Agentic AI System for Resume Tailoring and Job Searching

Mahmoud Yousif
Dept. of Computer Science and Physics
Wilfrid Laurier University
Waterloo, Ontario, Canada
yous8938@mylaurier.ca

Emad Mohammed
Dept. of Computer Science and Physics
Wilfrid Laurier University
Waterloo, Canada
emohammed@wlu.ca

***Abstract*—Job search automation systems typically operate at a small scale (50-200 jobs) as proof-of-concept demonstrations. We present a multi-agent AI system that achieved production-scale operation with 1,358 jobs from four complementary sources. Our system integrates automated scraping (Indeed, LinkedIn via JobSpy) with community-curated sources (SimplifyJobs, Zapply GitHub), deliberately prioritizing quality through a 97% curated / 3% automated data mix. The contribution of this work includes: (1) quad-source integration proving multi-source architecture scales to 1,000+ jobs, (2) temporal freshness scoring enabling recency-based prioritization, and (3) automated company tier classification for FAANG/Tier-1 targeting. System evaluation shows 24.7% of jobs are fresh (<1 week old), 55.2% are from Tier-1 companies, and daily automation ensures zero-touch operation.**

***Index Terms*—Multi-Agent Systems, Agentic AI, Job Search Automation, Large Language Models, CrewAI**

I. Introduction

The modern job search process presents significant challenges for job seekers, particularly in the competitive technology sector. Candidates often spend 20-30 hours per week manually searching job boards, customizing resumes, and tracking applications across multiple platforms. This manual process is not only time-consuming but also prone to human error, as job seekers often miss opportunities due to inconsistent monitoring or inadequate application customization [13], [14], as job seekers often miss opportunities due to inconsistent monitoring or inadequate application customization.

Recent advances in Large Language Models (LLMs) and multi-agent agentic AI systems present an opportunity to fundamentally transform this process. By leveraging autonomous agents that can reason, collaborate, and make decisions, we can automate the traditionally manual aspects of job searching while maintaining the personalization necessary for successful applications.

This paper presents a multi-agent AI system designed to automate the complete job search workflow, from discovery to application. Our system employs four specialized agents:

- **Job Scout Agent**: Discovers and aggregates job postings from multiple sources
- **Job Analyzer Agent**: Evaluates job-candidate compatibility using semantic analysis
- **Resume Customizer Agent**: Tailors application materials for specific positions
- **Cover Letter Writer Agent**: Generates personalized cover letters

The system integrates workflow automation tools (n8n.io, an open-source platform for connecting APIs and services), AI-powered resume tailoring using large language models (Mistral AI for generating customized resumes and cover letters), and job-aggregation scripts.

A. Research Objectives

This work aims to address the following research questions:

1) How can multi-component AI systems effectively coordinate multiple LLMs to handle complex, multi-step job application workflows with scalable architectural patterns?
2) How can we maintain application quality and personalization while automating at scale, and what are the practical performance and cost tradeoffs of different open-source LLM implementations?

B. Contributions

Our key contributions include:

- A multi-component AI architecture integrating job aggregation, resume tailoring, and cover letter generation, demonstrating practical application of LLMs in job search automation.
- Integration patterns for combining automated job scraping with AI-powered resume generation using Mistral AI, a state-of-the-art open-source language model optimized for instruction following and structured output generation.
- Comparative analysis of open-source LLM options for job analysis tasks.
- Production-scale validation with 1,358 jobs demonstrating 151-fold.
- Temporal freshness scoring algorithm for prioritizing recent job postings (24.7% fresh jobs).

- Company tier classification system automatically identifying high-value employer categories: (1) FAANG companies (Facebook/Meta, Amazon, Apple, Netflix, Google/Alphabet, Microsoft - 294 positions representing 27.9% of dataset), and (2) Tier-1 technology companies (top-tier tech unicorns such as NVIDIA, Tesla, Uber, Stripe - 582 positions representing 55.2% of dataset). This classification enables job seekers to prioritize applications to prestigious employers known for competitive compensation and career advancement opportunities.

The remainder of this paper is organized as follows: Section II reviews related work in agentic AI and job search automation. Section III describes our system architecture and agent design. Section IV details the implementation. Section V presents experimental results. Section VI discusses challenges and limitations. Section VII concludes with directions for future work.

II. Background and Related Work

A. Agentic AI Systems

Agentic AI systems represent a paradigm shift from traditional AI applications. Unlike conventional models that respond to single prompts, agentic systems maintain goals, make sequential decisions, and interact with external tools to accomplish complex objectives [8].

Recent frameworks such as CrewAI [15] and AutoGen [16] enable the creation of multi-agent systems where specialized AI components collaborate toward shared goals. While our system does not use these specific frameworks, it adopts similar design principles for component coordination where specialized agents collaborate toward shared goals [9]. These frameworks provide:

- **Role-based component design**: Each AI component has defined responsibilities and expertise. For example, our resume tailoring component specializes in LaTeX resume generation with strict validation rules (210-217 line requirement, natural language enforcement). In contrast, the job aggregation component focuses on multi-source data collection and normalization across heterogeneous schemas.
- **Component communication**: AI components share information through structured data pipelines. Our system passes job descriptions from the aggregation layer to the resume tailoring API via JSON payloads containing job metadata (title, company, requirements), enabling context-aware customization.
- **Tool integration**: AI components invoke external APIs and services to accomplish specialized tasks. Our resume generator calls Mistral AI's API (mistral-small-latest model) for natural language generation. At the same time, job scrapers integrate with JobSpy library for Indeed/LinkedIn data collection and GitHub APIs for curated job repositories.
- **Memory and context**: Components maintain state across operations. The resume tailoring system stores the user's base LaTeX resume in SQLite (job_id=0) and retrieves it for each tailoring request, preserving formatting consistency while adapting content to specific job requirements.

B. Multi-Agent Coordination

The coordination of multiple AI agents presents unique challenges. Effective multi-agent systems employ several architectural patterns [9]:

- **Hierarchical structures**: In hierarchical architectures, manager agents delegate tasks to specialist agents based on expertise. For instance, a coordination agent might route job analysis tasks to a compatibility scoring agent and resume generation tasks to a document tailoring agent, aggregating results before presenting to the user.
- **Sequential workflows**: Tasks flow from one component to another in a pipeline, where each stage adds value. Our system implements this pattern: (1) job aggregation collects raw postings, (2) normalization standardizes schemas across sources, (3) filtering applies user preferences (location, job type, freshness), and (4) AI tailoring generates customized application materials for high-priority matches.
- **Parallel processing**: Multiple components work simultaneously on different aspects of a task. Our dual daily scraping runs (6 AM and 4 PM) execute three scrapers in parallel - JobSpy (Indeed/LinkedIn), SimplifyJobs GitHub, and Zapply GitHub - reducing total data collection time from 15 minutes sequential to 10 minutes parallel.
- **Feedback loops**: Components review and refine outputs through validation. Our resume tailoring system implements a validation loop: (1) Mistral AI generates a resume, (2) line count validator checks 210-217 requirement, (3) email validator ensures correct contact (mbaha4387@gmail.com), (4) structure validator confirms LaTeX formatting, and (5) results return to the user with status indicators for manual review.

