

Four Cornerstones: A Buyer's Framework for AI Governance Tooling

By Eric Pace

August 2026

Disclosure: I am a paid advisor to TrueFoundry. I was also a TrueFoundry customer before that, which is the part of this that matters. Everything below is written from the buying side. I got there the long way, building these systems for other companies before I had to own the implementation myself.

If you can't see something, how do you say something?

The first time I felt the need to support a governance initiative was the day I owned one. The challenge was to make it an enabling force inside a Fortune 500-sized enterprise, and anyone will tell you "governance" and "enabler" are oxymorons in the enterprise. So there we were, in one of our first AI Council reviews, and my Finance Business Partner (an informal role on the council at the onset) asked the question: "How many models are we running in production, and what do they cost us?"

Ten people in the room. Nine blank stares. That was a blank stare from me (Chief AI Officer), Strategy, Architecture & Engineering, Research & Development, Product, Cybersecurity, Infrastructure & Operations, Data & Data Privacy, and Legal & Compliance.

We had a policy. Three pages, plus links to our code management guidelines and our standards for using generated images. Approved, circulated, filed. It was a good policy... but it could not answer the question.

Nobody in that room was negligent. We had a policy where we needed a platform. A policy describes what people are permitted to do. Governance is the machinery that makes the permitted thing the easy thing. One gets written by a committee while the other gets built or bought, deployed, and wired into your production path, where it either exists or it does not. Where it gets in the way or it does not.

Most teams are in that room with us. Of 200-plus enterprise AI leaders running agents in live production, surveyed by TrueFoundry in March and April of 2026: 76% had no unified logging across all their models and agents. 56% had no centralized control layer. Half could not confirm that every tool endpoint their agents call is authenticated. And 61% had no

spend caps on agentic workflows. Now think about the ones who answered yes. Much of that is a veneer of control at best.

Read that last stat twice. Six in ten shipped software that can call paid APIs in a loop, with no ceiling. We were in that six in ten, and we found out live, in front of Finance.

Gartner expects more than 40% of agentic AI projects to be canceled by the end of 2027. That is a finance number wearing a technology costume. Projects die when nobody can answer four questions: what did it cost, who approved it, what did it touch, and can you prove it. Without those answers, a project does not survive its first budget review. Or its first incident, whichever arrives sooner.

The regulatory clock started too. The EU AI Act's enforcement framework and transparency obligations came into application on August 2, 2026. The AI Omnibus package, in force since July, moved the high-risk Annex III deadlines to December 2027. That bought time. The obligations live right now are the ones about explaining what your systems do.

All that, and you have to know that long before this is about cost, it is about visibility and control. You cannot manage, maintain, monitor, or measure what you cannot see.

So let's narrow the question to make this easy. What has to be in place before prompts or agents touch production, and how does governance make shipping faster instead of slower?

Four things. A gateway, runtime security, observability, and inference classification.

They are laid out here in the order that makes them easiest to explain. The order to buy them in is nuanced, and that is the last section of this paper.

Cornerstone 1: The Gateway

What it is: a single control point that proxies all of your AI traffic, regardless of where it comes from, who is using it, or what model supports the workflow.

It comes first because it is the only one of the four that creates a chokepoint. Without it, you end up chasing every user around the halls of the organization, hoping to understand where they are using AI and how. You can write the best logging standard in the world, then realize months later that it only governs the teams that choose to follow it.

The objection arrives immediately, and it is fair. You have just built a single point of failure. But, you have **not**... if you build and deploy it right. A gateway is a layer: multiple instances, in multiple places, scaled horizontally, under one management plane. You centralize the control and distribute the infrastructure. If a vendor's answer to this involves one deployment you cannot scale out, you do not need what they are selling.

The failure mode is compounding and silent. Teams hold their own provider keys. Nobody rotates a credential; nobody knows what breaks when they do. Spend lands on a corporate card statement under "software" and isn't attributed to anything. Then someone senior asks which models the company runs and for what, and the answer takes three weeks (if you're lucky) to assemble and comes back incomplete.

Your developers call that autonomy. Your auditor has a different word for it.

