing pool of candidates and job openings. Traditional keyword-based matching systems fail to capture the semantic nuances of professional qualifications, leading to suboptimal matches and increased manual review time for recruiters.

1.2 Motivation

This project aims to develop an intelligent job-candidate matching system that leverages:

- **Large Language Models (LLMs)** for understanding unstructured resume and job description content
- **Retrieval-Augmented Generation (RAG)** for semantic search capabilities
- **Agentic Architecture** for orchestrating complex multi-step matching workflows

The goal is to automate the extraction of structured information from PDF documents and provide semantically-aware candidate recommendations that go beyond simple keyword matching.

1.3 Key Objectives

1. Extract structured data from unstructured PDF resumes and job descriptions
2. Enable semantic search using vector embeddings
3. Implement a sophisticated multi-factor matching algorithm
4. Provide an API-driven system for integration with recruitment workflows

---

2. Dataset and Preprocessing

2.1 Dataset Description

The system utilizes two primary data sources:

Dataset	Description	Format
Resume_Dataset	Professional resumes across 24 categories (Accountant, IT, Healthcare, etc.)	PDF files
Jobs Positions	Job descriptions for accounting positions	PDF files

The Resume_Dataset contains categorized resumes spanning diverse professional domains including: Accountant, Advocate, Agriculture, Apparel, Arts, Automobile, Aviation, Banking, BPO, Business Development, Chef, Construction, Consultant, Designer, Digital Media, Engineering, Finance, Fitness, Healthcare, HR, Information Technology, Public Relations, Sales, and Teacher.

2.2 Preprocessing Pipeline

The preprocessing workflow follows a structured RAG ingestion pipeline:

```
PDF Document → Text Extraction → LLM Structuring → Vector Embedding → Dual Storage
```

Step 1: Text Extraction
PDF documents are parsed using the `pdf-parse` library, extracting raw text content while handling multi-page documents and removing control characters that could interfere with downstream processing.

Step 2: LLM-Based Structuring
Raw text is processed by Google Gemini 2.5 Flash Lite to extract structured information:

- **For Candidates:** Name, email, phone, skills (with proficiency levels), work experience, education, certifications, and professional summary
- **For Jobs:** Title, company, location, employment type, required/preferred skills, salary range, and job description summary

Step 3: Vector Embedding
Professional summaries are converted to 768-dimensional vectors using Google's `text-embedding-004` model, enabling semantic similarity search.

Step 4: Dual Storage

- **PostgreSQL:** Stores structured data for SQL-based filtering and queries
- **Qdrant Vector Database:** Stores embeddings for semantic similarity search

---

3. System Architecture

3.1 Agentic Pipeline Design

The system implements a multi-agent architecture with specialized agents handling different aspects of the matching workflow:

```
                    ┌──────────────────────────┐
                    │   OrchestratorAgent      │
                    │   (Task Coordinator)     │
                    └────────────┬─────────────┘
                                 │
                                 │
              ┌──────────────────┼──────────────────┐
              │                  │                  │
              ▼                  ▼                  ▼
       ┌─────────────────┐ ┌─────────────────┐ ┌─────────────────┐
```

```
    │  Candidate        │ │  Job Processing │ │  Future Agents  │
    │  Ingestion        │ │  Agent          │ │  (Extensible)   │
    │  Agent            │ │                 │ │                 │
    └───────────────────┘ └─────────────────┘ └─────────────────┘
```

OrchestratorAgent: Routes incoming tasks to appropriate worker agents based on task type, managing the overall workflow coordination.

CandidateIngestionAgent: Handles the complete RAG pipeline for CV processing—from PDF parsing through LLM extraction to dual storage in PostgreSQL and Qdrant.

JobProcessingAgent: Processes job descriptions and executes the dual-search matching strategy to find suitable candidates.

3.2 RAG Implementation

The RAG system combines two search strategies for comprehensive candidate matching:

Strategy	Method	Strength
SQL Search	PostgreSQL queries on structured fields	Precise filtering (skills, experience years, location)
Vector Search	Qdrant cosine similarity	Semantic understanding of professional profiles

Candidates found through both strategies receive a "dual match" bonus, as this indicates strong alignment on both explicit criteria and semantic relevance.

