AI Job Agent — Technical Project Report

Author: Kiran Gowda Ramanagara Jayaram
Course: INFO 7375 — Generative AI
Project Type: Semi-autonomous job discovery, tailoring, and recruiter-outreach system using LLMs, RAG, a learned RoleFit model, and an online bandit learner.

---

1. Abstract

This project builds a pragmatic, human-in-the-loop assistant that helps a job-seeker to (1) ingest real postings from multiple ATS sources, (2) rank jobs by estimated fit, (3) generate tailored artifacts (resume bullets, cover letter, interview prep, STAR answers), (4) discover recruiter work emails, and (5) learn—via a multi-armed bandit—which outreach template yields the best outcomes. The system ships as a Streamlit app with a SQLite persistence layer, and provides end-to-end reproducibility scripts.

Core techniques demonstrated: Prompt Engineering, Retrieval-Augmented Generation (RAG), Synthetic Data + classical ML (logistic regression), and Online Learning (Thompson sampling). The design intentionally avoids portal auto-submit and keeps a human in the loop for ethical and practical reasons.

---

2. System Architecture

```
flowchart TB
  subgraph S[Streamlit App]
    Q[Queue Tab] -->|artifact prompts| G[LLM Artifact Gen]
    Q --> H[Hunter.io Finder]
    Q --> C[(CRM: contacts, outreach)]
    Q --> B[(Bandit state)]
    Q --> RAG[RAG Store]
    D[Dashboard Tab] --> C
    D --> B
  end

  ING[Ingestion Scripts] --> JDB[(jobs.db)]
  JDB --> RF[RoleFit v2 Scorer]
  RF --> Q
  RAG --> G
  C --> B
```

Persistence:

- `db/jobs.db` (SQLite): job postings, fit scores, artifacts metadata.
- `db/crm.py` tables: `Contact`, `OutreachEvent`.
- `data/bandit_state.json` (and optional `bandit.sqlite3`): per-template Beta parameters by bucket.

---

3. Data Model and Interfaces

3.1 Job schema (simplified)

```
JobPosting(id, company, title, location, jd_text, posted_at, status)
FitScore(id, job_id, total, created_at)
Artifact(id, job_id, resume_path, cover_letter_path, qa_json,
created_at)
```

3.2 CRM schema

```
Contact(id, name, email, title, company, linkedin_url, source,
created_at)
OutreachEvent(id, contact_id, job_id, channel, template_name, outcome,
notes, created_at)
```

3.3 Bandit key

A **bucket** groups events so the learner generalizes appropriately. We use:

```
 bucket = f"role:{title_lower}|company:{company_lower}"
```

Alternative bucketings (role-only, company-only, seniority bins) are supported.

---

4. Ingestion and Filtering

Sources: Greenhouse, Lever, Ashby (via curated slugs or ad-hoc). Each script normalizes title, location, and description fields.

4.1 Location filter

Given a set of allowed cities/regions `L`, a posting passes if

$$\exists \ell \in L:\ \text{match}\left(\text{location},\ \ell\right) = 1\,.$$

We implement `match` with regexes over common variants (for example, `"Remote - USA"`, `"San Francisco, CA"`).

4.2 Role filter

Let the target role strings be `R = {r_1,\dots,r_m}`. A posting passes if

$$\max_{r\in R} \text{sim}_t(\text{title}, r) \geq \tau_t,$$

where `sim_t` is a token-level similarity (normalized overlap and n-gram tf-idf cosine). Threshold `\tau_t` is tuned to be permissive, and misclassifications are absorbed by the downstream fit model.

4.3 Keyword filter (optional)

A posting passes if required keywords `K` all appear in JD text:

$$\forall k \in K:\ 1\left[k \in \text{JD}\right] = 1\,.$$

---

5. RoleFit v2 — Model and Features

Objective: rank jobs by probability of fit (p(y=1x)) given a user profile.

