The Brilliant Employee · 10

The loop that would learn backwards

The Incident Files. Before the learning loop was ever switched on, a controlled sweep of its inputs showed it promoting the expensive model exactly when that model was failing.

sgnk.ai/brilliant ↗

Sagnik Mitra

Why this exists

sgnk is a one-person studio. A language model writes a large share of the production code. You do not need your own system to get answers out of a model. You need one to find out when the answers were wrong, because wrong work reads exactly like right work and no prompt fixes that.

Three things from that week, none visible from inside the tool: a sizing step ignored on 796 of 816 tasks, two fixes I had written up as done that were never made, and a detector whose 1,112 catches were mostly one test string of my own.

THE SYSTEM, AS OF 5 AUGUST 2026, 00:39Z	
rules written	11 June, 55 days ago
ledger running	27 June, 39 days
tasks logged	2,959, across 3,397 rows
controls on tool calls	**6 registered, 3 of them armed**

You can rent the model. You cannot rent the part that tells you it was wrong.

Five parts, and where today sits

Five parts sit around the model. What changes from one piece of writing to the next is which of them is under the microscope.

the Gate	sizes a job before a model is picked
the Gambler	recommends which model gets the job
the Rails	refuse the irreversible, before it runs
the Judge	grades finished work pass or fail
the Ledger	at least one append-only line per task

Listed in the order they act on a task, not the order they were built.

Today cuts across more than one of them, so no row is marked. Nothing below is assumed knowledge; everything this post needs, it explains.

This page is the map. Nothing here needs another post to make sense.

The Gambler, before the switch

The Gambler is the part of this workplace that recommends which AI model gets each job. It keeps a running score per model, and it carries an escalation rule: when a confidence number for a family of tasks drops below a bar, hand the work to the stronger, more expensive model.

> **Plain words: ARMING.** connecting a learning loop's decisions to live behaviour. Until then it only writes advisory rows: what it would have done.

This loop had run unarmed its entire life, and arming it was next on the plan. Before connecting anything, I wanted to see which way the maths actually pointed. Not by rereading the code. By measuring it.

Before you connect a wire, check which way the current flows.

The instrument: a controlled sweep

Hold every input fixed except one. Walk that one input across its full range. Record the decision at each step. If the decisions move in the direction you intended, the loop is safe to arm. If they move the other way, you have found a sign error that no code review would ever show you, because a sign error compiles, runs, and logs like a healthy policy.

Here the cheap model, sonnet, was held fixed at E[theta]=0.818, a strong track record. The expensive model, opus, was swept from 0.005, failing almost always, to 0.995, succeeding almost always. Five rows.

You cannot see the direction of a learning loop from its code. Sweep it.

Five rows, one flip

SONNET HELD AT E[THETA]=0.818, OPUS SWEPT	
opus E[θ]=0.005	decision=opus · p25=0.064
opus E[θ]=0.100	decision=opus · p25=0.262
opus E[θ]=0.500	decision=opus · p25=0.545
opus E[θ]=0.900	**decision=sonnet · p25=0.850**
opus E[θ]=0.995	**decision=sonnet · p25=0.973**
verbatim from the sweep record, 2026-08-02	

Read it top to bottom. When the expensive model succeeds one time in two hundred, the decision is: give it the work. When it succeeds 199 times in 200, the decision is: take the work away. The flip sits between 0.500 and 0.900, and it points exactly the wrong way.

Escalate to the expensive model precisely when it is failing: that was the policy.

Why: a transfer term, weighted 0.5

```
TRANSFER=0.5  opus a=40 b=1
  (opus works)  -> stay sonnet
TRANSFER=0.5  opus a=1  b=40
  (opus fails)  -> escalate to opus
TRANSFER=0    both -> opus

live: SGNK_BANDIT_TRANSFER = 0
```

Weighted 0.5, a strong result for the expensive model argued for staying cheap and a weak one argued for escalating to it. The live setting is 0.

A sign error does not look like a bug. It looks like a policy.

The sweep also refuted my own audit

An earlier audit of this loop carried a headline claim: the decision never compares the arms at all, so the loop is dead, not dangerous. The sweep refutes that as stated. Walk one input and the decisions move. The loop responds to its inputs. The defect is not deadness. It is direction.

The distinction matters more than it sounds. A dead loop wastes nothing but the effort of building it. A live loop with an inverted sign converts every observation into the opposite lesson, at full speed, with full confidence, and every dashboard reports it as learning.

Refute your own audits with instruments, not with re-reads.

Live status, stated plainly

WHAT ACTUALLY RAN, AND WHAT WAS FIXED	
negative rewards written	once, on 29 July
what it learned	4 negative marks, all wrong
time until gated off	74m 10s
decision rows on record	every one advisory, none live
the pick it computes	never assigned to anything
the fix	**SGNK_BANDIT_TRANSFER=0**
fix status	applied, verified in live settings
one environment variable removed the inversion	

So the backwards loop never routed a real task. The routing hook computes its pick, validates it, logs it, and assigns it to nothing. The sweep ran before arming precisely because arming was next.

The cheapest time to catch a backwards loop is before it costs anything.

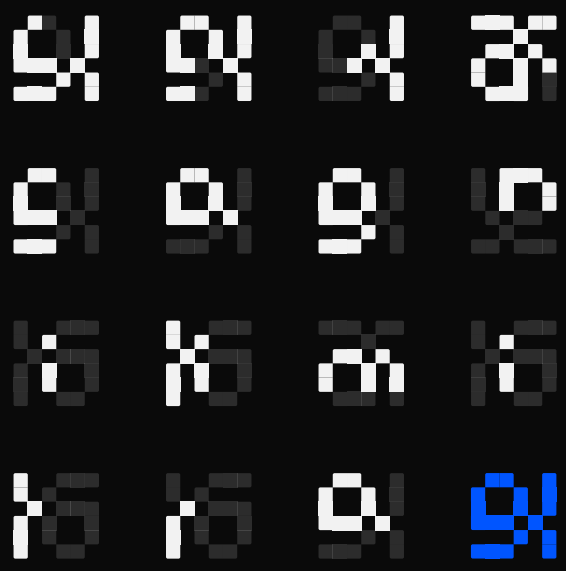
What the sweep buys you

1. Hold everything fixed. Walk one input across its range.
2. Watch the decisions move, and check the direction, not just the liveness.
3. Run it before arming, not after the first bill.

A loop that learns backwards is worse than no loop. No loop leaves your routing exactly as it was. A backwards loop spends real money converting real evidence into the opposite policy, and it will look busy and healthy the whole time, because it genuinely is learning. Just not your lesson.

> **REJECTED.** Arming the loop first and watching the dashboards for weirdness. Dashboards report activity, not direction.

Test the direction of a learning loop while it still costs nothing to be wrong.



A learning loop is not proven by the fact that it learns, but by the direction it learns in.

sgnk.ai/brilliant ↗

Sagnik Mitra

I build AI systems and the machinery that keeps them honest: gates, ledgers, judges, and rails. Product design for years, engineering the whole time. This series is what the building actually taught me.

`sgnk.ai · in/sagnikmitra · github.com/sagnikmitra`