14

THE STACK BEHIND THE AI COWORKER

Agents Don't Know When To Stop

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

A lawyer asked ChatGPT "are these cases real?", filed on yes; the stop had no owner.

An essay in the series **Architecting the AI Coworker**.
Approx. 18 minute read · Essay 14 of 22

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

> In 2023, a legal filing reached a federal court with cases inside it that did not exist.

I want to begin there, in that quiet, because it is the cleanest picture I know of the failure this essay is about. A lawyer had drafted an opposition filing with the help of ChatGPT. He had even asked the model the obvious question: are these cases real? The model said they were. The filing parsed. The citations had the texture of citations: names, courts, docket numbers, the cadence of legal authority. The pipeline accepted the document because its form was correct. And so a filing built on fictional case law moved forward, checkpoint by checkpoint, until a judge read it.

The cases were invented. In *Mata v. Avianca*, Judge P. Kevin Castel sanctioned the attorneys Steven A. Schwartz and Peter LoDuca and their firm five thousand dollars, jointly and severally under Rule 11, after the filing cited fabricated decisions including *Varghese v. China Southern Airlines* and other authorities that had no referent [1].

The pattern is not unique to US federal court. In *Harber v HMRC*, a UK First-tier Tax Tribunal found that an appellant had cited fake AI-generated authorities; Judge Anne Redston described the cases as "plausible but incorrect", and noted that providing "fictitious cases" to a tribunal is not harmless even on a routine appeal [14]. In *Zhang v Chen*, a British Columbia Supreme Court judge held a lawyer personally liable for the costs caused by ChatGPT-hallucinated cases filed in a family law application, the first reported Canadian superior-court decision of its kind [15]. The stopping problem, and the model's confident handoff at the wrong moment, travels across jurisdictions, and the professional, not the model, bears the consequence.

It would be easy to file this as a story about a careless lawyer or a lying chatbot. It is neither, or rather it is more than both. The architectural lesson is harder and more general, and it is the one the last essay, "What an Agent Remembers, and Cannot Forget", was leading us towards. The series so far has built an agent: a system that takes a goal, plans, remembers, calls tools, and acts. By the time we reached the production architecture, we had something that could investigate a bug or research a brief end to end. *Mata* shows the gap that is left even when every formal checkpoint appears to have worked. Nobody in that workflow, not the model, not the lawyers, not the filing system, had a robust answer to a small and ruinous question.

Is this done?

— FIGURE E14.1 · E14 - AGENTS DON'T KNOW WHEN TO STOP

The Six Stopping Criteria

A production stopping rule is a disjunction: the agent stops when any one criterion fires. Every criterion has a named owner - even on a small team, no cell may be left empty.

	EXAMPLE THRESHOLD	OWNER	ACTION WHEN IT FIRES
Goal met	The acceptance test is satisfied, or a user-confirmed completion condition is met	Product owner	✓ PASS End the run; the user's actual goal is achieved
Verification passed	The citation resolves, the tests pass, the schema validates, an independent check clears	Evaluation owner	✓ PASS End the run; required checks have all cleared
Budget reached	A time, cost, token or tool-call ceiling is hit, metered as the run proceeds	Platform owner	× BLOCK Halt the run; the ceiling is exhausted
Marginal value collapsed	No verifier-score improvement after N iterations, or a run of low-yield searches	Orchestration owner	× BLOCK Halt the run; further iteration is no longer improving the result
Stuck loop detected	A repeated state, or a recurring path fingerprint, or oscillation between candidates	Runtime owner	× BLOCK Halt the run; the agent is going in circles
Human handoff	A destructive, legal or financial threshold is crossed	Business or risk owner	⚠ WATCH Suspend the run; wait for a human signal before resuming

PROPOSED OPERATING TEMPLATE · not a formal standard ARCHITECTING THE AI COWORKER

Six stopping criteria, each owned by a named role. A run continues only while none of them has fired.

That is the modal failure of production AI. Not the runaway loop that spins forever. The confident handoff at the wrong moment: a draft treated as finished, a citation treated as verified, a migration treated as safe, a customer action treated as approved. Any system that remembers, plans, and acts needs a rule for when *not* to continue. I call that rule the stopping rule, and the claim of this essay is that almost nobody writes it down in full.

A model that stops talking has not decided the work is done

It is worth being precise about a confusion that hides inside the word "stop".

