Why Smart Models Drift

By Rahul Jindal

One night, a routine email went to the wrong inbox. A little automation I run sends me a status note every evening. For weeks it landed in my personal inbox, where it should. That night it showed up in my work inbox instead. Same note, same words, every fact correct. Just the wrong door.

Nothing broke. No error, no crash, no bad logic. The model that runs it is a good one, and it had sent that note to the right place dozens of times. Ask it straight out which inbox is correct and it tells you, instantly. That night it picked the other one anyway.

This is the kind of failure that bites you now. Not the model being dumb. The model being completely reasonable about a question nobody answered.

The smartest models trip on the dumbest things

Andrej Karpathy has a name for it: jagged intelligence. The same model that cracks a hard proof will also tell you 9.11 is bigger than 9.9, or miscount the r's in “strawberry.” Brilliant and brittle in the same breath. Capability isn't a smooth hill that rises with size. It's a mountain range with cliffs.

My wrong email is the boring, everyday version of the same thing. And once you see why it happens, you start seeing it everywhere.

It wasn't a mistake. It was a blank.

We think of a mistake as someone missing something. Knowledge, skill, attention. You fix it by topping up whatever was short. Teach them, train them, slow them down.

None of that fits here. The model had the knowledge. It had done the job right far more than wrong. It doesn't get tired and it wasn't rushing. Hunt for a flaw in the model and you come back empty.

The flaw wasn't in the model. It was in the instruction.

The rule said: send me the note. It never said which address. And the same rules listed all three of my addresses as fine to use. So when the model reached the point of naming a recipient, it faced an open question with several right-ish answers and no stated favorite. It had to fill the blank. It filled it.

Every run starts from nothing

Here is the part people miss. The model didn't remember sending the note to my personal inbox last night, or the forty nights before. It builds no habit. Each run begins cold, from the instructions in front of it. Nothing carries forward unless it is written down where the next run can read it.

So the streak I had watched for weeks wasn't a settled rule. It was the same likely answer coming up again and again. Give a question one obvious answer and you get that answer most of the time. It looks like a law. It's a lean.

> ***"Forty correct runs in a row weren't a habit. They were forty coin flips that happened to land the same way."***

That changes the whole diagnosis. The model didn't drift in the sense of changing or decaying. Night forty-one was the same model as night one. What looked like fixed behavior was a coin that lands one way most of the time. Flip it long enough and the other face shows. Not if. When.

This isn't a metaphor

The coin is real, and you can measure it. A model has a dial called temperature: it sets how much the model is allowed to gamble on a less likely word. Turn it up for surprising writing, down for safe writing. At zero, the model is told not to gamble at all, just pick its single most likely next word every time, which should make it repeat the same answer word for word. A team at Thinking Machines Lab, the outfit Mira Murati started after leaving OpenAI, ran one prompt through a model a thousand times at temperature zero and got eighty different answers. Not because anyone asked for randomness, but because

the math runs in batches on the chips underneath, and the batch size shifts with whatever else the server is handling that second. Same prompt, same settings, eighty outcomes. The flip is baked into the hardware.

And the model itself doesn't hold still. In 2023, researchers at Stanford and Berkeley tested one version of GPT-4 three months apart on the same questions. On one set of prime-number problems, its accuracy fell from 97.6 percent to 2.4 percent. It hadn't forgotten what a prime is. It had quietly stopped following the step-by-step instruction it used to follow, and every tool built on the old behavior moved with it. Nobody got a memo. The ground just shifted.

So you are building on something that gives slightly different answers to the same question, and changes its mind over months without telling you. That is the raw material. The work is to build something dependable on top of it anyway.

The blank fills itself from whatever's nearby

Picture every task as a form. The instructions fill some boxes. The model fills the rest, reaching for whatever looks most natural in the moment.

Usually that's a gift. It is why these models feel so capable: they fill the unstated boxes the way a sharp colleague would. But natural and correct are not the same thing, and they split at exactly the boxes nobody filled.

My work address looked natural. It's really mine. It was on the approved list. And it showed up elsewhere in the same instructions, for an unrelated reason, inside a search filter. None of that made it the right place to send the note. All of it made the model a touch more likely to grab it. The wrong answer was sitting right there in the context, wearing a name tag.

You might think the cure is to spell everything out. Researchers who study this found the opposite. Stuff a model with enough requirements and its accuracy on each one slides, because now it's juggling twenty rules and dropping one. You can't win by saying less, and you can't win by burying it in more. You win by saying the few things that must not be wrong, plainly, and nailing them down.

A smarter model doesn't save you. It can sink you.

The hopeful guess is that a better model drifts less. Often it's the reverse.