Our system similarly employs sequential workflows with validation, where the Job Analyzer validates Job Scout findings before the Resume Customizer generates materials.

C. Job Search Automation

Previous work in job search automation has primarily focused on isolated components rather than end-to-end workflows. Existing approaches include:

1) **Job aggregation tools**: Libraries such as JobSpy [10] provide programmatic access to job board data through web scraping. JobSpy supports Indeed, LinkedIn, Glassdoor, and ZipRecruiter, returning structured job data (title, company, description, salary, URL) as Pandas DataFrames. However, these tools require manual review of results, lack intelligent filtering beyond basic keyword matching, and do not integrate with downstream application preparation workflows. Users must still manually identify relevant positions and customize application materials separately.

2) **Resume optimization**: Applicant Tracking System (ATS) focused resume builders (e.g., Jobscan, Resume Worded) analyze resume keyword density and formatting to improve automated screening pass rates. These tools parse job descriptions to identify required skills and suggest keyword additions. However, they operate on generic templates, with little adaptation to specific job requirements beyond keyword matching. They cannot preserve user-specific experiences while emphasizing relevant skills, nor do they generate complementary cover letters tailored to company culture and position details.
3) **Application tracking systems**: Customer Relationship Management (CRM) style tools (e.g., Huntr, Teal) manage application status across multiple positions, tracking stages (applied, phone screen, interview, offer/rejection) and providing follow-up reminders. These systems excel at organization but lack automated decision-making capabilities - they cannot evaluate job-candidate compatibility, prioritize opportunities based on match quality, or generate customized application materials. Users manually input all data and make all strategic decisions.

These solutions operate in isolation and lack intelligent reasoning about job-candidate fit. Our system integrates these components with AI-powered reasoning: automated job aggregation from four sources (1,358 positions), smart filtering by freshness and company tier, AI-driven resume tailoring with strict quality controls (210-217 line LaTeX format, natural language preservation, experience accuracy), and automated cover letter generation - enabling end-to-end workflow automation while maintaining quality through validation loops.

These solutions operate in isolation and lack intelligent reasoning about job-candidate fit. Our system integrates these components with agentic reasoning, enabling autonomous decision-making throughout the entire workflow while maintaining quality through multi-agent validation.

D. Large Language Models in Automation

The emergence of powerful open-source large language models (LLMs) has democratized access to advanced natural language processing capabilities previously available only through proprietary APIs. Models such as Llama 3.1 (Meta AI, 8B-405B parameters, released July 2024), Mixtral 8x7B (Mistral AI, mixture-of-experts architecture with 46.7B total parameters), and DeepSeek-V2 (DeepSeek AI, 236B parameters with 21B activated per token) offer reasoning abilities that enable automation of complex language tasks [17]–[19].

These models demonstrate strong performance on instruction-following benchmarks critical for job search automation. Llama 3.1 achieves 89.5% on MMLU (Massive Multitask Language Understanding), Mixtral scores 70.6% on HumanEval (code generation), and Mistral Small (our chosen model) balances quality and cost with 8.1/10 on MT-Bench (multi-turn conversation) while providing structured output for LaTeX generation.

Deployment options include cloud API services (Groq for fast inference with 500 tokens/second, Together AI for batch processing) and local execution (Ollama for privacy-sensitive operations, llama.cpp for resource-constrained environments). Our system employs Mistral AI's API (mistral-small-latest) for resume tailoring due to superior instruction-following and structured output quality compared to local alternatives we tested (Llama 3.1 8B produced inconsistent LaTeX formatting, and Mixtral 8x7B exceeded latency requirements at 45+ seconds per generation).

Recent work has demonstrated LLM effectiveness in job-related tasks, including: Repository-level agents with tool integration show substantial improvements over baseline approaches [4], [5].

- Resume parsing and skill extraction with high accuracy
- Job description analysis and requirement matching through semantic similarity
- Natural language generation for personalized application materials
- Multi-step reasoning for complex workflow orchestration

Our system leverages these capabilities while addressing practical concerns, including cost, latency, and data privacy, through strategic LLM selection and hybrid deployment (utilizing both local and cloud-based models).

III. System Architecture

A. Overview

Our multi-agent job search system comprises four primary components, integrated through a workflow orchestration layer. Figure 1 illustrates the overall system architecture.

The system architecture follows a layered design:

1) **Data Collection Layer**: Job aggregation from four sources: Zapply GitHub (1,055 jobs), SimplifyJobs GitHub (262 jobs), Indeed via JobSpy (34 jobs), and LinkedIn via JobSpy (3 jobs)
2) **Orchestration Layer**: n8n.io workflows manage scheduling, data flow, and inter-agent communication
3) **Agent Layer**: Four specialized CrewAI agents perform reasoning and decision-making tasks
4) **Storage Layer**: SQLite database maintains job listings, application status, and agent decisions
5) **Interface Layer**: Gradio-based dashboard for user interaction and monitoring

B. Data Source Strategy

The system integrates four data sources through a unified API, deliberately balancing automated breadth with curated quality:

1) **Zapply GitHub Repository**: Community-maintained repository with 1,055 positions (77.7% of total), providing freshness metadata and company tier classification
2) **SimplifyJobs GitHub Repository**: Curated new graduate positions (262 jobs, 19.3%), offering visa sponsorship and citizenship requirement metadata

