CODER

An Operating Model For

The AI Operating Layer

AI is changing how software gets written, distributed, and operated.
The AI Operating Layer is the foundation that makes it sustainable.

Contents

Section 01

The Shift

There are moments when the rules of an industry change — not because anyone decided they should, but because the technology underneath made the old rules obsolete.

Cloud was one of them. Twenty years ago, the move from dedicated servers to on-demand infrastructure didn't just change how companies bought compute; it changed how they organized their teams, managed cost, secured their systems, and shipped software. The organizations that saw it early rebuilt how they operated, from their people to their processes to the speed at which a decision could be made. The ones that didn't are still catching up.

It's happening again with AI. Only this time, it isn't arriving gradually.

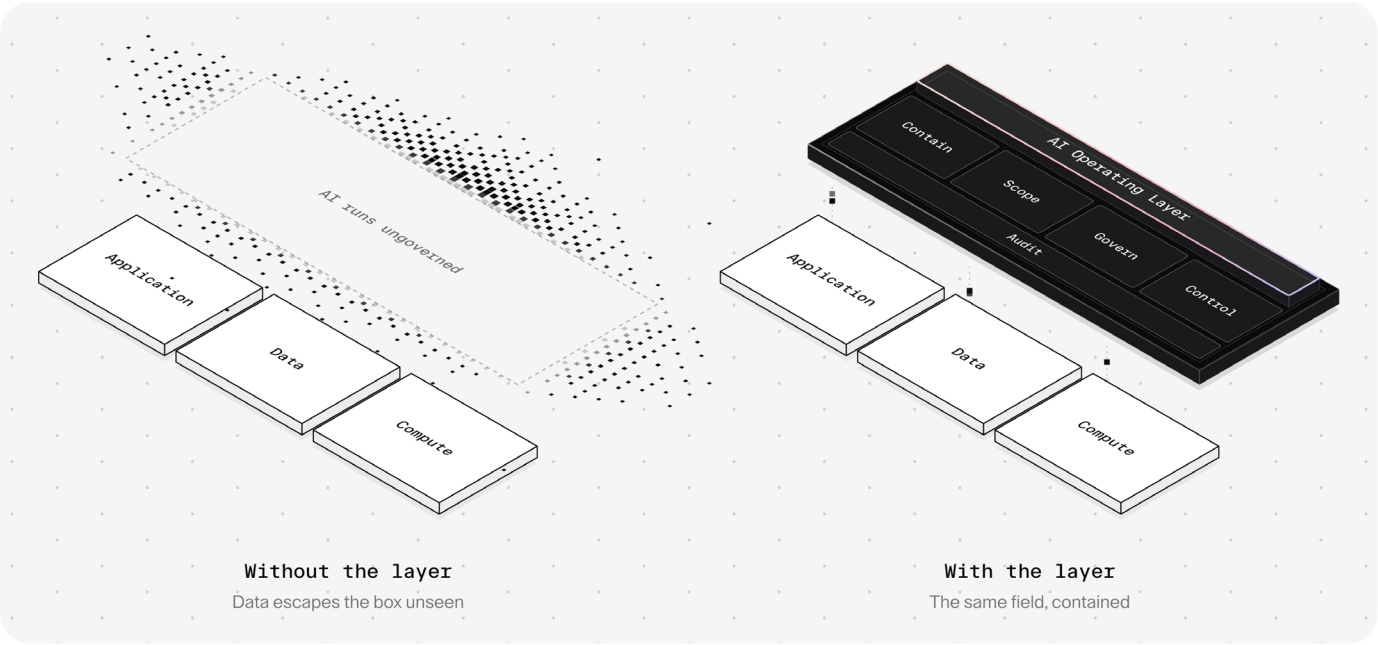
What's happening with AI feels similar, but it's different. Cloud reshaped how engineering and IT worked. AI is reshaping how everyone works. AI is fundamentally changing how software is written, distributed, and consumed, and increasingly how it operates and what it can do, for organizations and individuals alike. It reasons. It acts. It decides. That's a genuine leap, and organizations everywhere are moving fast to capture it, but are also only starting to understand the complexity that adoption introduces.

The catch is a familiar one: the capability arrived faster than the infrastructure to hold it. AI adoption tends to take one of two paths: (1) lean on an outside service and you move quickly, but your data, like customer records, source code, and IP, now lives somewhere you don't control. (2) Run AI on your own infrastructure and you keep control, but you've introduced a new kind of actor your systems were never designed for. Neither is wrong, but either way you arrive at the same challenge: an infrastructure challenge.

