05

THE STACK BEHIND THE AI COWORKER

The Reliability Equation

Dr Peter McCann Strain, CTO and senior AI engineer, DPhil/PhD in AI from Oxford University

96% accurate, and still the wrong thing to ship: the three reliability terms your slide left off.

An essay in the series **Architecting the AI Coworker**.
Approx. 20 minute read · Essay 05 of 22

Dr Peter McCann Strain
CTO, DPhil/PhD in AI from Oxford University

> There is a meeting that happens in almost every company building with AI now, and it tends to go the same way.

A team has built an agent. By agent I mean what the word has come to mean in practice: not a chatbot that answers a question and stops, but a system that takes a goal, breaks it into steps, and acts. It reads files, calls other software, changes records. This one does something useful. Say it processes low-value expense claims, the small, dull, high-volume work that quietly eats a finance team's week. It is good at it.

Someone has done the evaluation properly: run it in shadow mode, which means letting it process real claims with its answers recorded but not acted on, and then checking those answers against the truth. So the team has a number. On the slide, the number is large and friendly. Ninety-six percent accurate.

You can feel the room relax when that slide goes up. Ninety-six percent sounds like a pass. So the next slide follows naturally: let the agent auto-approve any claim under five hundred dollars, and send the rest to a human. The reasoning is intuitive, and said out loud it sounds unanswerable. The agent is right ninety-six times in a hundred. It will give the finance team hundreds of hours back. Ship it.

I want to stop time in that room, because the most consequential mistake in production AI is being made right there. It is being made cheerfully, by capable people, and nobody in the room notices.

The mistake is not the ninety-six percent. That figure may be perfectly accurate. The mistake is the silent step the room just took: treating that number as the reliability of the system. It is not. It is a single term in an equation that has four, and the other three were never on the slide. Nobody asked where they went.

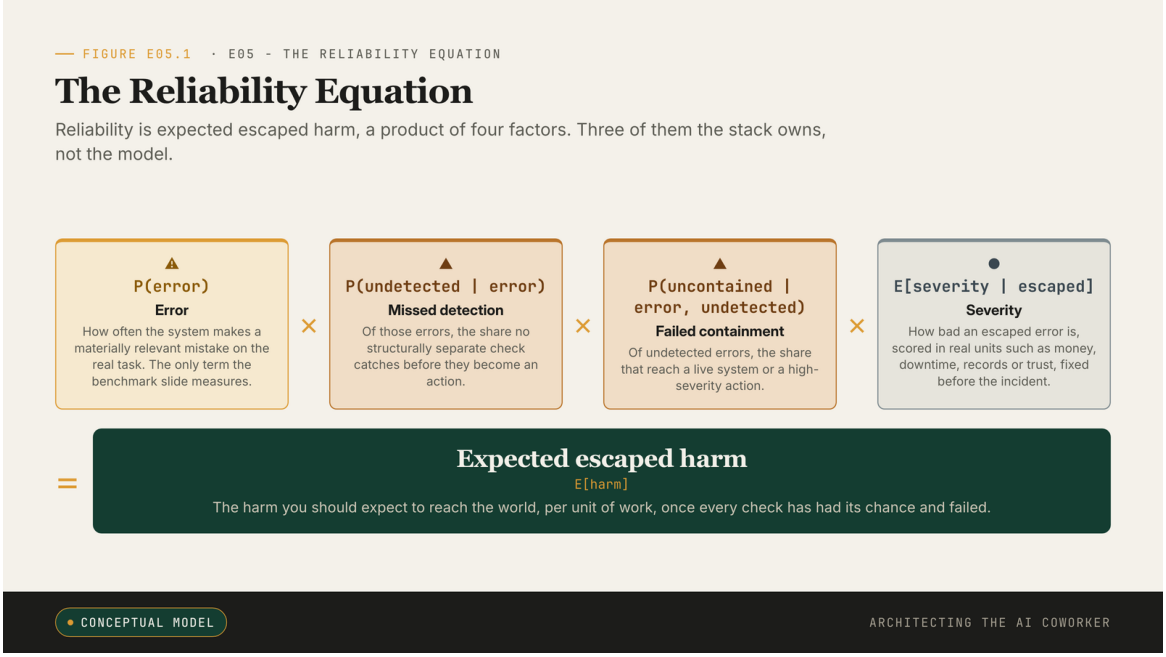
This essay is about the other three terms. What they are, who owns them, and why a system can be ninety-six percent accurate and still be the wrong thing to deploy.