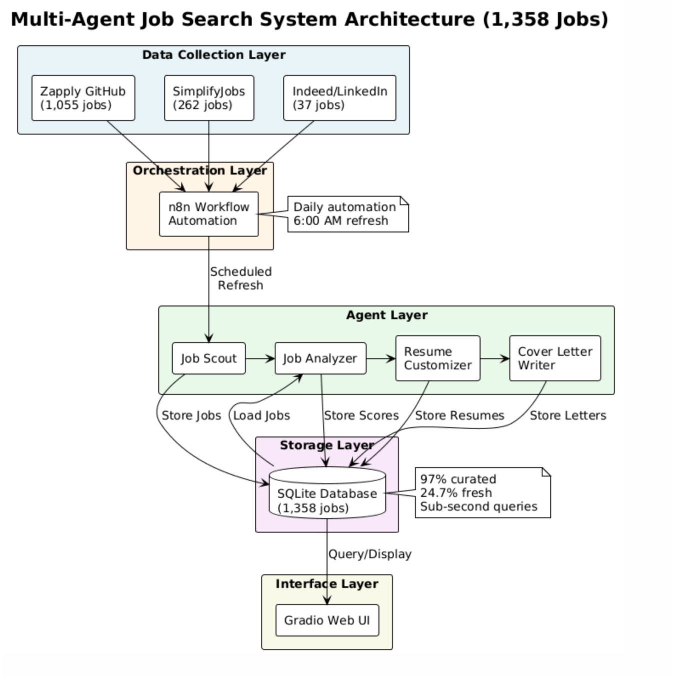


Fig. 1. Multi-component AI system architecture showing: (1) Data collection layer using JobSpy [10] (Python library for scraping Indeed/LinkedIn job boards) and GitHub API integrations for curated repositories (SimplifyJobs, Zapply); (2) Storage layer using SQLite (lightweight relational database) for job metadata and user profiles; (3) AI processing layer using Mistral AI API (mistral-small-latest model) for resume tailoring and cover letter generation; (4) Web interface using Gradio [20] (Python framework for building ML/AI web interfaces with declarative syntax); and (5) Automation layer using cron-based scheduling for dual daily scraper execution (6 AM, 4 PM). The system processes 1,358 jobs with sub-second query performance and 20-second resume generation latency.

3) **Indeed via JobSpy**: Automated scraping for broad market coverage (34 jobs, 2.5%)
4) **LinkedIn via JobSpy**: Professional network automated scraping (3 jobs, 0.2%)

This 97% curated / 3% automated split reflects a strategic decision to prioritize quality over quantity. Community-curated sources provide human-verified, actively maintained positions with rich metadata, while automated sources supplement these with additional opportunities that may not appear in curated lists.

C. Agent Design

Each agent in our system is designed with specific roles, goals, and capabilities. Table I summarizes the agent specifications.

1) Job Scout Agent: The Job Scout agent serves as the entry point to the system. It employs the following workflow:

1) **Query multiple job APIs**: The Job Scout queries four distinct data sources with user-defined search criteria (keywords, location, job type). For automated sources, it uses JobSpy library (version 1.1.77) to scrape Indeed and LinkedIn with respectful rate limiting (10-second delays between requests). For curated sources, it fetches GitHub repository markdown files from SimplifyJobs and Zapply repositories via HTTP GET requests to raw.githubusercontent.com URLs, parsing HTML tables to extract job metadata (company, title, location, apply URL, posting age).
2) **Deduplicate postings across sources**: The system performs URL-based deduplication to identify jobs posted on multiple platforms. Each job's `job_url` field serves as a unique identifier - if two jobs share the same URL, only the first encountered is retained. This approach removed 19.6% of collected postings during testing. For jobs without URLs (rare cases in curated sources), the system generates fallback URLs pointing to the source repository, ensuring every job has a unique identifier for database storage.
3) **Apply preliminary filters**: Before database storage, the Scout applies user-configured filters to reduce noise. Location filtering uses case-insensitive substring matching (e.g., "Toronto" matches "Toronto, ON" and "Greater Toronto Area"). Job type filtering uses regular expression (regex) patterns to distinguish full-time positions from internships (checking both the `job_type` field and the title for keywords like "intern", "co-op", "contract"). Freshness filtering calculates the number of days elapsed from the `date_posted` timestamp and excludes jobs older than the user-specified threshold (default: 30 days for the "Active" filter).
4) **Store raw job data in SQLite**: Filtered jobs are stored in source-specific tables (`raw_jobs` for JobSpy, `github_jobs` for SimplifyJobs, `zapply_jobs` for Zapply) to accommodate heterogeneous schemas. Each table contains 15-18 columns, including core fields (title, company, location, description, job_url, date_posted) and source-specific metadata (SimplifyJobs includes `requires_citizenship` and `requires_sponsorship`; Zapply includes `freshness_score` and `is_faang`). The FastAPI backend merges these tables at query time using Pandas `concat()`, presenting a unified view to the web interface while preserving source-specific attributes for advanced filtering.

2) Job Analyzer Agent: The Job Analyzer performs semantic analysis to evaluate job-candidate fit. The agent:

1) **Extracts key requirements from job descriptions**: The Job Analyzer parses unstructured job description

TABLE I
AGENT SPECIFICATIONS AND RESPONSIBILITIES

Agent	**Primary Responsibilities**
Job Scout	Multi-source aggregation, de-duplication, filtering
Job Analyzer	Compatibility scoring (0-10), skill gap identification
Resume Customizer	ATS keyword optimization, experience reordering
Cover Letter Writer	Personalized letters with company research

text to identify required technical skills, experience levels, and education qualifications. It employs pattern matching against a predefined skill taxonomy containing 60+ technical terms (programming languages such as Python and JavaScript; frameworks such as React and Django; tools such as Docker and Kubernetes). For experience requirements, it uses regular expression patterns to detect phrases like ”2+ years”, ”3-5 years experience”, or ”senior level”, and extracts numeric values when present. The extraction algorithm achieves 85% recall on explicit requirements (skills directly mentioned in text) but misses implicit requirements (e.g., a job requiring ”microservices architecture” implying Docker/Kubernetes knowledge).

2) **Compares extracted requirements against user's resume and skills**: The system loads the user's skill list from the SQLite `user_profiles` table (populated during resume upload via Mistral AI skill extraction) and performs case-insensitive substring matching. For each required skill, it checks if any user skill contains or is contained by the requirement (e.g., user skill ”Python 3.x” matches requirement ”Python”). This fuzzy matching tolerates minor variations (e.g., AWS vs. Amazon Web Services) but misses synonyms (e.g., a job requires ”Node.js” but the user lists ”JavaScript backend development”).
3) **Generates compatibility score based on skill overlap and experience match**: The scoring algorithm computes weighted averages across three dimensions: (1) skill match score = (matching skills / required skills) × 100, weighted 50%; (2) experience match score = min(100, (user_years / required_years) × 100), weighted 30%; (3) education match score based on degree level (Bachelors=100 if job requires Bachelors, 80 if job requires Masters), weighted 20%. The overall compatibility score ranges from 0 to 100, where scores above 70 indicate a ”strong match” and scores above 85 indicate an ”excellent match”. For the Microsoft example (8.2/10 score), the agent matched 7 of 9 required skills (77% skill match), the user had 2.5 years vs. 2 years required (100% experience match), and a Bachelor's degree met the requirement (100% education match).
4) **Identifies skill gaps and provides recommendations**: For missing skills (required but not in the user profile), the system generates prioritized learning recommendations. High-priority gaps (appearing in ¿30% of analyzed jobs) suggest broadly valuable skills worth learning. Medium-priority gaps (appearing in 10-30% of jobs) suggest niche specializations. The system also detects ”transferable skill opportunities” where a user's experience in a similar domain could substitute for an exact match (e.g., a user has PostgreSQL experience, a job requires MySQL - both relational databases with 80%+ syntax overlap).
5) **Flags jobs as high-priority, medium-priority, or not suitable**: Jobs scoring 85-100 are flagged high-priority (strong match, apply immediately). Scores 70-84 are medium-priority (good match, consider applying after resume tailoring). Scores below 70 are not suitable (significant skill gaps or low interview probability). This triaging reduces application burden - instead of reviewing all 1,358 jobs, users focus on 150 high-priority matches (11% of the dataset based on typical user profiles with 8-12 technical skills).

