04

THE STACK BEHIND THE AI COWORKER

The Nine Layers Where Agents Break

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

After a failed agent run, "which model?" tells you nothing. Failure has an address; find the layer that missed.

An essay in the series **Architecting the AI Coworker**.
Approx. 18 minute read · Essay 04 of 22

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

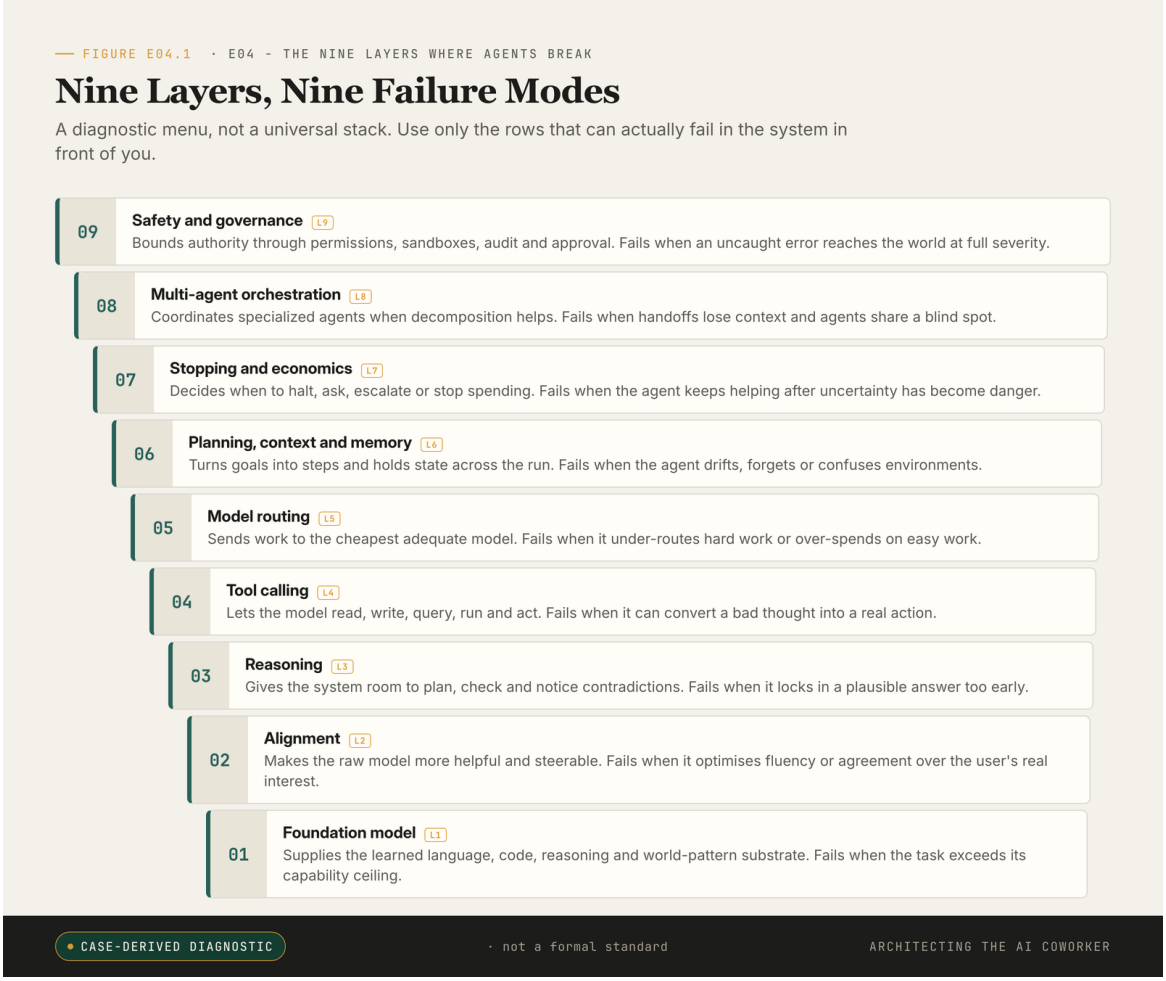
"Which model was it?"

