20

THE STACK BEHIND THE AI COWORKER

You Cannot Benchmark a Coworker

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

A green benchmark is not a hiring decision: four questions a system must pass before it touches a customer.

An essay in the series **Architecting the AI Coworker**.
Approx. 27 minute read · Essay 20 of 22

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

> A support lead I was sitting with last quarter had two slides open on her laptop, and both of them were green. The first slide said the new AI "coworker" could answer customer questions. The second said the evaluation had passed. Her product team wanted to move quickly: let the system send routine replies on its own, instead of queuing every draft for a human to read first. Two greens, one decision, and a meeting booked for Friday to approve it.

I asked her one question. Green on what, exactly? The room went quiet, and that quiet is the subject of this essay.

Here is the quiet, put into words. You can benchmark a model. You cannot benchmark a coworker. A benchmark scores a clean run of a narrow task. A coworker is something you trust with a class of work, around real customers, over many days, in the cases nobody scripted. The second green slide on that laptop was a benchmark result dressed up as a hiring decision, and the gap between those two things is wide enough to lose a quarter in.

The series so far has replaced the illusion of a single clever thing with a stack: model, tools, retrieval, memory, permissions and oversight. It has shown that reliability is expected escaped harm rather than accuracy, and argued that compliance is evidence a system produces as it runs. Evaluation is where all of that gets tested before a system is allowed near a real customer. It is not the leaderboard. It is the promotion board: the moment a system stands in front of a panel and asks to be trusted with more authority than it had yesterday. Most evaluation programs, I have found, are interviewing the wrong candidate for the wrong job.

The word "coworker" is useful shorthand when you are talking to users. It is the wrong object to put in front of a benchmark. What you are actually evaluating is a system that can classify an incoming message, retrieve the relevant policy, draft a reply, offer store credit below some threshold, escalate the risky cases, and write a running record of what it did and why. The honest question is not whether that system can perform on one clean run in a demo. It is whether repeated runs, under the way real customers actually behave, with tools and carried state and escalation and an audit trail, are safe, correct, contained, and attributable. That is a question about a colleague, and no leaderboard was ever built to answer it.

Put plainly: evaluation asks whether the system has earned the next action before it takes it. The launch question is no longer "can it answer?" It is "what should it be allowed to do?"

The four questions a promotion board has to answer

The rest of this essay rests on a single structure, so let me set it down once and hold it.

> **The Four-Question Promotion-Board Spine**

> A launch decision has to answer these four, in order. They are not the same question asked four ways.
>
> 1. **Capability.** Can the agent do the task on a clean run with a well-formed request and tools that behave?
> 2. **Reliability.** Across many repeated, realistic trials of the same family of task, how often does the system work, and how wide is the spread?
> 3. **Autonomy fitness.** At what class of action is it safe to operate, granted action by action rather than once for the whole agent?
> 4. **Governance readiness.** Does every run leave the evidence an auditor, a regulator or an incident review would need?
>
> Capability is the question public benchmarks were built to answer. The other three are where the launch decision actually lives, and a leaderboard is silent on all of them.

Take them one at a time, because each carries its own discipline.

Capability asks whether the agent can do the task with a well-formed request and tools that behave. A benchmark, a standard task set used to compare systems, estimates this and nothing else. It tells you the agent can answer a refund question. It does not tell you what happens the moment a real request arrives malformed.

Reliability is a distribution, not a single number. The question is: across many repeated trials of the same family of task, with realistic variation in what the user says and does, how often does the system work, and how wide is the spread? The cleanest anchor I know is tau-bench, a benchmark that places a tool-using agent inside a moving conversation with a simulated customer in retail and airline settings, rather than against a frozen prompt [1]. tau-bench introduced a measure called pass^k: out of k repeated trials of the same task, the proportion in which the agent succeeds *every single time*. This is not the same as pass@k, which asks only whether at least one of k attempts works. The difference matters because your customer does not get to pick the lucky run. Anthropic's guidance for evaluating agents draws exactly this line, and treats the all-pass version as the customer-facing reliability question [2]. I use pass^8 as a working default through this essay, and I should be honest that the 8 is mine, not a standard. It is a round number large enough to expose variance a single run hides and small enough to run affordably against a real policy set. The right k for you depends on how much variance you can tolerate and how much each trial costs; the principle, not the integer, is what carries.

> **Choosing k without pretending k is magic.** Pick a k of your own by treating repeated trials as a tool, not a ritual, and applying them unevenly:
>
> - **Stratify.** Group tasks by action class (classify, draft, send, refund, close) and by severity, so a wrong order-status reply and a wrong refund over the threshold are not averaged together.