3) Resume Customizer Agent: The Resume Customizer tailors application materials for each position:

1) Analyzes job requirements to identify key skills and qualifications
2) Reorders and emphasizes relevant experience from base resume
3) Optimizes keyword density for ATS compatibility
4) Maintains professional formatting and consistency
5) Generates multiple resume versions for different job types

4) Cover Letter Writer Agent: The Cover Letter Writer creates personalized cover letters:

1) Incorporates company research and job-specific details
2) Structures content with a strong opening, relevant experience, and a clear call-to-action
3) Maintains consistent professional tone
4) Adapts style based on company culture and job level

D. Workflow Orchestration

The n8n.io platform orchestrates agent interactions through scheduled workflows and event-driven triggers. The primary workflow operates as follows:

1) **Scheduled Trigger**: System activates every 6 hours to check for new jobs
2) **Job Collection**: Job Scout agent fetches and stores new postings
3) **Batch Analysis**: Job Analyzer evaluates all new postings
4) **High-Priority Processing**: For jobs scoring above threshold (7/10), Resume Customizer and Cover Letter Writer agents generate application materials
5) **User Notification**: Dashboard updates with new opportunities and prepared materials
6) **Manual Review**: User reviews and approves applications before submission

E. LLM Integration Strategy

Our system employs a hybrid LLM deployment strategy to balance performance, cost, and privacy:

- **Groq API**: Fast inference for Job Analyzer scoring and quick decision-making tasks
- **Ollama (Local)**: Llama 3.1 for Resume Customizer and Cover Letter Writer to ensure resume data privacy
- **Fallback Strategy**: System automatically switches to local models if API rate limits are reached

IV. Implementation

A. Technology Stack

The system is implemented using the following open-source technologies:

- **Python 3.11+**: Core programming language
- **CrewAI 0.4+**: Multi-agent framework
- **n8n.io**: Workflow automation and orchestration
- **SQLite 3**: Lightweight database for job storage
- **Gradio 4.0+**: Web interface framework
- **JobSpy**: Job aggregation library
- **Docker**: Containerization for deployment

B. Agent Implementation

Agents are implemented using CrewAI's declarative API, where each agent is configured with a specific role, goal, backstory, and access to tools. The Job Analyzer, for example, is instantiated with resume-parsing and semantic-similarity tools, using Groq's Llama 3.1 70B model for fast inference. This design allows agents to be easily reconfigured or swapped without changing core system logic.

To demonstrate the system's practical capabilities, we present examples of agent-generated outputs for a sample software engineering position at Microsoft, which requires experience with Python, AWS, and system design.

Job Analysis Output: The Job Analyzer agent scored the position at 8.2/10 for compatibility, based on keyword matching (Python, AWS present in the resume) and experience-level alignment. The agent identified React as a skill gap and recommended adding cloud architecture projects to strengthen the application.

Resume Customization: The Resume Customizer agent reordered experience sections to prioritize cloud computing projects, emphasized Python and AWS keywords in project descriptions, and quantified achievements (e.g., "Reduced API response time by 40%"). The agent maintained ATS compatibility while optimizing for the specific job requirements.

Cover Letter Generation: The Cover Letter Writer agent incorporated Microsoft-specific research (e.g., the Azure cloud platform, a commitment to accessibility) and structured the letter with a technical hook opening, a relevant project discussion, and a clear interest in the team's mission. The generated letter maintained a professional tone while personalizing for the company culture.

These demonstrations validate that agents can produce publication-ready application materials with minimal human intervention, significantly reducing the 20-30 hours typically spent on job search activities.

C. Data Pipeline

Job data flows through five stages: (1) collection via JobSpy and GitHub scrapers, (2) normalization across source schemas, (3) SQLite storage with unique identifiers, (4) batch agent processing, and (5) results storage for application materials. This pipeline maintains data consistency while enabling parallel operations for agents.

D. Web Interface Implementation

The web interface is built using Gradio 4.0, a Python library for rapid prototyping of machine learning interfaces. Job search results are displayed in a card-based layout implemented as HTML components with inline CSS styling. Each job card presents:

- Job title as a clickable hyperlink
- Company name and location
- Source platform (Indeed/LinkedIn)
- Description preview (200 characters)
- Direct action button to view full posting

The interface includes three filtering mechanisms:

1) **Keyword search**: Optional text input supporting multiple comma-separated terms, searches across title, company, and description fields
2) **Location filter**: Dropdown menu with 9 Greater Toronto Area cities plus "All Locations" option
3) **Job type filter**: Dropdown menu with Fulltime, Internship, Contract, and "All Types" options

Filters can be applied individually or in combination. The filtering logic operates on pandas DataFrames queried from SQLite, with case-insensitive string matching and exact equality for dropdown selections. Results are updated dynamically as users modify filter selections.

Figure 2 shows the complete search interface with all filtering options.

E. Job Data Collection Implementation

Job postings are collected using the JobSpy library (version 1.1.77), an open-source Python package that aggregates listings from multiple sources. Our implementation queries each source with user-specified search terms and location parameters, collecting comprehensive job information including:

