Please Don’t Say That: Stopping AI From Giving Bad Advice, Leaking Data, or Breaking the Rules

Architecture beats wishful prompting

Ian Collier · 5 August 2026

The most common questions we get, by some distance, are about AI mistakes — especially the expensive ones. Typically that means financial, regulatory or legal.

What stops a new AI system from giving away personal or private client data? What prevents it from promising something a business can’t deliver? What if it gives poor financial advice, or strays into services that breach FCA rules? And how do you reduce the risk of the AI saying something which costs a client — or the business — serious money, especially when the “AI did it” defence remains untested in court?

Those risks are real. The moment an AI system can see client files, private information, previous advice or proprietary data, the question is no longer whether mistakes are possible. It becomes how you stop it from crossing the line: producing the wrong thing, for the wrong person, or without the right controls.

This is not a capability problem. The protections described in this post can all be built. What matters is how much risk a business is prepared to carry. In most engagements we go pre-armed and expecting to build a low risk build — purely because low risk over generic already produces much better accuracy. But there’s also a very low risk model and, of course, a banking-grade zero-risk version that offers little practical value for most businesses in this automation wave — and largely negates the point of adopting AI in the first place.

Outside of a purely generic AI setup, four viable options exist:

- A practical structure (low risk)
- Gold-standard protection (very low risk)
- A hybrid version that starts practical, selectively adopts stronger controls, and is built so protection can be raised or reduced over time
- The zero-risk model, which is effectively useless for most business applications

We have already built the hybrid version and prepared a gold-standard build for a legal practice. The recommendation we give to most businesses is to start with the practical structure and then add further protections where required — in other words, the hybrid approach.

First, what does a badly designed AI integration look like

Surprisingly, it's never in the dramatic way people imagine. No system announces that it is about to leak something, and because AI output always arrives fluent, formatted and entirely confident, nothing about it invites a second look.

The failure points are structural. A system provided with real company data, or simply left free to draw from its own general knowledge, will go wrong in four predictable ways unless each is deliberately constrained from the outset. These problems appear most when broad internal material is made available through retrieval (RAG), but they are not limited to it.

Retrieval looks for relevance, not for permission

Left alone, an index of data will return the closest match to a question. Closeness and permissions are unrelated properties. Unless entitlement is evaluated at query time against the identity and permissions of whoever is actually asking, the retriever will faithfully return the most relevant material in the corpus — which can be your most sensitive material. Specific beats general on every similarity measure.

Permissions sit atop of whole files; leaks happen in the small pieces

Source systems grant rights over files, folders and matters. Retrieval iterates over fragments of those files. No fragment carries any rights of its own unless it is deliberately propagated down to it, so a document somebody is entitled to open can also contain other passages and data that were never intended to leave the building. Container-level permissions cannot see inside the wider container.

Generation dissolves provenance — and invents what it does not have

Once data enters an AI context window it becomes flat. System instructions, retrieved documents and user input occupy the same space just without a security boundary. The model has no dependable way to distinguish what it was given in confidence from what it is allowed to restate, or from what it simply believes it already knows. When retrieval returns little or no data, it will often fall back onto its own general knowledge and produce an answer anyway. The output arrives with no record of how it was arrived at, or what it was assembled from. It can therefore not be audited, not be selectively redacted, and can never be shown to have been safe after the event.

Every connection is another way out, and combining information just makes it worse

Each integration becomes a destination for material that was assessed entirely separately. Because an answer can be assembled from many fragments — or from a mixture of retrieved data and the model's own knowledge — a system can disclose a combination of information that no single source would have ever disclosed on its own. At this point, nothing was ever improperly accessed; however something was nonetheless inferred or invented, and then stated as fact.

No attack, social engineering, or bad intent is required. It is simply the default behaviour of a system pointed at real company data, or left alone to answer from what it thinks it knows.

The first two are largely consequences of a single early decision — putting broad internal material into one retrieval pile and then asking the system to be careful about it. Both are significantly reduced if that decision goes the other way. The last two are not. Loss of provenance, confident invention from general knowledge, and disclosure by combination are real whatever you build, and each has to be defended against properly.

So, just build a stronger prompt, right? Wrong!