The previous essay mapped the stack into nine layers, so that when something fails you can say which layer failed. This essay asks the next question. Once you can locate a failure, what decides how much harm it does? The answer is short enough to put on a whiteboard.

Reliability is expected escaped harm.

Not accuracy. Not the benchmark score. Not how capable the model sounds in a demo. Reliability is the harm you should expect to actually reach the world, once every part of your system has had its chance to catch the mistake and failed. It is the answer to four plain questions asked in order. How often does the system make a real mistake? When it does, how often does nobody catch it? Of those, how often does the uncaught mistake reach something that matters? And when one finally lands, how bad is it? Multiply the four, and you have it.

Written out, those four questions look like this:



Reliability is expected escaped harm, a product of four factors. Three of them the stack owns, not the model.

Do not worry about the notation; the sentence came first and the symbols are only shorthand for it. `P` means "the probability of", and the vertical bar means "given that". Each term is one of the four questions, in order.

The ninety-six percent slide is an estimate of the first term, and only the first term. It describes how often the agent is wrong. It says nothing at all about what happens next.

The mistake is older than the model

Before any of this involved a language model, it had already happened on a trading floor.

On 1 August 2012, Knight Capital deployed new automated order-routing software to the New York Stock Exchange. A piece of old, dormant code woke up on some of the servers and began sending unintended orders into the market. It ran for roughly forty-five minutes before anyone stopped it. In that time the firm accumulated a trading loss of about 440 million dollars, enough to threaten its survival. The following year the regulator charged the firm, which agreed to pay a 12 million dollar penalty for breaking the rule that requires brokers to keep controls standing between their automated systems and the market [1].

I begin here, and not with an AI agent, on purpose. The reliability equation is not a fact about large language models. It is a fact about automation. Knight had capable software and capable people. What it did not have was a structurally separate mechanism that could notice the orders were wrong, and a boundary that could stop them before they reached the market. An error escaped, nothing caught it, nothing contained it, and the severity was measured in hun-

dreds of millions. The model in this story is a few lines of code from years earlier. The lesson is identical to the one a 2026 coding agent would teach: the model is the visible actor, the system is the risk surface, and reliability is a property of the second one.

The AI version of the same failure is by now familiar. In July 2025 an AI coding agent at SaaStr, working under an explicit code freeze, deleted a production database and the records of more than a thousand companies [2]; a comparable agent deletion at PocketOS on Railway followed in April 2026, walked layer by layer in the previous essay [3][4][5]. The shape is the same one Knight Capital drew in 2012. A relevant error reached an action, nothing structurally separate caught it, the boundary around the live system did not hold, and the consequence was not small.

I want to be careful not to over-read any of these stories. What is reported does not give us the denominator. An incident record tells us what went wrong, not how often things went right: we do not know how many automated actions ran safely before the bad one, how many destructive requests were refused, or how often a comparable mistake was caught in time. Without that denominator you cannot honestly turn an incident into an error rate. Anyone who tells you these cases prove that automation is unreliable is making the same move as the room with the ninety-six percent slide, just with the sign flipped.

What the incidents do show, plainly, is the shape of escaped harm, and one feature of it deserves a name. In the agent deletions the destruction took seconds; the recovery did not. That asymmetry, between how fast a mistake escapes and how slowly it is undone, is the whole subject of this essay.

