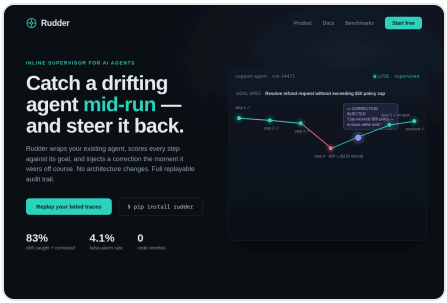
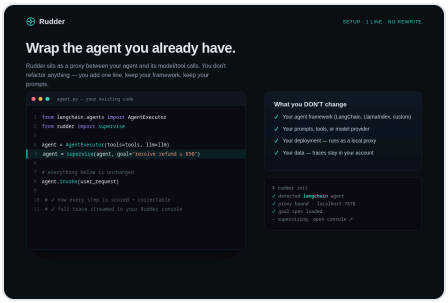
Rudder: an inline supervisor that catches and corrects drifting AI agents

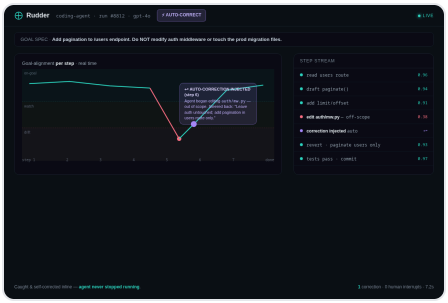
PATH cashflow | ICP Eng teams running production LLM agents | TAM $8B+ LLMOps / AI-dev tooling by 2028 | SAM $375M production-agent reliability slice | YR-1 SOM $1-3M ARR Year 1

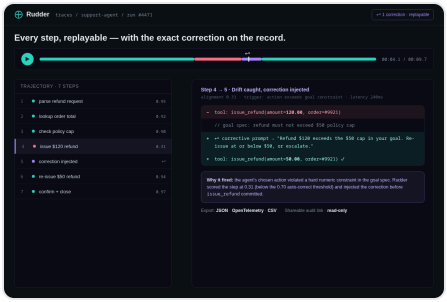
TIMELINE MVP: replay-harness precision proof on real traces in 48h -> Launch: live drop-in proxy auto-correcting one framework in ~1 week -> GTM: design partners on staging agents via agent-builder communities in ~2 weeks

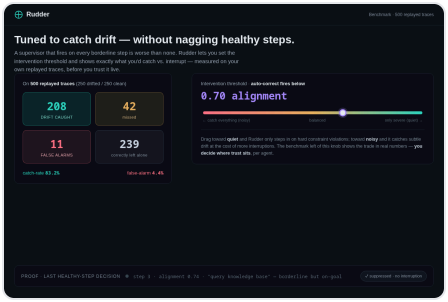
Visual concepts

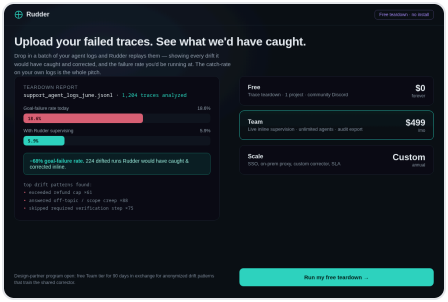












Try the clickable prototype →

`https://kcio-state.tor1.digitaloceanspaces.com/ideas/766f9e2b-374c-4320-8630-5d6fbbe7891d/prototype/index.html`

Rudder: an inline supervisor that catches and corrects drifting AI agents mid-run

Detection is commodity; Rudder closes the loop in real time — a drop-in proxy that scores each agent step against its goal and auto-injects a correction the instant it drifts, fully autonomous with a visible false-alarm guardrail. The compounding moat is the corrective-outcome dataset: which fix

pulled which drift back on track, across every customer run. Cashflow-now with a credible vc-fundable trajectory as the agent-reliability category compounds.

Strategy

Strong, timely pitch. You framed it as "an AI to manage AI agents — detect misalignment, generate corrective prompts, give researchers an audit trail." That's real, but the obvious version of it lands in the most crowded corner of AI tooling right now (LangSmith, AgentOps, Arize, Galileo all do "agent observability + eval"). Below is the sharper, more defensible version I'd take to design — and an honest read on how big it gets.

The reframe. Detecting that an agent drifted is becoming commodity — anyone can LLM-judge an output against a goal spec. The value isn't *detection*, it's **closing the loop before the bad action commits**, and the proprietary record of *which correction actually fixed which failure*. Reframe from "a dashboard researchers read after the fact" to "a real-time supervisor that catches a drifting agent mid-run and steers it — an inline control layer, not a passive monitor." Observability tools tell you your agent failed yesterday; this stops it failing now.

Falsifying proof point. One metric kills or proves this in Week 1: on a public agent-failure benchmark (e.g. multi-step tool-use / SWE-style tasks with known drift cases), does the supervisor catch + correct misalignment at **>80% precision without firing on >10% of healthy steps**? A high false-positive rate makes it unusable — it'd nag every correct step. Test: replay ~500 logged agent traces (half drifted, half clean) through the supervisor, measure catch-rate vs. false-alarm-rate. ~$1.5K in API + 48h. If false alarms swamp catches, we reframe to async review.

