Resource-Constrained Agentic AI for Technical Talent Sourcing

Written Report

Abstract

This report documents the design and implementation of a CPU-only *agentic* AI prototype for semi-automated technical talent sourcing. The system integrates a LinkedIn scraping workflow, a semantic embedding and scoring module, and a structured reasoning layer orchestrated through Ollama and `pydantic`-based tool schemas. A key engineering challenge was achieving reliable tool calling and reasoning under strict computational constraints (no GPU), necessitating empirical evaluation of multiple open-source models: `deepseek-r1:1.5b`, `granite3-dense:2b`, `llama3-groq-tool-use:8b`, and `granite4:micro`. The final solution adopted `Granite4-Micro(3b)` (released publicly on September 2) for its superior balance of determinism, resource efficiency, and tool invocation fidelity. This document details the problem framing, architectural methodology, implementation decisions, experimental observations, and improvement directions aligned with evaluation dimensions: problem analysis, technical implementation, communication clarity, and innovation.

1 Introduction

Automating candidate discovery and ranking remains nontrivial due to heterogeneous profile structures, shifting front-end markup (e.g. LinkedIn class name volatility), and the need for nuanced semantic alignment with job descriptions (JDs). The project objective was to construct a modular, inspectable, and locally executable AI pipeline that:

- Extracts candidate profiles via targeted search-driven scraping.
- Normalizes and semantically embeds candidate text for comparison against a JD.
- Produces ranked candidates with structured rationale.
- Employs an agentic control loop for selective tool invocation (scrape, extract, embed, score, explain).

The deliverables comprised (i) a working prototype, (ii) documented source code, (iii) a 5-minute demonstration video, and (iv) this written report. Evaluation emphasized analytical rigor in scoping, architectural soundness, clarity of exposition, and innovative adaptation to computational limits.

2 Methodology

2.1 Problem Analysis and Scope

The scope was intentionally constrained to the high-value pipeline stages:

1. **Acquisition**: Retrieval of candidate leads and raw profile sections (headline, experience, skills, education).
2. **Representation**: Embedding of candidate aggregates and JD text into a shared semantic vector space.

3. **Ranking**: Hybrid scoring combining cosine similarity with heuristic modifiers (recency weighting, skill coverage density).
4. **Reasoning Layer**: Lightweight LLM-mediated rationalization and tool sequencing.

Out-of-scope for the prototype phase: production-grade compliance auditing, bias mitigation analytics, and multi-lingual expansion (reserved for future iterations).

2.2 Constraint-Driven Model Selection

Operating strictly on CPU imposed constraints on memory footprint, inference latency, and energy efficiency. Empirical evaluation revealed:

- `deepseek-r1:1.5b`: Over-produced reasoning chains ("over-thinking") leading to latency inflation and delayed tool dispatch.
- Models < 3B parameters: Insufficient reliability in emitting correct structured tool call schemas (malformed JSON / missing arguments).
- `llama3-groq-tool-use:8b`: Adequate tool discipline but heavier memory and occasional throttling under batch extraction.
- Discovery (September 2) and integration of IBM open-source `Granite` family improved determinism; `granite4:micro` (2.1 GB) delivered superior tool adherence while sustaining acceptable CPU throughput.

2.3 Agentic Orchestration Design

Instead of a pre-built orchestration framework, a minimal custom orchestrator was implemented using `pydantic` for defining tool schemas and Ollama for local LLM reasoning. The orchestration loop handled:

- Intent recognition (deciding which tool to use next).
- Validation of structured outputs against predefined `pydantic` schemas.
- Execution of deterministic tool functions (scrape, extract, embed, score).
- Controlled reasoning steps to minimize latency and prevent runaway token usage.

2.4 Virtual Environment Isolation

Three dedicated Python virtual environments bolster reproducibility and reduce dependency drift:

`venv_linkedin`
: Selenium / HTTP stack for scraping (selectors updated manually when LinkedIn DOM mutates).

`venv_scorer`
: Embeddings, vector similarity, and ranking logic (Faiss / sentence-transformers or CPU-friendly alternative).

`venv_agent`
Ollama orchestration logic, `pydantic` schemas, and prompt templates.

Users can reconstruct environments via README instructions; Postman collection (`HR-AI-Agent.postman_coll` supports modular endpoint verification.