- **Repeat where variance matters.** A deterministic schema check does not need eight runs; a model-mediated escalation decision does.
- **Hide a holdout.** Keep tasks the build team has not seen, so the suite cannot be quietly tuned until it passes.
- **Report a range, not a decimal.** A confidence interval (or an honest range) beats a single point estimate from a handful of trials.
- **Inspect failed traces.** The integer tells you how often; the trace tells you why.
- **Refresh when the world moves.** Policy, tools, model, or user distribution changes invalidate any k chosen against last quarter's world.

> The number you publish is only ever as current as the world you drew it from. Treat the six points above as a tick-and-record checklist, refreshed every time the world they were measured against does.

Autonomy fitness asks at what class of action the system is safe to operate. The answer can differ for "classify a message", "draft a reply", "send a reply", "offer store credit", "issue a refund" and "close an account", even though the same model is behind all of them. One green light does not cover the lot. A system can be entirely fit to draft and entirely unfit to send. And the rung an action sits on is not granted once. It is held on condition: if live traces show drift, missing fields, failed retrieval or a rising escalation rate, that action class drops a rung until it is re-evaluated. A promotion board that cannot demote is not a promotion board; it is a launch ceremony.

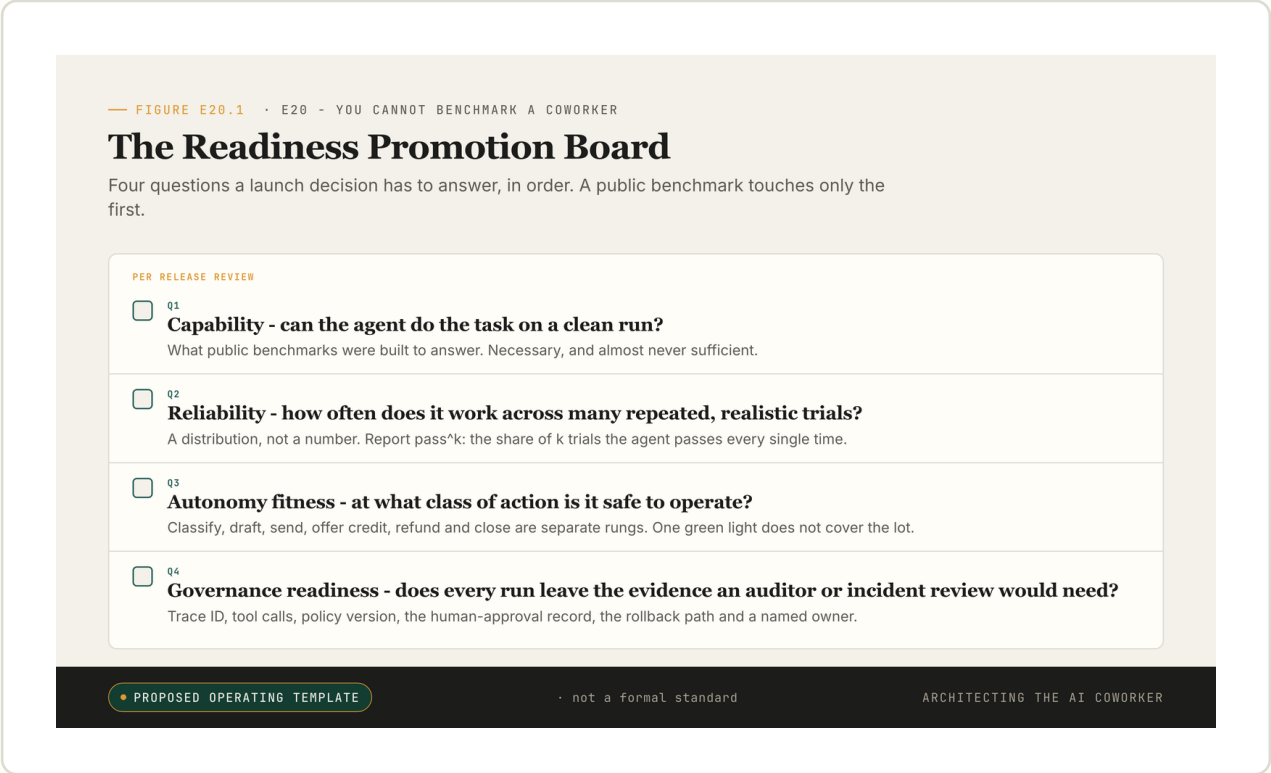
Governance readiness asks whether the system emits the evidence a regulator, an auditor or your own incident review would need: a trace ID, the tool calls it made, the policy version it read, the record of any human approval, the rollback path, and a named owner. The unit here is the trace, the running record of one whole run; a span is one timed unit inside it, a single tool call or model call. Shared vocabularies are emerging to describe them. The OpenTelemetry conventions for generative AI, and the OpenInference specification, give teams a common way to describe model calls, tool calls and retrievals, and to join them into spans and traces. The field names and status values can still change as the standards mature [345].