5.1 Feature engineering

For a job posting `j` and profile `p`:

1. **TF-IDF cosine** between JD text and profile summary:

$$\text{cos}\left(\mathbf{t}^{(j)}, \mathbf{t}^{(p)}\right) = \frac{\left\langle \mathbf{t}^{(j)}, \mathbf{t}^{(p)} \right\rangle}{\left\|\mathbf{t}^{(j)}\right\| \left\|\mathbf{t}^{(p)}\right\|}\,.$$

2. **Embedding cosine** (if enabled) using a fixed embedding model (()):

$$\text{cos}\left(\phi(\text{JD}), \phi(\text{profile})\right)\,.$$

3. **Keyword indicators**: (x_k = [k]) for must-have and nice-to-have sets.

4. **Title similarity**: bigram tf-idf cosine between normalized title and preferred role strings (max over targets).
5. **Geography features**: binary match by city/region; distance bin if coordinates are known.

Final feature vector: (x ^d) with standardized continuous components.

5.2 Classifier

We use logistic regression with L2 regularization:

$$\Pr\left(y = 1 \mid x\right) = \sigma\left(\mathbf{w}^\top x + b\right), \quad \sigma(z) = \frac{1}{1 + e^{-z}},$$

with parameters ((,b)) learned by minimizing the regularized negative log-likelihood:

$$\min_{\mathbf{w},b} \sum_{i=1}^{n} \left[-y_i \log p_i - \left(1 - y_i\right) \log\left(1 - p_i\right)\right] + \lambda \|\mathbf{w}\|_2^2,$$

where (p_i=(^x_i+b)).

5.3 Training data

Two regimes are supported:

- **Supervised** on labeled historical data if available.
- **Synthetic** (default): `scripts/seed_synthetic.py` creates semi-realistic pairs (JD, label) by perturbing template JDs and sampling labels via simple heuristics (keyword coverage, title match, location match). This seeds a reasonable initial classifier saved as `models/fit_clf.joblib`.

5.4 Calibration and ranking

Probabilities are used directly as scores. If needed, Platt scaling or isotonic regression can calibrate (p) on a held-out set.

5.5 Evaluation protocol

Given a labeled set ({(x_i,y_i)}):

- Precision/Recall/F1:

$$\text{Precision} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}},\ \text{Recall} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}},\ \text{F1} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}.$$

- ROC-AUC via threshold sweep on (p).
- PR-AUC for the positive class. We report 5-fold CV means and standard deviations to demonstrate stability.

---

6. Retrieval-Augmented Generation (RAG)

Artifacts (resume bullets, cover letter, prep notes, STAR answers) must reference job-specific context. We build a tiny vector store per job session.

6.1 Store

Let (= {d_1,,d_M}) be snippets including: JD, company blurb, role summary, and any saved notes. We compute embeddings (= {(d_m)}). At query time with prompt context (q), we retrieve top-k docs by cosine:

$$\text{NN}_k(q) = \underset{m}{\arg\operatorname{topk}} \cos\Big(\phi(q),\, \phi(d_m)\Big).$$

`rag.store.SimpleStore` abstracts the embedding backend (OpenAI embeddings or local encoder) and the exact index (in-memory cosine; can be swapped for FAISS/ Chroma without code changes to agents).

6.2 Prompt assembly

For artifact type (a{,,,}), we build a structured prompt:

```
SYSTEM: You are an expert career coach and technical writing
assistant.
CONTEXT: {top-k snippets from RAG}
RESUME: {base_resume_excerpt}
JOB: {company, title, role summary}
INSTRUCTIONS: {style, length, specificity, bullet grammar}
OUTPUT_FORMAT: {markdown bullets / sections}
```

This reduces hallucinations and keeps outputs grounded in the JD.

---

7. Outreach Learning as a Multi-Armed Bandit

We model the choice of outreach template as an arm in a contextual bandit where the **context** is the bucket (role/company). The response is encoded as a scalar reward (r) derived from the observed outcome.

