Shashank Mishra Test Task

Objective (what success looks like)

Deliver a deploy-ready HR AI system that:

1. Ingests sample CVs, JDs, and interview feedback (CSV / text / simple PDF → parsed).
2. Produces candidate match scores for each JD (explainable ranking + top-k output).
3. Runs feedback sentiment + alignment analysis.
4. Runs a simple RL agent that learns to make hiring recommendations (hire / reject / assign task / hold) based on prior matches + feedback rewards.
5. Exposes results via: a) JSON/CSV export, and b) a minimal Flask API endpoint (POST inputs → JSON decision).
6. Includes visualizations, a README, and a 5–8 minute walkthrough video.
   Delivered in a single GitHub repo + walkthrough video submitted via WhatsApp and reviewed in person.

Deliverables (deliver at end of 56 hours)

1. GitHub repo with code, requirements.txt, README, and sample data.
2. notebooks/ demo notebook showing E2E flow and visualizations.
3. app/ directory with a Flask API exposing /match and /decision.
4. /models/ saved model artifacts (or script to re-train quickly).
5. reports/ with CSV/JSON output for sample run + charts (png).
6. Short screen recording (5–8 min) explaining architecture, results, and how to run.
7. Short writeup: RL reward function, agent policy, limitations & next steps.

Tech stack

- Python 3.10+
- Pandas, NumPy
- scikit-learn (TF-IDF, similarity, classifiers)

- SentenceTransformers or OpenAI embeddings (local TF-IDF fallback allowed)
- NLTK / VADER / TextBlob (sentiment)
- Gymnasium (or custom loop) for simple RL simulation
- Flask for API
- MongoDB or simple JSON/CSV for persistence
- Matplotlib / Plotly for visualizations
- Optional: Dockerfile (bonus)

Repo skeleton (suggested)

```
hr-ai-shashank/
├─ app/
│   ├─ api.py            # Flask endpoints
│   └─ utils.py
├─ notebooks/
│   └─ demo.ipynb
├─ data/
│   ├─ sample_cvs.csv
│   ├─ sample_jds.csv
│   └─ sample_feedbacks.csv
├─ models/
├─ outputs/
│   └─ run_2025-08-XX.json
├─ utils/
│   ├─ text_preproc.py
│   ├─ embedding.py
│   ├─ matcher.py
│   ├─ sentiment.py
│   └─ rl_agent.py
├─ requirements.txt
└─ README.md
```

7-Day / 56-Hour Plan (8 hrs each day)

Day 1 — Setup & E2E design (8 hrs)

- Create repo, virtualenv, requirements.txt.

- Prepare sample dataset (20–50 synthetic CVs + 5–10 JDs + 40 feedback samples). Format: CSV with columns (id, name, location, skills, experience_months, education, resume_text, social_links, national_id_masked).
- Draft system architecture diagram and dataflow (ingest → preprocess → embed → match → feedback analysis → RL agent → decision → API).
- Implement basic text preprocessing utilities (utils/text_preproc.py).
- Goal: run pipeline that reads CSVs and prints counts.

Day 2 — Embeddings & CV ↔ JD Matching (8 hrs)

- Implement embedding approach:
    - Preferred: SentenceTransformers (all-MiniLM-L6-v2) for sentence embeddings.
    - Fallback: TF-IDF + SVD.
- Build utils/matcher.py:
    - Compute cosine similarity between CV and JD embeddings.
    - Output top-k matches and similarity scores.
    - Add simple rule boosts (location match, specific skill present → +0.05 score).
-
- Unit test matcher on sample data and store outputs/match_results.csv.
- Visualize similarity distribution (histogram, top 5 matches per JD).

Day 3 — CV Parsing, Feature Engineering & Explainability (8 hrs)

- Improve CV parsing: extract key features (years experience, top 5 skills via keyword matching).
- Create a feature vector combining embedding similarity + structured features (experience normalized, education level, skill match count).
- Train a simple classifier/regressor that maps feature vector → baseline compatibility score (use a small labeled set or heuristics).
- Implement explainability output per candidate (top reasons: e.g., “Skill overlap: Python, ML; Location match; Experience: 24 months”).
- Add a small UI or script to display candidate profile + explainability bullet points.

Day 4 — Sentiment Analysis of Interview Feedback (8 hrs)

- Implement utils/sentiment.py using VADER/TextBlob; also try a light transformer model if possible.
- Process sample_feedbacks.csv: label as positive/neutral/negative and compute sentiment scores.
- Create alignment score: Combine recruiter sentiment + candidate self-feedback sentiment → alignment metric.
- Visualize sentiment breakdown (pie chart) and per-candidate sentiment timeline (if multi-feedback).