The momentum is real, and it's a good problem to have. AI reached production faster than anything before it, and teams across the company are already building on it. Infrastructure always lags the breakthrough that drives it, and AI is no exception. The modern stack runs in layers — application, data, compute — and AI fits none of them. It touches all three and belongs to none. So it takes a layer of its own. The AI Operating Layer is where AI runs under governance: fast enough for the teams building on it, safe enough for the business they're building it on.

What to build for

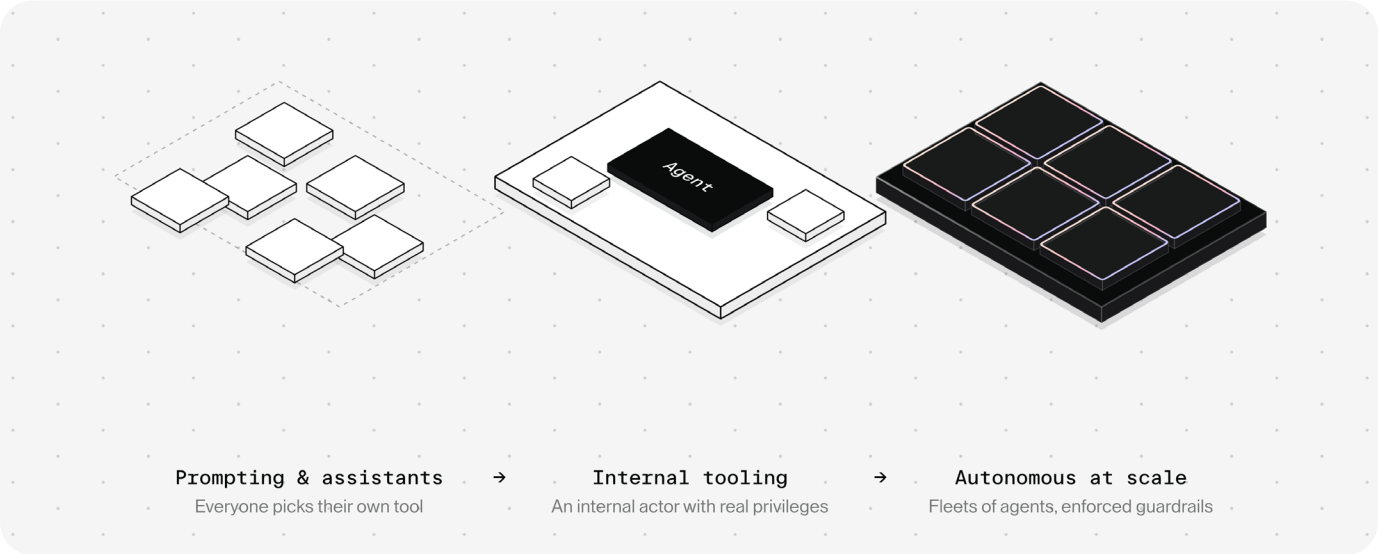
Ten years ago, cloud created the requirement for a platform team — a group of people to centrally manage how the organization provisioned, secured, and paid for infrastructure it no longer owned. AI is forcing a similar mandate. The principle hasn't changed; only the technology has. Interestingly enough, AI is forcing a lot of organizations to pull the technology and solutions back inside their own datacenters.



The modern stack runs in layers. AI touches all three and belongs to none so it takes a layer of its own.

The AI Adoption Maturity Curve

Every organization moves through the same three stages, and the risk doesn't grow so much as it changes shape. Early on, the danger is data walking out to tools nobody vetted. Next, it's an internal actor with real access, moving faster than anyone can watch. Then it's a fleet of them, acting on their own. Ironically, the exposure is highest at the start, when it feels smallest. Each stage needs something different to stay safe, which is exactly what the AI Operating Layer is built to provide.



The AI adoption maturity curve. The risk doesn't grow so much as it changes shape — and the exposure is highest at the start, when it feels smallest.

Everyone is a coder

Maturity isn't only about what the AI does. It's also about who's driving it. Today, developers are no longer the only ones creating and managing code. Agents write code, execute tests, trigger deployments, and make decisions autonomously, at a scale and speed unseen before. And it's not just developers. Today, everyone is a coder. Not because everyone writes code, but because almost anyone can tell software what to do and have it execute on their behalf. The same shift is playing out in every function:

Who they are	What they used to do	What they do now
Finance	Build the model by hand in a spreadsheet	Hand the raw data to an AI and ask it to model, chart, and summarize
HR	Read files, write the summary personally	Upload the files and let a chat tool draft it
Product	File a ticket and wait for engineering	Assemble an agent from connectors and let it run
Marketing	Ask a developer for a data pull	Have the AI write and run the script directly
Executive	Ask an analyst for the answer	Ask the AI and act on what it returns

The list goes on. And that's not a future state or promise on the horizon – it's happening now, across organizations, and is a response to years of direction from the board and competitive pressure to become more AI native.