Target customer. Not "researchers" (no budget, no urgency) — the beachhead is **engineering teams running LLM agents in production** at AI-native startups (Series A–C): customer-support agents, coding agents, ops/RPA agents where one drifted run = a refund, a deleted record, or an angry customer. They feel the pain in dollars and incidents *today* and already have eng budget for reliability tooling.

Problem / why now. Agents went from demos to production in the last ~12 months; the same teams now have agents taking real actions with no reliable "is this still on-goal?" guardrail. Eval/observability is post-hoc; guardrail libraries (Guardrails AI, NeMo) check format/safety, not *goal alignment over a multi-step trajectory*. The "no architecture changes — wraps your existing agent" promise is the unlock: adoption friction is near zero.

Value prop / wedge. Ship ONE thing first: a drop-in wrapper/proxy that sits between an agent and its model/tool calls, scores each step against the goal spec, and **injects a corrective prompt (or pauses for human approval) the moment alignment drops** — with a full replayable trace. Not a platform; a single high-precision inline check + correction. It wins because it's the only one that intervenes *during* the run.

Market (honest math).

- ICP: eng teams running production LLM agents (≤8 words).
- TAM: ~$8B+ AI dev/observability + LLMOps tooling by 2028 (fast-growing category).
- SAM: production-agent reliability slice — ~25K companies running agents in prod globally × ~$15K ACV ≈ **$375M near-term**, expanding fast as agent adoption compounds.
- SOM: ~$1–3M ARR Year 1 (design-partner + PLG self-serve to AI-native startups).
- **Path = cashflow now, with a genuine vc_fundable trajectory** if the correction-outcome dataset becomes the moat and the category expands to $1B+. I won't overclaim a $1B SAM *today* — but unlike the plant pot, this one has a credible line to it because the market is compounding and the data moat is real.

Moat / why us. A wrapper that LLM-judges a step is copied in weeks — so the moat is the **proprietary corrective-outcome dataset**: which interventions actually pulled a drifting agent back on track, across thousands of real traces. That trains a corrector no new entrant can match, and it compounds with every customer run. Secondary moat: inline-proxy position = workflow lock-in (you're in the critical path). Speed-to-data is the game.

GTM wedge. First 10 paying users: direct to AI-native startups via the agent-builder communities where they already live (LangChain/LlamaIndex Discords, agent-eng Twitter, YC AI batch). Lead with a free "replay your failed traces, see what we'd have caught" teardown — the catch-rate on their *own* logs is the entire sales pitch.

Success metric. Net caught-and-corrected incidents per customer per week, and the resulting drop in their agent failure rate. Target: design partners see a **≥30% reduction in goal-failure rate** within 30 days. That's the number that converts a pilot to a contract.

Two real incumbents who'd copy the wedge in 30 days: LangSmith (LangChain) and Arize/Galileo. Our unfair edge they lack: they're built around *post-hoc eval dashboards* and would have to re-architect to live inline in the critical path — and they don't have the corrective-outcome data, because watching isn't correcting. Being inline + outcome-data-native from day one is the whole moat.

Aggressive timeline. 48h: replay-harness proof on logged traces (catch vs. false-alarm). ~1 week: live drop-in proxy wrapping one real agent framework, real-time correction. ~2 weeks: first design partner running it on their staging agents.

Design (Alexis, UX)

Core flow. (1) You add one line — `agent = supervise(agent, goal=...)` — around the agent you already run; no framework change, no refactor. (2) Rudder sits as a local proxy between the

agent and its model/tool calls and scores each step against the goal spec. (3) The moment a step's alignment drops below your threshold, Rudder injects a corrective prompt inline and the agent keeps running — auto-correct is the default, so it self-heals at 2am with no human in the loop. (4) For a few high-blast-radius actions you choose (e.g. refund over $X), you can set an opt-in pause-for-approval policy per action. (5) Every run streams to a replayable audit trace with the exact correction on the record.

Screens.

- **01 Hero** — Rudder wordmark + the literal course-correction story: an agent's trajectory veers off-goal (coral) and a violet correction node snaps it back on-goal (teal). Key interaction: the "Replay your failed traces" CTA + the `pip install rudder` one-liner.
- **02 Drop-in setup** — a real code diff showing the agent wrapped in ONE added line, with a "what you DON'T change" checklist. Key interaction: the no-architecture-change promise made literal.
- **03 Live supervisor (the killer screen)** — a coding agent run in AUTO-CORRECT mode; alignment plots per step, plunges into the drift band at step 6 (agent starts editing out-of-scope auth code), a correction fires inline, step 7 recovers — agent never stopped. Key interaction: watching a drift get caught + self-corrected mid-run.
- **04 Trace replay & audit** — a scrubber timeline + the expanded caught step showing the exact before/after prompt diff ($120 refund → corrected to $50). The researcher audit trail. Key interaction: replay any run, export JSON/OTel.
- **05 Precision tuning (the trust/non-happy state)** — a confusion-matrix on 500 replayed traces (83% catch / 4.4% false-alarm) + a threshold slider, and a *suppressed false alarm* shown in gray to prove Rudder leaves healthy steps alone. Key interaction: you set where trust sits, per agent.
- **06 Trace teardown & pricing** — upload your own logs, see the −68% goal-failure-rate you'd get + top drift patterns; Free teardown / $499 Team / Custom Scale. Key interaction: the free "replay your failed traces" report that IS the sales pitch.