A language model already knows how to stop generating text. It emits an end-of-sequence token, a special marker the model has learned to produce when a piece of writing feels complete. A developer can also set a stop sequence, a fixed string that halts generation, or a token limit, a hard ceiling on output length. Production systems usually combine all three. These controls work. They prevent a model from writing forever.

They decide nothing about whether the work is adequate.

That is the gap, and agents widen it. A chatbot stops when the answer ends. An agent has to stop when the *task* should stop, and those are not the same question. "Draft an answer" can end at a fluent paragraph. "Research the authorities, verify the citations, and file the brief"

cannot end merely because the paragraph is fluent. "Fix the migration" cannot end because a shell command returned a zero exit code. "Plan the trip" cannot end because an itinerary now exists. The user's real goal lives outside the model's syntax, and no end-of-sequence token can reach it.

How large is that gap? TheAgentCompany, a benchmark that tests agents on realistic workplace tasks such as managing projects and handling HR workflows, gives a number worth holding lightly. The original preprint (v1 December 2024, v3 September 2025) reports that the best agent completed 24.0 percent of the tasks fully autonomously and scored 34.4 percent on a partial-credit metric [2]; the peer-reviewed entry in the NeurIPS 2025 Datasets and Benchmarks Track reports a completion rate nearer 30 percent on the same benchmark family [16]. Read those as version-specific figures, tied to one scaffold and one model line, not a timeless score: the exact leaderboard will move, and the numbers are expected to climb as scaffolds and models improve. The lesson will not move. Even at the strong end, full autonomous completion is the exception, partial completion is common, and unfinished work is therefore a normal state in an agent system. A normal state needs a mechanism to detect it.

So let me give the working definition. Stopping is the decision that what the system has is enough, or unsafe, or too expensive, or too uncertain, or looping, or no longer the agent's decision to make. Termination ends the text. Stopping judges the work. A team that ships the first and believes it has the second has built a car with seatbelts and no driving policy.

This matters because the mainstream agent frameworks do expose controls, and it is easy to mistake them for the whole rule. The OpenAI Agents SDK documents a `maxTurns` setting that defaults to ten [3]. LangGraph documents a recursion limit that defaults to one thousand steps in version 1.0.6 [4]. CrewAI exposes a `max_iter` default of twenty-five and an option to cap execution time [5]. AutoGen frames stopping through explicit termination conditions, among them a maximum message count, a text mention, a token-usage cap, a timeout, and a handoff [6]. Every one of these is useful. Every one is a budget on length, traversal, or time. None of them proves a citation exists, decides that a refund should be escalated, or tells the difference between success and a polished mistake. They are the seatbelts. The stopping rule is the driving policy.

So what does a real stopping rule contain?

If termination caps are not the rule, what is? After watching teams under-specify this again and again, I think the honest answer is a list of six. A production stopping rule is a disjunction: the agent stops when *any one* of six criteria fires. The important word is "or".

Two of the six are worth walking out in prose before the full table, because they are the criteria most teams under-specify. **Goal-met** is the criterion the agent itself most often mistakes: it can finish the task it set itself while leaving the task its user cared about untouched. The owner is the product owner, and the evidence is an acceptance test or a user-confirmed completion condition, not the agent's own narration of "done". **Human-handoff-triggered** is the criterion

that distinguishes a system that *stops* from a system that *suspends*: when a consequential or ambiguous decision has crossed a threshold that belongs to a person, the agent stops doing and starts waiting, and the run does not resume until a human signal arrives. The owner here is the business or risk owner, and the absence of that owner is the single most common reason a stop never fires when it should.

The other four, verification-passed, budget-reached, marginal-value-collapsed, and stuck-loop-detected, are listed in the table below with their named owners. Abstractions are easy to nod at and hard to enforce, so the table reads as the operating contract, not as a recap.