The trigger is earlier than most teams think. Stand the gateway up before the second use case. Retrofitting a front door onto a dozen direct integrations is slow, and more to the point it is political: every team that built its own path has to give something up, and each of them will want to know why they are being punished for shipping.

The most useful question in a gateway evaluation is one line. Can a developer bypass it? If the answer involves the word "policy," you are buying a suggestion. Make the gateway the path of least resistance and it becomes a control point. Anything else and your coverage is voluntary.

And the exception is the whole game. The first time you wave through a direct-to-model integration just this once, the policy stops being uniform, and the exceptions never stop at one. Route the urgent case through the plane too.

The checklist below cannot tell you this part: the gateway has to be deployable where your models already run. If a third of your inference sits on-prem behind a sovereignty rule and the gateway is SaaS-only, you have built a chokepoint for the traffic that was easy and you have a blind spot for the rest. Coverage is a deployment question before it is a feature question.

And nothing routes itself. Before you stand it up, find what is already running. Expense records, proxy logs, SaaS admin consoles, and asking teams directly. Then move each one behind the gateway on a date, with a name against it. A gateway that waits for traffic to arrive collects the traffic of people who had no other option.

One thing before the list. Not everyone needs to buy this. If you are running a handful of use cases against a single hosted provider, a thin proxy with authentication, logging, and a cost read does the job, and you should not be shopping for a platform. The requirements below start to matter when you route to more than one provider, when more than one team is building, or when someone outside your team has to be able to answer what AI costs.

Your requirements list will be longer than this one and specific to your environment. What follows is the shorter set: the places where products differ once you get real and start testing.

What to demand:

Requirement	How to test it
Ask this before anything else: bypass detection	Can a developer route around it? Then the harder question: what share of your model traffic is not passing through the gateway today, and how would the product know? Most cannot answer the second one.
Unified API across providers and self-hosted models	Ask for the provider list and the model-update cadence. New frontier models should land in days, not quarters.
Key rotation and revocation without downtime	Rotate a live key in the POC. Watch what breaks.
Identity integration: SSO, RBAC, SCIM 2.0 provisioning	Revoke a user's access and confirm their credentials die everywhere, automatically.
Budget enforcement: hard limits and threshold alerting	Set a hard cap in the POC and try to exceed it. Then ask what fires at 80%. Without the alert, you find out you hit the cap when something breaks.
Latency overhead, measured separately from end-to-end time to first token	Every vendor publishes an overhead figure. Get theirs in writing, then verify it yourself at p95 and p99, under load, with streaming enabled. If you cannot reproduce their number, make them show you exactly what they measured and on what.
Data residency and deployment flexibility: SaaS, VPC, on-prem, air-gapped	Confirm log residency separately from inference residency. They are often governed by different initiatives, teams, or contracts.
Failover and routing across providers	Kill a provider in the POC. Measure time to recovery.
The control layer stays yours to configure	Can you see and change routing rules, failover logic, and policy yourself, or are they set inside the product? Change one routing rule in the POC without a support ticket. A control plane you cannot inspect is somebody else's control plane.

Cornerstone 2: Runtime Security

What it is: real-time enforcement in the request and response path, before the damage.

Your security program was built around software that does what it was written to do. LLM systems drop that guarantee. Agentic systems drop it further, because the system now picks its own actions and calls tools nobody authorized for that specific step. Many of those agents are spun up for a single task and torn down when it finishes, so the thing that took the action may be gone before anyone looks for it.

A chatbot has one risk surface: the text it returns. Grade the output and most of it is covered. An agent wraps that model in an execution loop, so it calls tools, keeps state across steps, and produces effects in your systems. The question is no longer whether it said something harmful. It is whether it did something harmful, and whether you could have stopped it.

Here is the number that should stop a CISO cold. 78% of these teams run six or more active tool or MCP endpoints. 51% cannot confirm all of them are authenticated.

Cannot confirm. The inventory does not exist.

That is a lateral movement path with a natural language interface bolted to the front.

The useful goal is not a model that can't be fooled. Assume it will be. Build so that when it happens, nothing important is within reach. Every requirement below is a version of that same move, shortening the distance a bad instruction can travel before something stops it.

