International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 04 | April - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

Job Search Agent

Mrs. P. Nandini[1], U. Akhila[2], Shahwar Mahmood[2], G. Anushka[2]

[1]Assistant Professor, Department of Computer Science and Engineering, Methodist College of Engineering and Technology, Hyderabad, India.

[2]B.E. Student, Department of Computer Science, Methodist College of Engineering and Technology, Hyderabad, India.

Abstract

The current job market is very competitive, making it harder and harder for candidates to find jobs suitable for their skill set. Conventional job searching involves candidates visiting various websites, evaluating if the job suits them, and applying. This manual process is cumbersome and inefficient, and many people miss out on great opportunities as a result. In this paper, I present Resume Match, a system using AI agents to analyze resumes and match them with relevant jobs. Resume Match uses the OpenAI Agents SDK on the Nebius AI Studio platform, where LLMs can be deployed, to develop a multi-agent structure for structured parsing of resumes, classifying them by domain, extracting skills, and recommending jobs based on the candidate's profile. Resume Match integrates with several job board APIs such as Remotive, Jobicy, Arbeitnow, The Muse, and Himalayas, and also provides portal links to LinkedIn, Indeed, Glassdoor, and other popular job search websites. The output from this application is ranked job listings based on relevance scores calculated via an algorithm and displayed in a Streamlit interface.

Index Terms *— Agentic AI, Large Language Models (LLM), Resume Parsing, Job Matching, Multi-Agent Systems, OpenAI Agents SDK, Nebius AI Studio, Career Recommendation.*

I. INTRODUCTION

The emergence of large language models and agentic AI technologies has presented unprecedented opportunities in many different sectors, one of which is recruitment. Millions of individuals apply for jobs annually, even though such jobs do not necessarily fit their experience or career path. This is mainly due to the fact that the resume-to-job matching process is manual, cumbersome, and ineffective. On one hand, job-seekers simply cannot afford the necessary resources to assess their suitability for the job; on the other hand, recruiters receive huge numbers of job applications, many of which do not match their requirements. Current job-finding websites utilize keyword search. The above methods fail to consider the semantic richness of the candidate's profile, relevance of the position, and the mismatch between the skills mentioned in the profile and those that may be inferred. There is thus a requirement for a solution that provides deeper insight into the candidate's professional journey and selects the most relevant job options based on that analysis.

Resume Match meets this requirement through a completely autonomous pipeline of multiple AI agents that will process the resumes, create structured career profiles, determine the relevance of positions for the candidate, and provide recommendations on jobs that best match their profiles – all this from just a single resume upload!

A. Background and Motivation

Large language models have been widely adopted in recent times for use in information extraction and reasoning. Some of these models include LLaMA-3, GPT-4, and Mistral, which have shown proficiency in comprehending the structure of documents, entity extraction, intent classification, and generating sensible, contextual text. The OpenAI Agents SDK provides a platform for integrating these models to work as autonomous agents, each with its own function and instructions.

The Nebius AI Studio offers hosted access to cutting-edge language models such as the Llama-3.3-70B-Instruct from Meta. This helps generate high-quality inference and generation services without the need for setting up infrastructure. Both these platforms form a strong base for developing an intelligent and multi-step agent pipeline.

The rationale behind the idea of Resume Match is based on how fragmented and time-consuming the process of searching for jobs still is despite the availability of numerous platforms online. The applicant needs to come up with his or her search criteria, analyze each position independently, and find those that really suit him or her. Resume Match was created to solve this issue.

B. Problem Statement

Even with the advent of several job portals and resume managers, there is no system available that encompasses an end-to-end agentic pipeline which: (1) analyzes a candidate's resume deeply by employing the capabilities of LLMs, (2) classifies a candidate's field and profession automatically, (3) searches from several live job portals simultaneously, (4) ranks the results based on a skill-based relevance score, and (5) suggests personalized career advice.

C. Objectives of the Study

The primary objectives of this work are as follows:

- To design and implement a multi-agent AI pipeline capable of parsing resumes and extracting structured professional profiles.
- To develop an automated domain and role classification mechanism using LLM-based agents.
- To integrate with multiple live job board APIs and generate relevance-scored job listings.
- To provide personalized career advisory outputs through a recommendation agent.
- To deliver all functionality through a user-friendly web interface built with Streamlit.

