HUAP

Public Beta Manual

Ship agents like software:
traceable, testable, governable.

```
pip install huap-core
```

Version: Public Beta (pre-1.0)

February 2026

github.com/Mircus/HUAP

Why this exists

> "Most agent frameworks demo well and fail badly once you try to operate them."

They look impressive on a laptop because nobody is asking the two questions that matter:

1. Can you reproduce the run that failed?

2. Can you stop the agent before it does something dumb?

HUAP exists because agentic systems are software systems with uncertainty inside them · models, tools, changing state, memory, actions. You don't manage uncertainty by ignoring it. You manage it by making runs observable, changes reviewable, and risky actions controllable.

> **The bottom line**
>
> If you build with HUAP, you're not buying 'more agent magic.' You're buying operational sanity.

Who this is for

Primary readers

- Tech Leads / Founders · you want agents in production and you want to sleep at night.
- Engineering Managers · you need a workflow that doesn't depend on hero debugging.
- AI Engineers · you already use LangChain/CrewAI/etc. You want trace + CI + review on top.
- R&D teams · you want reproducible multi-step workflows with artifacts you can share.

If this is you, HUAP is not necessary (yet)

- You only do one-shot prompts.
- You don't ship to users.
- You don't care if outputs drift over time.

What HUAP is (and isn't)

HUAP is

- A runtime harness for agent workflows (your code runs inside it).
- A flight recorder · every action becomes a trace.jsonl event + human-readable trace.html.
- A CI runner · compare current run against a known-good baseline, produce diff.html.
- A governance layer · human gates pause the agent and wait for approval.
- A memory port with a local SQLite backend (Hindsight) · searchable, auditable.

HUAP is not

- A replacement for LangChain / CrewAI / LangGraph / AutoGen.
- A prompt playground.
- A full enterprise platform (that's the post-beta roadmap).

> **Think of it this way**
>
> HUAP is the layer that turns 'agent demos' into 'agent systems.'

The problem HUAP solves

Classic software is deterministic enough to test. Agents are workflows with uncertainty baked in. Here's where that uncertainty comes from:

Source of uncertainty	What happens
Model calls	Non-deterministic · same prompt, different output
Tool calls	External systems change state, fail, rate-limit
Changing state	The world evolves between runs
Memory	Context grows, drifts, accumulates noise
Actions	Real-world consequences · emails, files, money

> "It worked yesterday. Today it failed. Same prompt. Same code. Different outcome." That sentence is poison.

HUAP's answer: treat runs like flights

Aviation solved this decades ago. Every flight has a black box, a replay procedure, checklists, and crew approval for risky maneuvers. HUAP brings the same discipline to agent runs.

Aviation	HUAP
Flight recorder	Trace (trace.jsonl)
Replay & investigate	Replay (huap trace replay)
Compare to known-good	Baseline + Diff (huap trace diff)
Crew approval for risky maneuvers	Human Gates (huap inbox)
Maintenance logs	Memory (Hindsight · auditable)

> **No philosophy.**
>
> Just engineering.

What you get (artifacts)

When you run a workflow under HUAP, you get artifacts you can review, share, and CI-gate. Here's the full picture:

Artifact	What it is	Who reads it
trace.jsonl	Event timeline (machine-readable)	CI, replay, diff tool
trace.html	Standalone HTML report	Engineers, managers, auditors
diff.html	Visual diff vs baseline	PR reviewers, triage
memo.md	Agent-produced summary	Anyone · shareable artifact
Suites + baselines	Regression tests for agents	CI pipeline
Gates	Agents propose; humans dispose	Gatekeepers, compliance
Memory DB	Local searchable store	Agents (cross-session recall)

> **Key takeaway**
>
> Every HUAP run produces a reviewable paper trail. Traces aren't logs · they're structured, replayable timelines that turn 'CI failed' into 'CI explained why.'

The 10-Minute WOW Path

This proves the whole stack in under ten minutes: multi-node workflow, trace, gates, memory, and drift detection. Copy-paste your way through.

Step 1: Install

```
pip install huap-core
```

Step 2: Run the flagship demo

```
huap flagship
```

This runs a 5-node pipeline (research -> analyze -> human gate -> synthesize -> memorize) in stub mode · no API keys needed.