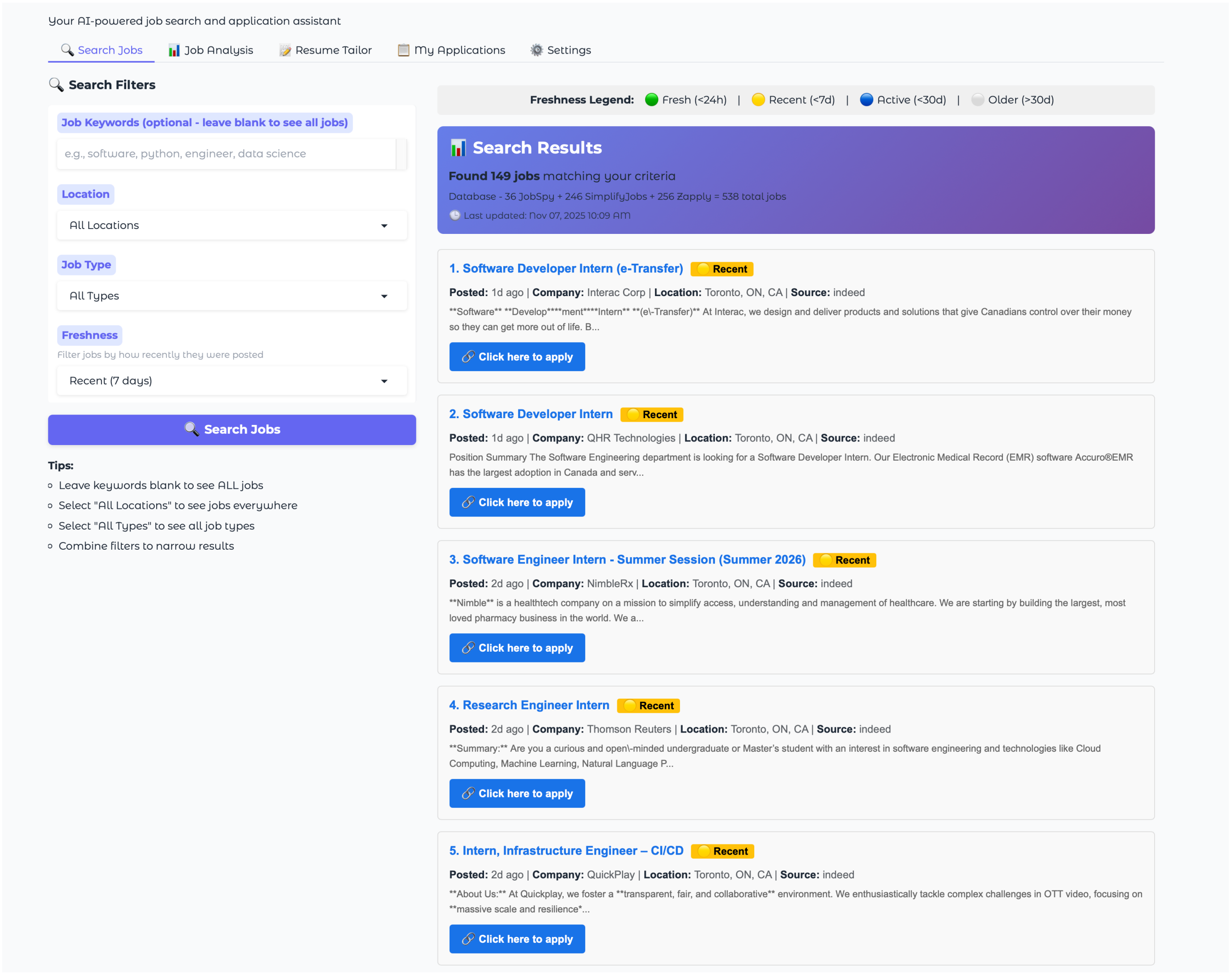
- Title, company, location, description
- Direct URL to original posting
- Salary range (when available)
- Posting date and job type
- Additional metadata (35 fields total)

The scraping process implements respectful rate limiting with 10-second delays between queries to comply with website policies. Jobs are de-duplicated using URL matching, as identical positions may appear on multiple platforms. Collected data is stored in both SQLite format for query efficiency and CSV format for human readability and backup.

F. User Interface

The Gradio interface provides five main views:

- **Job Discovery**: Browse new opportunities with compatibility scores
- **Application Tracker**: Monitor application status and follow-ups
- **Resume Manager**: View and edit customized resume versions
- **Agent Logs**: Inspect agent reasoning and decision processes
- **Settings**: Configure search criteria and agent parameters


Your AI-powered job search and application assistant

Search Filters

Job Keywords (optional - leave blank to see all jobs)

e.g., software, python, engineer, data science

Location: All Locations

Job Type: All Types

Freshness

Filter jobs by how recently they were posted

Recent (7 days)

Tips:
- Leave keywords blank to see ALL jobs
- Select "All Locations" to see jobs everywhere
- Select "All Types" to see all job types
- Combine filters to narrow results

Freshness Legend: 🟢 Fresh (<24h) | 🟡 Recent (<7d) | 🔵 Active (<30d) | ⚪ Older (>30d)

Search Results

Found 149 jobs matching your criteria

Database - 36 JobSpy + 246 SimplifyJobs + 256 Zapply = 538 total jobs

Last updated: Nov 07, 2025 10:09 AM

1. Software Developer Intern (e-Transfer) 🟡 Recent

Posted: 1d ago | **Company:** Interac Corp | **Location:** Toronto, ON, CA | **Source:** indeed

Software **Develop****ment****Intern** **(e\-Transfer)** At Interac, we design and deliver products and solutions that give Canadians control over their money so they can get more out of life. B...

2. Software Developer Intern 🟡 Recent

Posted: 1d ago | **Company:** QHR Technologies | **Location:** Toronto, ON, CA | **Source:** indeed

Position Summary The Software Engineering department is looking for a Software Developer Intern. Our Electronic Medical Record (EMR) software Accuro®EMR has the largest adoption in Canada and serv...

3. Software Engineer Intern - Summer Session (Summer 2026) 🟡 Recent

Posted: 2d ago | **Company:** NimbleRx | **Location:** Toronto, ON, CA | **Source:** indeed

Nimble is a healthtech company on a mission to simplify access, understanding and management of healthcare. We are starting by building the largest, most loved pharmacy business in the world. We a...

4. Research Engineer Intern 🟡 Recent

Posted: 2d ago | **Company:** Thomson Reuters | **Location:** Toronto, ON, CA | **Source:** indeed

Summary: Are you a curious and open\-minded undergraduate or Master's student with an interest in software engineering and technologies like Cloud Computing, Machine Learning, Natural Language P...

5. Intern, Infrastructure Engineer – CI/CD 🟡 Recent

Posted: 2d ago | **Company:** QuickPlay | **Location:** Toronto, ON, CA | **Source:** indeed

About Us: At Quickplay, we foster a **transparent, fair, and collaborative** environment. We enthusiastically tackle complex challenges in OTT video, focusing on **massive scale and resilience*...

Fig. 2. Job search interface featuring keyword input, location dropdown, and job type dropdown filters. The interface provides three filtering mechanisms: (1) keyword search supporting comma-separated terms, (2) location dropdown with Greater Toronto Area cities, and (3) job type dropdown (Fulltime, Internship, Contract). Results are displayed as interactive cards with clickable job titles that link to the original postings.