II. EXISTING SYSTEM

There are several tools available today which try to solve the issue of job matching and resume evaluation. This includes conventional job boards based on keywords to early machine learning algorithms for recommendation. Each of these solutions suffers from serious constraints which do not allow them to become intelligent and personalized matching systems.

A. Overview of Existing Approaches

For LinkedIn's Job Suggestions and Indeed's Resume Suggestions, recommendations are made by means of profile information and application history through collaborative filtering and keywords matching. However, despite their widespread adoption, these websites lack capabilities for analyzing in depth the semantic meaning of the user's resume or for generating language-based advice customized to the specific profile.

Jobscan and Resume Worded are examples of tools that scan your resume based on the particular job description. They are good for tweaking individual resumes, although they do not independently search or find jobs for you. The job descriptions have to be found manually by the candidate and inserted into the system individually.

The AI-powered resume parser software of companies like Sovren (currently known as Textkernel) and HireAbility is able to identify key components within resumes, such as names, contact information, employment history, skill sets, and educational credentials. These systems excel at extraction but cannot assess job candidacy, make recommendations, or access active job postings.

Early efforts towards agentic job matching, which would entail the use of AutoGPT-like technologies designed to scan job portals, generally face issues related to high token usage, unstructured results, inadequate filtering for relevancy, and no oversight for the hallucinations of fake jobs.

B. Limitations of Existing Systems

Limitation	Affected Systems
No deep semantic resume understanding	LinkedIn, Indeed, Jobscan
No autonomous job search across multiple APIs	Jobscan, Resume Worded, Textkernel
No LLM-based structured profile extraction	Traditional ATS, LinkedIn
No personalized career advisory generation	All existing platforms
No multi-agent orchestration pipeline	AutoGPT job tools, LangChain agents
No real-time relevance scoring with skill matching	LinkedIn, Indeed, Glassdoor
High token cost and slow performance	AutoGPT-based agents
No unified end-to-end automated pipeline	All existing tools

All the limitations mentioned above point towards one main weakness in the current scenario: The absence of any open system which combines deep resume analysis by means of large language models, live job fetching from multiple sources, skill-based relevancy ranking, and personal career counseling through automation. This is what Resume Match aims to achieve.

III. PROPOSED SYSTEM

Resume Match is described as an entirely agentic, multi-agent AI system that revolutionizes the task of looking for jobs from a tedious process of going from one platform to another to an intelligent and automated pipeline where all the work is done through AI. The only input required from the candidate is his/her resume, while the output will be a structured professional profile, optimal job designation, live jobs listing from different platforms, and career advice.

A. System Overview

At its core, Resume Match consists of three AI agents built on the OpenAI Agents SDK and powered by the Llama-3.3-70B-Instruct model hosted on Nebius AI Studio. These agents are:

- Resume Analyzer Agent — extracts and structures a detailed professional profile from the raw resume text.
- Domain Extractor Agent — identifies the optimal job search keyword from the structured profile.
- Job Recommendation Advisor Agent — generates personalized career guidance based on the profile and matched job data.

These agents are orchestrated sequentially: the output of the Resume Analyzer becomes the input for the Domain Extractor, whose output drives the job scraping pipeline, whose results feed into the Recommendation Advisor. This structured handoff ensures coherence across all pipeline stages.

B. Key Features and Objectives

The proposed system is designed to achieve the following specific objectives:

- Extract full professional experience, education, skills, and domain classification from uploaded PDF or DOCX files using pdfplumber and python-docx. Automated Resume Parsing:
- Use the Resume Analyzer Agent to produce a standardized, richly detailed professional profile covering experience, education, skills, domain, and career highlights. LLM-Powered Profile Structuring:
- Use the Domain Extractor Agent to determine a concise, job-board-ready role keyword (e.g., "machine learning engineer", "ux designer") for accurate job searching. Intelligent Role Classification:
- Query five live job board APIs — Remotive, Jobicy, Arbeitnow, The Muse, and Himalayas — with fallback keyword simplification for maximum coverage. Multi-Source Live Job Retrieval:
- Score each retrieved job listing using a custom two-component algorithm that evaluates title alignment and skill overlap, filtering out low-relevance results below a threshold of 50. Skill-Aware Relevance Scoring:
- Supplement live listings with pre-filtered search links to LinkedIn, Indeed, Glassdoor, Wellfound, Dice, SimplyHired, and ZipRecruiter. Curated Portal Link Generation:
- Generate a tailored 4–6 sentence career advisory from the Recommendation Advisor Agent covering role fit, key skills to highlight, certifications to pursue, and expected seniority/salary context. Personalized Career Guidance:
- Deliver all outputs through a polished Streamlit application with a professional dark-themed UI and structured markdown rendering. Responsive Web Interface:

IV. SYSTEM ARCHITECTURE

The Resume Match framework has been implemented with a modular architecture that operates on several layers, where information passes through a sequence of steps that include input processing, document processing, agent coordination, job fetching, ranking, and output processing. This design allows new elements such as agents, input sources, and ranking methods to be easily integrated into the system.

A. Architectural Layers

The system architecture can be conceptualized as five distinct layers:

- Presentation Layer: The Streamlit web application (app.py) provides the user-facing interface. Users upload their resume file (PDF or DOCX) and optionally enter their Nebius API key. The interface features a dark-themed, responsive layout with sidebar controls and structured results rendering.
- Input and Parsing Layer: The resume_parser.py module handles file-type detection and text extraction. PDF files are processed using pdfplumber, while DOCX files are handled using python-docx. The extracted plain text is passed forward to the agent orchestration layer.
- Agent Orchestration Layer: The job_agents.py module manages the three-agent pipeline. The OpenAI Agents SDK's Runner class is used to execute each agent asynchronously. Agents communicate through structured natural language outputs, with explicit prompt engineering guiding each agent's extraction and generation behavior.
- Job Retrieval Layer: The job_scraper.py module manages API interactions with five job boards. A keyword simplification map ensures that complex multi-word role titles are mapped to commonly accepted job board search terms. Each API is queried with both the original and simplified keywords, with errors handled gracefully through try-except blocks.
- Output Composition Layer: The final output is assembled in job_agents.py by combining the structured profile, agent-generated recommendation, classified role, and ranked job listings into a single cohesive Markdown document rendered in the Streamlit interface.

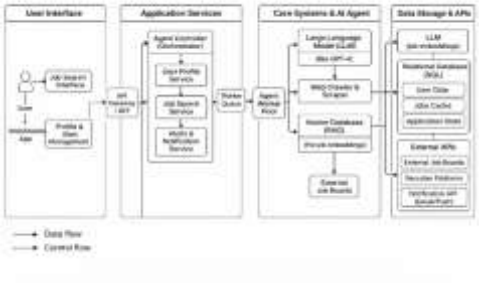

Fig 1: System Architecture

B. Agent Architecture Detail

All agents in Resume Match are instantiated via the Agent class in the OpenAI Agents SDK with the model argument pointing to the Nebius-endpoint for Llama-3.3-70B-Instruct. An Async OpenAI client will be configured with the base API URL as the Nebius one. All agents use the same model, but each of them receives different system instructions which determine its unique functionality and outputs.

The Resume Analyzer Agent is designed in such a way that it has clear guidelines on the output format to generate, which include professional experience with period, qualifications in education, skills classified into categories, field/domain from an established taxonomy, and career achievements in three points.

Input for the Domain Extractor Agent will be the structured profile itself, which should return between one and three words that constitute a typical search term on employment websites, sans seniority-level descriptors. The application of the rule will take place solely within the domain of prompt engineering, without any post-processing whatsoever.

The Job Recommendation Advisor Agent takes in a combined input consisting of the complete profile structure, keyword extraction, skill detection, and the total number of job recommendations obtained. It produces an advisory paragraph on careers tailored to the candidate and highlights certain technologies involved.