The standard response is to treat all of the failures above as a prompting problem. So write firmer instructions. Buy a platform that advertises “guardrails”. Add a line in capital letters at the top of your North Star that says “Never Disclose Confidential Information”.

None of that is a control. A system prompt asks. It does not prevent. It carries no enforcement mechanisms — it just means more text in the context window, competing for attention with whatever data was retrieved and with whatever the user typed. It holds only for as long as the model keeps choosing to honour it.

Which it does, mostly. Right up until somebody rephrases a request, or the retrieval step returns something unexpected, or the model decides the current request is clearly an exception. It is a single layer, and single layers fail.

That is precisely the argument for independent controls — both inside and outside the model — that can be tested separately. These are not prompting problems. They are architectural problems. Proper data protection requires controls that are separated enough for each one to be isolated and attacked deliberately, while the others still hold if one gives way.

1. The “Practical” Solution (low risk)

This is the architecture that works for the vast majority of professional businesses. It is deliberately layered, before and after generation, so that each part can be tested independently. Each layer audits the one before it. It follows the pipeline described earlier in which every control sits at the point where any one of the four failures occur. It naturally becomes a cornerstone of solution 2 (the very low-risk Gold-standard model) and is our default build.

Two classes of data, handled in completely different ways

Almost every leak described above begins with the same decision: put everything into a single data pile and simply ask retrieval to be careful. The first implemented control mechanism in the practical solution does none of that.

The uploaded corpus — the material sitting in the index waiting to be matched against a question — should only contain beige / general company data. Who the business is, what it does, which services it offers, how it works, standing policy and things like ‘about us’. Nothing stakeholder-specific. Nothing personal. Nothing anybody has to rely on. Nothing that is not already on the website. It is safe not because it is heavily guarded, it’s safe because of what it is. If the whole index were printed out and left on a train, the damage would be zero.

Everything else — client records, figures, matter history, anything specific to one person or one company — does not go into any RAG index. Nor does it go into a second database. This is the point at which most builds take the easy road and regret it later.

The temptation is to import the sensitive data. Copy it into a data store, classify it on entry, and then let the AI query that on demand. This is much easier to build and the responses will be faster. When dealing with sensitive material it is entirely the wrong answer — because you have just created a second permanent copy of the most critical data in the business, with its own permissions, its own backups, its own deletion

obligations and its own attack surface. The original problem was that sensitive data was reachable. Now there's two copies of it.

The answer is to not import any sensitive data at all. Instead, fetch it as required on demand, via a single request, by API or webhook querying the system that already holds it. The data is then used and discarded. Nothing persists. There is no second copy to secure, reconcile, or include in a subject access response.

What on-demand does not mean is unchecked. Every imported payload gets parsed, classified and validated before the model is even allowed near it. Fields are matched against what this request was entitled to ask for, so a source system returning more than requested — which happens routinely — gets trimmed rather than passed whole. Sensitivity, client boundary and regulatory flags are applied to what remains. Anything unrecognised, unexpected or ambiguous is dropped or escalated.

One field matters more than any other, and it is the one most often lost in transit: status. A figure is not simply a number. It is filed, or drafted, estimated, or superseded — and that distinction has to travel with the value into the model. A provisional figure stated as a settled one is how a system commits a company to something nobody agreed to. Data arrives with its status attached or it does not arrive at all.

You start with the pinch points. What is this system actually for? Which questions does it need to answer, for whom, and what data is required? Only then do you decide what is safe to fetch, and on what conditions. Those two lists rarely match, and the gap between them is deliberate. Fetch only what you would be willing to display to a logged-in user in a secure web portal — and only at the moment they ask for it.

The consequence that matters: at any given moment the system holds nothing sensitive. The general corpus is dull by design. Sensitive data exists inside the request for the length of that request and is then discarded. No accumulated store to breach. No store to keep in step with source.

Where document content has to be exposed, excision errs on the side of removing rather than rewriting. Surviving text stays identical to source. A model asked to paraphrase sensitive content into a "safe" summary will eventually alter meaning while still reading perfectly well. In an advice context, fluent and subtly wrong is the worst possible combination. Where confidence is low, the content is withheld and flagged for human approval — or an API triggers a send directly from source.