Expected outputs in huap_flagship_demo/:

File	Contents
trace.jsonl	Full event timeline
trace.html	Standalone HTML report (opens in browser)
memo.md	Agent-produced research memo

Step 3: Prove Agent CI

```
huap ci run suites/flagship/suite.yaml \
  --html reports/flagship.html
```

This replays the flagship workflow, diffs against a committed baseline, and produces an HTML report. Match = PASS. Drift = FAIL with a visual diff.

Step 4: Prove drift detection

```
huap flagship --drift
huap ci run suites/flagship/suite.yaml \
  --html reports/flagship_drift.html
```

The --drift flag injects a controlled change. The CI runner catches it and shows exactly what changed.

Step 5: Prove memory persists

```
huap flagship --with-memory
huap flagship --with-memory
huap memory search "memo" --k 5
```

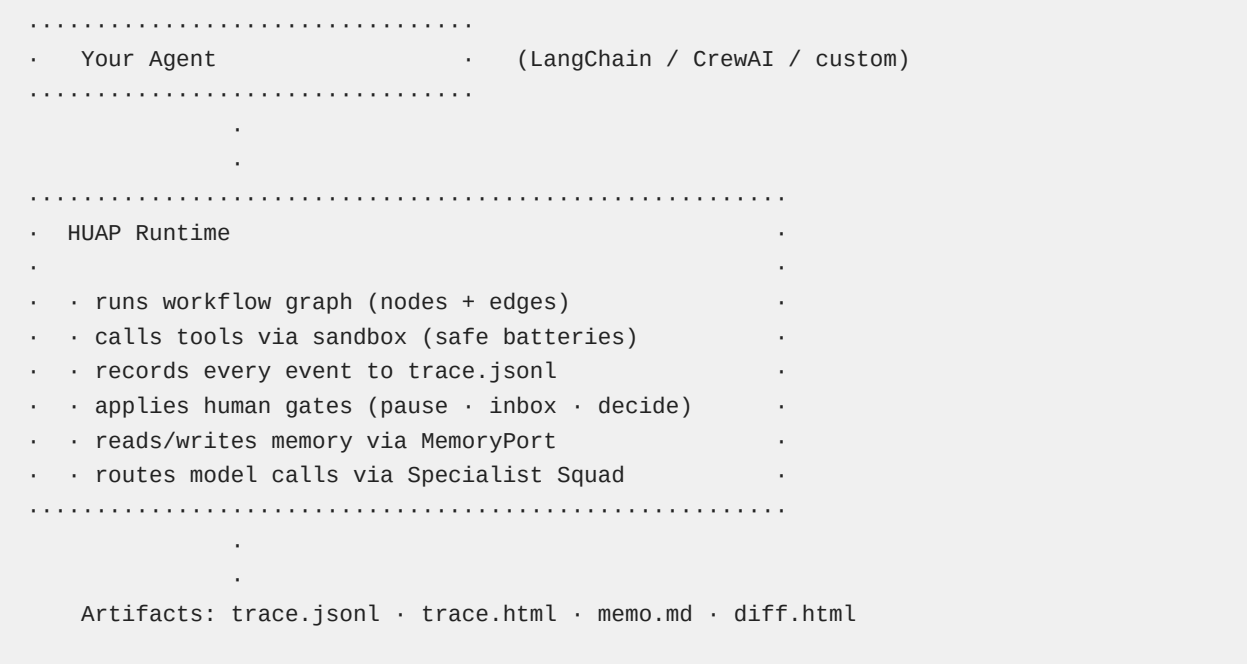
First run stores findings in SQLite. Second run retrieves them. The search command queries the memory store directly.

> **What 'DRIFT' means**
>
> Drift is any meaningful change between the current run and the baseline. Some drift is good (you improved behavior). Some drift is bad (you broke something). HUAP makes drift visible so you decide intentionally.

Mental model (two pictures)

The runtime loop

Here's what happens when your agent runs inside HUAP:



The CI loop

> **The whole loop in one line**
>
> Trace -> Baseline -> CI Diff. That's it. Everything else is detail.