This four-question spine is not the only way to grade a machine's authority, and it helps to say what it is not. There is an older literature on graded human-automation control: Sheridan and Verplank's 1978 ten-level scale of automation, and the SAE J3016 six-level taxonomy for driving automation, both rank "how much of the decision the machine makes" on a single dimension. The spine here is doing something different. It asks four orthogonal questions about a single action class, can it, how reliably, at what authority, with what evidence, rather than one question about a whole system. The graded view is not wrong; it answers a later question, and the next essay's autonomy ladder returns to it deliberately.

Capability tells you whether the agent can answer a refund question. The other three decide whether a useful system has earned more authority. That is the gap between a green slide and a launch, and it is the gap the support lead's Friday meeting was about to fall into.

The promotion board only works, though, if its evaluators do, and a handful of design failures recur across evaluator types. Naming them is what stops you reaching for the wrong one.

- **Brittle oracle**: the test's correct answer is over-specified, so a model that solved the right problem differently fails.
- **Rubric vagueness**: the test asks for a quality that has no shared definition; verdicts vary by who scored.
- **Shared scaffold**: the evaluator uses the same harness as the model under test, so both fail in correlated ways.
- **Sample skew**: the test set does not represent the distribution the agent will actually meet in production.



The four questions a launch decision has to answer. Capability is the only one a public benchmark touches.

Where a capable agent still fails

Make this concrete with the support lead's own queue, and I will keep it for the rest of the essay. Her agent handles messages of a few recurring shapes: "Where is my order?", "Can I return this?", "Your policy is unfair", and "I need a refund." The business wants the agent to send the low-risk replies on its own and route the rest to a human.

Put the four questions to that agent, one row at a time, and watch where a capable agent still fails. The point of the walk is not to redefine the questions; it is to see which rows a capable system leaves blank.

Capability is the easy row. The agent can plainly answer a refund question. That row passes.

Reliability is harder. The review asks for an evaluation set built from real and synthetic support tasks, run many times each, with an explicit pass^k target for any customer-facing action. A single accuracy number, however large, leaves the row blank.

Autonomy fitness is where most demos come apart. Classify, retrieve, draft, send, offer credit, issue refund and close ticket have to be pulled apart into distinct actions, each with its own risk tier, with a named list of which actions may run autonomously and a feature flag that demotes a send back to human review. A demo that says "human in the loop" while nobody in the room can name which actions actually require approval has not earned the row.

Governance readiness is the row that quietly fails on storage. Store only the final answer and the transcript, and there is no way to reconstruct what the agent did. The row needs every trial to record the retrieved policy, the tool calls, the final action, the model and policy version, the latency and the trace ID, in a structure compatible with the standards above, and it needs customer-facing messages attributable to the support agent and a named operational owner, rather than sent under a borrowed human identity.

That is the spine: legible first, then bounded, then recoverable. What each action is and how reliably it works makes the system legible. Which rung each action sits at, and who it is attributable to, makes it bounded. A correction path for a wrong reply, a reversal path for a wrong refund, and a launch decision resting on something better than a stale benchmark make it recoverable. If a part of that has no real answer, that part is the work. The review is meant to be uncomfortable. A capable support agent can fail it, and that is the point: capability tells you the agent can answer the refund question, and the review tells you whether it should be allowed to send the answer, move the money, or close the ticket.

If I were buying a support system rather than building one, there is a single procurement question I would ask above all others. What is your pass^8 on a representative set of support tasks drawn from our actual policies, and can we inspect the failed traces? If the vendor can only give you pass@1 on a public benchmark, you have learned something real. You have not learned whether the system is reliable enough to face your customers.

The honest vendor has a fair rebuttal here, and it is worth meeting squarely: no one can hand you pass^8 on your policies pre-sale, because they have not seen your policies, your tools, or your escalation rules. That is true, and it is the reason the procurement question should not stay a gotcha. Convert it into an acceptance test. Demand the vendor's pass^8 on their own published reference suite as the baseline, then write the real bar into the contract: an agreed pass^8 on a task set you draw, run during a paid pilot, with failed-trace inspection, as a launch gate the vendor has to clear before the system touches a live customer. A number a vendor can legitimately dodge becomes a number a contract can enforce.