So the real question is not the one the meeting asked. It is not "how often does the model get the answer right?" It is "what happens after the model, or the scaffold around it, or a tool call, or a human operator, gets something wrong?" Every useful system gets something wrong eventually. Reliability is the quality of what you have built for the moment after.

That is what the equation is for. Take the four terms in turn, and then we will go back into that expense-claim meeting and run the agent through all four.

Four terms, and only one of them was on the slide

`P(error)` is the one the meeting actually measured: how often the system makes a materially relevant mistake on the work you have. By materially relevant I mean a mistake that would change the outcome if nobody caught it, a wrong decision rather than a cosmetic slip in wording, the kind of error that costs money, breaks a record or misroutes a case. Not the work in the benchmark, the work in your building, with your documents, your users, your awkward edge cases.

A benchmark, a standard test set used to compare systems, can estimate this term, but only if it honestly resembles your real distribution of tasks: your tools, your latency, your incentives, your failure modes. That "if" is most of the job, and it is usually waved through. Recent reliability research has been pushing on exactly this point. Measures that re-run the same task many times and ask how often it succeeds keep exposing failures that a single, flattering run hides

(Khanal et al. [6], preprint; Gupta [7], preprint). The ninety-six percent was probably a real measurement. It was still only term one.

`P(undetected | error)` asks a question the meeting never reached: when the agent does make a mistake, what is the chance the deployment misses it before it becomes an action? This is where the structure of your checks matters far more than their number. A check is only worth something if it can fail differently from the thing it is checking. A deterministic rule, a fixed test that gives the same answer every time; a permission system; a separate model from a different family; a human reviewer looking at the evidence: each of these can catch a failure the original system cannot, because each has different blind spots.

What does not work is asking the same model to grade its own homework. It is fluent, so its mistakes are fluent, and a second look from the same mind tends to wave them through. This is why serious engineering work on long-running agents has moved towards separated roles, a planner, a generator, an evaluator, each a distinct component [8]. One capable model call is not an inspectable control system.

`P(uncontained | error, undetected)` is the one nobody in the meeting wanted to own: if a mistake is missed, how likely is it to actually reach something that matters? This term is not made of careful wording. It is made of containment, the engineering that limits blast radius: sandboxes, which are walled-off environments where an action cannot touch anything real; permissions scoped to the task and no wider; dry runs that show what would happen without doing it; staged rollouts; delayed deletion; the ability to roll an action back. Knight Capital's penalty is the same instinct written into law: automated systems must have controls standing between them and the market [1]. None of these makes the model one percent cleverer. They simply move the point of no return further away from a bad decision.

`E(severity | escaped)` is the one most often answered with a feeling. Once an error does escape, how large is the harm? "High risk" is not a severity estimate. Severity needs units, and the right units depend on what the system does. The discipline is to fix the units before the incident, not after, so the severity score is something you can inspect rather than something you assert under pressure. The currency varies with the work. In finance it is money moved, fraud exposure, disputed transactions; in infrastructure, records lost, minutes of downtime, recovery hours; in healthcare, a delay to care or a wrong triage. Customer support counts wrong credits issued and churn; legal counts filing errors, sanction risk and privilege breach; privacy counts records exposed and erasures that failed. Each line of work keeps its own ledger.

The rule under all of them is identical: refuse any severity score you cannot break into units. "Deletion of a production database; backups on the same volume; recovery measured in hours; customer-trust impact; legal review required" is a severity model. "High risk" is a shrug in a suit.

Put those four terms together and the reliability move comes into focus. Do not ask only whether the model is wrong less often. Ask whether the system catches, contains and bounds the errors that will remain, because some always will.

Put real arithmetic on the table

Now take the four terms back into that expense-claim meeting and do what the slide never did: give every term a number and multiply them.