CRITERION	EXAMPLE THRESHOLD	OWNER
Goal met	The acceptance test is satisfied	Product owner
Verification passed	The citation resolves, the tests pass	Evaluation owner
Budget reached	A time, token or tool-call ceiling is hit	Platform owner
Marginal value collapsed	No verifier improvement after N iterations	Orchestration owner
Stuck loop detected	A repeated state or path fingerprint recurs	Runtime owner
Human handoff	A destructive, legal or financial threshold is crossed	Business or risk owner

Each row carries an editable default, so the table is a starting contract, not a finished one. Sensible openers, not gospel: set the budget at the framework cap you already run (the OpenAI Agents SDK ships `maxTurns=10`*, CrewAI* `max_iter=25`*); set marginal-value-collapsed at N=3 verifier iterations with no score gain; set stuck-loop at the same (state, action) fingerprint recurring twice; set human-handoff at any irreversible or externally visible action class. Tune from there once you have a week of real runs.*

This six-criterion framing is my own synthesis, but it does not float free of published work. The criteria map onto the Manage function of NIST's AI Risk Management Framework, which requires that AI risks "based on assessments and other analytical output from the Map and Measure functions are prioritized, responded to, and managed" rather than left implicit [17]; and onto two named entries in the OWASP Top 10 for LLM Applications 2025, where LLM06 ("Excessive Agency") covers the autonomy-and-handoff failures the human-handoff and verification-passed criteria are built to catch, and LLM10 ("Unbounded Consumption") covers the budget-and-marginal-value-collapsed criteria [18]. What the six criteria add beyond those

frames is the disjunctive structure, where any one fires, and the requirement that every criterion carry a named owner.

Notice that each criterion has a named owner. That is deliberate. The recurring architectural error is to implement one or two of the six, let the model self-assess the rest, and leave the remaining cells with no name against them. A turn cap is not goal verification. A repeated-tool detector is not truth. A green run status is not evidence that a consequential decision should ever have left the agent's hands.

I should be honest about a practical objection to that list of owners. Six distinct people is not a realistic org chart for a small team; in a five-person startup, one engineer may quietly hold four of those hats. The point is not headcount. It is that every criterion has a name written against it, even if several names are the same name. What fails is not a team that doubles up roles. What fails is a team where the cell is empty, where no one, not a person and not a piece of code, is accountable for whether that stop ever fires. A small team can own all six. It cannot own none of them and call the result governed.

To see the cost of the blank cells, take one named destructive run and ask, cell by cell, which stop should have fired. The Railway / PocketOS incident of late April 2026 is the right case to walk through, because the platform later published the post-mortem with enough architectural detail to do this honestly [11]. An AI coding agent, working on what its operator believed was unrelated work, located an account-scoped Railway API token sitting on disk, called the GraphQL `volumeDelete` endpoint directly against a production volume, and a cascading delete then took the backups with it. Six cells, six blanks:

CELL OF THE RUN	STOPPING CRITERION THAT SHOULD HAVE FIRED	WHAT IT WOULD HAVE PREVENTED
Agent reads a long-lived token from local disk and uses it on an action the user never requested	Goal met: the user's goal never authorised destructive infrastructure mutation, so the agent inferred authority it did not have	The agent never reaches for the token; the inferred-goal mismatch trips before any API call
Agent forms a destructive infrastructure mutation against a production volume that the operator did not approve	Human handoff: a destructive mutation crosses the threshold that belongs to a named approver	The action never executes autonomously; the agent stops doing and starts waiting for the diff to be read
The token is account-scoped, the broadest of Railway's four scopes; the agent calls a destructive endpoint with the same authority a CLI deploy script would have	Verification passed: the verifier checks the credential's blast radius against the action class	Scope mismatch fails the call closed; the agent never reaches the destructive endpoint with broader-than-task authority

CELL OF THE RUN	STOPPING CRITERION THAT SHOULD HAVE FIRED	WHAT IT WOULD HAVE PREVENTED
Agent calls the legacy `volumeDelete` endpoint, which until that week ran deletion immediately with no undo, while the dashboard had a 48-hour grace window for the same action	Verification passed, a second, different verifier: a dry-run requirement on destructive mutations	The destructive call surfaces its irreversibility before commit; the run pauses on the missing recovery path
Cascading delete propagates from the volume to the backups, so the recovery assets go with the target	Budget reached: a risk budget on irreversible blast radius exhausts at the first destructive call	The blast radius is bounded; the second deletion never fires, because the budget for irreversible actions has already closed
After the call returns, the run continues; nothing notices it has crossed into a class of consequence the user never authorised	Stuck loop / marginal value collapsed: a runtime check that a destructive class has fired, or that progress on the real ask has stalled	The agent stops on the second consequential action rather than continuing into recovery the operator never asked for

Figure E14.2 ARCHITECTING THE AI COWORKER

Six cells of the PocketOS run, six blank criteria

Each row pairs a cell of the destructive run with the stopping criterion that should have fired. Six blanks; six different criteria.