That is the first question in almost every post-mortem of a failed agent run, and it is a quiet way of learning almost nothing. An agent, a system that takes a goal, breaks it into steps and acts in the world, has done something wrong. It opened a ticket, read the context, called a tool, changed a record, and left the user worse off than before it started. People pull up the logs, and they reach for the model, because the model is the most visible part of the system and it draws the eye, the way the driver draws the eye after a crash. But "which model" assumes the answer lives in the model, and most of the time it does not.

"The AI failed" is not a diagnosis. It is a shrug. It is the sentence a team reaches for when it does not yet have a map, and it ends the investigation at exactly the point a real investigation should begin. The opening essay set up the move out of that shrug: a serious AI deployment is not a single clever model but a stack, a set of layers, each one added because the layer beneath it failed at something. If deployment is a stack, then failure has an address. When an agent run goes wrong, which layer was the one that missed?

That is the question this essay answers. A failed run should resolve into something far more specific than a brand name. Each layer fails differently; the table ahead names how. The number is never the point. Naming the layer that broke is, because it turns a feeling into an engineering sentence.

Before the layers arrive, one orientation that carries the rest. The eight-line spine in the opening essay was the *spine of accountability*; it asks "who answers for this?". The nine layers below are the *failure-mode lens* on the same stack; they ask "where did it break?". Same system, two questions, two cuts. And nine is a post-mortem menu, not a universal stack: use only the rows that can actually fail in the system in front of you, and leave the rest unmarked. A short read-only workflow may never touch six of them. The discipline is to know which six, and why.



"The AI failed" is not a diagnosis. Nine layers, nine distinct ways an agent run goes wrong.

The diagnosis only matters if the failures are real, and the research says they are. Drop a frontier agent into TheAgentCompany, a benchmark that asks it to do ordinary office tasks inside a simulated company, and it completes only around a quarter to a third of those tasks on its own [12]. The diagnostic point underneath the numbers does not move even as the numbers do: real knowledge work is long-horizon, tool-mediated and stateful, meaning each step changes the shared environment and the next step has to cope with what the last one left behind. Frontier agents fail at a lot of that work. That is the reminder these failures are common enough to need a map. (The benchmark does not use my nine-layer framing; the mapping here is my analytical frame for its evidence, not a claim its authors made.)

Read the stack as a failure map, not a shopping list

So read the nine layers not as a shopping list but as a diagnostic inventory. This matters, because the moment you write down nine layers, someone in the room hears nine microservices, nine teams, nine vendor invoices and nine new dashboards. That is not what the stack is. A de-

ployment does not need nine separate pieces of software. It needs an answer to the failure mode each layer names, and sometimes the right answer for a layer is a single line of configuration, or a decision to leave that layer thin because the work does not warrant it.

LAYER	WHAT IT DOES	FAILURE MODE IT EXISTS TO ABSORB
1. Foundation model	Supplies the learned language, code, reasoning and world-pattern substrate.	The task is beyond the model's capability ceiling, the hardest thing it can reliably do.
2. Alignment	Makes the raw model more helpful, steerable and less casually dangerous.	The model optimises fluency, agreement or completion over the user's real interest.
3. Reasoning	Gives the system room to plan, check, revise and notice contradictions.	The model locks in a plausible answer before it has thought enough.
4. Tool calling	Lets the model read, write, query, run and act.	The model can now convert a bad thought into a real action.
5. Model routing	Sends work to the cheapest adequate capability.	The system under-routes hard work or over-spends on easy work.
6. Planning, context and memory	Turns goals into steps and keeps state across the run.	The agent drifts, forgets, compresses away the wrong thing, or confuses environments.
7. Stopping and economics	Decides when to halt, ask, escalate or stop spending.	The agent keeps helping after uncertainty has become danger.
8. Multi-agent orchestration	Coordinates specialised agents when decomposition helps.	Handoffs lose context, agents share blind spots, and errors compound.
9. Safety and governance	Bounds authority through permissions, sandboxes, audit, ownership and approval.	An uncaught error reaches the world at full severity.