That contract bar is not only good practice. In a growing number of places it is becoming the law, and the same four-question spine maps onto the rules cleanly. Take one obligation and work it through. Europe's high-risk AI rules turn the reliability evaluation into a logging duty: the system must record its own events automatically across its working life, and a pass^8 number with no event log behind it has not met the obligation it claims to measure, because the same rules set a separate accuracy, robustness and cybersecurity bar that the log is what proves [17]. The pass rate is not the obligation; the logged trace behind it is.

The medical-device regulator in the United States shows the same mechanism governing change rather than a single launch. A sponsor can pre-commit a description of the modifications it expects to make, a protocol for making them, and an assessment of their impact, which turns iterative evaluation from a one-time regulatory event into a governed lifecycle: each later change is judged against representative tasks, repeated trials and a recorded decision about the next action, not re-litigated from scratch [16].

Other regimes reach for the same pairing of reliability and logged evidence, each turning it into a duty you can be held to:

- The US standards body's risk-management framework makes measurement a named function of governing AI [15].
- The UK regulator's draft guidance on automated decisions and profiling is open for consultation through 29 May 2026, with final guidance expected in summer 2026 [18].
- Canada's federal rules require an impact assessment before an automated decision system is used [19].
- Canada's financial regulator will, from 1 May 2027, require every federally regulated financial institution to manage model risk across each model's whole life, whether built in-house or bought in, AI and machine-learning systems explicitly included [20].

Four jurisdictions, one shape: prove reliability, and keep the evidence that proves it. But a logged pass rate is only as honest as the trace beneath it, which is the next thing the review has to learn to read.

Read the trajectory, not the answer

The most important habit this review teaches is to stop treating the clean final answer as the interesting part of a run. It is the least interesting part.

Here is the failure the support lead's queue throws up most often, and it is the everyday case the review is built to expose. (It is not a claim about a real public incident; it is the ordinary shape of the work.) A customer asks for a refund on an item bought 43 days ago. The policy allows an automatic refund only within 30 days, unless the item was damaged on arrival. The agent retrieves the general returns policy but misses the exception table for late returns. It drafts a confident, warm apology and tells the customer the refund is available. It then calls the `refund.quote` tool and prepares a refund over the value threshold that should require sign-off. No escalation occurs, because the model reports it is "confident." The trace shows no human

approval screen. And when the same task is run eight times across small variations in policy wording, the same failure appears in three of the eight runs: five of eight pass, three fail, so pass^8, the all-pass measure, is not met on that action class.

Now watch the same task clear the gates after the fix, so the threshold is not just a thing the agent fails. Add the late-return exception table to what retrieval can reach, and place a hard gate on `refund.quote` above the value threshold. Re-run, and the agent retrieves the late-return exception, declines the automatic refund, and routes the over-threshold case to the approval screen instead of preparing the payout. The trace now records policy version v4.2, the escalation event, and the named owner who holds the approval. Across the same eight variations, the agent now passes all eight: the action class clears pass^8, where before it managed only five of eight. That all-pass trajectory, set beside the five-of-eight one, is what "earned the next action" looks like in the trace: a second reader can open both runs and see exactly which step moved.

For that "second reader" to be more than a hope, the verdict has to be a rule rather than a judgement call, so two reviewers scoring the same traces land on the same decision. Write the threshold down as a table and apply it action by action.

Reliability gate to autonomy rung (apply per action class).

PASSING RUNS ON HOLDOUT (K=8)	UNGATED OVER-THRESHOLD TOOL CALLS	VERDICT
8/8 (pass^8 met)	zero	Promote to send-class autonomy
8/8 (pass^8 met)	one or more	Hold at draft; fix the gate, re-run
7/8 or below (pass^8 fails)	any	Hold at draft; inspect failed traces

A send-class action (send a reply, issue a refund, close a ticket) is promoted to run autonomously only when it passes all eight trials on the stratified holdout, pass^8 met, AND the traces show zero ungated tool calls above the action's value threshold. A single ungated over-threshold call holds the action at draft even when all eight pass. Anything short of a clean eight-of-eight holds the action at draft regardless of the gate, with mandatory failed-trace inspection before re-evaluation.

The 43-day refund case reads cleanly against this rule. Before the fix it passed only five of eight, with an ungated `refund.quote` above the threshold: a hold at draft on two independent counts. After the fix it passes all eight (pass^8 met) with the over-threshold call gated to the approval screen: a promote. Two

> readers scoring those same two trace sets reach the same verdict because the rule, not their judgement, decides it.

That five-of-eight-to-all-eight move is therefore not a vibe a reviewer reports; it is a row that crosses a written line.