Before any of this becomes a purchase, inventory what you already run. Four surfaces need securing and you own controls for most of them. The model: where it came from, what is embedded in it, whether its provenance is known. The access, in both directions: who can reach the AI, and what the AI can reach. The data in prompts, outputs, logs, and anything the agent retrieves or remembers. And the application: the same segmentation, hosting, and integrity controls you apply to any system. Map each control objective to something you operate today or to something you have to add. That map is the input to the budget conversation, and it usually turns up coverage you already paid for.

And do not write the whole rulebook before you can enforce a line of it. Months spent building a requirements library gets you a rulebook for the system you had when you started. That is the same failure from the opening of this paper, working in a different department.

One note on architecture, because this gets argued about. There are two enforcement points here. A gateway brokers and logs the calls. A runtime firewall inspects what moves through it. Vendors increasingly sell both as one purchase, but that doesn't mean it is the right buy. Write the requirements separately, so the decision stays yours. Otherwise your security vendor picks your gateway, or your gateway vendor picks your security.

What to demand:

Requirement	How to test it
Inspection in the request and response path, with the classes named	Ask for the list: injection, sensitive-data leakage, agent misuse, malicious code, unsafe URLs, off-topic use. Then bring your own red team prompts. Vendor demos use vendor test cases.
PII detection and redaction, inbound and outbound	Test with your data shapes, including the messy or consequential ones.
Policy enforcement before tool invocation, not after	This is the agentic-specific one. Ask to see a blocked tool call in a trace.
Centralized tool and MCP registry with an approval workflow	If tools can be registered without approval, you're not controlling anything. Then ask what was inspected inside the MCP server before it was approved, which is a different question.

Requirement	How to test it
Per-agent and per-tool authorization, scoped narrowly	An agent should hold the narrowest credential that lets it finish the job. Then test the hops: do those credentials flow through the gateway, and does anything watch for drift between the gateway and the agent, or between one agent and the next?
Agent identity with a lifecycle, not a shared credential	Can each agent get a unique, verifiable identity with short-lived task-scoped credentials, provisioned and de-provisioned automatically? Then the question that separates products: when one agent calls another, does the call carry that identity, or does the second agent inherit trust?
Guardrail extensibility	Every enterprise has that one rule nobody else has. Confirm you can add it without a vendor release.
Irreversible actions treated differently from reversible ones	Ask how the product tells a tool call you can undo from one you cannot, and what it does differently for the second. If both take the same approval path, that decision is being made by whoever wrote the prompt.
Controls mapped to a named framework, not a vendor taxonomy	Ask which controls map to NIST AI RMF, ISO/IEC 42001, and the OWASP Top 10s for LLM and Agentic Applications. A vendor's private risk categories cannot be audited against anything.
Independent enforcement, swappable in both directions	Can you replace the gateway without replacing the guardrail layer, and the reverse? Does enforcement run in the path on its own, or does it need that vendor's routing? Then price the exit: the bundle against the two parts bought separately.

Cornerstone 3: Observability

What it is: a unified, queryable record of every model and agent interaction, with enough structure to answer questions you have not thought of yet.

This is where most programs are weakest, and the weakness hides because the logs do exist. They sit in multiple systems, in multiple formats, with no shared identifier. So "what did this agent do last Tuesday" turns into someone reconstructing it by hand, and likely backing into an answer.

You already own observability. Every enterprise does, and that is the problem with the word. It has been in the building longer than AI has, usually owned by a team that inherited it, and in plenty of organizations it was never done well on the systems it was built for: dashboards nobody opens, alerts everyone mutes, traces that stop at the service boundary. It has become part of the noise. Now that same practice is pointed at LLM systems. Infrastructure health tells you almost nothing there. What you need is the reasoning path: which tool the agent picked, and what it had retrieved when it picked it. Your existing stack was not built to hold that, and the team that runs it has not been asked to.

The 76% without fully unified logging is the headline. The number underneath it matters more once agents are involved. Fewer than half have full step-by-step tracing. Another 49% capture the output and nothing about the sequence that produced it. They can see that an

agent ran. What happened inside the run is dark. For one model call, survivable. For a twelve-step agent that called four tools and retried twice, that gap is the difference between a root cause and a shrug emoji.