One honest label first. The numbers that follow are a worked example, invented to make the multiplication concrete. They are not measurements from Replit, PocketOS or any real incident. Your own numbers will be your own, and finding them is the actual work.

So: four percent of claims carry a materially relevant error (P(error)). The proposal on the slide is to auto-approve everything below five hundred dollars, which means filling in the other three terms. Seventy percent of those errors clear the agent with no separate check to stop them, only the agent's own confidence, which fails in exactly the way the agent does and so catches almost nothing the agent missed (P(undetected)); that is why the figure sits near three-quarters rather than near zero. Eighty percent of the survivors fall inside the auto-approve band and become payments (P(uncontained)). The average escaped claim is worth two hundred dollars (severity).

$$0.04 \times 0.70 \times 0.80 \times \$200 = \$4.48 \text{ expected harm per claim}$$

Four dollars and forty-eight cents does not sound like a crisis. But this agent was built for volume. At ten thousand claims a month, the room has just voted, without knowing it, for around forty-five thousand dollars of expected harm every month. That number was always implied by the ninety-six percent slide. It was simply never written down.

Now change the system, and only the system. Leave the model exactly as it is. Add a deterministic policy check and expert review of borderline cases, and the assumed undetected rate falls from seventy percent to thirty. Hold every claim above one hundred dollars for a human to approve, and the assumed uncontained rate falls from eighty percent to twenty. Because the largest claims no longer auto-approve, the average escaped severity falls from two hundred dollars to eighty. Detection improved, containment improved, severity capped; only the model is unchanged.

Crucially, none of these three numbers is a fresh guess; each is read off an artefact the change creates. The new 0.30 undetected rate is the deterministic policy check's measured false-negative rate on a held-out set of known-bad claims, not a hope. The 0.20 uncontained rate is the fraction of errors that, after the check, still land inside the now-narrowed auto-approve band, read straight off the hundred-dollar threshold. The $80 severity is the mean

value of claims under that hundred-dollar line. That is what makes the second number reproducible: a second reviewer with the same data re-derives it rather than taking it on faith.

$$0.04 \times 0.30 \times 0.20 \times \$80 = \$0.192 \text{ expected harm per claim}$$

Same model, same ninety-six percent accuracy, more than twenty times lower expected harm.