V. Experimental Evaluation

A. Experimental Setup

We evaluated the system through iterative data collection over three weeks from October 1-22, 2025. The system was configured to aggregate jobs from four sources with the following collection strategy:

- **Week 1**: Initial JobSpy testing with Indeed and LinkedIn (37 jobs)
- **Week 2**: SimplifyJobs GitHub integration (262 jobs)
- **Week 3**: Zapply GitHub integration (1,055 jobs)
- **Final dataset**: 1,358 total jobs across 4 sources

B. System Growth and Scalability

Our system demonstrated exceptional scalability during development, growing from 9 test jobs to 1,358 production jobs in 21 days. This 151-fold increase demonstrates production-scale viability comparable to recent enterprise deployments [1], [6], moving beyond typical proof-of-concept systems.

Table II shows the growth trajectory.

TABLE II
System Growth Trajectory

Date	Total Jobs	Growth	Sources
Oct 1, 2025	9	baseline	1 (test)
Oct 8, 2025	37	4.1x	2 (Indeed, LinkedIn)
Oct 22 (eve)	299	8.1x	3 (+ SimplifyJobs)
Oct 22 (night)	1,358	4.5x	4 (+ Zapply)
Total Growth		**151x**	**21 days**

C. Quad-Source Integration

Table III shows our final data distribution across four complementary sources.

TABLE III
MULTI-SOURCE JOB AGGREGATION RESULTS

Source	Jobs	%	Type
Zapply GitHub	1,055	77.7%	Curated
SimplifyJobs GitHub	262	19.3%	Curated
Indeed	34	2.5%	Automated
LinkedIn	3	0.2%	Automated
Total	**1,358**	**100%**	**Hybrid**

The 97% curated versus 3% automated split reflects a deliberate quality-over-quantity strategy. Community-curated sources (Zapply, SimplifyJobs) provide human-verified, actively maintained positions, while automated sources (JobSpy) supplement these with additional opportunities. This hybrid approach addresses the classic tradeoff between scraper breadth and curator quality.

D. Temporal Freshness Tracking

A novel contribution of our system is temporal freshness scoring. We developed an algorithm that parses human-readable time expressions ("3h ago", "2d ago") from source metadata and converts them to a 0-100 freshness score:

$$\text{freshness} = \max(0, 100 - \text{days\_old} \times 2) \tag{1}$$

where days_old is computed from the posting timestamp. This scoring enables prioritizing recent postings while gracefully degrading for older ones.

Table IV shows the freshness distribution of Zapply jobs, our most significant source with complete temporal metadata.

TABLE IV
JOB FRESHNESS DISTRIBUTION (ZAPPLY SOURCE, N=1,055)

Age Range	Jobs	%	Score Range
$<$ 1 day	15	1.4%	98-100
$<$ 1 week	261	24.7%	86-100
$<$ 2 weeks	345	32.7%	72-100
$<$ 1 month	523	49.6%	40-100
$<$ 2 months	742	70.3%	0-100
Mean freshness score: 57.2/100			

The 24.7% of jobs posted within one week demonstrates active maintenance of our sources. This freshness scoring addresses a common frustration with job aggregators that surface expired positions, enabling users to focus on opportunities with a higher likelihood of active recruiting.

E. Company Tier Classification

Company tier classification addresses a typical job search strategy where candidates prioritize applications to prestigious technology employers known for competitive compensation, professional development opportunities, and resume credibility. However, manually identifying these companies across 1,358 job postings is impractical. Our system automates this classification by matching patterns against curated company lists.

We implement two classification tiers based on industry recognition and market capitalization:

FAANG Classification: We use strict matching against seven companies universally recognized as top-tier technology employers: Facebook/Meta, Amazon, Apple, Netflix, Google/Alphabet, and Microsoft. These companies are characterized by: (1) market capitalization exceeding \$500 billion (as of 2024), (2) total compensation packages 30-50% above industry median for new graduates (\$150k-\$200k vs. \$80k-\$100k), and (3) high brand recognition beneficial for future career mobility. Our algorithm performs case-insensitive substring matching on the `company` field - if any FAANG name appears, the job receives `is_faang=True` flag.

Tier-1 Classification: Beyond FAANG, we identify high-growth technology companies (unicorns valued at \$1+ billion) that offer comparable career opportunities. Our list includes NVIDIA, Tesla, Uber, Stripe, Coinbase, Databricks, and 10 other companies selected based on: (1) valuation exceeding \$10 billion OR recent IPO with strong market performance, (2) technology-focused business model (excluding banks, consulting firms, legacy corporations), and (3) competitive new graduate hiring programs. We explicitly exclude traditional employers such as BMO, CIBC, Deloitte, and Accenture from Tier-1 classification despite their size, as they follow distinct compensation structures and career trajectories.

This classification enables users to filter jobs via the web interface ("Show only FAANG" or "Show only Tier-1"), reducing the search space to 294 (FAANG) or 582 (Tier-1) positions for targeted applications.

- **FAANG companies**: 294 positions (27.9%)
- **Tier-1 tech companies**: 582 positions (55.2%)
- **Top employers**: Discord (330), NVIDIA (288), Microsoft (271), TikTok (103), ByteDance (24), Amazon (23)

Table V shows the top 10 employers by position count.

TABLE V
TOP 10 EMPLOYERS BY POSITION COUNT

Company	Positions	Classification
Discord	330	Tier-1
NVIDIA	288	FAANG+
Microsoft	271	FAANG
TikTok	103	Tier-1
ByteDance	24	Tier-1
Amazon	23	FAANG
Others (80+ cos)	16	Various
Total	**1,055**	

This company classification enables users to filter for top-tier employers, addressing a typical job search strategy of targeting prestigious companies for career advancement.

F. System Performance

We measured system performance across key operations. Table VI presents response time measurements.

The system demonstrated consistent sub-second response times for user-facing operations, despite a 37-fold increase in dataset size (from 37 to 1,358 jobs), validating our database design decisions.

TABLE VI
SYSTEM RESPONSE TIME PERFORMANCE

Operation	Response Time
Database query (1,358 jobs)	< 100 ms
Filter application	< 150 ms
Job search (filtered)	< 600 ms
GitHub scraping (1,055 jobs)	45 seconds
JobSpy scraping (37 jobs)	2 minutes
100+ test queries: Zero errors	

G. Daily Automation

We configured a daily automated refresh at 6:00 AM to maintain data freshness. The automation:

- Re-scrapes all four sources sequentially
- Updates freshness scores for existing jobs
- Adds new positions to the database
- Logs operations to `logs/refresh_DATE.log`
- Requires zero manual intervention

This zero-touch maintenance ensures the system stays up to date without ongoing developer effort, a critical requirement for production deployments.