The strongest version of this control: where even momentary exposure is too much, the model is not given the data at all. It determines what needs to happen and triggers it through the system that already holds the information. The sensitive content never touches the model. For high-risk material, reliance on existing system connections should be the first thing the AI triggers, not the last.

Entitlement enforced at the point of request

Deciding what may be fetched determines what can be reached at all. It does not determine who may reach it. Every fetch is authorised against the identity of the person making the request, at the moment they ask, against permissions in the source system rather than a synced copy. Most leaks die here, because the material never enters the room.

Constrained generation, with every claim bound to its source

The model answers only from permitted material. Each statement carries a reference back to the record it came from. That binding restores the provenance the context window destroys. Where a claim cannot be grounded, or would cross into regulated advice, the system declines. A system that says "I can't answer that from the data I hold" is not a weaker system. It is a system with enforced boundaries. The person still receives an answer — even if that answer is an email being sent or a referral to a human.

The independent final review

Fetching on demand does not remove the need for a final audit of the proposed output. Data that was correctly fetched and classified on the way in can still end up in an output that says more than it should, because the problem is created at assembly. What's required is a separate independent agent checking output against policy before it goes anywhere. It does not generate. It only passes, redacts or blocks. Because it did not write the answer, it has no investment in defending it.

Edge cases and sign-off

Data flagged as sensitive and bound for an approved recipient can be stopped for human approval. Content is fingerprinted so what gets approved is provable against what was sent. Sign-off happens through an authenticated channel kept separate from the AI conversation. The model does not get to participate in the decision about whether the model was right. Every destination for output is registered and scoped in advance.

Every layer is built and then attacked in a Parallel environment before any real client sees output. Output stays internal until the structure holds under pressure.