3 Implementation Details

3.1 System Architecture

- **LinkedIn Scraper**: Query-based candidate discovery; resilient retry logic; structured JSON export. DOM fragility acknowledged—class selectors versioned for quick patching.
- **Profile Extractor**: Section segmentation, minimal cleaning (HTML stripping, whitespace normalization), and canonical labeling.
- **Embedding & Scoring**: JD and candidate text concatenations embedded; cosine baseline fused with:
  - Recency factor (exponential or linear decay over years).
  - Skill coverage ratio (matched / declared skills).
  - Optional penalty for sparse experience tokens.
- **Agent Layer (Ollama + Pydantic)**:
  1. *Plan Step*: LLM determines next high-level intent.
  2. *Schema Validation*: Each tool call validated using `pydantic` models.
  3. *Execution*: Deterministic Python tool execution.
  4. *Rationale Generation*: `granite4:micro` provides concise explanations for candidate ranking.

3.2 Data Structures

Each candidate object encapsulates:

- `id`, `profile_url`
- `sections`: `headline`, `experiences[]`, `skills[]`, `education[]`,
- `embedding`: R^d
- `scores`: {`similarity`, `recency`, `skill_coverage`, `total`}

This structure enables forthcoming attribution (e.g. per-feature contribution charts).
PS: `languages[]` was not used in scoring to avoid overloading the model and requests.

3.3 Tool Invocation Discipline

Prompt templates enforced:

- JSON schema constraints (strict key ordering).
- Temperature = 0 for deterministic answers (better for tool calling formatting).
- Role separation: planning vs. explanation to reduce reasoning verbosity.

3.4 Reliability Strategies

- **Non-Greedy Inference**: Models below 3B easily hallucinate when overloaded; inputs were trimmed to maintain schema compliance.
- **Schema Enforcement**: `pydantic` validation ensured output stability across model variations.
- **Selector Drift Handling**: Centralized selector map with version tag; failure triggers logging for quick patch deployment.

4 Results & Discussion

4.1 Functional Outcomes

The prototype achieved:

- End-to-end candidate pipeline: search → extraction → scoring → rationale generation.
- Deterministic ranking reproducibility given fixed weight configuration.
- Modular testing via Postman collection enabling isolated endpoint verification.

4.2 Performance Observations

- CPU latency per candidate (extraction + embedding + scoring) remained within practical bounds for interactive usage.
- Embedding pre-filtering reduced candidates passed to the *explain* stage, minimizing LLM token usage.
- `deepseek-r1:1.5b` underperformed due to excessive introspective reasoning (inefficient loops).

4.3 Model Behavior Insights

- Models below ∼3B parameters exhibited schema instability (missing arguments, malformed JSON).
- `llama3-groq-tool-use:8b` delivered stable structure but with high CPU memory pressure.
- `granite4:micro (3b)` achieved the best balance of structure, determinism, and speed.

4.4 Innovation Dimensions

- **Constraint-Aware Architecture**: Local, CPU-only operation through careful tool-call frequency and schema enforcement.
- **Hybrid Scoring**: Fusion of semantic similarity and domain heuristics improved ranking stability.
- **Schema-Driven Control**: Pydantic-based orchestration offered transparent and auditable decision logic.

4.5 Challenges and Mitigations

CPU-Only Limitation
Mitigated via quantized or inherently small models; reduced reasoning steps.

Model Instability
Enforced low-temperature, schema-constrained prompts; validated responses pre-execution.

Scraper Fragility
Externalized selector configuration for rapid adaptation.

Over-Reasoning (DeepSeek)
Replaced with model emphasizing concise reasoning and tool discipline.

4.6 Limitations

- No production-grade anti-blocking or compliance mechanisms yet.
- Limited evaluation set; ranking precision not benchmarked at scale.
- Heuristic weighting not yet learned from user feedback loops.

4.7 Future Work

- Integrate adaptive weight tuning (Bayesian or bandit-style).
- Bias and fairness diagnostics on skill attribution.
- Add MCP integration for centralized, architecture-agnostic coordination.
- Extend with LinkedIn message automation or portfolio scraping.

5 Conclusion

A fully CPU-operable agentic AI pipeline for technical talent sourcing was designed and implemented using Ollama and Pydantic-based orchestration. The system demonstrates that disciplined schema enforcement and lightweight reasoning can overcome hardware constraints. The hybrid scoring approach enhanced relevance consistency, while the modular design supports future extensions. Future work will focus on adaptive scoring, compliance robustness, and interpretability tooling, advancing the system across the dimensions of problem analysis, technical implementation, clarity, and innovation.

End of Report.