Notice what the final reply looks like at the bottom of that run: a warm, well-written apology offering a refund. A reviewer reading only the answer would tick it as helpful. The error is upstream, in a missed exception table and a tool call nobody gated, and it is only visible if you read the trajectory rather than the output. Read it that way and every one of the four questions has something to say: reliability catches the three-in-eight variance, autonomy fitness catches the ungated over-threshold tool call and the missing approval screen, governance readiness catches a trace that cannot reconstruct what happened. This failure is not exotic. It is ordinary. That is exactly why evaluation has to include repeated trials, checks on tool calls and state, evidence that escalation happened, and a deliberate review of failed traces.

To catch failures of that shape you need more than one kind of evaluator, because each kind reads a different sort of evidence and each has a characteristic blind spot. A deterministic check reads machine state: it can confirm a schema is valid, a policy lookup returned, a refund amount sits within bounds, the final environment state matches what was expected. Its weakness is the brittle oracle: the test quietly encodes the wrong success condition, and OpenAI's 2026 decision to retire SWE-bench Verified is the live warning that even a carefully curated test set can become flawed or contaminated over time [7].

Where an exact check is too narrow, a model judge reads meaning instead, scoring tone, policy fit and helpfulness. Its failure mode is rubric vagueness: it rewards plausible support prose rather than the specific policy obligation, which is why concrete rubrics, order randomisation and human calibration samples matter [8][2]. *Disclosure: a co-author of HealthBench's rubric development is a member of the author's family.* The trace reviewer, sometimes itself an agent, reads the path and catches a wrong retrieval or an unnecessary tool call. Its danger is the shared scaffold, judging with the same blind spots as the agent under test, so it should reason over structured traces rather than vibes [3][4][5]. The human is the last evaluator, and the only one who reads the cases none of the others anticipated. Sample skew is the human's weakness, the pull towards famous failures or easy random cases, which is why samples should be stratified by risk, length, tool use, low confidence and reviewer disagreement [8][2].

The human evaluator carries one more weakness that no amount of training removes, and it is worth dwelling on because it is the one most launch plans wave away. The "but our reviewers are trained" defence is weaker than it looks. Parasuraman and Manzey's 2010 synthesis of the automation-complacency literature documents that both naive and expert operators over-rely on automation as attentional load rises, and that simple practice does not eliminate the effect [21]. A 2025 randomised clinical trial by Qazi and colleagues then shows the same pattern alive in the LLM era: physicians who had completed twenty hours of formal AI-literacy training saw

their diagnostic accuracy drop from 84.9% to 73.3% when exposed to flawed ChatGPT-4o suggestions, a fourteen-point adjusted decrement that training did not eliminate [22]. Approval-fatigue is therefore not a willpower problem you can train away. It is an evaluator-design problem, and the only durable fix is to keep the queue short enough, the rubric specific enough, and the routing structured enough that a tired reviewer cannot wave through a fluent mistake.

I should be honest about those four labels. "Brittle oracle", "rubric vagueness", "shared scaffold" and "sample skew" are diagnostic names for evaluator-design problems, not results lifted from a single paper. The source-backed claims underneath them are narrower and firmer: benchmark tests can become flawed or contaminated [7]; vendor guidance recommends task-specific evaluations, traces and human calibration [2]; and trace standards make step-level review more practical [3][4].

One design rule holds all of this together. The evaluator should be structurally different from the generator wherever the claim matters. A different model family helps. A different scaffold helps more. A different type of evidence helps most. If your generator is fluent and your judge is fluent in the same way, the judge will quietly approve the generator's fluent mistakes. For the support agent, that rule means the final answer must not be the only object you evaluate. The environment state, the trace, the retrieved policy, the tool call, the escalation decision and the human approval screen each need their own check.

Why is the clean answer so misleading?

There is one more reason the clean answer misleads, and it is the gap between a static prompt and a real customer. The static prompt is too polite. Customers are not.

They change their mind mid-conversation. They paste angry text from a forum. They ask for a refund and then mention the item was used. They give an order number with one digit wrong. They ask a policy question in the middle of a complaint. They say "never mind" after the tool call has already been prepared. None of that shows up in a one-shot prompt, and all of it shows up in a real support queue.

This is why interactional evaluation, testing the agent inside a moving conversation rather than against a frozen prompt, matters. tau-bench evaluates agents inside exactly those dynamic interactions. Its original experiments reported that state-of-the-art function-calling agents, agents that decide for themselves when to invoke a tool such as a refund or a lookup rather than waiting to be told, succeeded on fewer than half the tasks, and that the all-pass measure in the retail setting was lower still, roughly one trial in four [1]. I am deliberately not quoting those figures to the decimal, because tau-bench is version-specific and the numbers move with every model release; the lesson is what stays fixed. Repeated interaction exposes a variance that clean prompts hide.