Day 5 — RL Agent Design & Simulated Training (8 hrs)

- Design RL formulation:
    - State: [matching_score, sentiment_score, experience_norm, location_match_flag, prev_decision]
    - Actions: {HIRE, REJECT, ASSIGN_TASK, HOLD}
    - Reward: +1 if future feedback (simulated) matches decision (e.g., HIRE & good performance), -1 for mismatch; small penalty for false hires.
- Implement a simple environment (custom Gym-like) with simulated outcomes:
    - Use heuristics to simulate “true” candidate performance from features (so agent can learn).
- Implement Q-learning or policy-gradient simple agent (tabular Q or small MLP).
- Train agent for a few episodes; track reward curve.
- Save agent policy.

Day 6 — Decision Engine + API + Visualization (8 hrs)

- Build Decision Engine that combines matcher output, sentiment alignment, and agent policy to output final recommendation and explanation.
- Build minimal Flask API:
    - POST /evaluate — payload: CV text + JD text + feedbacks → returns JSON with match score, sentiment, agent_decision, explanation.
    - GET /top_candidates — returns top N for a JD.
- Run end-to-end demo: POST sample JD + CVs and show JSON results.
- Create dashboard screenshots or static HTML to view results.

Day 7 — Polish, Test, Docs & Walkthrough (8 hrs)

- Add tests & CLI scripts (run_pipeline.py).
- Final visualizations (confusion matrices for simulated labels, reward curve).
- Write README: architecture, how to run, how RL was set up, limitations, next steps.
- Record 5–8 minute walkthrough video.
- Zip repo or push to GitHub and prepare WhatsApp submission message.

RL Agent / Reward Function (concise design)

- State vector: [sim_score (0-1), sentiment_score (-1..1 normalized), exp_norm (0-1), location_flag (0/1), prev_action_onehot]
- Actions: 0:HIRE, 1:REJECT, 2:ASSIGN_TASK, 3:HOLD
- Reward design (example):
  - If action == HIRE:
    - +1.0 if simulated_performance >= 0.7
    - -1.0 if simulated_performance <= 0.4
    - +0.2 small bonus if sentiment_score > 0.5
  - If action == ASSIGN_TASK:
    - +0.6 if task_success (simulate via match_score and exp)
    - -0.4 otherwise
  - REJECT:
    - +0.5 if simulated_performance < 0.4
    - -0.6 if performance > 0.7
  - HOLD: small negative reward (-0.05) to discourage indefinite holding
- Train with Q-learning or small policy network; log episodic rewards and show convergence chart.

AI Agent architecture (practical)

- Agent type: Hybrid — rule + learned policy.
    - Rules (fast): immediate filters (CTC mismatch, location exclusion).
    - Learned policy (RL): handles ambiguous cases and learns from feedback/rewards.
- Optionally: implement a LangChain-style orchestration (chain of tools) where:
    - Tool A: Embed & Match
    - Tool B: Sentiment Analyzer
    - Tool C: Decision Agent (RL)
    - Orchestration layer calls tools sequentially.

Visualizations to include

- Similarity histogram & top-k match table
- Per-JD top 10 candidates bar chart
- Sentiment breakdown pie chart
- Reward per episode curve (RL)
- Explainability example (per candidate: feature contributions)

Evaluation Rubric (how you will grade)

- Functionality (40%) — End-to-end pipeline works and API returns expected JSON.
- Modeling & RL (25%) — Reasonable matching & working RL training loop with learning curve.
- Explainability & UX (15%) — Per-candidate explanations and clear visualizations.
- Code Quality & Docs (10%) — Readable, modular code + clear README.
- Delivery (10%) — Video walkthrough clarity + timely WhatsApp submission.

Data / Privacy note

- Use synthetic or anonymized sample CVs. Mask or omit any real national IDs. If you use public CV samples, redact PII. We will not accept unredacted personal data.

Learning resources (RL & Agent + NLP) — search keywords + note about broken links

Some YouTube links and external resources may become unavailable. If a link breaks, use the keywords below on YouTube/Google. The objective is concept mastery — any high-quality tutorial is fine.

Recommended searches:

- "Reinforcement Learning tutorial Sentdex", "Q-learning python gym tutorial", "custom RL environment python".
- "AutoGPT explained", "LangChain agents tutorial", "building AI agents with python".
- "SentenceTransformers tutorial", "TF-IDF cosine similarity python".
- "VADER sentiment analysis python", "TextBlob sentiment tutorial".
- "Flask REST API python tutorial".

Suggested specific watch/read (if available):

- Intro to RL by Sentdex — search by title.
- LangChain Agents overview — search.
- SentenceTransformers docs & examples — search.
- VADER sentiment analysis tutorial — search.

Bonuses (award extra points)

- Dockerfile + docker build/run.
- Streamlit dashboard to view top candidates.
- Simple CI (GitHub Actions) to run linting / basic tests.