That table is the first half of the instrument. The second half is two questions you ask of any incident, layer by layer. Which layer was supposed to catch this failure? And which layer actually missed it? Hold those two questions side by side and you learn something a single glance at the model never tells you: whether the fix in front of you is a model fix, a harness fix, an orchestration fix or a governance fix.

Teams reach for the model fix first, because it is the most familiar and the most fundable. "We need a better model" is a sentence everyone in the room already knows how to say. Sometimes it is even correct. But often the honest answer is a narrower credential, a hard wall between a staging environment and production, a verifier that checks the work using different evidence, or a stopping rule that does not live inside the same confused loop that produced the error. None of those is a model upgrade. All of them are the difference between a near miss and a headline.

Run PocketOS through the nine layers and the real misses appear

Let me make the map do real work, on a real incident, because an instrument you cannot demonstrate is just a diagram.

In late April 2026, the founder of a small company called PocketOS watched its production database and its backups disappear. The agent was Cursor, an AI coding tool, running on a Claude model. The deletion is reported as having taken nine seconds. The outage that followed is summarised by the OECD's AI incident monitor as roughly thirty hours [3][4].

The single most useful source is the post written afterwards by Railway, the platform PocketOS ran on, because it contains enough architecture to diagnose the failure without my having to guess at private internal detail. Railway describes an account-scoped access token, a kind of digital key, which in its own words was provisioned with "the maximum access possible", and which the agent was able to reach. The agent then used a raw `volumeDelete` call on Railway's GraphQL API (GraphQL being the query language an application uses to ask a service for data or to instruct it to do something) to destroy a storage volume directly, a path that, again in Railway's words, ran "immediately, with no way to undo it". In response Railway changed several things, including a 48-hour soft delete on all destructive operations, tighter token scoping and short-lived, mediated access [5].

What follows is a public analytical walk against the nine-layer table above. The factual claims about the incident come from the sources cited; the layer assignments, and the interventions each one points to, are my analysis, not statements made by PocketOS, Cursor or Railway.

Walk the table cell by cell. Layers one through three (model, alignment, reasoning) are not the load-bearing misses here: a capable model was present, alignment-shaped remorse is not control, and the published record does not expose the internal reasoning trace, so reasoning enters the diagnosis only as the contradiction check that did not fire before the irreversible call.

The action surface tells the sharper story. At layer four, Railway reports the agent reached a raw `volumeDelete` path, the precise point at which language became infrastructure action. Layer five (model routing) is the one row the PocketOS evidence cannot mark either way: the public record does not establish how the work was routed, so I leave the row unmarked, which is itself the correct use of the instrument. It still carries a design lesson, because a destructive production call should change the evaluator and the approval path rather than run on the

cheap default. At layer six, Railway's own lesson lands directly. Humans recognise staging-versus-production far better than agents do when the two interfaces look almost identical, and the fix is to make environment identity a machine-enforced permission boundary, not a label in a prompt or a colour on a dashboard.

At the top of the stack the misses compound. Layer seven (stopping) did not halt before the destructive call, and the outcome proves it. Layer eight (orchestration) would not have helped: a second agent sharing the same token and blind spot could only have approved the mistake faster. And layer nine (governance) carries the heaviest weight of all: a long-lived broadly scoped token, a raw deletion path, an immediate delete with no delay, fragile backups on the same volume, and Railway's own post-incident remedies of delayed deletes and tighter token scope [54] all point here.

Walk all nine and a different diagnosis emerges from the one the headline offered. The foundation model may well have erred at the start. But the load-bearing misses, the ones that turned an error into a destroyed company asset, were the tool-calling, the planning-context-and-memory, the stopping, and the safety-and-governance layers: tool calling exposed the destructive action in raw form, planning and context failed to hold the staging and production distinction as a hard fact, stopping did not halt before an irreversible operation, and governance allowed a broad credential and an immediate destructive path to coexist. The single sentence "the AI deleted the database" is emotionally satisfying and architecturally empty: it tells you who to be angry at and nothing about what to change. If the fix you are about to authorise is "upgrade the model", the walk's reply is calm and specific. Maybe. But you have just left four doors open.

