AI Regression Runner — High-Level Design

Purpose

A lightweight system to automatically test that HiBob's AI agents give consistent answers to payroll questions. It compares AI-generated answers to expected reference answers using semantic similarity. It is run as part of CI and on schedule.

The system will consist of a CLI runner that will have a set of questions and expected answers. This runner will request the needed AI-Agent and will compare the response with the expected answer using embeddings. The runner will mock the tools responses used by the AI-Agent to ensure consistent testing environment without external dependencies.

Directory Layout and Scenario Example

Directory structure:

```
tests/
└ AI-Agent/.    ← for example us_payroll
   ├ why_net_lower.yaml
   └ __files/
     payslips/999999.json        ← stub response for /payslips
```

Test Yaml example

```
test_name: why_net_lower

variables:
  currentPayDetailsId: 999999
  employeeId:       123456

user_input: "Why is my net pay lower this month?"
```

```
expected_answer: |
  Your net pay dropped because a one-time bonus
  was taxed at the 22 % supplemental federal rate.

semantic_threshold: 0.88

tool_fixtures:
  paySlips:                # ← tool_id as configured in the AI agent
    - request:
        payDetailsIds: [999999]
        region: US
      response_file: payslips/999999.json

    - request:                  # ❷ second call (previous period)
        payDetailsIds: [999998]
        region: US
      response_file: payslips/999998.json

  paySlipsSummary:         # second tool
    - request:
        payDetailsIds: [999999]
        region: US
      response_file: summaries/999999.json
```

System Flow Diagram

```
flowchart TD
    %% ──────────────── GitHub Actions job ────────────────
    subgraph ga [GitHub Actions job]
        checkout[1 Checkout repo]
        buildImage[2 Build agent image]
        compose[3 docker-compose up]
        checkout --> buildImage
        buildImage --> compose
    end
```

```
  %% ──────────────── docker-compose services ────────────────
  subgraph dc [docker-compose]
    runner["Runner CLI (fat JAR)"]
    agent["AI Agent container :8081"]
    wm["Stub Server (FastAPI) :8080"]
    runner -- "HTTP (SSE)" --> agent
    agent  -- HTTP --> wm
  end

  %% ──────────────── orchestration links ────────────────
  compose --> runner
  runner --> csv["results.csv + exit-code"]
  csv --> gh[(GitHub)]
```

Responsibilities

Component	Description
Runner CLI	Parses YAMLs, starts stub server, streams questions, checks results
Stub Server (FastAPI)	Returns fixture JSON for every tool call
Agent Container	Runs the actual AI logic (same build as prod)
GitHub CI	Builds agent container, runs test suite, uploads results

Grading Logic

Each answer is graded by comparing its embedding to the expected_answer from YAML.

- Embeddings model: ??? → text-embedding-3-small
- Cosine similarity
- Pass if cosine ≥ semantic_threshold

Tool Base-URL Overrides in Test

During regression tests all tool base URLs are redirected to the single stub server so no live backend is contacted.

PAY_DETAILS_API_BASE=http://stub:8080

US_PAYROLL_API_BASE=http://stub:8080

UK_PAYROLL_API_BASE=http://stub:8080

CI Strategy

- One matrix job per agent slug (us_payroll, uk_payroll, ...)
- Workflow triggers on PRs touching tests or agent code, and weekly schedule
- Runner executed as

  ```
  python -m ai_runner --agent us_payroll
  ```

Output

- results.csv — scenario ID, pass/fail, latency
- runner.log — stdout/stderr logs

Failures cause exit code 1, failing the pipeline.