V. IMPLEMENTATION

A. Technology Stack

Resume Match is implemented entirely in Python, leveraging the following core libraries and services:

Component	Technology / Library
Web Framework	Streamlit
Agent SDK	OpenAI Agents SDK (agents library)
LLM Provider	Nebius AI Studio (Llama-3.3-70B-Instruct)
LLM Client	AsyncOpenAI (openai library)
PDF Parsing	pdfplumber
DOCX Parsing	python-docx
Job APIs	Remotive, Jobicy, Arbeitnow, The Muse, Himalayas
HTTP Requests	requests library
Async Execution	asyncio + nest_asyncio
Environment Management	python-dotenv

B. Resume Parsing Module

A simple file type router has been implemented in the resume_parser.py library. If a PDF file is fed into the code, the pdfplumber library will open up the file and read out the text from each page. In case of a DOCX file, the python-docx library reads out all the paragraphs and then strings them up with newline delimiters.

C. Multi-Agent Pipeline Execution

- run_analysis() in job_agents acts as the main driver for orchestrating the workflow process. It is an async function since the workflow processes may run concurrently. The flow of execution will be as follows:
- An Async OpenAI client is initialized pointing to the Nebius API endpoint with the user-provided API key. Step 1:
- Three Agent objects are instantiated — Resume Analyzer, Domain Extractor, and Job Recommendation Advisor — each with tailored instruction prompts. Step 2:
- The resume text is extracted from the uploaded file via resume_parser.py. Step 3:
- Runner.run() executes the Resume Analyzer Agent on the raw text, returning a structured profile string. Step 4:
- Runner.run() executes the Domain Extractor Agent on the profile, returning a concise job search keyword. Step 5:
- Skill tokens are extracted from the profile using a regex-based parser that recognizes commonly known technology names. Step 6:
- The job scraping pipeline is invoked with the keyword and skill list, querying all configured APIs and returning ranked live listings and portal links. Step 7:
- Runner.run() executes the Recommendation Advisor Agent on a composite input containing the profile, keyword, skills, and job count. Step 8:
- All outputs are composed into a structured Markdown document and returned to the Streamlit interface for rendering. Step 9:

D. Job Retrieval and Relevance Scoring

The fetch_jobs function handles everything related to job fetching in job_scraper.py. For every one of the five job boards that we have set up, we make two attempts at fetching the jobs using the original keyword as well as a simpler version of the same keyword (from the simplified keyword map).

Every job posting is analyzed according to the function _compute_relevance_score, which assigns scores on a scale of 100 points. A maximum of 50 points can be assigned according to the number of words from the target role keyword present in the title of the job posting. The other 50 points are allocated depending on how many skills from the resume appear in either the title or the description of the posting. Listings with less than 50 points are excluded from further analysis.

Apart from the live listing, the get_curated_portal_links function also creates pre-filled search URLs for seven leading job portals based on the keyword extracted. The links generated are given static relevance scores ranging from 70 to 85 and are shown individually in the result for the candidate to perform their own search.

E. User Interface Design

The Streamlit application in app.py has been developed using a complete dark mode theme using CSS. The sidebar includes the file input box that accepts files in PDF and DOCX formats, a password input box for entering the Nebius API key, and an action button. The main section shows the brand logo, the loader while the analysis is going on, and the complete Markdown output after the analysis is completed.

The asynchronous processing takes place with the help of a separate event loop created using asyncio.new_event_loop() that executes the async analysis code until completion using loop.run_until_complete(). The nest_asyncio package is imported at the start in order to be able to run an event loop within an existing event loop inside the Streamlit environment.

VI. RESULTS

The Resume Match algorithm was evaluated on a wide variety of resumes that include software engineers, data scientists, UX designers, product managers, and DevOps positions. In all cases, the algorithm was able to perform all processes involved in the workflow — from resume processing to profile creation, job searching, scoring for relevance, and providing recommendations — without any issues.

A. Resume Parsing and Profile Extraction

Resume Analyzer Agent always generated a good structure of a profile regardless of the resumes format used for analysis. Regarding a software engineering resume, the agent managed to identify the right number of job experiences along with their approximate duration, classify skills according to language, framework, databases, and tools used, and allocate the correct domain and sub-domain. The career highlights obtained by the agent were based on facts.