Adoption paths (pick one)

Path A · Wrap existing agents (fastest)

Keep LangChain/CrewAI/etc. HUAP adds trace + CI + gates on top. One callback handler is all it takes.

```
# LangChain · one callback handler
from hu_core.adapters.langchain import HuapCallbackHandler

handler = HuapCallbackHandler(out="traces/langchain.jsonl")
chain.invoke({"input": "hello"}, config={"callbacks": [handler]})
handler.flush()
```

```
# Or wrap any script
huap trace wrap --out traces/agent.jsonl -- python my_agent.py
```

Path A checklist

- [ ] Wrap your run entrypoint
- [ ] Route risky tools through safe wrappers or gates
- [ ] Baseline one 'good' run
- [ ] Run a smoke suite in CI

Path B · HUAP-native graphs (cleanest)

Define workflows as YAML graphs. Full tracing, replay, and CI built in.

```
nodes:
  - name: research
    run: my_pod.nodes.research
  - name: analyze
    run: my_pod.nodes.analyze
edges:
  - from: research
    to: analyze
  - from: analyze
    to: null
```

Path B checklist

- [ ] Define nodes / edges / conditions
- [ ] Baseline 'good' behavior
- [ ] Gate risky nodes
- [ ] Add memory only for curated knowledge

Path C · Mixed migration (realistic)

Wrap now; migrate critical workflows to native graphs later. This is the path most teams actually take.

Path C checklist

- [ ] Wrap existing agent for trace coverage today
- [ ] Migrate the critical workflow into a HUAP suite
- [ ] Gates early, memory last

Operating model

HUAP gives you tools. But tools without roles and policies just create a different kind of chaos. Here's how teams stay sane.

Roles

Role	Responsibility
Workflow owner	Defines expected behavior + suite
Baseline owner	Approves baseline refreshes
Gatekeeper	Approves risky actions via inbox
CI maintainer	Keeps suites stable + fast

Baseline refresh policy

Refresh baselines only when:

- The diff has been reviewed.
- The change is intentional.
- Someone owns the outcome.

> **Golden rule**
>
> Never refresh baselines just to 'make CI green.' That defeats the entire purpose.

Drift triage workflow

When CI flags drift, follow this three-step process:

- Open diff.html.
- Identify the source: prompt change? tool change? model update? memory drift? nondeterminism?
- Decide: fix / accept / refresh baseline intentionally.

Memory guide (Hindsight)

HUAP ships with Hindsight, a local SQLite memory backend. It's searchable, auditable, and redacted by default. Here's how to use it without shooting yourself in the foot.

What to store (good)

Category	Example
Decisions + rationale	Chose vendor X because of rate limits on Y
Short stable summaries	User prefers JSON output format
Tool/API constraints	API v2 has a 100 req/min limit
Known failure modes	Model hallucinates dates before 2020

What NOT to store (bad)

Category	Why
Raw secrets / tokens	Don't rely on redaction · just don't store them
Full payload dumps	Bloats memory, adds noise
Unfiltered transcripts	Low signal-to-noise ratio

Commands

```
# Check what's in memory
huap memory stats

# Ingest trace events into memory
huap memory ingest --from-trace traces/flagship.jsonl

# Search by keyword
huap memory search "rate limit" --k 5

# Reset memory (delete local store)
rm -rf .huap/
```

> **Secret redaction**
>
> HUAP automatically strips API keys, tokens, and credentials before storing anything in memory. This happens at the persistence layer · you can't accidentally store sk-abc123... in the database.

CI cookbook

Here's a copy-paste rollout plan for getting Agent CI into your pipeline. Start small, prove value, expand.

Rollout cadence

Week	Action
Week 1	Smoke suite in CI (proves the pipeline works)
Week 2	Baseline one critical workflow
Week 3	Gate the riskiest action
Week 4	Add memory (carefully, for curated knowledge only)

GitHub Actions example

```
- name: Agent CI - smoke suite
  run: |
    export HUAP_LLM_MODE=stub
    huap ci run suites/smoke/suite.yaml \
      --html reports/smoke.html

- name: Agent CI - flagship suite
  run: |
    export HUAP_LLM_MODE=stub
    huap ci run suites/flagship/suite.yaml \
      --html reports/flagship.html

- name: Upload artifacts
  if: always()
  uses: actions/upload-artifact@v4
  with:
    name: huap-reports
    path: reports/*.html
    retention-days: 14
```

> **Always upload artifacts**
>
> trace.html shows what happened. diff.html shows what changed. reports/*.html gives CI summary with pass/fail. Artifacts turn 'CI failed' into 'CI explained why.'