Practical Architecture Diagram

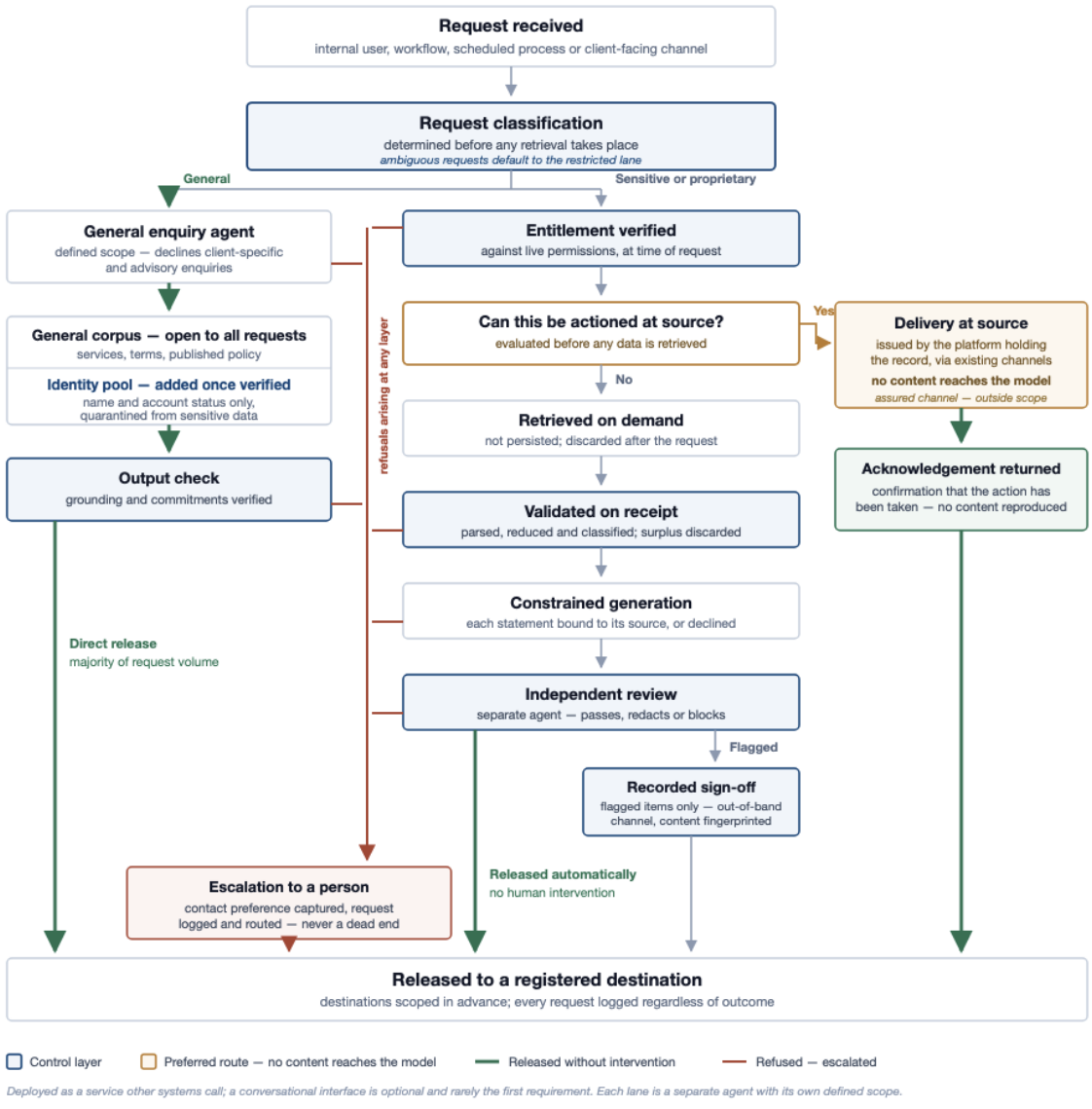

Figure 1 — Practical architecture: control layers, preferred routes (no content reaches the model), automatic release, and escalation paths.

2. The Gold Standard approach (very low risk)

For exceptional, highest-risk environments — large regulated financial services, automated legal environments or anywhere the cost of a single failure is existential — the bar rises.

The practical architecture already separates data classes, enforces entitlement at request time, constrains generation and adds an independent review layer. The gold standard bolts on additional isolation the practical version deliberately does not attempt.

- Infrastructure isolation. Dedicated tenancy, segregated networks, and compute that runs nobody else's workload.
- Fully tool-mediated access. No free-form retrieval. The model can only call a fixed set of narrow, individually authorised queries via webhook or API.
- Zero AI output delivery, applied universally. The AI is never provided with client-specific or sensitive information. It talks around the subject at a general level and presents available services or next steps. Those are then triggered automatically or passed to a human. The model never generates the sensitive content.
- Hybrid controls. Deterministic rules handle the hard boundaries. LLM review is reserved for nuance.
- Maker-checker dual control. High-risk releases require two authorised people.
- Continuous automated red-teaming and monitoring after go-live.
- Immutable audit logs.

Correct where the risk profile justifies the cost and overhead. For most professional businesses it is more protection than the actual risk requires on day one.

3. The Hybrid Approach — Some from column A, some from column B

For most professional businesses the gold-standard provides more protection than the actual risk requires, especially on day one. This is why we also offer a hybrid approach. It is deliberately pick-and-mix. You start with the practical structure as the foundation, then selectively add stronger controls only where they are needed. The system is designed from the outset so protection can be raised later without a rebuild.

In practice most hybrid set-ups begin practical and migrate toward higher security only as real usage data shows where the residual risk actually sits.

4. The banking-grade solution (the zero risk option)

There is exactly one way to ensure complete safety: keep all data disconnected from the AI. This essentially means the value-add capabilities of AI are switched off. It works. It is also the least useful option for almost nobody reading this will want.

The point of listing every option, including the impractical one, is simple. The moment you decide an AI system can see client data, internal notes or past advice, you have accepted some degree of risk — however small. That is usually the right commercial decision. But it should be a decision you make deliberately, with open eyes, not something that happens because a supplier switched on a feature.

How does it stack up? Show me some real life scenarios

Architecture is easy to draw and harder to defend. Here are three typical examples run through the practical model.

"Could you send me the witness statements for tomorrow's hearing?" — from a 3rd party barrister.

Classified as sensitive. Entitlement checked first. The statements already sit in the document management system, so the AI identifies which ones are required and triggers their release through that platform's own secure channel. Counsel receives the documents. The model never opens a witness statement. Nothing privileged passes through it.

"What is my tax liability for 2024/25?" — from a business client.