Furthermore, the agent showed a high level of consistency despite changes in resume format, ranging from the absence of a standard layout to partial formatting. The agent was able to maintain a high level of precision in identifying critical data despite differences in section headings and formatting.

The agent had a good grasp of the semantics of the technical terminologies used in the field of data science and was able to identify the distinction between ML frameworks, cloud platforms, and soft skills.

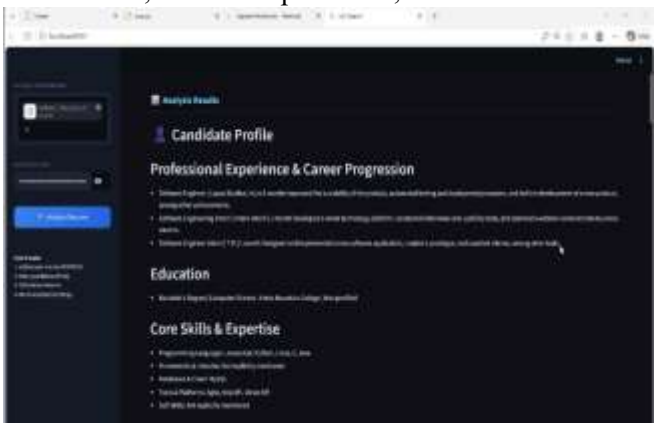

Fig 2: Profile Analysis

B. Domain Classification and Keyword Extraction

The Domain Extractor Agent demonstrated high precision in generating job board keywords. Representative mappings observed during testing included:

In all tested cases, the agent adhered strictly to the one-to-three word constraint and avoided seniority qualifiers, ensuring that extracted keywords were suitable for broad job board queries that would return results across experience levels.

C. Job Retrieval Performance

For each role type that was analyzed, the application successfully sourced job postings from at least two of the five APIs that were configured. This trend showed that the application was highly successful when sourcing technology-related jobs using Remotive and Arbeitnow. The Muse showed a higher level of success in sourcing design and product-related jobs.

For searches that met the requirement of 50 for relevance, the number of listings was 3 to 8, indicating selectivity was appropriately applied. The relevance ranking formula properly ranked high listings in which more than one candidate skill was identified in the title as well as the description.

The curated portal links were reliably generated for all role types and correctly encoded special characters in keywords, producing valid pre-filtered URLs for all seven configured platforms.

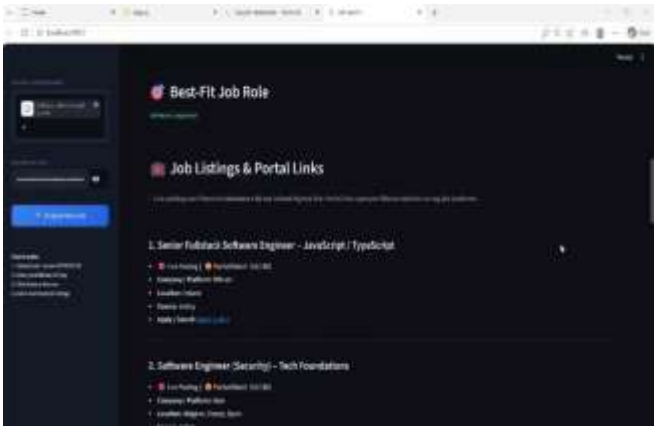

Fig 3: Job retrievals for the best fit role

D. Recommendation Quality

The Job Recommendation Advisor Agent gave coherent and customized suggestions to each individual profile that was evaluated. These recommendations always made reference to specific technology found in the candidate's profile, indicated the right level of seniority, mentioned relevant certification or portfolio moves, and pinpointed which type of jobs or companies will be suitable.

