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 via LangGraph and locally hosted lightweight large language models (LLMs) served by Ollama. 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

Rather than a linear procedural script, a graph-based state machine (LangGraph) was adopted:

- Nodes map to *intent classes*: search expansion, candidate extraction, embedding & scoring, explanation generation.
- Edges encode termination or refinement conditions (e.g. plateau in marginal relevance gain).
- LLM involvement minimized to discrete decision and explanation phases to cap cumulative latency.

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`
: Orchestration (LangGraph, Ollama client bindings) 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 (LangGraph + Ollama)**:
  1. *Plan Node*: Assess whether to fetch more candidates or refine scores.
  2. *Tool Nodes*: Deterministic Python functions (scrape, extract, embed, rank).
  3. *Explain Node*: `granite4:micro` generates succinct rationales and outreach hints.
  4. *Halt Condition*: Converged top-k set or no new high-relevance profiles.

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[]`(wasn't used in the scoring to not overload requests, because adding it shouldn't be as a scored embedding

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 cross-contamination of reasoning verbosity.

3.4 Iterative Model Evaluation

A controlled harness logged:

- Rationale token length (to detect drift / verbosity inflation).

`granite4:micro` exhibited the highest syntactic compliance with minimal over-expansion.

3.5 Representative Orchestration Pseudocode

```
state = init_state(job_description)
while not termination(state):
    next_intent = llm_plan(state.summary())
    if next_intent == "scrape":
        leads = discover_candidates(state.query)
        enqueue(leads, state)
    elif next_intent == "extract":
        raw = extract_profile(dequeue(state.pending))
        profile = normalize(raw)
        profile.embedding = embed(profile.text)
        profile.score = composite(profile, state.jd_embedding)
        state.profiles.append(profile)
    elif next_intent == "refine":
        adjust_weights(state.metrics)
    else:
        break

topk = select_top_k(state.profiles, k=10)
rationales = llm_explain(topk, state.job_description)
persist(topk, rationales)
```

3.6 Reliability Strategies

- **Non-Greedy-Inference**: Even the the chosen model has 126k context window, we choose to not fit in the context large batches of data (e.g. job description, json exctracted information files). Small models (3b/8b) tend to hallucinate or drift from formatting when over loaded.

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 the number of candidates passed to the *explain* stage, minimizing LLM token expenditure.
- `deepseek-r1:1.5b` underperformed due to excessive introspective reasoning (inefficient loops).

4.3 Model Behavior Insights

- Other Models below ∼3B parameters exhibited schema instability (missing arguments, malformed JSON).
- `llama3-groq-tool-use:8b` delivered stable structure but at higher CPU memory pressure.
- `granite4:micro (3b)` combined structural compliance with compact footprint, enabling multitool orchestration without swap thrashing.

4.4 Innovation Dimensions

- **Constraint-Aware Architecture**: Deliberate minimization of LLM invocation frequency while preserving agentic adaptability.
- **Hybrid Scoring**: Fusion of semantic similarity and domain heuristics improved stability versus pure cosine ranking.
- **Graph-Oriented Control**: LangGraph representation facilitates forthcoming insertion of fairness or bias audit nodes without restructuring core 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 tool-call parsimony.

4.6 Limitations

- Absence of production-grade anti-blocking / compliance mechanisms.
- Limited evaluation set (no large-scale benchmark of ranking precision).
- 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.

- MCP integration for a more centralized system, and more adaptable system to achritecture changes.
- simple message linkedin message approaching tool, or portfolio detecter and scrapper for email.

5 Conclusion

A fully CPU-operable agentic AI pipeline for technical talent sourcing was designed and implemented, demonstrating that disciplined architectural decomposition and empirically guided model selection can overcome hardware constraints. The adoption of `granite4:micro` following the September 2 open-source release enabled reliable, resource-aware tool orchestration where smaller models failed. The hybrid scoring approach enhanced relevance consistency, while the graph-based control structure positions the system for controlled extensibility. Key lessons include the importance of early constraint anchoring, aggressive reduction of superfluous reasoning, and strict schema validation for tool invocation. Future enhancements will target learned scoring calibration, compliance robustness, and interpretability tooling, further advancing the system along the evaluated axes of problem analysis, technical implementation, communication clarity, and innovation.

End of Report.