Entitlement is straightforward. The real risk is status. A figure from a filed return is a fact. The identical figure from a draft computation is provisional — and stated without qualification it becomes a commitment the company never made. So the outcome depends on what the fetched data carries with it. Filed and final: released bound to its source. Draft or estimated: flagged by the review layer and either stopped for approval or returned with the qualification attached. A number without its status is not an answer, it's an exposure.

"What time do I have to be in court tomorrow?" — from a client.

Client-specific, so restricted lane regardless of how mundane it looks. Entitlement checked, diary read live, answer released. This is also where fetching on demand earns its keep. Court listings move. A system answering from a store refreshed overnight will eventually send somebody to the wrong hearing — confidently, in writing. Reading the live record is the control.

Two of those three requests are better served by triggering an action at source than by generating anything at all. That distribution — most work done without the model touching anything sensitive — is the argument for the whole design.

What this looks like from your side of the table

What do you need to hand over, in what format, and what comes back? Less than most people expect — and none of it is a file dump.

Connections, not data dumps.

For anything operational — ledgers, client records, matters, diaries — you authorise a read-only encrypted connection (APIs and webhooks) to the systems you already run. Together we identify the pinch points and define the narrowest set of data required for each type of request.

Sensitive or private data is not stored in a second database. It is fetched on demand for the specific query, cleaned and checked in memory, injected for that request only, and then discarded. Where even that exposure is too high, we do not ask the AI to generate the output at all. Instead we trigger the required action through your existing systems so the sensitive content never passes through the model.

General, non-sensitive material can be prepared and stored if useful. Private and sensitive data is not. The core protection is simple: give the AI the minimum data required, at the moment it is required, and nothing more.

A modest pile of general material.

The only material indexed as files is the least sensitive information you own: what the business does, services, standard terms, policies, the questions clients ask repeatedly. Most of this already exists.

The rules only you know.

This is usually the most valuable conversation in the engagement. Who may see what. What must never go out without a partner reading it first. Where advice stops and a regulated boundary begins. What you would never put in writing to a client. That discussion produces the classification and handling rules that every layer enforces.

A handful of examples, good and bad.

Correspondence you were happy to send. Something that would have been unacceptable, and why. A small set of real examples calibrates the review layer far better than abstract policy alone.

You do not need to tidy anything first. “We should sort our data out before we look at AI” is one of the most common reasons businesses never start, and it gets the order wrong. Messy is normal. Messy is expected.

What comes back

For general questions the system can answer directly from the approved general material. For anything that touches client-specific or sensitive data, the AI is given only the minimum information required for that exact request, at that moment. In many cases it does not generate the final content at all. Instead it identifies the required action and triggers it through your existing systems (or passes it to a human).

Drafts that involve sensitive material are prepared for human approval rather than released automatically. Nothing writes back into your systems on day one. Everything runs alongside the people currently doing the work until it is demonstrably better and safer than the manual process.

Underneath it all sits a log of every request: what was asked, what data was fetched, what (if anything) was released, who asked for it and who approved it.

Ok, cool, we want chat, we want voice, document retrieval, agentic workflow…

Yes, of course. All of that, and more besides.

The structures described above are not separate products for each channel. They are the foundation that everything else runs on. One controlled core, one consistent set of rules, one place where data access, review and approval are enforced. Chat, voice, document retrieval, internal advice tools, client-facing outputs — they are all simply different endpoints served by the same underlying system.

You do not build a new brain for every interface. You build one properly controlled nervous system, then connect the channels to it.

What this actually comes down to

The technical question is not whether AI can see your data. It is whether you can control what it is allowed to do with it.

Most professional businesses do not need banking-grade isolation. They need a clear separation between general material and sensitive data, on-demand access rather than bulk exposure, independent review of outputs, and a human approval step where it still matters. That combination is achievable, testable, and proportionate.

The work begins with a practical conversation: where sensitive data currently sits, who can already reach it, and which processes would genuinely benefit from controlled assistance. From there the right level of protection becomes obvious.

When the architecture is right, the scary questions largely disappear. Ignore the architecture or get it wrong and no amount of prompting will stop it failing.

About the author

Ian Collier is a technology entrepreneur and AI systems architect with more than 30 years' experience building production software and automation platforms. He previously founded and sold a software business to Yahoo!, and now focuses on practical, low-risk AI architectures for professional services firms — helping organisations adopt AI without exposing client data, regulatory boundaries, or commercial commitments.