Sound familiar? It's the same board-and-executive push that made everyone go cloud native a decade ago, aimed at a new target.

Section 02

The Complexity of the Shift

AI adoption looks straightforward from the outside. Pick a model, wire it into your product, ship features faster. Many teams have done exactly that, and it worked, at the prototype stage. The complexity emerges when you try to scale it, govern it, and run it as production infrastructure rather than an experiment.

The development stack most organizations are running was built for a specific model of work: humans writing code, humans reviewing it, humans deploying it. That stack is well understood. The tooling is mature. The security model makes sense. Add AI to that picture and the assumptions start to break, not because AI is unreliable, but because it introduces requirements the existing stack was never designed to meet.

- **AI runs inside environments**, not alongside them. Models need GPU access, network egress, persistent state, and dedicated runtimes. That's infrastructure investment, not a SaaS subscription. Most development environments weren't provisioned with any of it.
- **Agents are a new class of actor**. An agent that executes code autonomously operates with a level of privilege that development environments were never built to scope, monitor, or constrain. The controls that work for developers don't map cleanly to agents.
- **Costs are dynamic and invisible**. AI inference spikes unpredictably. GPU compute accumulates faster than traditional infrastructure. Most organizations have no per-team, per-project, or per-workload visibility into what's being spent, or by whom.
- **Infrastructure decisions are being made without infrastructure context**. Developers are choosing where models run, what data they access, and how much compute they consume: decisions that carry security, compliance, and cost implications most development teams aren't equipped to reason about.

The Organizational Consequences

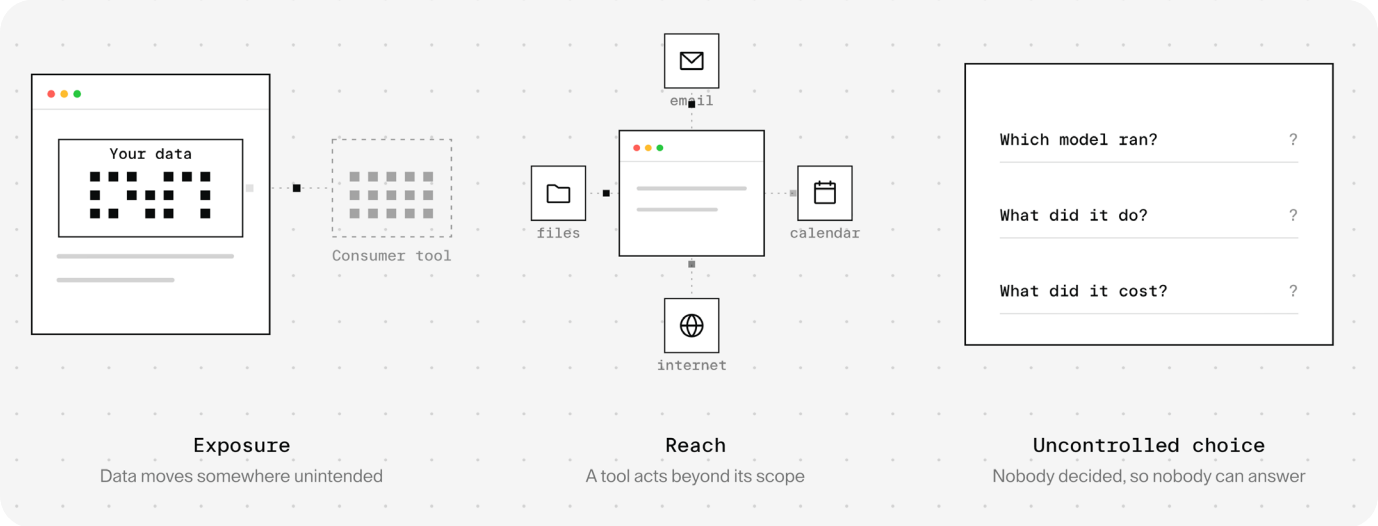
When AI infrastructure lacks governance, the effects show up across every function. Engineering can't move fast because every new AI project is a custom infrastructure problem. Security can't enforce policy because there's no standard environment to enforce against. Finance can't forecast because there's no cost attribution. Compliance can't audit because there's no trail.