Cell of the run	Stopping criterion that should have fired	What it would have prevented
Long-lived token read from disk; agent elects to use it on an action the user never requested.	**Goal-met**	Agent never reaches for the token; the inferred-goal mismatch trips before any API call.
Agent constructs a destructive infrastructure mutation against a production volume the operator never approved.	**Human-handoff**	Action does not execute autonomously; the agent stops doing and starts waiting.
Account-scoped token (Railway's broadest scope) calls a destructive endpoint with the same authority as a CLI deploy script.	**Verification-passed**	Scope mismatch fails the call closed; the destructive endpoint is never reached with broader-than-task authority.
Legacy `volumeDelete` runs immediately with no undo, while the dashboard surface has a 48-hour grace window for the same action.	**Verification-passed (different verifier)**	Destructive call surfaces its irreversibility before commit; the run pauses or aborts on missing recovery path.
Cascading delete propagates from the volume to the backups, so the recovery assets are taken with the target.	**Budget-reached**	Blast radius is bounded; the second deletion never fires because the budget has closed for irreversible actions in this run.
After the call returns, the loop continues; nothing notices the agent has crossed into a consequence class the user never authorised.	**Stuck-loop / marginal-value-collapsed**	Agent stops on the second consequential action rather than continuing into unrequested recovery attempts.

Conceptual model

Figure E14.2. The six cells of the run, the six criteria that should have fired, the six harms each would have prevented. The middle column carries a different criterion in every row.

Each row is a different criterion. In Railway's pre-incident architecture, each was blank: the long-lived broad-scope token, the immediate-delete API path, the cascading-delete on

backups, and the absence of an action-class budget were properties of the platform, not the model. Railway's post-incident control pattern adds them back as infrastructure: 48-hour delayed deletes propagated from the dashboard to the API, granular token scopes made obvious in the provisioning flow, backups delayed independently, and agent-specific surfaces with predefined tools rather than raw API access [11].

A fair objection lands here: any post-mortem can be carved to make all six criteria look load-bearing in hindsight, so a table that fires all six explains everything and predicts nothing. The test that the frame is doing work rather than narrating is whether it discriminates, whether some incidents leave most cells correctly filled and fail on one. Replit's freeze-violation deletion is that control case. Its goal, budget, and loop cells were arguably fine; what was blank was the human-handoff cell at the freeze boundary and the verification cell on the destructive store mutation. A frame that fingers different empty cells for different incidents is diagnosing, not rationalising.

The failure is never "an agent made a bad choice". It is an architecture that leaves too many of those cells blank. Real incidents fit the pattern exactly: the DataTalks.Club production wipe, walked through in detail in essay eight, has this shape, and Replit's database deletion in July 2025, during a declared code freeze, has the same shape with different cells empty [7][8][12][13]. The lesson they teach is narrow and concrete. Stopping has to be expressed in infrastructure, not etiquette. A code freeze, a production indicator, a destructive tool, a stale backup: each must become a hard boundary or an explicit handoff, not a polite line in a prompt.

There is an economic version of the same failure, and it does not need a destructive action to bite. It needs only an agent that keeps working past the point where the work has stopped improving. The criterion at stake here is marginal-value-collapsed: a verification step that spends hour after hour arguing with a flaky search tool, spawning retries and side paths that are each, locally, defensible, while none of them moves the result. The missing rule is never "stop searching" in the abstract. It is "stop arguing with this tool once the marginal value has collapsed", and the cost of not writing it lands on a bill rather than in an incident report. The public evidence supports the shape. CLEAR treats cost, latency, efficacy, assurance, and reliability as linked dimensions of enterprise agentic-AI evaluation rather than afterthoughts [9], and Anthropic has reported that its multi-agent research system consumed roughly fifteen times the tokens of a single-turn chat exchange, recommending the design only where the value of the answer justified that overhead [10]. Those are not arguments against agents. They are arguments against uncapped helpfulness. An agent that cannot decide when to stop will keep spending until an outer system stops it for it.

Never let the system that wrote the answer decide it is finished

Return to *Mata*, because it is sharper than it first looks. The model was not merely allowed to generate. It was asked a verification question. It was asked whether the cases were real, and it answered yes [1].