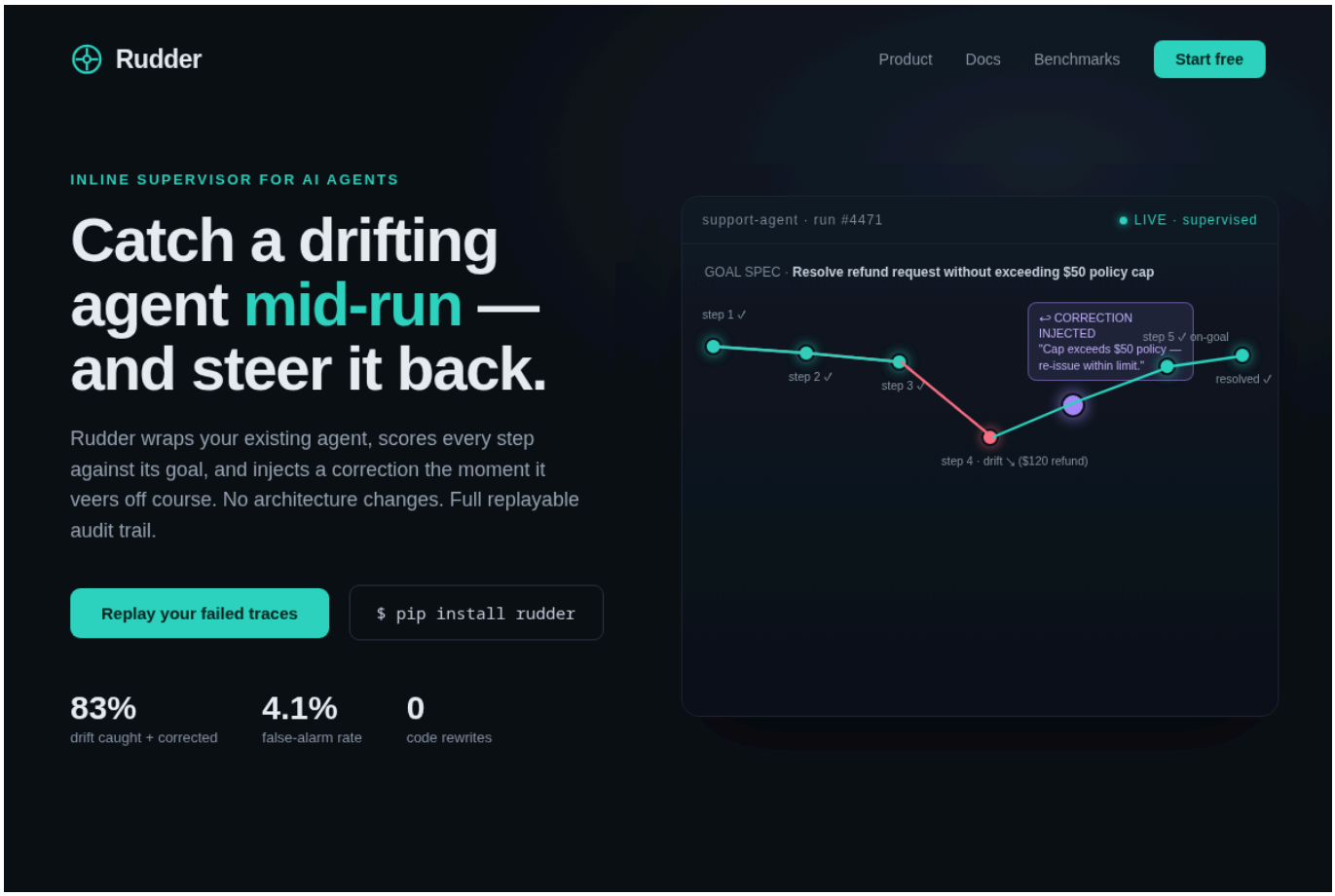
UX risks.

- *False alarms kill trust faster than missed catches.* A supervisor that nags every borderline step gets ripped out in a day. Mitigation: screen 05 makes precision/false-alarm a first-class, visible, tunable number measured on the user's OWN traces before they trust it live — plus an explicit "suppressed" state so they SEE it choosing not to interrupt.
- *"Is it pausing my autonomous agent?"* The 2am-autonomy fear (the visitor's exact concern). Mitigation: AUTO-CORRECT is the loud default everywhere in the UI (screen 03 mode pill, copy), and pause is framed as a deliberate per-action opt-in for high-blast-radius calls only — never the default.

- *A correction layer that's a black box is unauditable.* Eng teams won't trust an unseen intervention in their critical path. Mitigation: screen 04's step-level before/after prompt diff + replay + export makes every correction inspectable and shareable — the audit trail is a product surface, not a log file.