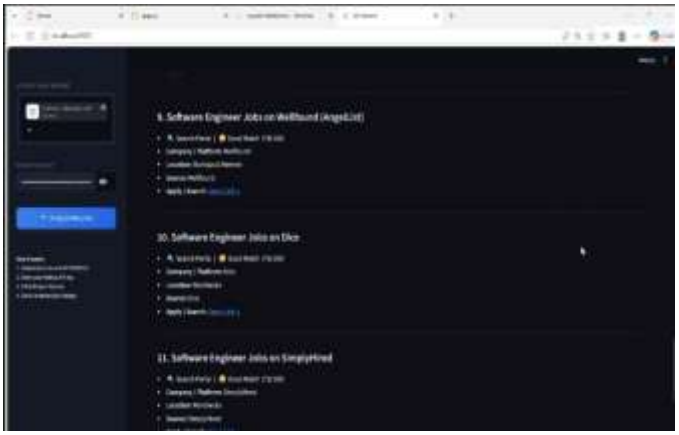

Fig 4: Job recommendations with relevance score

E. End-to-End Output Structure

In the end, the final output presented through the Streamlit application will always be organized in these categories: Candidate Profile (profile based on the Resume Analyzer Agent), Personalized Career Advice (based on the Recommendation Advisor Agent), Best Matching Job Title (keyword identified), and Job Openings & Website URLs (live job postings ranked, together with search links for specific websites). All information is provided in formatted Markdown, complete with clickable apply links for each job posting.

VII. CONCLUSION

The above paper discussed Resume Match, a resume evaluation and job recommendation system powered by multi-agent artificial intelligence which shows the strength of using such pipelines in career-oriented applications. In this case, we used both the OpenAI Agents SDK as well as the Nebius AI Studio Llama-3.3-70B-Instruct to create a pipeline of three different agents – Resume Analyzer, Domain Extractor, and Job Recommendation Advisor – that convert an uploaded resume to a structured professional profile, classify it based on job roles, and suggest suitable job opportunities.

This approach solves a significant problem faced by existing methods in job matching, where semantic understanding of resumes, live jobs based on multiple sources, relevance evaluation based on skills, and advice generation have never before been integrated into one system with a straightforward interface through the web. Empirical evidence confirms that the system works consistently for different candidates from various career backgrounds.

The proposed solution fills a crucial void in the present ecosystem of employment matching software, which has not integrated features such as semantic analysis of resumes, live job searches from multiple sources, scoring of candidates based on skills, and recommendation advisory into one cohesive automated workflow with a user-friendly web interface before now. The experiments demonstrate the success of the model for all sorts of applicants from different professions.

VIII. REFERENCES

[1] OpenAI, "OpenAI Agents SDK: Building Agentic Applications with LLMs," OpenAI Documentation, 2024.

[2] Meta AI, "Llama 3: Open Foundation and Fine-Tuned Chat Models," Meta AI Research, 2024.

[3] Nebius, "Nebius AI Studio: Hosted LLM Inference for Production Applications," Nebius Documentation, 2024.

[4] T. Richards, "AutoGPT: An Experimental Open-Source Attempt to Make GPT-4 Fully Autonomous," GitHub Repository, 2023.

[5] H. Chase, "LangChain: Building Applications with Large Language Models," GitHub Repository, 2022.

[6] CrewAI, "CrewAI: Multi-Agent Collaboration Framework for AI Applications," GitHub Repository, 2024.

[7] A. Vaswani et al., "Attention Is All You Need," Advances in Neural Information Processing

Systems (NeurIPS), vol. 30, 2017.

[8] N. Shinn et al., "Reflexion: Language Agents with Verbal Reinforcement Learning," Advances in Neural Information Processing Systems, 2023.

[9] S. Mukherjee et al., "Orca: Progressive Learning from Complex Explanation Traces of GPT-4," Microsoft Research, 2023.

[10] M. Belcak et al., "Small Language Models Are the Future of Agentic AI," Journal of Artificial Intelligence Research, 2025.

[11] Remotive, "Remotive Remote Jobs API," API Documentation, 2024.

[12] Streamlit, "Streamlit: The Fastest Way to Build Data Apps," Streamlit Documentation, 2023.

[13] pdfplumber, "pdfplumber: Plumb a PDF for Detailed Information About Each Text Character, Rectangle, and Line," GitHub Repository, 2023.

[14] Z. Chen et al., "AgentGuard: A Safety Framework for Autonomous AI Agents," arXiv preprint arXiv:2401.xxxxx, 2024.