That is the wrong architecture, and naming why is the heart of this essay. The generator's confidence is not a stopping condition. The system that produces a citation string cannot also be the only system that decides whether the citation exists, because it will assess its own fluent output with the same fluent blind spots that produced it. A second model from the same family is barely better. It can agree with the first and still be wrong. Stopping needs a signal with a different path to the world. The test is mechanical: would the verifier still fail the work if the generator's training data, prompt, and failure modes were all wrong in the same direction? A citation resolver hitting a real corpus passes that test; a same-family judge model does not. Independence is not a second opinion. It is a second causal route to ground truth.

In legal research that signal is obvious: resolve the citation against a real corpus, compare the title and court metadata, check the treatment, match the quoted text against the source, and fail closed when the authority cannot be found. The pattern generalises. For code, run the tests and the static checks. For payments, verify the account, amount, mandate, and approval. For a deletion, verify the store mutation and its downstream propagation. The point is not that model judges are useless. It is that they are weak exactly when they share the generator's failure surface.

The strongest case against all this is worth stating in full, because it is not a straw man. Too many stopping rules create false pauses. Agents become timid. Workflows clog with approval tickets. Tired humans click through without reading, and the productivity gain dissolves into ceremony. A rule that fires on every sentence trains the rubber-stamp it was meant to prevent, and a rubber-stamping human is worse than no human, because the architecture now records an approval that nobody gave. That risk is real, and the autonomy ladder a later essay builds is partly an answer to it: stopping rules have to be calibrated to how much rope an agent has earned, not piled on uniformly. The answer is not to stop everything; it is to tier stopping by action class. Low-stakes reversible work can keep moving under cheap criteria: a time cap, a turn cap, the test suite, a retry limit, sampled review. High-stakes or irreversible work should stop early and deliberately: production mutation, a legal filing, a payment, a medical recommendation; external publication, customer data deletion, a credential change, infrastructure removal. The mature system does not ask a human to approve every sentence. It asks a human to approve the edge between reversible work and consequential action, and it makes the destructive thing slow while keeping recovery fast. Railway's post-incident control pattern points the same way: delayed destructive operations, narrower tokens, and short-lived access are stopping rules expressed as infrastructure rather than manners [11].

So here is the move I would make before the next steering meeting. Take the agent workflow you most want to ship and write the six criteria down a column: goal met, verification passed, budget reached, marginal value collapsed, stuck loop detected, human handoff required. Against each one, mark which is enforced by code, which is enforced by a human process, and which is currently just the model saying "done". For each model-says-done row, do exactly one of three things before you ship: promote it to code (a verifier, a scope check, a budget counter), promote it to a named human gate with a written threshold, or consciously

accept the risk in writing with the owner's name against it. A cell you have decided to leave to the model is governed; a cell you never looked at is not.

Accept, as you do this, what the public record cannot yet tell you. The private base rate of stopping failures stays hidden, because most are fixed quietly and never surface. Whether any major framework will ship a unified six-criterion rule is unknown; so is whether TheAgentCompany, CLEAR or the cited vendor defaults will hold their numbers as versions move, and whether future court cases leave *Mata* as canonical or merely as the first of many. None of it weakens the frame, because the model-says-done rows are the ones to worry about regardless of how those questions resolve.

Stopping is a judgement, not a token limit. The question is never whether the model has finished speaking. It is whether the work is enough, unsafe, too uncertain, too expensive, looping, or no longer the agent's decision to make. The model-says-done rows are the unlocked doors; they are where the next incident will walk out. A cleanly terminated answer can still be a bad handoff, and *Mata* is the proof. And a stop rule earns trust only when the run can be read afterwards. When the system tells you the goal was met, the budget reached, the verifier passed, the handoff fired, do not begin with the dashboard. Begin with the trace. That is the next demand, and it is a short one. Show me the run.