Runner Design

Technology stack

Layer	Choice	Why
CLI framework	**`click`**	Simple flags + colours
YAML loader	**`ruamel.yaml`**	Strict typing, comments preserved
HTTP/SSE client	**`httpx`** + **`httpx-sse`**	Async, supports streaming
Mock API server	**FastAPI** + **uvicorn**	Python-native, rapid stubbing
Embeddings	**OpenAI Python SDK**	`text-embedding-3-small`
CSV output	Python `csv` std-lib	No extra dep

High-level flow

1. **Arg parsing** – `-agent`, `-out`, `-semantic-timeout`, etc.
2. **Discover YAMLs** under `tests/<agent>/**/*.yaml`.
3. **Start stub server** (FastAPI) listening on `0.0.0.0:8080`.
4. **Load tool stubs** from each YAML's `tool_fixtures` block and register routes:
   - path = `/{{tool_id}}`

    - request matcher = body equality
    - response = file contents from `response_file`.
5. **Loop tests**:

```
for path in yaml_files:
    scn = Scenario.load(path)
    answer = ask_agent_sse(scn.user_input, scn.variables)
    sim = cosine(embed(answer), embed(scn.expected_answer))
    passed = sim >= scn.semantic_threshold
    csv_writer.writerow([scn.test_name, "PASS" if passed else "FAIL", sim])
    if not passed:
        failures += 1
```

6. **Write csv output**
7. sys.exit(1) if failures > 0, else sys.exit(0).

Failure handling

Failure	Runner action
Agent SSE times out	Mark test `TIMEOUT`, write 0.0 similarity, count as failure
Stub route not matched	Returns 500 → agent fails → test marked `FAIL`
OpenAI API error	Retry ×3, then mark `EMBEDDING_ERROR` and fail
Unexpected exception	Log stack trace, increment failure counter, continue loop

Output File

Requirement	Design
Columns	`test_name, status, similarity, error`
Status values	`PASS`, `FAIL`, `TIMEOUT`, `EMBEDDING_ERROR`, `STUB_MISS`
Similarity	Cosine value (0-1). Write `""` (empty) when similarity couldn't be computed.
Error	Short text if status is not `PASS`; empty string otherwise.
File-name pattern	`{agent_slug}_results_{YYYY-MM-DDTHH-MM-SSZ}.csv` Example: `us_payroll_results_2025-08-01T06-42-18Z.csv`
Retention	The runner saves the file under `/workspace/results/...` and the GitHub Actions workflow uploads it **as a CI artefact**, so it can be downloaded from the job page.

Sample CSV

```
test_name,status,similarity,error
why_net_lower,PASS,0.92,
bonus_tax,FAIL,0.63,"similarity below threshold"
missing_stub,TIMEOUT,,"agent did not finish within 60 s"
```

Task Checklist (MVP)

#	Task	Owner	Definition-of-Done
1	Scaffold Python package `ai_runner/` with `click` entry-point	Backend	`python -m ai_runner --help` prints usage
2	YAML loader & schema validation (`ruamel.yaml` , `pydantic`)	Backend	A sample scenario parses with no errors; invalid YAML fails CI
3	FastAPI stub server + dynamic route loader from `tool_fixtures`	Backend	Stub returns correct JSON for at least two tools and two calls to the same tool
4	SSE client (`httpx-sse`) to stream answers from AI-agent	Backend	Round-trip to local agent completes and yields full text
5	Embeddings grader (`openai` SDK) with cosine comparison	Backend	Similarity ≥ threshold passes, otherwise fails; unit-test proves both paths
6	CSV writer (`test_name, status, similarity, error`) and timestamped file-name pattern	Backend	File named `us_payroll_results_<timestamp>.csv` appears in `/workspace/results`
7	Failure handling & exit-codes (TIMEOUT, EMBEDDING_ERROR, STUB_MISS)	Backend	Integration test shows runner continues after one failure and exits 1 at the end

#	Task	Owner	Definition-of-Done
8	Docker image (`Dockerfile` or `uvicorn` base) for runner	DevOps	`docker run … ai-runner --agent us_payroll` passes local tests
9	`docker-compose.test.yml` (agent + stub + runner)	DevOps	`docker compose up` streams answers and produces results.csv
10	GitHub Actions workflow with **matrix** over `agent` slugs; uploads results as artefacts	DevOps	Two agent folders trigger two parallel jobs; artefacts visible in Actions UI
11	Weekly scheduled workflow (drift detection)	DevOps	Job runs every Monday 06:00 UTC and posts artefacts
12	Seed 3 reference scenarios under `tests/us_payroll/`	Payroll SME	All three pass in CI with similarity ≥ 0.88