3.3 Technology Stack

Component	Technology	Purpose
Backend Framework	NestJS + TypeScript	API server and dependency injection
LLM	Google Gemini 2.5 Flash Lite	Structured data extraction
Embeddings	Google text-embedding-004	768-dim semantic vectors
Vector Database	Qdrant	Similarity search with filtering
Relational Database	PostgreSQL	Structured data storage
Containerization	Docker Compose	Development environment

3.4 Matching Algorithm

The sophisticated matching algorithm evaluates candidates using a weighted multi-factor approach:

Factor	Weight	Description
Skill Match	35%	Required vs. optional skills presence

Factor	Weight	Description
Skill Proficiency	15%	Experience level alignment
Experience Years	20%	Total years vs. requirements
Location Match	10%	Geographic compatibility
Vector Similarity	15%	Semantic profile relevance
SQL Match Bonus	5%	Found via structured query

The algorithm includes:

- **Fuzzy skill matching:** Handles abbreviations (JS↔JavaScript, K8s↔Kubernetes)
- **Overqualification penalty:** Reduces score for significantly overqualified candidates
- **Missing skill detection:** Tracks required skills not found in candidate profile

---

4. Results and Analysis

4.1 System Capabilities

The implemented system successfully demonstrates:

1. **Automated PDF Processing:** Extracts and structures information from diverse resume formats
2. **Semantic Search:** Finds candidates with similar professional profiles even when exact keywords differ
3. **Multi-Factor Scoring:** Provides nuanced matching beyond simple keyword overlap
4. **Scalable Architecture:** Agent-based design allows adding new capabilities without restructuring

4.2 Real Matching Results

To validate the system, we tested with two different job positions from our dataset:

Test 1: Accounting Operations Manager

Metric	Value
Top Score	64/100
Best Skill Match	6/9 required skills (67%)
SQL Matches	19 candidates
Vector Matches	20 candidates
Dual Matches	4 candidates

Analysis: Moderate scores (60-64) reflect missing specialized skills like "Regulatory Compliance" and "Process Improvement" in candidate pool.

Test 2: Medical Billing Specialist

Metric	Value

Metric	Value
Top Score	81/100
Best Skill Match	9/10 required skills (90%)
SQL Matches	20 candidates
Vector Matches	20 candidates
Dual Matches	1 candidate

Analysis: Higher scores (72-81) indicate better alignment with common accounting skills. Notably, the top candidate was found **only through vector search**—SQL alone would have missed them.

Key Finding: The score variance (64 vs 81) proves the algorithm evaluates actual skill alignment rather than producing default scores. Different job requirements yield meaningfully different results.

4.3 API Endpoints

Endpoint	Method	Function
`/candidates/upload`	POST	Upload and process CV (PDF)
`/candidates/process-folder`	POST	Batch process resume folder
`/job-offers/match`	POST	Upload job description and find matches
`/job-offers/process-and-match`	POST	Process job PDF and return candidates

4.4 Limitations

1. **Language Dependency:** Currently optimized for English-language documents
2. **PDF Quality Sensitivity:** Text extraction quality depends on PDF formatting
3. **Cold Start:** New categories require initial data ingestion before meaningful matching
4. **LLM API Dependency:** Requires internet connectivity and API quotas
5. **No Learning Loop:** System doesn't learn from recruiter feedback on match quality

---

5. Conclusion and Future Improvements

5.1 Conclusion

This project successfully demonstrates an AI-powered job-candidate matching system that combines the strengths of Large Language Models for understanding unstructured documents with RAG-based semantic search for finding relevant matches. The agentic architecture provides a flexible, extensible framework for handling complex recruitment workflows.

Key achievements:

- End-to-end pipeline from PDF ingestion to ranked candidate recommendations
- Dual-search strategy combining precision filtering with semantic understanding
- Sophisticated multi-factor scoring algorithm with explainable results

- Clean, modular codebase ready for production extension

5.2 Future Improvements

Improvement	Description
Feedback Learning	Incorporate recruiter feedback to improve matching weights
Multi-language Support	Extend to non-English resumes and job descriptions
Interview Scheduling Agent	Add agent for automated interview coordination
Skill Ontology	Implement hierarchical skill relationships (Python → Programming → Technical)
Candidate Ranking Explanation	Provide detailed reasoning for each match score
Real-time Updates	Implement WebSocket notifications for new matches
Fine-tuned Embeddings	Train domain-specific embedding model on recruitment data

---

References

1. Google Generative AI Documentation - Gemini API
2. Qdrant Vector Database Documentation
3. NestJS Framework Documentation
4. LangChain RAG Patterns and Best Practices

This report documents the AI Job-Candidate Matching System developed as part of Niv Arad project.