Two further benchmarks make the same warning at workplace scale. TheAgentCompany evaluates agents on consequential office tasks, and its best agent completed only a minority of them fully autonomously, somewhere around a quarter to a third depending on the model and

the version of the study [9][10]. METR's time-horizon work measures the task length at which models succeed half the time, and finds that length has been doubling roughly every four to five months on recent models, while cautioning on its own live page that measurements above sixteen hours are noisy and should be handled with care [11][12]. I am giving both as ranges on purpose: a precise figure quoted and then disclaimed in the same breath is worse than an honest range. Neither result tells you your support agent will improve on any schedule. Together they tell you something you can act on: your evaluation suite cannot stay frozen while the workload and the models move underneath it.

So evaluation has to be interactional, repeated, and alive. There is one objection that does not collapse under scrutiny, and it is sharper than the cost complaint. It is not that bespoke evaluation is merely expensive. It is that a private evaluation suite is graded by the same organisation that wants the launch to succeed, with no external benchmark to check it against, so it can be quietly tuned until it passes. A public leaderboard, for all its faults, is at least adversarial. It is run by someone with no stake in your Friday meeting.

That objection is real, and it is the reason a promotion board needs more than one independent voice on the panel. The answer is not to abandon the private suite for the public one. It is to make the private suite resist its own grader.

Four defences against a coached test

There are four defences, and the point is that none of them works alone. So test each against the same hostile question: could a motivated grader still coach past it? The short version is the box below; the paragraphs after it walk each line.

> **To trust an in-house eval, require:**
>
> 1. Rotate hidden tasks (refreshed quarterly, drawn by a function the build team cannot inspect)
> 2. Route flagged traces to external review (structurally independent of the launch decision)
> 3. Publish failure modes, not just pass rates (in a fixed schema an outsider can query)
> 4. Tie evaluator changes to a separate sign-off (owned by a role with standing to halt a release)

A **rotated hidden-task pool** fails the moment the build team can see this quarter's rotation, so the rotation must be drawn by a function they cannot inspect: a separate evaluation team, an external auditor, or a deterministic schedule signed before the model's release. Keep a sample of last quarter's hidden tasks hidden too, so regression can still be checked.

Routing flagged traces to external review is theatre if the reviewer is paid by the same launch decision; it becomes real only when the reviewer is structurally independent. And "independent" is concrete: an internal red team reporting to a different VP, a customer-side acceptance test, a regulator-facing assurance vendor of the kind beginning to advertise under Europe's third-party conformity-assessment regime, or a sectoral evaluator built for clinical or financial use.

Publishing failure modes rather than pass rates is theatre if the failures are curated for press release; it becomes real when the publication carries a fixed schema an outsider can query without negotiation. The schema is short and unforgiving: action class, observed defect, frequency band, mitigation status, last-tested date, reported the way the US standards body expects measurement results to be reported [15].

And **tying evaluator changes to a separate sign-off** is theatre if the signer also approves launches; it becomes real when the evaluator-change ledger is owned by a governance role with standing to halt a release, and every change to the rubric, the task pool or the scoring code is timestamped against the model build it scored.

Taken together, the four do not make a private suite adversarial in the way a public benchmark is. They make it harder to tune quietly until it passes, and that is the falsifiable bar.

The SWE-bench Verified retirement is the public worked example of what happens when even one of those four is missing. OpenAI built a curated, hidden-where-possible test set, ran it for several quarters, and still concluded in February 2026 that contamination, scaffolding effects and test defects had moved the benchmark out of the band where it could measure frontier capability [7]. The retirement is itself the published-failure-mode artefact in action: an evaluator-change announcement, signed off, dated, tied to the model builds it had scored. A private suite without that capacity to retire itself in public is one quarter from being quietly tuned to pass.

There is a quieter way for a whole panel of evaluators to fail at once, and the four defences have to be read at fleet scale to catch it. Call it correlated failure. A high pass^k can still collapse if the cases that fail are not statistically independent but share an upstream cause, so that one bad day takes a whole fleet of agents down together. The pattern is well documented in the agent-tooling supply chain: a 2024 arbitrary-code-execution flaw in LangChain's symbolic-maths chain, traced to a maths library that evaluated untrusted input as live code, showed that a single library defect can compromise every agent that imports it, regardless of how reliable each agent looked in isolation [23]. Shared-context attacks have the same shape: a poisoned retrieval source, a shared system prompt, or a single misconfigured tool will fail in lockstep across every agent reading from it.