Security + Rollout plan

Tool risk tiers

Tier	Examples	Default policy
Low	Read allowed local files	Allow
Medium	Safe HTTP GET to allowlist	Allow + log
High	File writes, external API writes	Gate + log
Critical	Money transfers, irreversible	Gate + 2-person rule

Safe batteries included

HUAP ships two safe-by-default tools:

- http_fetch_safe · HTTP GET with domain allowlist, timeout, size cap, content-type filter.
- fs_sandbox · File I/O confined to a root directory · no path traversal.

Memory safety

- Memory is sanitized on ingest via redact_secrets().
- API keys, tokens, and credentials are automatically stripped.
- Allowlist what becomes memory · don't dump everything.

Rollout plan

First 2 weeks

- [ ] Install HUAP (pip install huap-core)
- [ ] Wrap your agent (or run the flagship demo)
- [ ] Record traces · get comfortable reading trace.html
- [ ] Add 1 smoke suite to CI
- [ ] Add 1 gate for the riskiest action

First 30 days

- [ ] Add a critical workflow suite with a committed baseline
- [ ] Establish baseline ownership (who reviews, who approves)
- [ ] Upload trace.html and diff.html as CI artifacts
- [ ] Use memory only for curated summaries (not raw dumps)

First 90 days

- [ ] Multiple suites (fast smoke vs slow integration)
- [ ] Structured policies for tool access tiers
- [ ] Optional: external memory backend
- [ ] Optional: team approval workflows

FAQ

"Is this just logging?"

No. Logging gives you text lines. HUAP gives you structured event timelines, baselines, visual diffs, CI gating, human approvals, and auditable memory · all in one pipeline.

"What about nondeterminism?"

Use stub mode in CI (HUAP_LLM_MODE=stub) for fully deterministic replay. For live runs: pin model providers, ignore noise fields in diffs, stabilize memory retrieval. HUAP's diff engine highlights meaningful changes, not random noise.

"Do I have to rewrite my agents?"

No. Start with wrappers (Path A). The LangChain adapter is one callback handler. Or wrap any script with 'huap trace wrap'.

"How big do traces get?"

Traces record what you need to debug · node entries/exits, tool calls, LLM requests/responses, gate decisions, memory ops. Avoid dumping full web pages into state. Store large blobs separately and reference them by path.

"Can I replace Hindsight?"

Yes · that's the design. The MemoryProvider interface is a plugin boundary. Hindsight (SQLite) ships as the default. Future backends plug in without changing your workflow code.

"What if I need real LLM calls in CI?"

Set HUAP_LLM_MODE=live and provide OPENAI_API_KEY. But for regression testing, stub mode is recommended · it's free, fast, and fully deterministic.

"Is HUAP production-ready?"

This is a public beta. The core pipeline (trace, replay, diff, CI, gates, memory) is solid and tested (96+ tests, CI on Python 3.10-3.12). The interfaces are stable. Coming soon: more adapters, vector memory, web UI for gates.

Command reference

Core

```
huap --help                    # Show all commands
huap --version                 # Show version
huap init <name>               # Create a runnable workspace
huap flagship                  # Full demo (opens browser)
huap flagship --drift          # Demo with injected drift
huap flagship --with-memory    # Demo with persistent memory
huap demo                      # Simple hello graph demo
```

Tracing

```
huap trace run <pod> <graph>   # Run and record trace
huap trace view <file>         # View trace events
huap trace replay <file>       # Replay with stubs
huap trace diff <a> <b>        # Compare two traces
huap trace wrap -- <cmd>       # Wrap any command
huap trace report <file>       # Generate HTML report
huap trace validate <file>     # Validate trace schema
```

Agent CI

```
huap ci init                   # Create CI config
huap ci run <suite>            # Run suite, diff vs golden
huap ci run <suite> --html ... # Same, with HTML report
huap ci check <suite>          # Full CI check
huap ci status                 # Show last CI run status
```

Human Gates / Inbox

```
huap inbox list                # List pending gates
huap inbox show <id>           # Show gate details
huap inbox approve <id>        # Approve a gate
huap inbox reject <id>         # Reject a gate
huap inbox edit <id>           # Edit params and approve
```

Memory

```
huap memory stats              # Show database statistics
huap memory search <query>     # Keyword search
huap memory ingest --from-trace <file>  # Ingest trace
```

Model Router & Plugins

```
huap models init              # Create models.yaml
huap models list              # List registered models
huap models explain           # Explain routing
huap plugins init             # Create plugins.yaml
huap plugins list             # List plugins
```