Money hides here too. 80% of AI production cost sits in orchestration, tool calls, and retries. Inference is 15 to 20% of the bill. If your cost visibility stops at token spend per model, you are watching the small number. And 41% find their operating costs only after deployment, which means the business case got approved against a fraction of the run rate.

Here is what makes this cornerstone different from the first two. You find out about a gateway failure in minutes, and about a blocked tool call the moment someone complains. Degradation is silent. A provider updates a model underneath you, or someone edits a prompt, and quality slides without throwing a single error. Nothing raises a hand. The only thing that surfaces it is a record you were already keeping before anything went wrong. That is why the instrumentation has to exist before you have a reason to want it.

This also does not have to start as a purchase. Passive monitoring at the gateway costs close to nothing: capture the calls, watch the usage, keep the record. If spend runs away or an incident forces the question, that same stream becomes metering, budgets, and enforcement without new plumbing. The reverse doesn't work. Retrofitting measurement across a few thousand existing builds almost never happens.

One more thing, and it is where most programs stop. Observability gets implemented as an obligation. Somebody asked for logs, logs exist, the box is checked, and nobody opens the dashboard again until an incident. That was survivable when the systems were deterministic. It is not now, because the record is the only place an improvement loop can start. Nobody sends you an alert when a workload has been overpaying for a frontier model all year, or when one agent retries three times to do what a smaller model does once. You go looking on a schedule, or you never find it. Nothing pages you about a system that is working expensively.

The test that separates products here is to hand the vendor an incident. "An agent made 400 calls to an external API in an hour last Thursday. Show me why." Then count the screens it takes.

What to demand:

Requirement	How to test it
Every call captured with full metadata: tokens, latency, user identity, model, cost	Export a day of logs. Try to answer an operational question with them.
Step-level tracing across LLM calls and tool calls, correlated by one trace ID	Ask for a trace of a multi-step agent run that includes a retry.

Requirement	How to test it
Cost attribution by team, application, and agent task, with rollup	Chargeback is the feature that makes finance an enabler.
OpenTelemetry compliance and export to your existing stack	You already own Datadog, Grafana, or Splunk. Do not buy a second pane of glass.
Real-time visibility, ahead of billing reconciliation	Only 24% of these organizations have full real-time token-level tracking. Be in that quartile.
Log residency and retention controls	Logs are data. They fall under the same governance and rules as the data in them.
What the agent retrieved, not just what it called	For a knowledge-grounded agent, ask to see the passages that reached the model for a specific answer, and the ranking that selected them. Tool calls tell you what it did. Retrieval tells you what it knew.
Access control on what the logs contain, not just who sees the dashboard	Prompts and responses hold whatever your users typed. Ask who can read the body of a logged call, how that is scoped, and whether it is separable from access to the metrics.

Cornerstone 4: Inference Classification

What it is: classification of what your AI traffic contains, on both sides of the call. What people are asking for, and whether what came back was any good.

The first three cornerstones establish that a thing happened, that it was permitted, what it cost, and what it touched. They give you a record you can query. What they cannot tell you is what the work was for, or whether it was any good. Reading that out of a million interactions is a classification problem, and it is the least mature layer in the category.

I should be direct about where this one sits, because I have been on the wrong end of it. We implemented the first three. A gateway, the security controls, the logging. We never built this one, and it was not for lack of understanding that we needed it. We knew we needed it badly. In a rush of competing priorities, this one slipped through the cracks. It wasn't neglect. We chose against pointing our AI engineering capacity at it, and there wasn't a commercially available tool at the time to buy instead.

We chose poorly.

This is also the one that gets waved at hardest in a sales cycle. Everyone has a dashboard. Fewer have a mechanism that samples production outputs, scores them against a rubric that reflects your business, and puts the failures in front of a human on a schedule. Fewer still close the loop, so what the inspection finds becomes a guardrail or a routing rule instead of a slide titled "Learnings."

The objection is that you already do this. You have evals in the build pipeline, a benchmark suite, a set of test prompts that run before a release. Keep them. They tell you about the model. They tell you nothing about your traffic, because your traffic is what your users typed, using your data, in untested waters. Pre-deployment evaluation is a quality gate. This is a quality signal. They are not interchangeable.