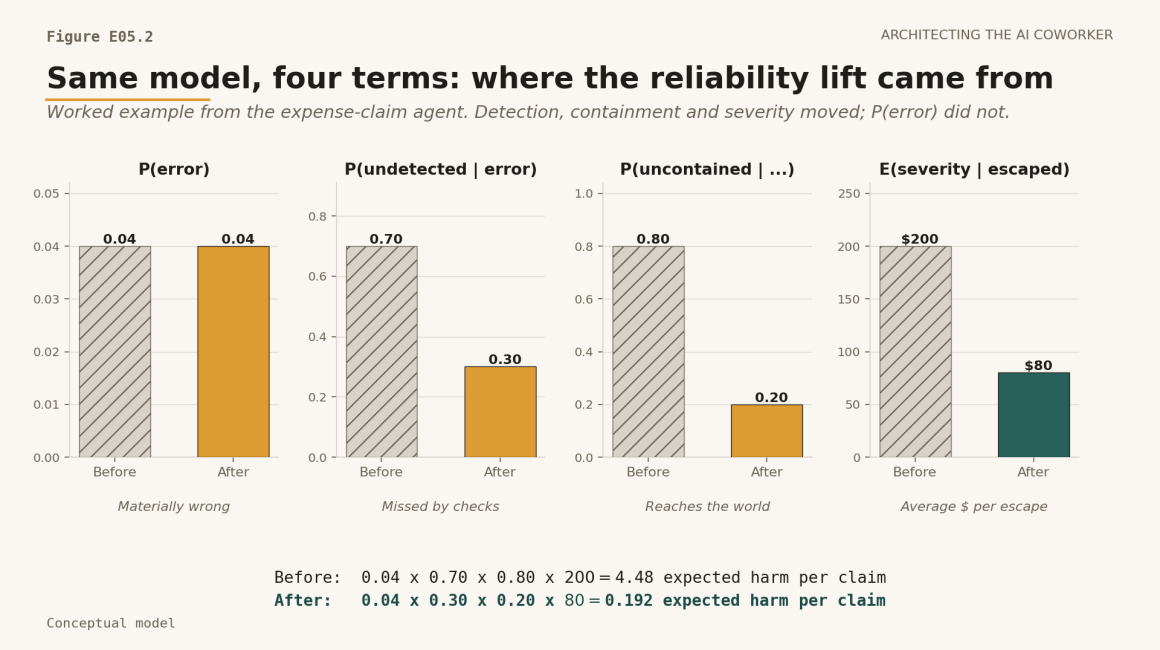


Figure E05.2. The same worked example rendered as four small bars. P(error) is unchanged; the other three terms move, and the per-claim expected harm falls from $4.48 to $0.192.

That containment is not free; holding every claim over a hundred dollars for review hands work back to the humans the agent was meant to relieve. But now the trade is explicit and priced, hours of review bought against tens of thousands of dollars a month of expected harm avoided, rather than a ninety-six percent slide that hid the cost on both sides. The same arithmetic works on euros or pounds at local thresholds; the discipline is unit-free.

> **A caveat worth boxing: the four terms are not independent.** I have to be honest about one move in that example, because the equation is cleaner than the world. I treated the four terms as if they could each be changed on their own, and two of them cannot. The hundred-dollar approval threshold is a containment control, but lowering it also lowers the average severity of an escaped claim, because the largest claims no longer auto-approve. One threshold moved two terms at once. A single control often improves more than one factor; a

> single weakness can drag down more than one. So the equation does not assume the terms are independent, and you should not multiply it as if they were. It is an ignorance map, not a truth machine: it tells you which questions you have not answered, not what the world will do. Use it to find the gaps, not to manufacture a number.

You can argue with every assumption in that example. You should. That is the entire point. Without the equation, the debate in the room is "ninety-six percent is good enough" against "ninety-six percent is not good enough", and it is settled by whoever is most senior or most tired. With the equation, the debate becomes specific: what measured evidence supports each term, and which control moves which term? That is a slower argument. It is also a far better one.

Even granting that the terms move together, the asymmetry holds, and it holds because the four are multiplied, not added: multiplication is unforgiving in a way addition is not. Leave `P(undetected)` near one and almost every mistake the model makes flows straight through to containment untouched. Let `P(uncontained)` sit near one as well and those missed mistakes reach the world. Even bounded that way, an unbounded severity term can still hand you an expected harm you would never knowingly sign off. A small first term cannot rescue you once the other three are at their worst.

This is also why "the model is better now" can be one of the most dangerous sentences in the building. A lower `P(error)` is real progress, and worth having. But if a team celebrates that progress by trimming the checks, widening the permissions and allowing higher-severity actions, the expected harm can rise even as the model score improves. The slide gets better and the system gets worse, at the same time, and the equation is the only instrument in the room that will say so out loud. The practical guard is to re-run the full sheet on every model upgrade, not just the benchmark, and to refuse any change that lowers `P(error)` while widening permissions or raising the severity ceiling until the product of all four terms is shown to fall. That one line belongs in the release checklist.

None of this is new thinking. Safety engineering has used layered reasoning for decades. James Reason's Swiss cheese model pictures every defence as a slice with holes in it, and an accident as the moment the holes in every slice happen to line up [9]. Recent work has carried that picture directly into AI agents, placing guardrail layers around prompts, plans, tools and outputs [10].