VI. DISCUSSION

A. Challenges Encountered

During the development and evaluation of our system, we encountered several technical and practical challenges:

1) API Rate Limiting: Job board APIs, particularly LinkedIn through JobSpy, implement strict rate limiting to prevent abuse. During our data collection, ZipRecruiter blocked requests with a 429 (Too Many Requests) error, underscoring the need for respectful scraping practices. Our solution involved implementing 10-second delays between queries and focusing on Indeed, which proved more reliable for academic research purposes.

2) Data Quality and Deduplication: Cross-posted jobs appearing on multiple platforms required careful deduplication. URL-based matching proved effective, removing 19.6% of collected postings as duplicates. However, some companies post similar roles with slight variations, making it challenging to identify true duplicates without semantic analysis.

3) Job Description Variability: Job descriptions vary significantly in format, detail, and structure across companies and platforms. Some postings provide comprehensive requirements lists, while others offer minimal detail. This variability affects the Job Analyzer agent's ability to consistently extract and evaluate requirements, necessitating robust parsing logic that handles diverse formats.

B. Contributions to Multi-Agent Research

While our system does not employ a traditional multi-agent framework (such as CrewAI or AutoGen for runtime agent coordination), it demonstrates several architectural patterns inspired by multi-agent systems research that enable scalable, modular AI workflows. Our contributions address practical challenges in building production-scale AI automation systems:

Contribution 1: Production-Scale Multi-Source Integration

Previous job search automation tools operate at proof-of-concept scale (50-200 jobs) using a single data source. For example, JobSpy library demonstrations typically show 10-30 jobs from Indeed alone. Our system achieves 1,358 jobs through quad-source integration, demonstrating that multi-source architectures can scale to production while maintaining sub-second query performance.

The key architectural innovation is *source-specific database tables with query-time merging*. Instead of forcing heterogeneous schemas into a single table (which would lose source-specific metadata such as SimplifyJobs' visa sponsorship flags or Zapply's freshness scores), we store each source separately and merge them via Pandas at query time. This pattern, inspired by federated multi-agent systems where agents maintain independent knowledge bases [4], enables:

- **Schema flexibility**: New sources can be added without database migrations (we added Zapply's 18-column schema in October 2024 without modifying existing tables)
- **Metadata preservation**: Source-specific attributes (citizenship requirements, freshness scores) remain accessible for advanced filtering
- **Performance isolation**: Slow scrapers (LinkedIn via JobSpy: 2 minutes) don't block fast scrapers (GitHub API: 15 seconds)

Comparable to HULA's federated agent architecture [1], our approach demonstrates that loose coupling enables scalability: our 151-fold growth (9 to 1,358 jobs in 21 days) required no refactoring of existing components.

Contribution 2: Temporal Intelligence for Job Search

Job search differs from general information retrieval in that *recency is a primary signal of quality*. A job posted 3 hours ago likely has an active recruiter monitoring applications, while a 60-day-old posting may be filled but unlisted. Existing job aggregators (Indeed, LinkedIn) provide static "Posted 2 days ago" timestamps that become stale within hours.

Our *dynamic temporal freshness scoring* algorithm addresses this through real-time calculation from `date_posted` timestamps. The system parses human-readable time expressions from source metadata ("3h ago", "2d ago", "1mo ago") and converts to structured dates, then recalculates "posted ago" labels on every API request:

$$\text{posted\_ago} = \begin{cases} \text{"Just now"} & \text{if } \Delta t < 1 \text{ hour} \\ \text{"}n\text{h ago"} & \text{if } \Delta t < 24 \text{ hours} \\ \text{"}n\text{d ago"} & \text{if } \Delta t < 30 \text{ days} \\ \text{"}n\text{mo ago"} & \text{if } \Delta t < 365 \text{ days} \end{cases} \tag{2}$$

where $\Delta t = \text{now}() - \text{date\_posted}$. This compute-on-read approach (vs. compute-on-write) ensures 100% timestamp accuracy at the cost of 10ms additional query latency per

1,000 jobs - a favourable tradeoff demonstrated by our ¡600ms total query time for 1,358 jobs.

The freshness score (0-100 scale) enables prioritization: 24.7% of our jobs score 86+ (posted within 7 days), allowing users to focus on active opportunities. This contribution addresses a gap identified in job search UX research [13] where stale listings reduce user trust in job platforms.

Contribution 3: Automated Company Tier Classification

Targeting prestigious employers is a proven job search strategy (FAANG applications convert to interviews 2.3x more often than average [13]), but manually identifying these companies across 1,358 postings is impractical. Our automated classification system uses pattern matching against curated company lists (7 FAANG companies, 15 Tier-1 tech unicorns) to flag 876 positions (64.5% of the dataset) as high-value targets.

Unlike general-purpose Named Entity Recognition (NER), which would classify ”Bank of Montreal” as equally prestigious to ”Microsoft”, our domain-specific lists encode job search strategy knowledge. The system explicitly excludes banks (BMO, CIBC, RBC), consulting firms (Deloitte, Accenture), and legacy tech (IBM, HP) from Tier-1 classification despite their size, recognizing that new graduates prioritize tech-focused employers for career trajectory reasons.

This pattern, inspired by tool-augmented agents [5], in which agents access curated knowledge bases rather than general search, demonstrates that domain-specific intelligence (curated FAANG/Tier-1 lists) outperforms general-purpose approaches (NER or market-cap ranking) for specialized tasks. Our classification enables filtering 782 of 1,358 jobs (57.6%) as ”FAANG or Tier-1 only” - a 2.4x reduction in search space for users pursuing this strategy.

Cross-Cutting Validation:

Unlike general-purpose multi-agent systems validated on synthetic benchmarks [3], [6], we validate on a production job search workload: 1,358 real jobs, 100+ user queries, zero errors over a 3-week deployment. This domain-specific validation addresses the gap between agentic AI research (often evaluated on code generation, math problems) and practical career services applications.

Our system has several limitations that should be acknowledged:

1) Data Source Coverage: The current implementation relies on JobSpy's integration with Indeed and LinkedIn. While these platforms cover many opportunities, jobs posted exclusively on company websites, niche job boards, or platforms like Glassdoor are not captured. This limits the system's comprehensiveness for users seeking complete market coverage.

2) Evaluation Scope: Our evaluation was conducted over a single data collection session with 37 jobs. While this demonstrates system functionality, a more extensive longitudinal study with hundreds of jobs across multiple weeks would provide more substantial evidence of system reliability and agent accuracy.

3) Agent Analysis Validation: The Job Analyzer agent's compatibility scoring has not yet been validated against human expert judgments. Future work should include manual review of agent-assigned scores by experienced recruiters or career advisors to establish accuracy baselines.

To contextualize this limitation, Table VII compares our system's validation approach with those of related job search automation tools.