The defence is not a higher k. It is the absence of shared single points of failure: rotate retrieval sources, separate tool credentials, vary the system prompt across deployments, and treat any common dependency as a correlated-risk surface the promotion board must see, not just the per-agent score. So the board needs a shared-dependency line on its own checklist, listing every retrieval source, tool credential and base prompt common to more than one deployed action class, and scoring that overlap in its own right. A per-action clean eight-of-eight across the whole fleet means nothing if all eight share the one library that a single bad call to `sympify` can detonate.

A public benchmark still has a place inside all of that. It is a scouting report. It can tell you a model family has improved at web tasks, coding tasks or workplace simulations [6][13][14]. It cannot tell you whether the agent will follow your refund policy, avoid disclosing the wrong customer's

order status, escalate a legal threat, preserve the approval screen, or reverse a mistaken store credit. OpenAI's 2026 post explaining why SWE-bench Verified no longer measures frontier coding capability is valuable precisely here: it shows that even a benchmark built to repair earlier flaws can reach the end of its useful life as frontier performance, contamination risk and test defects change [7]. The right posture is not "ignore benchmarks." It is "never let a benchmark do a promotion board's job, and never let the promotion board grade itself unwatched."

A few things this essay cannot tell you, because the launch decision is yours and not mine. I do not know your agent's real pass^8, because that number only exists once the task set is drawn from your customers, your policies, your tools and your escalation rules. I do not know how much of any public benchmark score comes from genuine capability rather than scaffold, contamination or evaluator design, without the benchmark's version and method. I do not know the right human-intervention rate for your support queue without seeing the cost, the harm, the volume and the quality of your reviewers. I do not even know whether today's trace-standard field names will still be the field names next year. Those gaps are not a weakness in the method. They are the work the method exists to point you at.

So take one support-like workflow your organisation wants to automate. Pick the single action you would most like to make autonomous, and ask what the current pass^8 is for that action on tasks drawn from your real policies. If the honest answer is "we have never measured it that way," that is your starting line.

Which returns us to the support lead and her two green slides, and to the Friday meeting. The second slide was never wrong, exactly. It was just answering a smaller question than the one the meeting had been booked to decide. A benchmark can tell you a model is capable. It cannot convene a promotion board, because a promotion board needs repeated reliability, a legible trace, a working escalation path, a recovery path, and an independent panel that the candidate cannot quietly coach. Once she had that, the Friday meeting could finally decide the thing the two green slides could not: not whether the system answers, but how much rope to give it. The rope is measured in rungs, and the ladder it climbs is where the next essay begins.

Carry This Forward

One move, then the next question.

Evaluation is a promotion board, not a leaderboard. The candidate is not a model in the abstract. It is a harnessed system asking a panel for permission to do a particular class of work under real policies, users, tools, traces, escalation paths and recovery limits, and the panel has to include a voice the candidate cannot coach.

The move: choose one action class and make the question concrete. Should this system classify, draft, send, refund, close, delete, file, or escalate? Put it in front of the four questions, can it do the task, how reliably across repeated trials, at what class of action is it safe, and does every run leave usable evidence, then decide what would make you raise, hold, or reduce its authority in the next release.

What it seeds: the answer becomes a rung. Permission is not granted to the whole agent at once; it is granted action by action. So the question the next essay takes up is the one a passed board hands you: now that this system has been promoted, exactly how far up the ladder does it go?

Three regulatory lenses US · EU · UK

Operating questions, not legal advice. The frameworks stay the same; the regulator changes.

US Can your evaluation produce evidence that procurement and NIST AI RMF can read as readiness for the action class?

EU Can your evaluation produce evidence sufficient for AI Act risk classification, logging and oversight expectations?

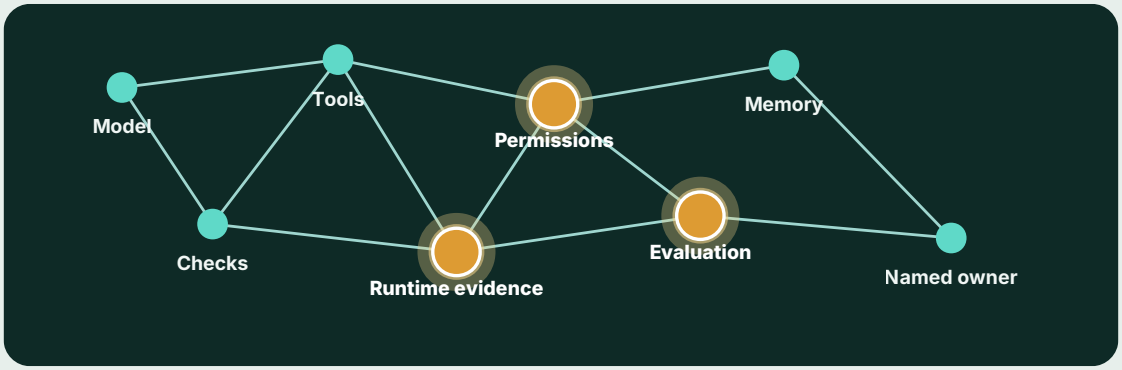
UK Can your evaluation produce evidence the ICO, sector regulators and the DSIT cyber code can read as readiness for the action class?