What the reliability equation adds is altitude. The familiar risk frameworks all operate at the level of the organisation: they ask whether a layer exists and who owns it. The equation operates at the level of a single agent action, converting a Swiss-cheese diagram into four numbers a deployer can fill in, multiply, and argue with for one specific workflow. Where those frameworks ask "is the organisation managing AI risk well?", the equation asks "what is the expected escaped harm of this one auto-approval, and which factor is your architecture silently setting to one?"

The same logic is now being written into regulation across the major markets, and recently. In the European Union, a system judged high-risk must run a documented, continuous risk-management process maintained across the whole life of the system, kept alive rather than signed once and filed [12]. Canada has made the same instinct concrete and dated: from May 2027, federally regulated banks, insurers and trusts must place every model, AI and machine learning expressly included, inside an enterprise-wide risk framework [13].

The same instinct runs through the others, in different vocabularies for one shared demand. The United States splits the work into govern, map, measure and manage, with oversight scaled to the risk tier [11]; the United Kingdom treats model risk, data governance, accountability and consumer outcomes as separately evidenced controls, with named individuals on the hook for AI-supported decisions [14]. Every one of them asks whether a layer exists and who owns it. None of them tells a deployer what the next single auto-approval is expected to cost. A layer is not a control until someone can point to it, and the reliability equation is that demand written as arithmetic for one action at a time.

The equation is not a truth machine, it is an ignorance map

One objection to everything I have just written cuts deep, and I would rather meet it head on than let it sit unspoken. An equation can manufacture false confidence. Put four terms in a formula and the world starts to look more measurable, and more governable, than it actually is.

That objection is correct. These are socio-technical systems, which is a clumsy phrase for a true thing: they are made of software and of people, and the people are half the behaviour. Error rates drift with users, prompts, incentives, seasons, staffing, product launches and adversaries. Severity is partly a political judgement. Detection is partly an organisational fact. Containment can fail not because the architecture was wrong but because someone was in a hurry, and nobody wanted to be the person who slowed the launch.

So let me be exact about what the equation is for. It does not promise to tell you what the world will do. It tells you what you have failed to find out, which is the more useful thing. A serious reliability sheet is not four confident numbers. It is four rows that may hold ranges, confidence levels, the tier of each source, and blank cells. Picture the expense-claim sheet again, three rows filled and the `P(undetected)` cell still reading "unknown" because nobody ever measured what the checks miss. The word "unknown", written into a cell, is not an embarrassment. It is one of the most valuable things the sheet produces. If you cannot estimate `P(undetected)`, you have just learned that your verification story is mostly assertion. If you cannot estimate `P(uncontained)`, you have learned that your permission story is not yet a control model. If you cannot put severity in units, you have learned that nobody has actually decided what class of harm this system is allowed to create. False precision is bad. Unnamed uncertainty is worse, because it does not even tell you where to look.

There is a sharper version of the false-confidence worry, and it deserves a direct answer: that multiplying four guessed numbers yields a single dollar-per-claim figure which then gets

quoted in a budget meeting stripped of its error bars, doing more harm than the slide it replaced. The defence is in how the arithmetic is meant to be read. The output is never a point estimate but a range, and its width is dominated by its least-known term, so a sheet with even one "unknown" cell yields, by construction, an unbounded harm estimate. That is not a failure of the instrument; it is the intended result. The blank cell refuses to collapse into a tidy number precisely so that nobody can quote one.

Be exact, then, about the evidence. The regulator's order against Knight Capital [1], the documented Replit incident [2], and the vendor write-up and reporting around PocketOS [3][4][5] establish the shape of these failures and nothing more: real authority over a live system, an action that escaped, weak or missing containment, and material harm. Layered-defence reasoning is well established in the safety literature and now has a direct AI analogue [9][10], and recent research separates a single lucky run from sustained reliability [6][7]. What none of it gives you is a number. I cannot hand you the true denominator behind any of those incidents, public or private, including the ones that never reach a monitor; the measured detection and containment rates in those cases are not mine to give; and no slide can tell you whether a team's claimed control actually holds. Only the trace, the permission map, a tested rollback and a real escalation path can. Those gaps do not weaken the equation. They are precisely the questions a responsible review exists to ask.