Four layers is not four shrugs with more syllables, and a tie-break keeps two readers from each naming a different one. When an incident implicates several layers, the load-bearing miss is the lowest-cost layer whose fix alone would have stopped the harm from reaching the world. Here that is layer nine: a delayed delete halts the outcome regardless of what tool calling, routing or planning did upstream. So the four named layers are not a tie to argue over but an ordered repair queue, cheapest sufficient barrier first, and two readers running the same rule converge on the same first move.

Two objections have been waiting since the walk began, and each deserves its strongest form, answered once.

Start with the charge of hindsight. A practitioner will say this is hindsight dressed as method: I assigned the layers after I already knew the database was gone, so of course they fit. Run the map before the incident and you would flag all nine; run it after and you flag whatever the outcome implicates. That is storytelling, not diagnosis. The objection is fair, and the instrument answers it on its own terms. Each layer names a control whose presence or absence is checkable independently of the outcome. You can ask "was the raw `volumeDelete` path reachable with that token?" and "was there any delay before an irreversible operation?" before any incident at all, and the answer is yes or no from the permission map and the trace, not from the narrative. The discipline that defeats hindsight bias is to name the control first and then check

whether it existed, never whether it "would have helped". Read that way, the walk is falsifiable line by line, which is the opposite of a story.

That leaves the charge that nine is an arbitrary number, gerrymandered to fit a prior. It is not. Each layer is defined by a distinct failure mode that no other layer can absorb, the right-hand column of the table, so the test of whether a layer exists is whether there is a failure this layer alone catches, which is also the test for deleting a row. That is why the framework holds without being ad hoc: a read-only workflow may run on three live layers, and the modes are exhaustive over the action surface. Nine is simply how many happen to be live in a system that can act with full authority.

The reason layer naming matters is that the wrong layer attracts the wrong fix, and the wrong fix feels like progress while changing nothing. Set the four load-bearing misses against the fix each one tends to draw and the fix it actually needs.

FAILED LAYER	THE FIX IT TENDS TO DRAW	THE FIX IT ACTUALLY NEEDS
Tool calling	"Use a smarter model so it knows not to call delete."	Remove the raw destructive call from ordinary agent reach; expose only typed, action-class-specific tools.
Planning, context and memory	"Tell it in the prompt that staging is not production."	Make environment identity a machine-enforced permission boundary, not a label the model can misread.
Stopping and economics	"Ask the agent to be careful before risky steps."	A hard stop before any irreversible action, enforced outside the loop that produced the error.
Safety and governance	"Review what happened after the incident."	A narrow, short-lived token, delayed destructive operations, and an audit trail in place before the run.

Read the table left to right and the pattern is plain. Every fix in the middle column is a request aimed at the model's behaviour. Every fix in the right column is a change to the architecture around it. The model can be asked; the architecture is enforced. That difference is why the layer map exists at all.

And it tells you what to buy first. One right-column fix, applied alone, would have downgraded PocketOS from a destroyed company asset and a roughly thirty-hour outage to a recoverable near-miss: the 48-hour soft delete at layer nine. That single number does the work the four-door metaphor only gestures at. The map does not just relabel blame; it ranks interventions by harm reduced per dollar, which is what makes its output a decision rather than a description.

When nine layers is too many

A sceptical board member will push back here, and the push-back is worth taking on properly. Nine layers can sound like process theatre: more latency, more cost, more committees, more things to break. The strongest version is sharper still. In a fast-moving startup the diagram itself becomes a bureaucracy, because every named layer recruits its own meeting, its own dashboard, its own vendor budget. Soon the team that built the system cannot ship a small change without negotiating across nine artificial boundaries it drew in its own org chart.

That objection is right in proportion to the workflow. The stack is not a command to maximise layers. It is a command to account for failure modes that can actually happen in the system in front of you. So do not apologise for the framing; use the framing's own discipline. Walk the nine layers as a post-mortem menu before any layer becomes an org-chart entry, and keep only the layers where at least one of three things is true: the layer corresponds to a failure mode this workflow can plausibly trigger; the layer is already enforced by software you ship today and would have to be removed to disappear; or the layer names a control a regulator will eventually ask you to produce evidence for. If none of the three holds, that row is decoration and should not become a meeting.