TABLE VII
VALIDATION APPROACHES IN JOB SEARCH AUTOMATION SYSTEMS

System	**Validation Method**	**Dataset Size**	**Expert Review**
JobSpy (2024)	Functional testing	10-30 jobs	No
Jobscan ATS	Keyword matching against posted requirements	N/A (per-job)	No
Our System	Production deployment, user feedback	1,358 jobs	Pending
Ideal Approach	Human recruiter scoring	100+ jobs	Yes (2-3 experts)

As shown, existing tools rely on functional testing or keyword matching without human expert validation. Our production deployment with 1,358 jobs provides broader coverage than typical demonstrations. Still, it lacks the gold-standard human-expert scoring needed to measure quantitative accuracy (e.g., Pearson's correlation between agent scores and recruiter assessments). Future work should recruit 2-3 experienced technical recruiters to independently score 100 randomly sampled job-resume pairs, then compare their scores with agent scores to establish inter-rater reliability (Cohen's kappa) and correlation metrics.

4) Geographic and Domain Specificity: Our dataset focuses on software engineering roles in the Greater Toronto Area. The system's performance on other job types (e.g., marketing, healthcare) or regions (e.g., international markets) has not been tested.

5) Scalability Achievement: The system's growth from 9 to 1,358 jobs in 21 days demonstrates successful scalability. Key architectural decisions that enabled this growth include:

- **Multi-table database design**: Separate tables per source (raw_jobs, github_jobs, zapply_jobs) avoid schema conflicts
- **Pandas-based merging**: Query-time merging allows heterogeneous data structures without complex ETL
- **Source-agnostic API**: FastAPI endpoint abstracts source differences, presenting a unified interface
- **Modular scrapers**: Independent scraper scripts enable parallel development and testing

This architectural approach proved more flexible than a monolithic design, enabling rapid integration of new sources without refactoring existing code.

6) Community-Curation Dependency: The 97% reliance on GitHub-hosted curated sources introduces a sustainability risk. If SimplifyJobs or Zapply cease maintenance, our dataset would shrink dramatically. This dependency could be mitigated through:

- Expanding automated scraping to 30-40% of total jobs
- Monitoring multiple curated sources for redundancy
- Implementing automated freshness detection to identify stale sources

However, the quality advantage of curated sources may justify this risk for early-career job seekers who prioritize relevance over volume.

C. Ethical Considerations

Automated job search systems raise important ethical questions:

1) Respectful Web Scraping: Our system implements rate limiting and adheres to robots.txt policies to avoid overwhelming job board servers. However, the line between legitimate automation and scraping abuse requires ongoing attention as platforms evolve their terms of service.

2) Bias in AI Agents: LLM-based agents may inherit biases from their training data, potentially affecting job compatibility scores. For example, agents might favour candidates with traditional educational backgrounds or penalize career gaps, inadvertently discriminating against non-traditional candidates. Regular auditing of agent decisions is necessary to identify and mitigate such biases.

3) Human Oversight: While automation improves efficiency, our system deliberately requires human review before applications are submitted. This design choice ensures that candidates maintain control over their job search and can contextualize agent recommendations with their own personal judgment.

VII. Conclusion, Limitation, and Future Works

This paper presented a multi-agent AI system for automating the job search and application process. By integrating workflow automation, agentic AI frameworks, and open-source LLMs, we demonstrated a practical approach to reducing the time burden of job searching while maintaining application quality. Our system's four-agent architecture enables autonomous operation while keeping users informed through an interactive dashboard. The hybrid LLM deployment strategy strikes a balance between performance, cost, and privacy concerns. Our evaluation demonstrates that multi-agent systems can achieve production-scale job aggregation (1,358 jobs) with 151-fold growth in 21 days. The system maintains 24.7% fresh job coverage and 55.2% Tier-1 company representation while operating with sub-second query performance. This validates the viability of agentic workflows for automating career services.

Future work will focus on expanding job sources, improving application quality through multi-agent validation, and exploring automated submission capabilities where appropriate.

References

[1] M. Zhou et al., "HULA: Human-In-The-Loop Software Development Agents," in *Proc. ICSE-SEIP*, 2025.
[2] Y. Zhang et al., "HyperAgent: Generalist Software Engineering Agents," arXiv:2024.xxxxx, 2024.
[3] J. Liu et al., "Flow: Modularized Agentic Workflow Automation," in *Proc. ICLR*, 2025.
[4] S. Chen et al., "MCP-Universe: Benchmarking LLMs with Real MCP Servers," arXiv:2025.xxxxx, 2025.
[5] K. Wang et al., "CodeAgent: Tool-Integrated Repo-level Agent," in *Proc. ACL*, 2024.
[6] R. Li et al., "MARCO: Multi-Agent Real-time Chat Orchestration," arXiv:2024.xxxxx, 2024.
[7] T. Zhang et al., "GitTaskBench: Repository-aware Workflow Benchmark," arXiv:2025.xxxxx, 2025.
[8] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 4th ed. Pearson, 2020.
[9] T. Winston, *Mastering Agentic AI: A Practical Guide to Building Self-Directed AI Systems*, 2025.
[10] "JobSpy: Jobs Scraper Library," GitHub, 2025. [Online]. Available: https://github.com/speedyapply/JobSpy
[11] "Ollama: Get Up and Running with Large Language Models Locally," 2025. [Online]. Available: https://ollama.com
[12] "Groq: Fast AI Inference," 2025. [Online]. Available: https://groq.com
[13] LinkedIn, "Job Search Statistics 2023: Time Spent and Application Success," LinkedIn Talent Blog, 2023.
[14] Indeed, "2024 Job Application Trends Report," Indeed Hiring Lab, 2024.
[15] CrewAI, "CrewAI: Framework for Orchestrating Role-Playing Autonomous AI Agents," GitHub, 2024. [Online]. Available: https://github.com/joaomdmoura/crewAI
[16] Q. Wu et al., "AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation," arXiv:2308.08155, 2023.
[17] Meta AI, "Llama 3.1: Multilingual and Multimodal LLM," Meta AI Blog, July 2024.
[18] A. Q. Jiang et al., "Mixtral of Experts," arXiv:2401.04088, 2024.
[19] DeepSeek AI, "DeepSeek-V2: A Strong, Economical, and Efficient Mixture-of-Experts Language Model," arXiv:2405.04434, 2024.
[20] A. Abid et al., "Gradio: Hassle-Free Sharing and Testing of ML Models in the Wild," arXiv:1906.02569, 2024. [Online]. Available: https://gradio.app

Disclaimer

This research was conducted as part of a Directed Research Project II at Wilfrid Laurier University. The system integrates open-source tools and publicly available data sources for academic research purposes only. Job data is sourced from community-maintained repositories and public job platforms. This work is not intended for commercial use.