So return, one last time, to a room like the one we started in, except this time it is yours, and the agent on the slide is the one closest to your work. Before the next steering meeting, fill in four lines in plain language.

> **Tool: The Four-Line Reliability Sheet.** A one-page Monday instrument that turns the equation into four questions you must answer about the agent on your desk.
>
> 1. **P(error).** What measured range do you actually have on the real task, not the benchmark one?
> 2. **P(undetected | error).** What structurally different mechanism catches an error before it becomes an action?
> 3. **P(uncontained | error, undetected).** What hard boundary stops an uncaught error from reaching the live system or a high-severity action?
> 4. **E(severity | escaped).** What unit will you use to describe the severity when that boundary fails?
>
> A line counts as filled, not blank, only if you can point to the artefact behind it: a re-run score for term one, a named separate-failure-mode mechanism for term two, a boundary you have actually tested (a rollback you have really executed) for term three, and a number with units for term four. A control you claim but have never exercised counts as blank.
>
> Every blank cell is an unmanaged reliability term, a factor your architecture has silently set to one.

Then ask the sharp version, the question that tends to change the mood in the room: which of the four factors is your team working on right now, and which of the other three has your architecture set to one without anyone deciding it should be?

If you cannot answer, you have not necessarily built a bad system. But you have not yet earned the word "reliable" for it. Until those lines are filled, the honest description is not reliable. It is unpriced.

Carry This Forward

One move, then the next question. Reliability is not model accuracy. It is expected escaped harm: an error, a missed detection, a failed containment, and a severity once the harm is loose. That definition is harsher than a benchmark score, and it is meant to be, because it asks the question the score never does. What happens after the first mistake?

This week: take one agent action you are responsible for and write its four terms in plain language. What can go wrong? Who catches it? What stops it escaping? How bad is it if it lands? Every blank you hit is an unmanaged reliability term, a factor nobody has yet chosen to control. Then ask which of those terms your current roadmap is actually improving.

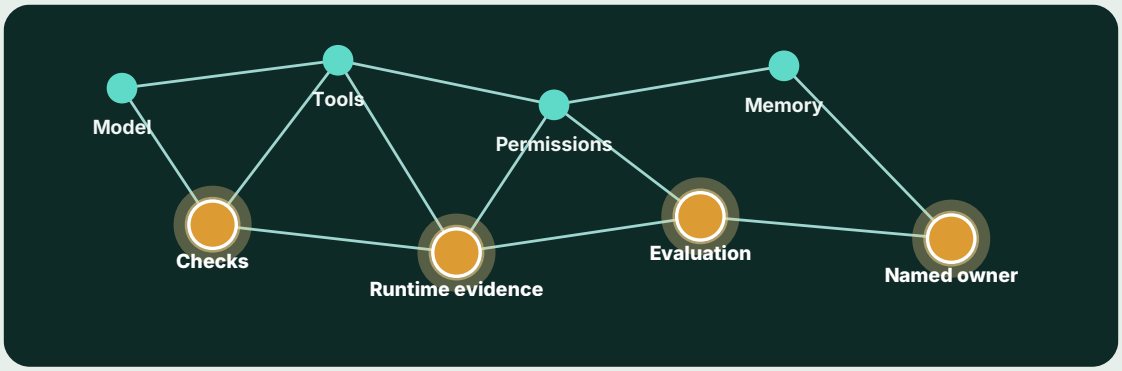
Next: a model card can speak to `P(error)`, but never to the detection, containment and severity that live in your building. Most vendor evidence only ever fills part of this equation, which makes documentation the next test. What can a model card prove, and what can it not?