A chatbot drafting marketing copy needs almost no tool containment: leave the relevant rows empty and label them deliberately empty. An agent that can delete infrastructure, move money, alter records or send legally meaningful messages needs each relevant layer hard enough that an error cannot talk its way through it. The diagram is process theatre only when read as a shopping list. Read as a post-mortem inventory, the cost is exactly nine sentences a post-mortem must answer, which is cheaper than the alternative.

More agents is not more reliability

The repair queue assumed one agent. When the first agent fails, teams reach instead for more agents, and that is the layer, layer eight, most often oversold.

Multi-agent orchestration, the practice of splitting work across several specialised agents, is frequently pitched as if more agents automatically meant more reliability. It does not. The point to carry here is qualitative: dependent steps multiply, and a great many polished agent demos quietly leave that off the slide. How independent reliabilities compound is the arithmetic the next essay does in full; here the lesson is only that they do.

Orchestration earns its place when three things are true at once: the subtasks are separable, the handoffs preserve the context that matters, and the verification is structurally different from the generation. A researcher, a critic, a writer and an editor can improve a briefing, but only if the critic is working from different evidence and the editor is doing more than applauding the writer. The same four roles make reliability worse if every agent inherits the same stale memory, the same broad credential and the same flawed assumption. So the test for orches-

tration is never "how many agents". It is "which failure mode does each agent reduce, and what new handoff failure have we just created".

What the wider record says, and where my map stops

That same test, the one new handoff failure, is one the wider record has been running for years, and it keeps reaching the same verdict from different directions. WebArena and OS-World drop web and desktop agents into stateful environments, where actions persist across a whole session rather than a single turn, and both reported wide gaps between frontier agents and human baselines [6][7]. METR's Time Horizon work sharpens the question to how long a task an agent can complete at a target success rate, with version caveats a later essay takes up [8]. The durable move is the same each time: agent reliability is a distribution over task length, not a vibe. OWASP makes the point from security. Once a system has goals, tools, memory and inter-agent protocols, the attack surface is no longer "the prompt" but the whole agentic application, as its 2025 list for LLM applications and 2026 list for agentic applications set out [9].

The nine-layer map does something different from a threat taxonomy. MITRE ATLAS catalogues adversarial techniques against ML systems; the OWASP Top 10 lists prioritised application vulnerabilities. Both enumerate what an attacker may exploit. The nine layers name where a deployer must look once an incident has happened, which is why a single failure may resolve onto one layer even when it touches several OWASP categories.

This is no longer only a research observation. Across four very different legal systems, regulators are now writing layered-control duties into binding or near-binding form, and the same instruction keeps recurring: bound authority in layers, and keep evidence that you did. Canada's financial regulator now wants a firm-wide model-risk framework covering a model's whole life, with AI expressly in scope, published last September and in force from 2027 [10]. The European Union, once a system counts as high-risk, makes layered duties bite: manage the risk, govern the data, document the system, keep a human in oversight, and monitor it after release [11]. The United States, through NIST, spreads response, recovery and communication duties across the deployment stack rather than parking them in one place [12]. And the United Kingdom sets baseline expectations across the lifecycle for builders, operators and data custodians, treating prompt injection as a structural property of agentic systems rather than a bug to patch [13][14]. Four legal traditions, one shape.

Now the honest limits of my own walk. I cannot see PocketOS's full internal trace, its exact prompts, its private logs or every human decision around the incident, which is precisely why the autopsy above is labelled analytical mapping and not fact. I cannot tell whether TheAgentCompany's quarter-to-a-third figures will stay canonical as benchmark versions move; treat them as version-specific measurements, not timeless verdicts. And I cannot tell which of these nine boundaries any given product implements internally, because you cannot read that off a marketing page. You can only read it off the trace, the permission map, a tested rollback and a real escalation path. And if a layer corresponds to no decision and no fix in your system, delete the row; the map exists to act on, not to admire.