A weak model that only ever saw my personal address has little to grab. One option, low chance of a wrong turn. A strong model knows more. It knows my work address is genuinely mine. It knows the note reaches me either way. It can build a tidy case for any of the three. The smarter the model, the more options look reasonable at the exact spots where the instruction goes quiet. More intelligence, wider field of plausible wrong answers.

> ***"Intelligence kills the errors of not knowing how. It does nothing for the errors of not being told which."***

So intelligence fixes one kind of error and leaves another standing. Competence errors fall: a strong model won't garble the message or invent an address that doesn't exist. Specification errors don't, because the right answer isn't a fact out in the world the model can reason toward. It's a preference, and the only person who holds it is the one who wrote the rule.

There is a nastier twist. A strong model writes smoothly and acts sure of itself, so the rare miss arrives looking exactly like the work that was right. My note was perfect except for where it went. No garbled text, no tell. The better the model, the better its mistakes hide. The errors stop being loud and obvious and turn quiet and plausible, which is far worse for whoever has to catch them.

Telling it to be careful does nothing

The tempting fix: tell the model to slow down. Double-check the address. Be careful.

Useless, and it helps to be exact about why. The model was already careful. It was just careful about the wrong thing: an instruction that never named the right answer. Tell a careful worker to be more careful about a vague order and all you get is more careful guessing. Pile on reminders and you lower the odds of a bad flip. You never reach zero,

because the thing making the variation is still there. The blank is still blank. Care doesn't fill it. It just weights the dice.

You can't make a coin land heads by asking the flipper to focus. The randomness isn't in the flipper's attitude. It's in the setup. If you need heads every time, you stop flipping.

Take the choice away

The fix that actually worked was one line. Write the right address into the rule and forbid the model from choosing. No discretion, no flip, no chance to grab the wrong thing from nearby. The blank is filled for good, by the person who knew the answer.

That's the whole move. Reliability doesn't come from a model deciding well every single time under pressure. It comes from settling the choices that can't change up front, writing them into the system, so the model never has to guess at them. Then let it do the open work that actually needs a brain.

So the real test of an automation isn't how smart the model is. It's how few high-stakes choices you've left for it to guess at. A sharp model with a sloppy instruction is a fast, fluent, confident source of the occasional disaster. A sharp model with the dangerous choices nailed down is something you can trust.

Which choices to pin

Don't pin everything. Pin every word and you've thrown away the reason you reached for a model at all. The skill is telling two kinds of blank apart.

One kind is dangerous: a wrong fill does real damage you can't wave off. Who an email goes to. A price. The account that gets paid. The thing a delete command points at. The one record a write updates. These have a right answer the model can't derive on its own, and a wrong answer that costs you. Pin them. Take the choice away.

The other kind is free: variety is harmless, or it's the point. How a summary is worded. Which example makes the case. The order of an explanation. Here the model's judgment is the whole value. Leave it open and let it run.

Nearly every reliability failure I have seen with capable AI is the same shape. A dangerous choice left open, filled from plausibility, and for a good long run the plausible answer was also the right one. Until it wasn't. The bug isn't in the model. It's in the line someone drew between what the system decides and what the model decides, and a choice that had to be fixed ended up on the wrong side of it.

> ***"The question that matters isn't how smart the model is. It's which decisions you're still letting it make."***

Your org already has this bug

Step back from the machine and you have seen this before, because people and teams fail the same way.

Most teams run on rules nobody wrote down. We always send it to finance first. We never ship on a Friday. We loop in legal on anything customer-facing. It holds because the people who know keep choosing it. Then someone new takes the seat, or the usual person is out, or the situation shifts an inch, and the rule that held for a year quietly breaks. Nobody decided to break it. The flip just came up the other way.

The fix is the same as the one-line fix on my email. For the calls that can't vary, you don't lean on people caring more or remembering better. You take the discretion out. Write it down. Build it into the step. Make the safe path the default and the risky one take a deliberate act. Move the decision off the person and into the system.

The reliable operation isn't the one with unusually careful people. It's the one that stopped depending on them for the things that can't go wrong.

The best model you can buy will still, now and then, do the wrong thing on a job it has done right a hundred times. Not because it got worse. Because somewhere in the instruction there was a blank, and this time it filled it with something plausible and wrong.

A better model won't save you. The next one will be smarter, fill the unsaid blanks even more convincingly, and hide its rare misses even better. The work is the unglamorous

part: find the decisions you can't afford to get wrong, and make them yourself, before the coin does it for you.

The intelligence is the easy half. The reliability is deciding, on purpose, what the model no longer gets to choose.