7.1 Reward mapping

Let outcomes be `sent`, `no_reply`, `positive_reply`, `interview`, `rejected`. We define weights (w_o) (configurable in code) and set:

$$r = w_{\text{sent}}\,1[o = \text{sent}] + w_{\text{no\_reply}}\,1\left[o = \text{no\_reply}\right] + w_{\text{pos}}\,1\left[o = \text{positive\_reply}\right] + w_{\text{int}}\,1[o = \text{interview}] + w_{\text{rej}}\,1\left[o = \text{rejected}\right].$$

A typical choice is (w_{}=1.0), (w_{}), and others near 0. Fractional rewards are supported by interpreting them as **pseudo-successes**.

7.2 Thompson sampling with Beta posteriors

For each bucket (b) and template/arm (k), maintain Beta parameters ((*{b,k},* {b,k})). With a Bernoulli likelihood and Beta prior, the posterior after observing reward (r) is

$$\left(\alpha_{b,k}, \beta_{b,k}\right) \leftarrow \left(\alpha_{b,k} + r,\ \beta_{b,k} + 1 - r\right).$$

At decision time, sample

$$\tilde{\theta}_{b,k} \sim \mathrm{Beta}\left(\alpha_{b,k}, \beta_{b,k}\right),$$

select (k^* = *k* {b,k}), and send the chosen template to the drafting agent. This procedure approximately maximizes expected reward while balancing exploration and exploitation.

7.3 Regret note

For optimal arm mean (^*) and sub-optimal arm (k) with gap (_k = ^* - _k), the expected Bayesian regret of Thompson sampling obeys a logarithmic bound of the form

$$\mathbb{E}\left[R_T\right] = \mathcal{O}\left(\sum_{k:\Delta_k>0} \frac{\log T}{\Delta_k}\right),$$

under standard assumptions (independent Bernoulli rewards, fixed gaps). This motivates maintaining separate buckets to reduce non-stationarity.

7.4 Persistence and cold-start

We initialize (*{b,k}=1,,*{b,k}=1) (uniform prior) unless historical data exists in `data/bandit_state.json`. Buckets are created lazily on first use.

---

8. Hunter.io Integration and Smart Domain Guessing

We expose two flows:

1. **Domain search**: `domain_search(domain, department)` returns candidate work emails (name, position, email, confidence).
2. **Name finder**: `email_finder(domain, first, last)` with optional `verify_email` for confidence.

When only the company name is known, **Smart Search** tries a ranked list of plausible domains

$$\mathscr{D}\big(\text{company}\big) = \{\text{company}\,.\,com,\ \text{company}\,.\,ai,\ \text{company}\,.\,io,\ \ldots\}$$

plus hand-coded exceptions (for example, `makenotion.com`, `notion.so`). Results are cached in session state so that “Save” works across reruns.

---

9. Prompt Engineering Details

9.1 Artifact prompts

- **Resume bullets**: enforce accomplishment-driven bullets, scope→action→impact grammar, numeric deltas if present.
- **Cover letter**: 3 short paragraphs, explicit alignment to role requirements, and a tailored closing.
- **Prep pack**: company mission, product lines, role charter, top 6 topic areas, question lists.
- **STAR answers**: 20 role-specific Q→A pairs with Situation/Task/Action/Result breakdown.

9.2 Outreach drafting prompt

Inputs: `{company, title, contact_name, contact_title, base_resume_excerpt, top-k JD snippets}`. Outputs: subject, email body, LinkedIn

DM draft, and the `template_used` that the bandit chose. We constrain tone to be concise, credible, and specific to the company.

9.3 Guardrails and error handling

- Max token budgets to prevent truncation.
- Deterministic formatting to make copying easy.
- Fallbacks when no RAG context is available (minimal viable prompt with profile and title only).