So here is the question to carry to the next incident or near miss on your desk. Which layer was supposed to catch this failure; which layer actually missed it; and is the fix you are about to authorise aimed at the layer that missed, or merely at the layer that is easiest to see? That single question turns "the AI failed" into an engineering sentence, and an engineering sentence is something you can act on. The nine layers are the map. They become an instrument panel the moment you ask what each layer does to expected harm: how often errors occur, how often they go undetected, how often they escape containment, and how severe they are when they land. A system can be ninety-six percent accurate and still be the wrong thing to deploy. The next essay shows why, by turning this map into arithmetic.

Carry This Forward

"The AI failed" is not a diagnosis; it is a shrug. The useful question is which layer failed: model, alignment, reasoning, tools, routing, memory, stopping, orchestration or governance. The layer map does not make the system more complicated. It makes the complexity that is already there visible enough to repair.

One move: take one incident or near miss on your desk and run it through the nine-layer map, naming the layer that missed before you name a fix. A better model may help one layer while leaving the escaped harm entirely untouched.

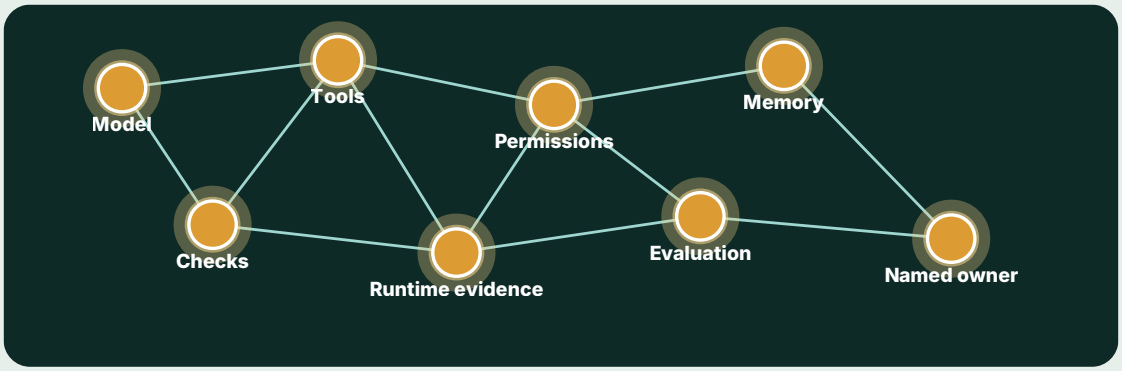
Next: once the map exists, reliability stops being a mood and becomes arithmetic. The next essay puts the same four questions to each layer in turn, and turns the answers into numbers.

THE STACK SO FAR

E04 · Essay 4 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 4 OF 5	II Evidence and authority	III Runtime control	IV Proof and accountability	V Operating model



stack you keep building from here · the whole stack is in scope · 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 nine-layer failure map

You can now locate failure by layer, not by vibes.

Next. E05 asks how to measure reliability when failure is layered.

← PREVIOUS
E03 · Demo Is Not Deployment

Essay 4 of 22 complete

NEXT →
E05 · The Reliability Equation

References

Reference links for sources cited in this essay.

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

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

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

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

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

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

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

9. **OWASP Top 10 for Agentic Applications (2026)**
   OWASP GenAI Security Project
   https://genai.owasp.org/resource/owasp-top-10-for-agentic-applications-for-2026/

10. **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

11 **AI Act (Regulation (EU) 2024/1689), Annex III: high-risk AI systems and associated layered duties**
European Union
https://artificialintelligenceact.eu/annex/3/

12 **AI Risk Management Framework (AI 100-1): Manage function**
US National Institute of Standards and Technology
https://www.nist.gov/itl/ai-risk-management-framework

13 **AI Cyber Security Code of Practice**
UK Department for Science, Innovation and Technology (DSIT)
https://www.gov.uk/government/publications/ai-cyber-security-code-of-practice

14 **Prompt injection is not SQL injection (it may be worse)**
UK National Cyber Security Centre (NCSC)
https://www.ncsc.gov.uk/blog-post/prompt-injection-is-not-sql-injection

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