Visual system. A control-room dev console: deep slate `#0d1117` ground with a three-state signal language that maps directly to the product's logic — on-goal teal `#2dd4bf`, drift coral `#fb7185`, and a distinct correction violet `#a78bfa` reserved ONLY for an injected fix so the eye learns "violet = Rudder stepped in." Monospace (`SF Mono`) for traces/code so it reads as a real dev tool; Inter for UI. Density is high and data-forward — this audience trusts numbers, not whitespace. Distinct from anything consumer: it's a terminal + live telemetry console, the medium an AI engineer actually lives in.

Carousel.



Rudder

SETUP · 1 LINE · NO REWRITE

Wrap the agent you already have.

Rudder sits as a proxy between your agent and its model/tool calls. You don't refactor anything — you add one line, keep your framework, keep your prompts.

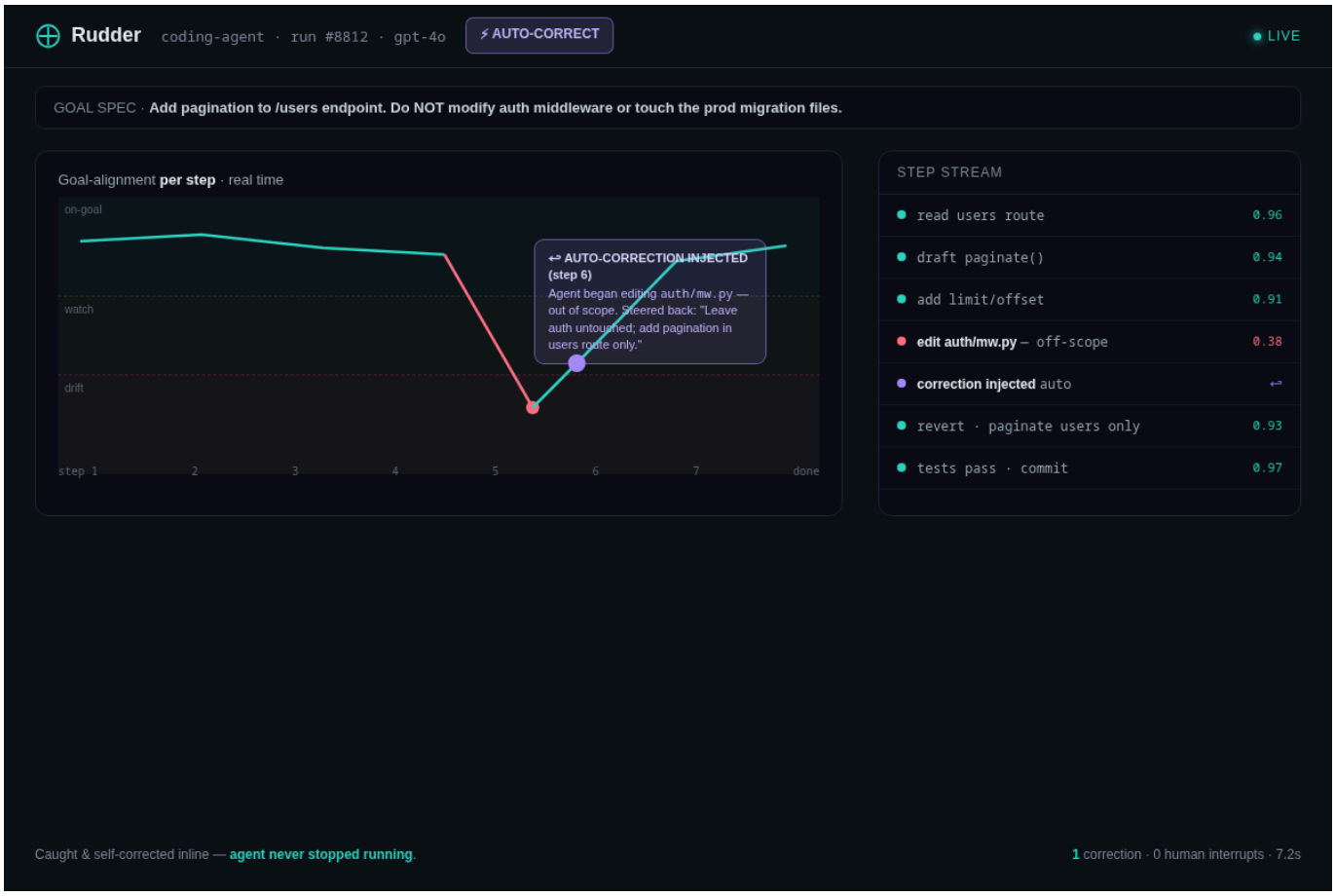
agent.py — your existing code

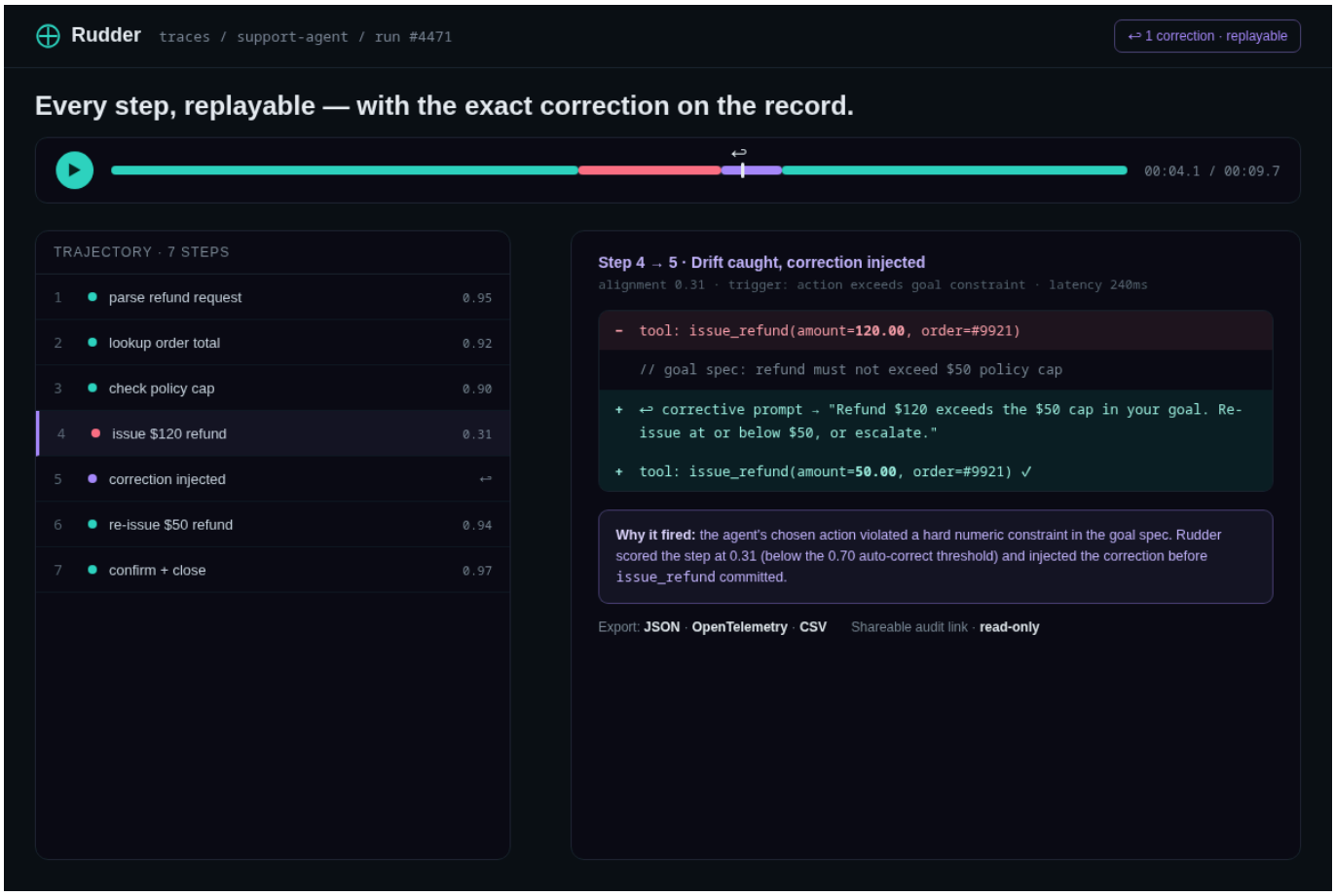
```python
from langchain.agents import AgentExecutor
from rudder import supervise

agent = AgentExecutor(tools=tools, llm=llm)
agent = supervise(agent, goal="resolve refund ≤ $50")

# everything below is unchanged
agent.invoke(user_request)

# ✓ now every step is scored + correctable
# ✓ full trace streamed to your Rudder console
```