Be skeptical of every vendor here, including the ones I work with. Gateway, security, and observability are commoditizing. Classification is thin across the platforms and crowded with point solutions. If a vendor tells you their evals story is finished, ask to see the loop.

The line that separates a product from a dashboard here is short. A finding that changes nothing is a report. If the findings pile up faster than anyone can act on them, you have bought a very good view of your quality problem.

You can also start this without buying anything. Pull a hundred production outputs a week, have a model score them against a rubric you write yourself, and put the failures in front of the person who knows what good looks like. Sorting text against written criteria is one of the things models are genuinely good at, which is why the first version of this costs an afternoon. It will tell you more about your quality than any dashboard you can purchase this quarter, and it will tell you what to look for when you do go shopping.

What to demand:

Requirement	How to test it
Multi-turn evaluation, not single-response scoring	Agents fail across conversations instead of answers. Ask to see a run where the model held for six turns and broke on the seventh. If every eval in the demo is one prompt and one response, they are testing a chatbot and you are buying for an agent. Bring unscripted conversations, not just your known-bad prompts.
Output sampling and classification on production traffic, beyond pre-deployment evals	Offline benchmarks tell you about the model. You need to know about your traffic.
Drift and quality regression detection tied to model version changes	Providers update models under you. You should find out from your tooling, not your users. Unscripted sampling is what catches this. A fixed test suite only ever tells you about the failures you already imagined.
A path from finding to enforcement	When classification catches a failure class, what changes automatically? Ask whether the scorer that caught it in evaluation is the same one running in production, or whether you are maintaining two.
Human review workflow with sane queueing	Someone has to look. Make the tooling respect their time.
Custom rubrics that encode your domain	Generic toxicity scoring is table stakes and nearly useless for a claims process or a trading desk.
Feedback capture from end users, joined to the trace	The cheapest signal you have, and the most commonly discarded.

Requirement	How to test it
A place to exercise the agent end-to-end without touching production	Can you run the full agent, tool calls included, against replicas of your systems? Evaluating the model in isolation tells you nothing about what the agent does when a tool returns something unexpected.
Findings that become test and training data, not a report	When an evaluation surfaces a failure class, ask what artifact comes out of it. A slide is a report. A regression test that runs on every prompt change, plus data that corrects the behavior, is a loop.
Classifiers you can retrain on your own taxonomy	Classification runs over everything, so the economics only work with a small model rather than a frontier one. Some vendors ship models already tuned for an industry or a task, which is a genuine head start. Ask what it was trained on, and whether you can retrain it on your categories. A confident label in somebody else's vocabulary is worse than no label.

As you focus on the cornerstones of your foundation strategy, this is usually the one to fund last, and always the one to contract for loosest. I meet vendors every week doing things I had not seen a month earlier, and this is the cornerstone most of them show up in. What you are shopping for today may turn out to be a feature of something that has not shipped yet. Buy for this year, and keep the exit cheap.

What the four cornerstones build

This part rarely makes it into the governance conversation, and it is the reason to sequence the work on purpose instead of buying it in a panic after an incident.

Each cornerstone produces a record. The gateway knows every model call your company makes and who made it. Runtime security knows what got attempted and what got refused. Observability knows the sequence, the cost, and the latency of every step. Classification knows which outputs held up, and what inputs were used to create them.

Individually, these all look like compliance artifacts. Step back and they become a structured, queryable account of how your company uses AI. Which teams. For what. At what cost. With what results.

Call it system context.

System context compounds and there is no easy button, because it is a record of your operations and your institutional memory. Nobody can sell it to you. Once it exists, your routing and build-versus-buy calls come off your own traffic: you know which workloads need a frontier model and which have been overpaying for one for a year. You walk into a provider renewal with per-workload consumption data and demonstrated portability, which is a different conversation than walking in holding an invoice. Capacity planning survives contact with agents, because you measured the amplification. 83% of these teams see token

amplification in agentic workflows, running 3 to 15 times the tokens they expected. And when the auditor arrives, the answer takes an afternoon.