THE STACK SO FAR

E20 · Essay 20 of 22 complete · Arc V: Operating model

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	II Evidence and authority	III Runtime control	IV Proof and accountability	V Operating model ESSAY 1 OF 3



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

You have just added.

The Readiness Promotion Board

You can now evaluate an agent through a promotion board, not a leaderboard.

Next. E21 asks which autonomy rung each action class has earned.

← PREVIOUS
E19 · Compliance Is Not a PDF

Essay 20 of 22 complete

NEXT →
E21 · The Autonomy Ladder

References

Reference links for sources cited in this essay.

1. **tau-bench**
Yao et al.
https://arxiv.org/abs/2406.12045

2. **Demystifying evals for AI agents**
Anthropic
https://www.anthropic.com/engineering/demystifying-evals-for-ai-agents

3. **OpenTelemetry GenAI spans**
OpenTelemetry
https://opentelemetry.io/docs/specs/semconv/gen-ai/gen-ai-spans/

4. **OpenTelemetry GenAI semantic conventions**
OpenTelemetry
https://opentelemetry.io/docs/specs/semconv/gen-ai/

5. **OpenInference specification**
Arize AI / OpenInference
https://arize-ai.github.io/openinference/spec/

6. **2026 AI Index Report**
Stanford HAI
https://hai.stanford.edu/ai-index/2026-ai-index-report

7. **Why SWE-bench Verified no longer measures frontier coding capabilities**
OpenAI
https://openai.com/index/why-we-no-longer-evaluate-swe-bench-verified/

8. **OpenAI HealthBench**
OpenAI
https://openai.com/index/healthbench/

9. **TheAgentCompany: Benchmarking LLM Agents on Consequential Real World Tasks**
Xu et al.
https://arxiv.org/abs/2412.14161

10. **TheAgentCompany OpenReview entry**
OpenReview / Xu et al.
https://openreview.net/forum?id=LZnKNApvhG

11 **Time Horizon 1.1**
METR
https://metr.org/blog/2026-1-29-time-horizon-1-1/

12 **Task-Completion Time Horizons of Frontier AI Models**
METR
https://metr.org/time-horizons/

13 **WebArena: A Realistic Web Environment for Building Autonomous Agents**
Zhou et al.
https://arxiv.org/abs/2307.13854

14 **OSWorld: Benchmarking Multimodal Agents for Open-Ended Tasks**
Xie et al.
https://arxiv.org/abs/2404.07972

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

16 **Marketing Submission Recommendations for a Predetermined Change Control Plan for Artificial Intelligence-Enabled Device Software Functions**
U.S. Food and Drug Administration
https://www.fda.gov/regulatory-information/search-fda-guidance-documents/marketing-submis-sion-recommendations-predetermined-change-control-plan-artificial-intelligence

17 **EU AI Act, Regulation 2024/1689 (Articles 12 and 15)**
European Union
https://eur-lex.europa.eu/eli/reg/2024/1689/oj/eng

18 **ICO consultation on draft guidance about automated decision-making, including profiling**
Information Commissioner's Office (UK)
https://ico.org.uk/about-the-ico/ico-and-stakeholder-consultations/2026/03/ico-consultation-on-the-draft-guidance-about-automated-decision-making-including-profiling/

19 **Directive on Automated Decision-Making**
Treasury Board of Canada Secretariat
https://www.tbs-sct.canada.ca/pol/doc-eng.aspx?id=32592

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

21 **Complacency and bias in human use of automation: An attentional integration**
Parasuraman, R., & Manzey, D. H.
https://doi.org/10.1177/0018720810376055

22 **Automation bias in large language model-assisted diagnostic reasoning among physicians trained in AI literacy: A randomized clinical trial**
Qazi, I. A. et al.
https://doi.org/10.1056/AIoa2501001

23 **CVE-2024-46946: arbitrary code execution in LangChain LLMSymbolicMathChain via sympy.sympify (CWE-20, CVSS 9.8)**
National Vulnerability Database (NIST)
https://nvd.nist.gov/vuln/detail/CVE-2024-46946

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