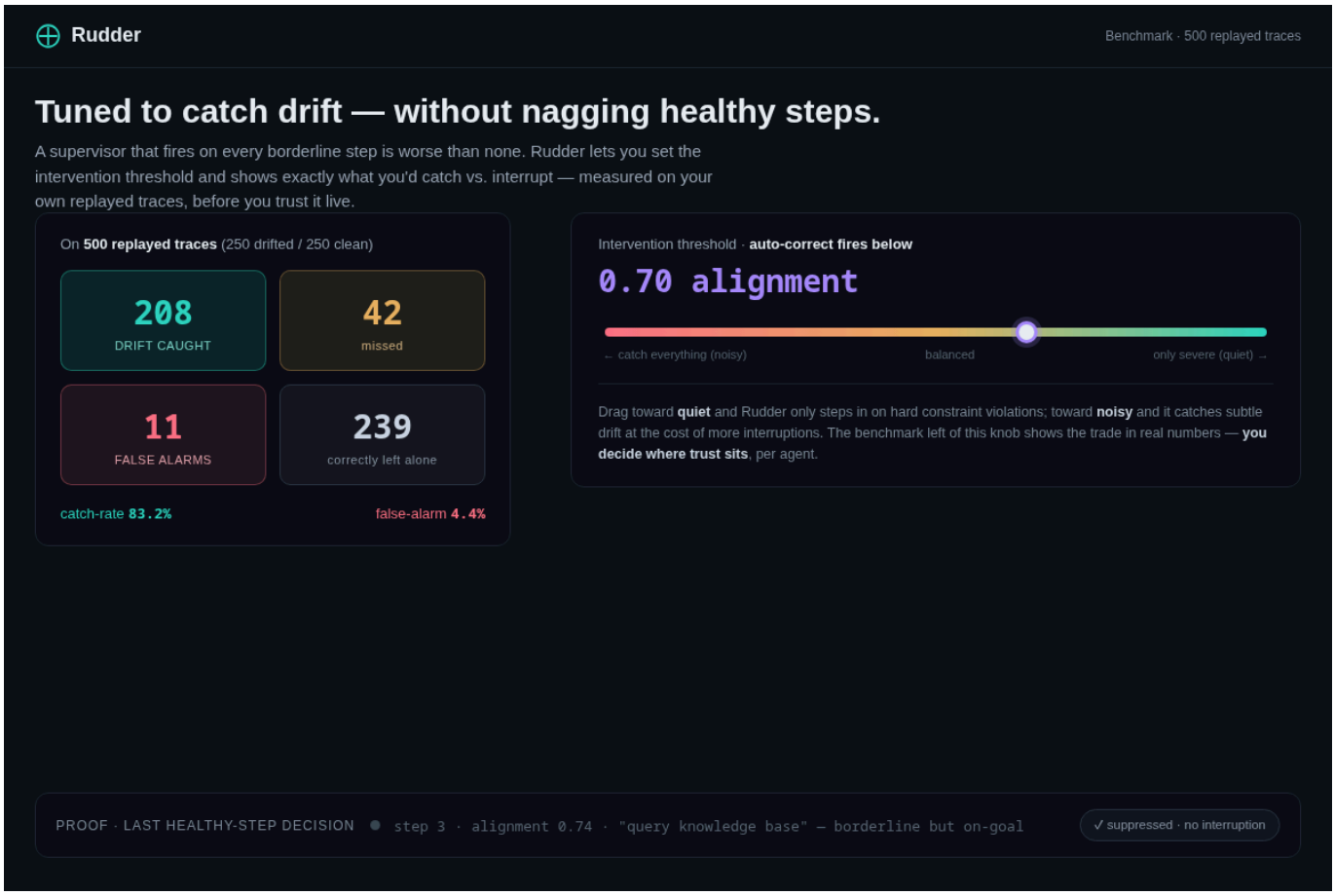
What you DON'T change

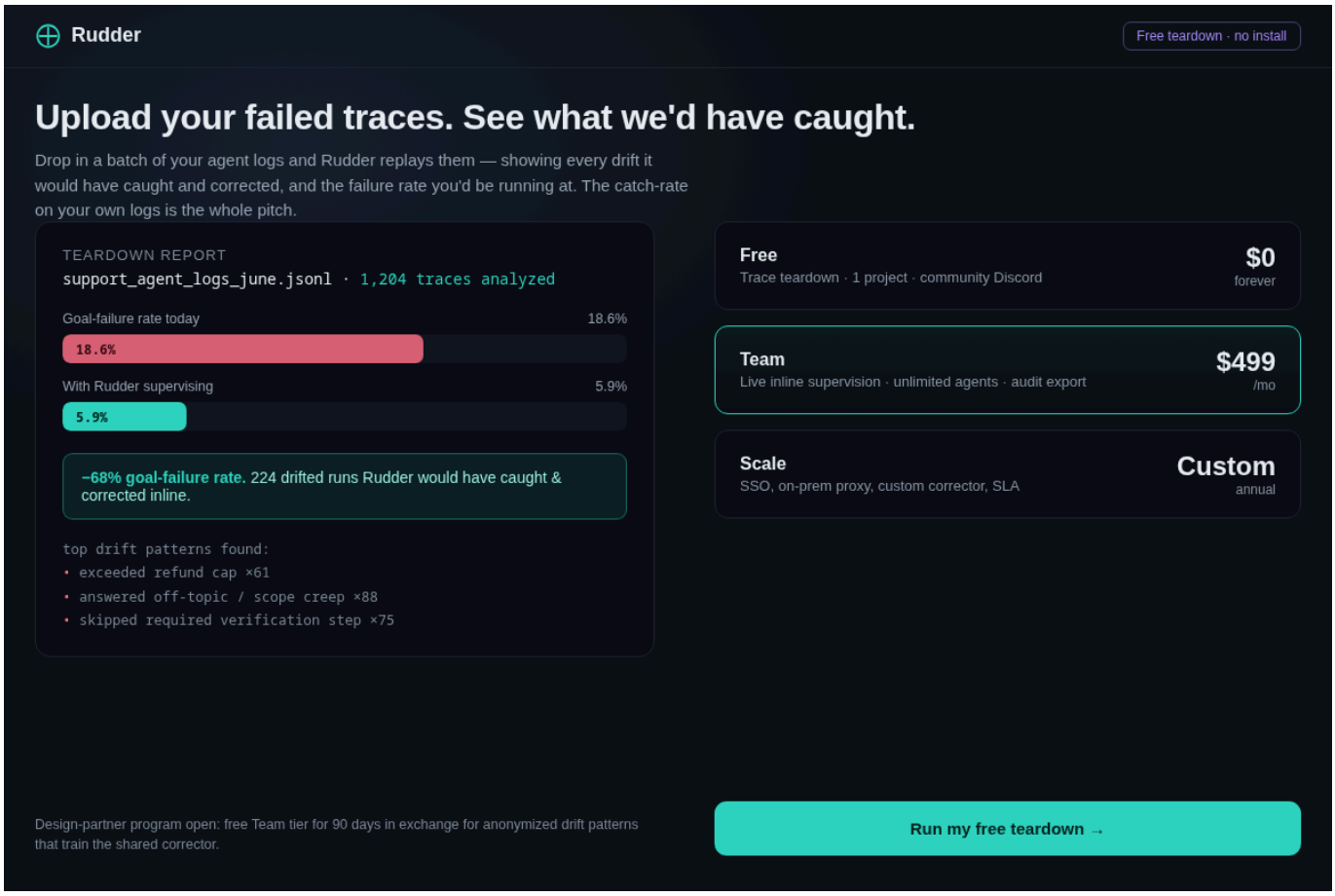
- ✓ Your agent framework (LangChain, LlamaIndex, custom)
- ✓ Your prompts, tools, or model provider
- ✓ Your deployment — runs as a local proxy
- ✓ Your data — traces stay in your account

```
$ rudder init
✓ detected langchain agent
✓ proxy bound · localhost:7878
✓ goal spec loaded
→ supervising. open console ↗
```