Here's the part people miss. Your competitor is shopping the same vendor list you are, and the feature grids converge fast enough that procurement ends up flipping a coin. Nobody sells the other thing, though. Eighteen months of your own traffic, priced out by workload, with every routing decision you made and what it cost you. That only accrues to whoever bothered to instrument early.

Then the half that matters more. System context is one side of what your company knows. The other side is tacit: the judgment in the heads of people who have run your operation for fifteen years and know which exceptions are normal, which customers get handled differently, why the obvious optimization is a trap, and when to ignore all three. None of it is written down. It is why generic AI deployments produce generic results.

The companies that pull ahead over the next three years will have both, wired together. System context tells you what happened at scale. Human context tells you what it meant. Agents running on the first alone will keep making confidently wrong calls, and everyone will blame the model.

You cannot collect the tacit half until the instrumented half exists, because there is nothing to attach it to. Which is why this is an investment in an asset. The compliance case pays for it. The context case is why you should want it.

Sequencing: what to do in what order

If you are starting from scattered keys and no control layer, the order matters. Two of these are dependencies and two are a judgment call.

1. **Gateway first.** It is the chokepoint. Everything else enforces through it. In the first ninety days, optimize for coverage over features. A gateway that 60% of your traffic bypasses is worth less than a simpler one carrying all of it.
2. **Observability second.** It is nearly free once the gateway is in place, it converts every argument after it from opinion to data, and everything downstream reads the record it produces.

Those two are the dependency. After that it forks. Runtime security and classification don't need each other, they just both need the record. So what you do third comes down to who is asking you for it, and how loudly.

Runtime security third, if it's your CISO or your data privacy lead doing the asking. Tighten from a permissive baseline once you can see what normal looks like. Lock it down before you have observability and you are guessing at your own traffic, which is how you

break something that was working in week two and spend the next month earning back permission to touch anything. And if they will not get off your back in the meantime, give them the grail journal.

Classification third, if it's Finance or the business wanting to know what all of this is for. It reads the record the first two produce, and it's the only one of the four that answers the question Finance asks right after the one about cost.

Whichever you take, the other is fourth. Ours came from Finance, and we funded the CISO's answer first anyway. An incident has a date on it.

I have watched teams skip the first two entirely. They buy an evaluation product first, because it demos well and because bad output is the failure that stings, then discover there is no unified pipe to plug it into. You cannot inspect traffic you do not route.

The one-page version

Take this into your next vendor conversation.

Cornerstone	The question that separates products
Gateway	Can a developer bypass it? Can I rotate a key right now without downtime?
Runtime Security	Can you block a tool call before it executes, and show me the trace?
Observability	Here is an incident. Show me why it happened, in one screen
Classification	Can you tell me what people are using this for? And when you find a failure class, what changes automatically?

And one question for all four. What happens to this data if I leave? A governance layer holding your operating history hostage has quietly become the biggest lock-in in your stack. Put export format and portability in the contract, while you still have something to trade.

You cannot govern what you cannot see. Start with the pipe.

Eric Pace is an enterprise technology operator and advisor. He spent most of his career implementing large-scale platform programs for global enterprises, then moved to the buying side where he served as Chief AI Officer at a Fortune 500-sized communications operator. He now advises AI infrastructure companies on go-to-market and enterprise readiness. He is a paid advisor to TrueFoundry.

Sources

- TrueFoundry enterprise AI survey, 200+ leaders with production agent deployments, fielded March to April 2026: [Business Wire release](#) and [survey landing page](#)
- Gartner, "Over 40% of Agentic AI Projects Will Be Canceled by End of 2027," 25 June 2025: [press release](#)
- EU AI Act application timeline and the AI Omnibus package: [European Commission](#)
- The evaluation criteria in this paper are mine. Where they overlap with vendor-published buyer guidance I cross-checked against it rather than copying from it: [How to Choose an AI Gateway](#) and [Enterprise AI Platform RFP Questions](#). The criteria you will not find in any vendor's published guidance, including bypass detection, agent identity lifecycle, irreversible action handling, retrieval visibility, export and portability, and independent enforcement, come from running this inside a Fortune 500.