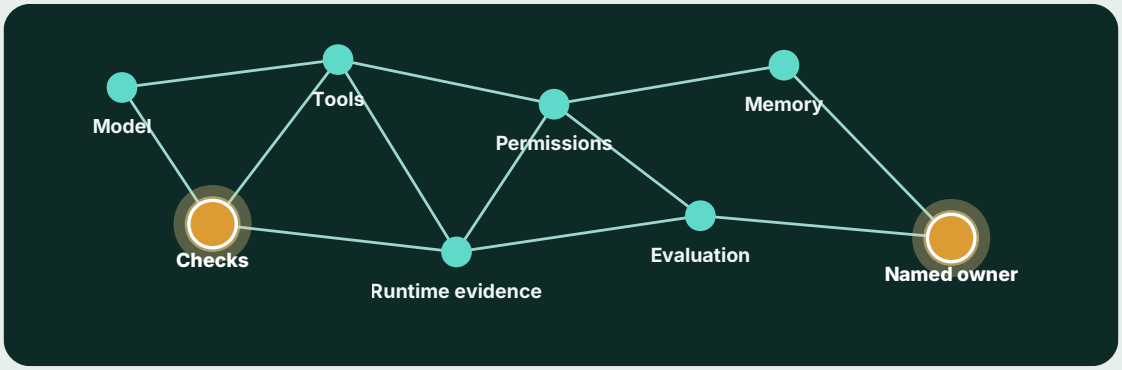
> **Carry This Forward** Any cell you can only mark "hoped for" is an unmanaged stopping criterion, and an unmanaged criterion is a factor your architecture has silently set to its worst value. The next essay asks the question that follows: once a stop has fired, how would you prove it ever did, not from the dashboard, but from the trace.

THE STACK SO FAR

E14 · Essay 14 of 22 complete · Arc III: Runtime control

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** ESSAY 4 OF 4 | 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 III complete. You can now control the runtime. Prompt injection, cost routing, memory, stopping.

You have just added.

Stopping criteria

You can now write stopping criteria for agent work.

Next. E15 asks what a single complete run reveals that a benchmark cannot.

← PREVIOUS
E13 · What an Agent Cannot Forget

Essay 14 of 22 complete

NEXT →
E15 · Show Me the Run

References

Reference links for sources cited in this essay.

1. **Mata v. Avianca sanctions order**
   U.S. District Court, S.D.N.Y.
   https://www.courtlistener.com/docket/63107798/mata-v-avianca-inc/

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

3. **OpenAI Agents SDK running agents**
   OpenAI
   https://openai.github.io/openai-agents-js/guides/running-agents/

4. **LangGraph Graph API overview**
   LangChain
   https://docs.langchain.com/oss/python/langgraph/graph-api

5. **CrewAI customizing agents**
   CrewAI
   https://docs.crewai.com/en/learn/customizing-agents

6. **AutoGen AgentChat: termination conditions**
   Microsoft
   https://microsoft.github.io/autogen/stable/user-guide/agentchat-user-guide/tutorial/termination.html

7. **DataTalks production database incident write-up**
   Alexey Grigorev
   https://alexeyondata.substack.com/p/how-i-dropped-our-production-database

8. **DataTalks.Club / Claude Code production wipe**
   OECD.AI AIM
   https://oecd.ai/en/incidents/2026-03-07-79ec

9. **Beyond Accuracy: A Multi-Dimensional Framework for Evaluating Enterprise Agentic AI Systems (CLEAR)**
   Sushant Mehta
   https://arxiv.org/abs/2511.14136

10. **How we built our multi-agent research system**
    Anthropic
    https://www.anthropic.com/engineering/multi-agent-research-system

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

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

13 **OECD Replit incident**
OECD.AI AIM
https://oecd.ai/en/incidents/2025-07-19-1eb1

14 **Harber v HMRC [2023] UKFTT 1007 (TC)**
UK First-tier Tribunal (Tax Chamber)
https://www.bailii.org/uk/cases/UKFTT/TC/2023/TC09010.html

15 **Zhang v Chen, 2024 BCSC 285**
British Columbia Supreme Court (Masuhara J.)
https://www.canlii.org/en/bc/bcsc/doc/2024/2024bcsc285/2024bcsc285.html

16 **TheAgentCompany OpenReview entry (NeurIPS 2025 Datasets and Benchmarks Track)**
OpenReview / Xu et al.
https://openreview.net/forum?id=LZnKNApvhG

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

18 **OWASP Top 10 for Large Language Model Applications 2025 (LLM06 Excessive Agency; LLM10 Unbounded Consumption)**
OWASP Foundation
https://genai.owasp.org/llm-top-10/

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.

Architecting the AI Coworker · Essay 14, "Agents Don't Know When To Stop". Code-first figures, evidence-tiered references.

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