---

10. User Interface and UX

The Streamlit app presents two tabs:

- **Queue**: expand a row to see JD, generate artifacts, run Smart Search, draft outreach, and log outcomes. Supports exporting a full ZIP packet per job (artifacts + CRM CSVs).
- **Dashboard**: heatmap of outcomes by company vs outcome type; download CSVs for jobs, contacts, and outreach events.

Design accents: larger font sizes for readability, animated containers for visual feedback, and clean button groupings. All forms are stateful to survive reruns.

---

11. Evaluation and Results

11.1 RoleFit model

Run:

```
python -m scripts.seed_synthetic
python -m scripts.train_fit_model
```

Report: 5-fold cross-validated ROC-AUC, PR-AUC, and F1. Also show a calibration plot (reliability curve) to validate probability estimates.

11.2 Bandit learning

Protocol:

1. Initialize equal priors and pick among 3–5 templates.
2. Log outcomes over time; periodically compute empirical means (_{b,k}=/(+)).

3. Plot template selection frequencies and moving average reward to show convergence.

11.3 Human evaluation of artifacts

Rubric: correctness, specificity to JD, readability, and actionability. Sample 10 jobs × 2 artifacts each; score on a 1–5 Likert scale.

---

12. Reproducibility and Configurability

- Deterministic seeds for synthetic data and model training.
- All API keys are provided via `.env` and never checked into version control.
- Profile targeting is centralized in `data/profile.yaml`.
- Swappable components: embedding model (RAG), vector index, and outreach templates.

---

13. Security, Privacy, and Ethics

- **Privacy**: stores only necessary recruiter contact info and user artifacts locally.
- **Consent & ToS**: Hunter.io usage conforms to their API policy; no scraping against robots directives.
- **Bias and fairness**: the RoleFit model uses transparent features; users can inspect reasons (keywords/overlaps). Avoids opaque auto-apply decisions.
- **Avoiding spam**: user remains in the loop; batched personalized outreach with logging discourages mass, untargeted messages.

---

14. Limitations

- Small-data setting for the bandit per bucket; convergence may be slow for rare companies/roles.
- RoleFit model is deliberately simple (logistic regression) for interpretability; stronger encoders could help.
- RAG quality depends on JD clarity; some vendor JDs are sparse or marketing-heavy.

---

15. Future Work

1. **Contextual bandits**: replace bucketed TS with linear Thompson sampling over job features to share statistical strength:

$$\Pr\left(r = 1 \mid x, k\right) = \sigma\left(\mathbf{w}_k^\top x\right).$$

2. **Hybrid fit model**: concatenate encoder embeddings with symbolic keyword features and train a calibrated gradient-boosted tree or light-weight transformer.
3. **Active learning**: ask the user for quick pairwise preferences between similar jobs to refine the scoring surface.
4. **Vector DB**: move the RAG store to FAISS/Chroma with persisted indices and add hybrid BM25 + dense retrieval.
5. **LLM-guardrails**: add automated style checks and hallucination detection using self-consistency or critique prompts.
6. **API surface**: expose a REST layer for scheduling ingestion and exporting reports.

---

16. How to Run (concise)

```
# 1) Install
python -m venv .venv && source .venv/bin/activate
pip install -r requirements.txt

# 2) Configure
cp .env.example .env  # or create .env with OPENAI_API_KEY,
HUNTER_API_KEY

# 3) Ingest jobs
python -m scripts.ingest_presets --limit 100
python -m scripts.cleanup_jobs

# 4) (Optional) Train RoleFit
python -m scripts.seed_synthetic && python -m scripts.train_fit_model

# 5) Run app
streamlit run app/app.py
```

---

17. Conclusion

This project demonstrates a full, production-shaped pipeline for a job-search copilot: scraped data, a learned ranking model, grounded LLM generations, recruiter discovery, and an online learner to optimize outreach style. The focus on interpretability, reproducibility, and human control makes it defensible in real-world use while still showcasing several core Generative AI techniques at a graduate level.