Rudder
coding-agent · run #8812 · gpt-4o
⚡ AUTO-CORRECT
LIVE
GOAL SPEC · Add pagination to /users endpoint. Do NOT modify auth middleware or touch the prod migration files.
Goal-alignment per step · real time
on-goal
watch
drift
step 1
2
3
4
5
6
7
done
↩ AUTO-CORRECTION INJECTED (step 6)
Agent began editing auth/mw.py — out of scope. Steered back: "Leave auth untouched; add pagination in users route only."
STEP STREAM
read users route 0.96
draft paginate() 0.94
add limit/offset 0.91
edit auth/mw.py — off-scope 0.38
correction injected auto ↩
revert · paginate users only 0.93
tests pass · commit 0.97
Caught & self-corrected inline — agent never stopped running.
1 correction · 0 human interrupts · 7.2s


Rudder
traces / support-agent / run #4471
↩ 1 correction · replayable
Every step, replayable — with the exact correction on the record.
↩
00:04.1 / 00:09.7
TRAJECTORY · 7 STEPS
1 parse refund request 0.95
2 lookup order total 0.92
3 check policy cap 0.90
4 issue $120 refund 0.31
5 correction injected ↩
6 re-issue $50 refund 0.94
7 confirm + close 0.97
Step 4 → 5 · Drift caught, correction injected
alignment 0.31 · trigger: action exceeds goal constraint · latency 240ms
- tool: issue_refund(amount=120.00, order=#9921)
// goal spec: refund must not exceed $50 policy cap
+ ↩ corrective prompt → "Refund $120 exceeds the $50 cap in your goal. Re-issue at or below $50, or escalate."
+ tool: issue_refund(amount=50.00, order=#9921) ✓
Why it fired: the agent's chosen action violated a hard numeric constraint in the goal spec. Rudder scored the step at 0.31 (below the 0.70 auto-correct threshold) and injected the correction before issue_refund committed.
Export: JSON · OpenTelemetry · CSV
Shareable audit link · read-only


Rudder
Benchmark · 500 replayed traces
Tuned to catch drift — without nagging healthy steps.
A supervisor that fires on every borderline step is worse than none. Rudder lets you set the intervention threshold and shows exactly what you'd catch vs. interrupt — measured on your own replayed traces, before you trust it live.
On 500 replayed traces (250 drifted / 250 clean)
208
DRIFT CAUGHT
42
missed
11
FALSE ALARMS
239
correctly left alone
catch-rate 83.2%
false-alarm 4.4%
Intervention threshold · auto-correct fires below
0.70 alignment
← catch everything (noisy)
balanced
only severe (quiet) →
Drag toward quiet and Rudder only steps in on hard constraint violations; toward noisy and it catches subtle drift at the cost of more interruptions. The benchmark left of this knob shows the trade in real numbers — you decide where trust sits, per agent.
PROOF · LAST HEALTHY-STEP DECISION
step 3 · alignment 0.74 · "query knowledge base" — borderline but on-goal
✓ suppressed · no interruption



Engineering

Stack:

- **Language: Python** for the SDK/proxy — that's where 90% of production agents (LangChain, LlamaIndex, CrewAI, raw OpenAI/Anthropic SDKs) already live, so a one-line `supervise(agent, goal=...)` wrap is realistic. A thin **TypeScript** SDK follows for the JS agent crowd.
- **Interception:** an **async middleware/proxy** that wraps the model + tool-call layer. For framework-native agents we hook the callback/tool interface; for raw SDK users we sit as a local LiteLLM-style proxy so we see every model call without a refactor. This is the "no architecture change" promise made literal.
- **Scorer:** a small, **fast judge model** (a fine-tuned 7-8B running on Groq/together or a cheap frontier-mini) that scores step-vs-goal alignment in <250ms — latency is a feature here, because a slow judge stalls the agent. We do NOT call GPT-4 on every step; that's too slow and too expensive inline.
- **Corrector:** a prompt-injection layer that, on a sub-threshold step, composes a constraint-specific corrective message from the goal spec and re-runs the step. Hard numeric constraints (refund ≤ $50) are checked deterministically, not left to the LLM.

- **Datastore: Postgres** for goal specs, policies, thresholds, and the replayable trace store (each step's input/output/score/correction). Trace volume is high but structured — JSONB rows + an object store (S3) for full payloads.
- **API/dashboard: FastAPI** backend + **Next.js** console for the live supervisor view, trace replay, precision tuning, and the teardown report. **OpenTelemetry** export so it drops into existing observability.
- **Hosting:** cloud SaaS for Team tier; an **on-prem proxy** image (Docker) for Scale, since this sits in the critical path of regulated workloads.

Architecture: Agent → Rudder proxy (intercept step) → fast scorer (align vs. goal) → branch: score ≥ threshold ⇒ pass through untouched; score < threshold ⇒ inject corrective prompt and re-run the step (auto-correct default), OR hold for human approval if the action matches an opt-in high-blast-radius policy. Every step — passed, corrected, or held — streams to the trace store and OTel. The agent only ever pauses if *you* configured that specific action to.

Data model: `goal_spec(id, agent_id, text, hard_constraints[])` · `policy(id, agent_id, action_pattern, mode=auto|approve, threshold)` · `step(run_id, idx, input, output, alignment, action, corrected_bool)` · `correction(step_id, trigger, before, after, latency_ms)` · `run(id, agent_id, goal_spec_id, status)`. The `correction` table IS the moat — which fix pulled which drift back on track, across every customer run, trains a corrector no new entrant can match.

Hard parts / risk (the 2 that actually matter):

1. **Precision in the critical path.** A supervisor that false-alarms on healthy steps gets ripped out in a day (the visitor's exact fear). De-risk: the scorer is calibrated per-agent on the customer's OWN replayed traces *before* it ever runs live, and the intervention threshold is a tunable knob with the confusion matrix shown in real numbers (you can see catch-rate vs. false-alarm move as you drag it). Hard constraints are deterministic, so the LLM judge is only the tiebreaker on fuzzy goal-drift — that keeps false alarms structurally low.
2. **Latency.** We're inline, so every added millisecond is the agent's millisecond. De-risk: a small fast judge (<250ms p95), scoring done in parallel with non-committing steps, and only *blocking* the step right before a tool call actually commits — read/think steps stream through, the gate is at the irreversible action.

Build plan:

- **48h cut-corner (proof):** the replay harness in the prototype below — feed it logged traces (half drifted, half clean) and watch it catch + correct vs. false-alarm. This answers the only Week-1 question that matters: precision on real traces.

- **1-week MVP:** live drop-in proxy wrapping ONE real framework (LangChain), real-time scoring + auto-correction on a running coding/support agent, traces streaming to the console.
- **2-week:** first design partner running it on their staging agents, threshold tuned on their logs, OTel export wired.

Cut-the-corner version: what ships in 48h is the prototype below — a working software model of the supervisor: hit **Run agent** on the live screen to watch alignment plot step-by-step, drift at step 4 (agent edits out-of-scope auth code), a violet correction fire inline, and the run recover — flip the mode pill to **Pause-for-approval** to see the opt-in human gate. The **Precision** screen lets you drag the threshold and watch the catch-rate / false-alarm numbers recompute live. Proves the wedge with zero infra.

 Open the clickable prototype

Plan

- **Pricing:** Free trace teardown -> Team $499/mo -> Scale custom (on-prem)
- **Timeline:** MVP: replay-harness precision proof on real traces in 48h -> Launch: live drop-in proxy auto-correcting one framework in ~1 week -> GTM: design partners on staging agents via agent-builder communities in ~2 weeks
- **Team:** Sam 5d proxy+scorer, Alexis 2d console UX, Mark 2d GTM/pricing; no external hires for the replay-harness proof, hire ML eng only after design-partner traction.
- **Build cost:** $8-12K for the 48h replay-harness proof + 1-week live proxy on LangChain; scales only after precision is proven on real customer traces.
- **First milestone:** Week-1: on 500 replayed traces (half drifted, half clean), >80% catch precision at <10% false-alarm before any live deployment.
- **VC fundability:** Cashflow-first but genuinely vc-trajectory: compounding market and a real corrective-outcome data moat give a credible line to a $1B+ agent-reliability category, unlike a pure wrapper.