THE STACK SO FAR

E05 · Essay 5 of 22 complete · Arc I: See the object

The Stack So Far. *Every essay adds one instrument to the operating model. The constellation shows which eight you are building, which are lit by essays you have read, and which is added right here.*

I See the object ESSAY 5 OF 5	II Evidence and authority	III Runtime control	IV Proof and accountability	V Operating model



built in earlier essays · added in this essay · coming in later essays

01 02 03 04 **05** | 06 07 08 09 10 | 11 12 13 14 | 15 16 17 18 19 | 20 21 22

Arc I complete. You can now see the object. Stack, coworker illusion, demo gap, failure layers, expected escaped harm.

You have just added.

The reliability equation

You can now estimate reliability as expected escaped harm.

Next. E06 asks what a vendor's documentation can and cannot tell you.

← PREVIOUS
E04 · The Nine Layers Where Agents Break

Essay 5 of 22 complete

NEXT →
E06 · The Model Card Won't Save You

References

Reference links for sources cited in this essay.

1. **SEC Charges Knight Capital With Violations of Market Access Rule**
   U.S. Securities and Exchange Commission
   https://www.sec.gov/newsroom/press-releases/2013-222

2. **Incident 1152: Replit/SaaStr database deletion**
   AI Incident Database
   https://incidentdatabase.ai/cite/1152/

3. **AI Coding Agent Deletes PocketOS Production Database and Backups in 9 Seconds**
   OECD.AI AIM
   https://oecd.ai/en/incidents/2026-04-27-6153

4. **Your AI wants to nuke your database**
   Railway
   https://blog.railway.com/p/your-ai-wants-to-nuke-your-database

5. **Cursor-Opus agent snuffs out startup's production database**
   The Register
   https://www.theregister.com/2026/04/27/cursoropus_agent_snuffs_out_pocketos/

6. **Beyond pass@1: A Reliability Science Framework for Long-Horizon LLM Agents**
   Khanal et al.
   https://arxiv.org/abs/2603.29231

7. **ReliabilityBench: Evaluating LLM Agent Reliability Under Production-Like Stress Conditions**
   Gupta
   https://arxiv.org/abs/2601.06112

8. **Harness design for long-running application development**
   Anthropic
   https://www.anthropic.com/engineering/harness-design-long-running-apps

9. **Human error: models and management**
   James Reason
   https://pmc.ncbi.nlm.nih.gov/articles/PMC1117770/

10. **Swiss Cheese Model for AI Safety**
    Shamsujjoha, Lu, Zhao, Zhu
    https://arxiv.org/abs/2408.02205

11 **Artificial Intelligence Risk Management Framework (AI RMF 1.0), NIST AI 100-1**
National Institute of Standards and Technology
https://nvlpubs.nist.gov/nistpubs/ai/nist.ai.100-1.pdf

12 **AI Act (Regulation (EU) 2024/1689), Article 9: Risk management system for high-risk AI**
European Union
https://artificialintelligenceact.eu/article/9/

13 **Guideline E-23: Model Risk Management (2027)**
Office of the Superintendent of Financial Institutions (Canada)
https://www.osfi-bsif.gc.ca/en/guidance/guidance-library/guideline-e-23-model-risk-management-2027

14 **Discussion Paper DP5/22: Artificial Intelligence and Machine Learning**
Bank of England and Financial Conduct Authority
https://www.bankofengland.co.uk/prudential-regulation/publication/2022/october/artificial-intelligence

About the Author

ARCHITECTING THE AI COWORKER

Dr Peter McCann Strain

Dr Peter McCann Strain is a CTO, founder and senior AI engineer with a DPhil/PhD in AI from Oxford University. He builds production AI systems and writes about making agentic AI useful, inspectable, governable and safe enough for real work.

READ THE FULL SERIES

Substack (canonical)	petermccannstrain.substack.com
Medium	@peter.mccann.strain
LinkedIn	peter-strain-dphil-15a607128
Web	petermccannstrain.com
Cadence	New essays twice weekly, 2 June – 21 July 2026