The Brilliant Employee · 01

The employee I never interviewed

A language model writes most of my production code. I could not interview it, so I built the checks into the room instead. This is what that machinery is, and what it caught.

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.

You cannot vet this hire

The employee is software: a language model that writes production code in my studio. One afternoon he came close to deleting a client's business, then told me in the same calm voice he uses for everything else.

The part worth sitting with is not the mistake. It is the tone. The same worker who moves a client's whole database before lunch will invent a setting that does not exist after it, and both answers arrive equally sure of themselves.

An interview samples a few answers and trusts them to stand for the whole. Nothing here holds still long enough for that to work.

THE WORKER, IN PLAIN TERMS	
what it is	a language model in a loop
what it can do	read files, run commands, call APIs
what it remembers	nothing, once the session ends
how it fails	**fluently, and at the same speed**

Reliability cannot live in the hire.

A sentence is not a fence

What I got wrong first was assuming that writing a rule down was the same as enforcing it.

Everything you tell a language model is text. It lands in the same space as your question, the conversation so far, and whatever got fetched along the way, and then it is weighed against all of it. A capitalised DO NOT is not a different kind of object. It is more text, and sometimes it loses the argument.

> **Plain words: A CONTROL.** An ordinary program that sits between the worker and the action, runs before the action does, and can refuse. It holds no opinion and has nothing to persuade with, because it is not made of language.

```
~/.sgnk/bin/sgnk-bash-guard.sh    header
# By the system's own LR#48 (a textual prohibition
# is advisory, never the gate)
```

The first clause of the guard's header comment, wrapped to fit. It is there because the lesson was expensive.

A rule the model reads is advice. A program on the boundary is a control.

Six are registered. Three can refuse.

A control is a script registered against a tool, and it runs first. Scripts end with a number: 0 means carry on. This harness reads 2 as a refusal, and on a 2 the call never happens.

```
~/.claude/settings.json    hooks.PreToolUse
{
  "matcher": "Bash",
  "hooks": [{
    "type": "command",
    "command": "bash ~/.sgnk/bin/sgnk-bash-guard.sh",
    "timeout": 5
  }]
}
```

matcher picks the tool. command is the program that gets to say no. The home directory is shortened to a tilde here; the file carries the absolute path.

Six sit on tool calls: shell, file edits, skills, agent spawning, and two on anything reaching outside. Only three are armed. Two can never stop anything by contract, and a third waits behind an unset flag.

Registered is not armed.

One task, and where it can be stopped

I type a sentence.
`on submit`

The job is sized before a model is picked.
`the Gate`

A model is recommended, cheapest first.
`the Gambler`

It asks to run a command.
`the Rails`

catastrophic only, refuse

It asks to read something off the web.
`the Rails`

untrusted source, refuse

The work is graded pass or fail, by a grader I do not fully trust.
`the Judge`

One line is written down, and never edited.
`the Ledger`

28 hooks across 8 moments in a session. The Gate is fail-open on purpose: a triage nudge that can wedge a turn is worse than no triage. The Judge is graded too, and the one calibration of the Gate scored 0.1 against a bar of 0.7.

Two of the seven steps can refuse, and both only for a short list.

Most of it only warns, on purpose

Here is the part I would skip if I were selling you something. My shell guard has two tiers, and the wide one does not block. It logs the hit, names the rule I am about to walk into, and lets me through.

```
~/.sgnk/bin/sgnk-bash-guard.sh    the always-block list
git push .*--force
DROP (TABLE|DATABASE|SCHEMA) | TRUNCATE | dropdb
vercel (rm|remove|rollback)
aws .* (delete|terminate|remove|rm)
```

Four of ten, abbreviated. This list exits 2 whatever mode the guard is in, though three doors sit above it: a bypass variable, a one-shot human approval token, and a recursion guard for nested runs.

Everything wider only warns, and that is the lesson. A gate that blocks everything gets bypassed within a week, and a bypassed gate protects nothing.

> **ASIDE.** 541 warnings between 11 July and 4 August. The guard matches the text of a command, not the intent behind it, so grepping its own deny-list counts as a hit. It flagged one of mine while I wrote this slide.

A gate that blocks everything gets switched off. Then you have no gate.

Ok, continue, and silence do not count

The heaviest of the eight standing rules governs anything irreversible. It does not say be careful. It specifies a ritual, and the ritual is what makes it checkable.

1. Name the operation in plain language.
2. State the blast radius. How many rows, what breaks, when the last backup ran.
3. Say what was checked to confirm it is safe.
4. Ask, and then stop.
5. Wait for an explicit yes.
6. Do it, then verify the state afterwards and report it.

Every operation is its own approval. Yesterday's yes does not carry.

> **ASIDE.** Ok, continue, a thumbs up, and silence are each written into the rule as not-approval. Every one of them had to be listed, because every one of them had been tried.

Only a fresh yes counts.

Every finished task leaves one line

A memory he can edit is not evidence. So the record lives outside him: one line appended per finished task, in a file nothing rewrites. This is a real line from the session that produced this deck.

~/.sgnk/traces/2026-08-04.jsonl

```
"gate_tier":       "floor",
"decided_model":   "sonnet",
"decision_mode":   "advisory",
"decision_real_n": 0,
"model":           "claude-opus-5",
```

Five of thirty fields, reindented to fit. Values unchanged.

Read in order, it indicts itself. The cheap model was recommended. The expensive one ran anyway, because the recommendation is advisory and nothing enforces it. And the count of live observations behind that recommendation was zero, which is what a system looks like before it has learned anything.

One line per finished task since 27 June.
47 of them carry a verdict I gave myself.

The eighth rule is about the rules

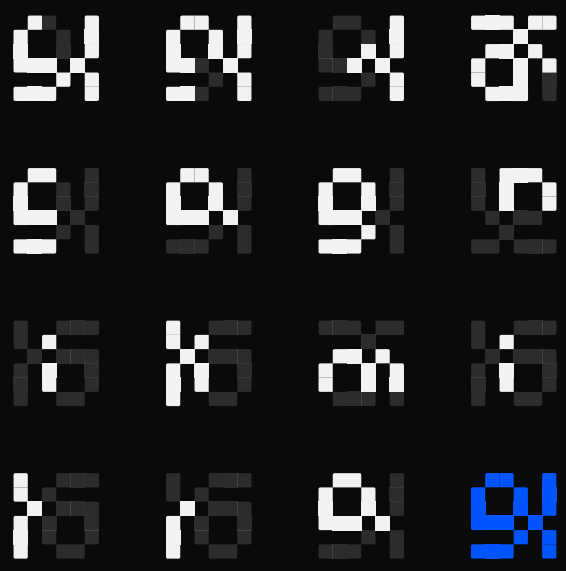
Seven rules govern the work. The eighth governs the rulebook. Every correction, mine or one he catches in himself, becomes a numbered line in a second block that may only ever be added to.

```
~/.claude/CLAUDE.md    Learned Rules
66. Run every harness assertion under BOTH bash
    and zsh before trusting its verdict.
67. A tool that writes must verify the write
    LANDED before reporting success.
68. A test whose subject is a rare fault proves
    nothing until it reproduces the fault.
```

The last three, abbreviated to their first sentence. 67 was written the day a tool printed REGISTERED twice while storing nothing.

The eight rules are still eight. The block they started is at 68, numbered one to sixty-eight with no gaps. He reads it at the start of every session and can only append to it. He forgets everything else between versions. This is the one thing that accumulates.

The original eight have not changed since June. The 68 above them are scar tissue.



I never interviewed him. So I built a room that does the vetting instead, on every task, forever.

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`