The Brilliant Employee · 07

The Scorekeeper With an Eraser

On 2026-07-26 my report read 79.3% across the whole scoreboard. Matched to the writer's own seeds it read 86.4%. The reporting layer was subtracting a head start the live system never granted, and the expensive model's points vanished before anyone saw them.

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 keeps score

Inside this workplace, the Gambler is the part of the system that recommends which AI model gets each job. It learns by keeping score per model: every finished task lands as a win or a loss on that model's arm, like a bookmaker adjusting odds.

Two programs touch that scoreboard. The live system writes it, one task at a time. A reporting tool reads it and prints the headline win rates I actually look at. The writer and the reader were built in different sessions, by the same distracted manager. Me.

> **Plain words: ARM.** one model's running score: successes and failures, plus a starting assumption called a prior

Every scoreboard has a writer and a reader. They can disagree.

Two scoreboards

The live system seeds each arm deliberately. The cheap default model starts with a small benefit of the doubt, roughly four assumed wins, because it has a long track record. The expensive model starts from nothing: one nominal win, one nominal loss. Zero head start in its prior. The points above that prior came from offline eval runs carried at half weight, and the state file tags every one of those arms seeded.

The reporting tool assumed one rulebook for everyone. Hardcoded into it: every arm began with the same four-win head start. So before printing real wins, it subtracted four. From every arm. Including the one that never got them.

> **ASIDE.** In Beta-prior terms: the live seeds were (4,1) for the cheap model, (1,1) for the expensive one, (4,2) and (2,1) for domain arms. The reader assumed (4,1) flat.

A hardcoded assumption is a fact nobody voted on.

Three minus four is zero

The expensive model's arm sat at a tally of 3.0. The reporting tool subtracted its imagined head start of 4 and got minus 1. Code does not print negative wins, so it clamped the result to zero.

```
    pa, pb = prior
    a = float(arm.get("a", pa))
    b = float(arm.get("b", pb))
    return max(0.0, a - pa), max(0.0, b - pb)
```

1. arm tally reads 3.0
2. reader subtracts an assumed 4-win head start
3. 3.0 minus 4 = minus 1
4. max(0, minus 1) = 0
5. printed: zero real wins

Every success the expensive model had earned was erased before a human ever saw it. Not one bad night. Every report, silently, for as long as both programs existed.

Clamping to zero turns a math error into a clean-looking lie.

Caught mid-write

2026-07-26. I am writing the audit that is supposed to be the honest one, and the headline is already typed: the pooled scoreboard reads 79.3%, nine units of evidence. House rule: re-derive every headline number at write time. So I re-derived it.

ABOUT TO PUBLISH	RE-DERIVED, PRIORS MATCHED
pooled win rate **79.3%**	pooled win rate **86.4%**
interval **55.9 to 95.2**	interval **69.6 to 97.0**
weighted evidence **9**	weighted evidence **17**

Eight units of evidence had been vanishing from every report. Worth being exact about the unit: every point on these six arms came from offline eval runs carried at half weight, so 17 is weighted evidence, not seventeen live jobs. Note the direction: the bug did not flatter the system. It made the expensive employee look worse than he was.

Re-derive the headline at write time, from the producer's own files.

Where 86.4 actually comes from

The corrected figure reproduces from the audit's own line: 15.0 successes, 2.0 failures, all of it offline eval evidence at half weight, plus the one legitimate prior. (15 + 4) / (17 + 5) = 86.4%. Show the work, or the number is a rumor with decimals.

21%

of the corrected numerator is prior, not evidence

That footnote matters. Even the fixed figure is one-fifth assumption by mass, pooled across six arms that are not really comparable to each other. A number can be correctly computed and still be soft in the middle. The audit says so in the same breath it prints it.

State how much of a number is evidence and how much is assumption.

Better, and still not quotable

The house abstention rule: an arm needs at least 5 real observations before its win rate means anything. MIN_N = 5, and there is no exception for numbers that flatter.

0

6

arms that had cleared the five-observation floor on 2026-07-26

So the correction produced a strange artifact: a number that got better AND stayed unreportable. The audit published it anyway, wearing its caveat in the same sentence: 86.4%, and no single arm is quotable yet. That sentence, the improvement and the abstention riding together, is the whole discipline in miniature.

A correction can raise the number without earning it a headline.

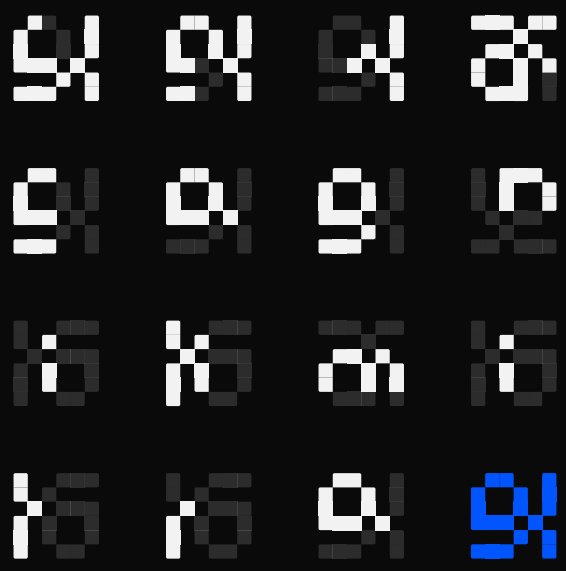
The rule this bought

Two programs exchanging data through a file is the cheapest integration there is, and the most quietly dangerous. The reader must accept what the writer actually writes: same field names, same units, same priors. A missing key is UNKNOWN, never coerced to zero.

And the rule I wrote down: any number bound for a headline gets re-derived by a script at write time, and that script names the file and the key its data came from. The reporting tool in this story still does neither. A hand-carried number has no way to notice it is wrong.

> **REJECTED.** A dashboard that caches yesterday's computed headline. If the number is not re-derived on read, it is a screenshot, not a measurement.

The reader's assumptions must match the writer's. Check, do not trust.



Re-derive every headline number at write time, and make the reader's assumptions match the writer's. The full arithmetic and the audit trail.

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`