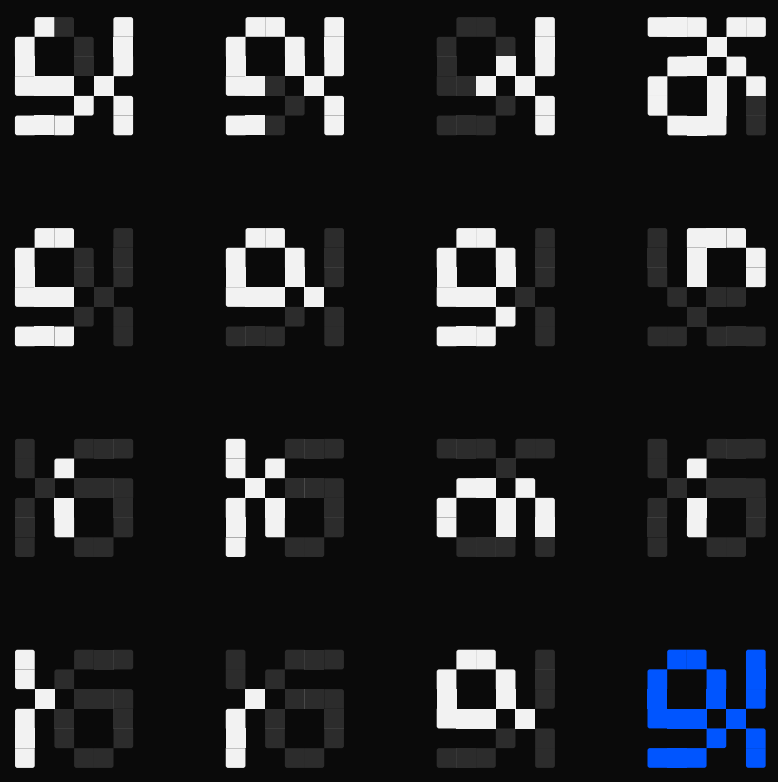


The Brilliant Employee · 10

My learning loop would have rewarded failure

Nothing that learns gets wired to live behaviour here until a sweep has shown which way it moves while unarmed. This loop came back inverted: it promoted the expensive model exactly when that model was failing.

sgnk.ai/brilliant ↗

Sagnik Mitra

Why this exists

A language model writes a large share of my production code. Wrong work reads exactly like right work, and no prompt fixes that. So the system below exists to find out when the answers were wrong.

From one week, none of it visible from inside the tool: a sizing step ignored on 796 of 816 tasks, and two fixes written up as done that were never made.

THE SYSTEM, AS OF 7 AUGUST 2026, 22:55Z

rules written	11 June, 57 days ago
ledger running	27 June, 41 days
tasks in the ledger	3,140, across 3,680 rows
controls on tool calls	**6 registered, 4 that can refuse**

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. Each post takes one of them.

sizing check	sizes a job before a model is picked
model picker	recommends which model gets the job
safety stops	refuse the irreversible, before it runs
grader	grades finished work pass or fail
task log	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, so no row is marked.

> **ASIDE.** Only the safety stops can refuse. The other four observe: one advises, one recommends, one grades after the fact, one records.

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

A low score is meant to buy a better model

The model picker is the part of this workplace that recommends which AI model gets each job. It keeps a running score per model, and it carries one escalation rule.

- a running score, kept per model and per task family
  `the number is called p25`
- the score falls under the bar
  `0.60 in the default band`
- the job moves to the stronger, more expensive model
  `escalation, the remedy for a poor score`

> **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. Before connecting anything I wanted to see which way the maths pointed.

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

A unit test would have gone green on this

1. Hold every input fixed except one.
2. Walk that one input across its full range.
3. Record the decision at each step.

If the decisions move in the direction you intended, the loop is safe to arm.

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.

> **ASIDE.** The fair objection is that a sign error is what a unit test exists to catch. An assertion encodes the value I already expect, and my expectation was the flaw, so it would have gone green.

A sweep encodes nothing. It moves one input end to end and reports which way the outcomes travel.

An assertion encodes what you expect. A sweep encodes nothing and reports the direction.

The worse it did, the more work it got

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.

> **ASIDE.** Escalation is supposed to be the remedy for a poor score. Here the loop stood ready to reward failure with promotion.

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

One shared term inverted the whole rule

routing-bandit-decide.sh:128, verbatim

```
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
```

p25 was meant to measure the cheap model's record. Weighted 0.5, the expensive model's record bled into it, so a strong result for that model argued for staying cheap and a weak one argued for escalating to it.

THE ONLY ESCALATION-ELIGIBLE ARM IN THE SYSTEM	
p25 at TRANSFER=0.5	**0.541, under the 0.60 bar**
p25 at TRANSFER=0	0.690

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

I had audited this loop and got it wrong

AN EARLIER AUDIT	THE SWEEP
said	said
the loop is dead, not dangerous	the decisions move
method	method
a re-read	walk one input

The sweep refutes that as stated. Walk one input and the decisions move. The defect is not deadness. It is direction.

> **ASIDE.** 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.

It never routed a single real task

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.

A backwards loop is worse than no loop

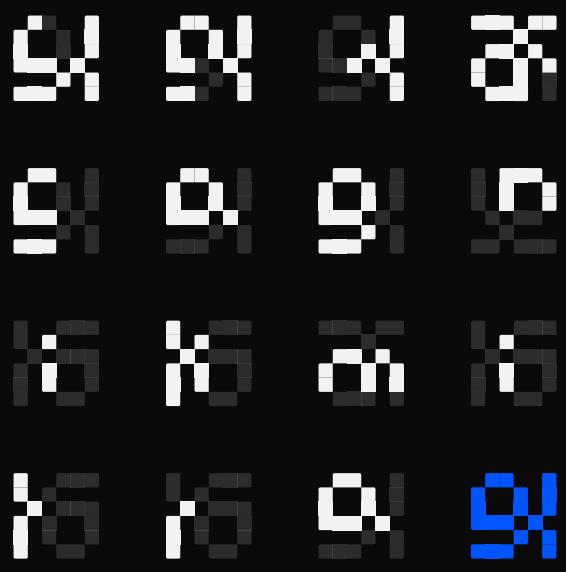
1. Hold everything fixed. Walk one input across its range.
2. Watch the decisions move, and check the direction, not just the liveness.
3. Run it before arming, not after the first bill.

That test costs minutes, and it must run before arming, because afterwards the same defect costs money and teaches your system the opposite of every lesson it observes.

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.



Sweep first, arm second.

Before the wire is connected is the only time being wrong about direction is free.

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`