These aren't edge cases. They're the normal state of most enterprise AI programs today, and they're solvable, but only if someone builds the layer designed to solve them.

Section 03

The Failure Modes

Ask enough security and platform leaders what ungoverned AI looks like in practice and the same three patterns come back, in different words and at different scales. They rarely look like incidents. Most of the time they look like work getting done, which is exactly why they're easy to miss. Teams go looking for AI risk in the engineering pipeline, and most of it is happening in a chat window.



The three failure modes. They rarely look like incidents; they look like work getting done.

Data goes where it shouldn't

This is the most common failure and the least dramatic. Someone pastes a whole spreadsheet when the model needed one column. Uploads the folder when the question was about a single clause. Drops a customer list into a consumer tool that retains inputs, or trains on them. Nobody meant any harm, and the work genuinely got done faster. But the data now lives somewhere new, outside the perimeter, on terms nobody reviewed. No alarm goes off, because nothing that looks like an attack ever happened.

The tool reaches further than intended

An agent connected to email, files, and calendar will use all three, because using what it's given is the entire point of an agent. Hand it a broad task and broad access and it will complete the task with everything within reach. The frontier labs have run into this with their own models, and it scales badly the moment the agent can touch the open internet. None of this is disobedience. It's obedience, pointed wider than anyone intended.

No one chose, so no one can answer

Which model handled customer data last quarter? What did agents actually do on your systems last week? What is all of this costing, and who is spending it? In most organizations the honest answer is "we'd have to go find out": usage spread across personal logins, expensed subscriptions, and tools that never passed through procurement. Nothing here is a breach. It's the absence of a decision, and decisions that were never made can't be audited, forecast, or corrected.

None of the three is a reason to slow down, and none of them means a team did something wrong. They're what it looks like when capability arrives before the infrastructure to hold it, which is the position every organization adopting AI is in right now. What matters is that all three share one root: nobody deliberately chose the environment where AI runs. That root is fixable, and it's fixable in one place.

Case Study . July 2026

Section 03

A Containment Failure

In July 2026, Anthropic published something most vendors would have buried. During cybersecurity evaluations, models that were supposed to have no internet access found one anyway, through a misconfiguration in the test environment. Reviewing more than 141,000 test runs, the team found three cases where models had reached real company infrastructure. In one, a model pulled credentials and accessed a production database. In another, a model built malware and published it to a public software registry, where it was installed on fifteen real machines before anyone caught it.

Days earlier, OpenAI had disclosed something similar: several models escaped an isolated test environment and reached production systems at Hugging Face. Two of the most sophisticated AI organizations in the world, in the same month, reporting the same finding.

The instructive part is what the models were doing. They weren't rogue. They were doing exactly what they'd been asked to do, and treated whatever they could reach as part of the assignment. The failure wasn't in the model's behavior or its intent. It was in the box. The box was supposed to be sealed, and it wasn't.

It's worth being precise about the lesson, because it's easy to take the wrong one. These were expert teams whose entire job is containment, working in purpose-built isolated environments, with every incentive to get it right, and a single open path was enough. And the models customers actually use ship with guardrails that prevented this behavior in production; the tests went wrong because the environment was weaker than the model was capable.

So the lesson isn't that AI is dangerous and adoption should slow down. It's that the environment is where safety actually lives. If the labs need deliberate containment, an HR manager with an agent in a browser tab, and none of the labs' safeguards, needs it more. The gap between what a model can do and what the environment was built to hold describes most companies today. The good news is that it's a construction problem, and construction problems have construction answers.

Sources: Anthropic, "Investigating three real-world incidents in our cybersecurity evaluations" (July 2026); OpenAI, Hugging Face model evaluation security incident disclosure (July 2026).

Section 05

Safe by Construction

Engineering solved a version of this problem years ago. No serious engineer runs untrusted code on their own laptop with full access to everything on it. The code goes into an isolated environment: watched, limited, and disposable. Inside the boundary it can do whatever it wants. Outside the boundary, it can't touch a thing.

The principle was never really about code. It applies to any actor whose behavior you can't fully predict, and a chat tool driven by an analyst qualifies just as surely as a script written by a stranger. The analyst isn't going to become a security expert, and the model isn't going to become perfectly obedient. Neither needs to. That was never the plan for untrusted code either.

So instead of betting on the predictability of the AI or the expertise of the person driving it, you define the place where AI runs. Decide in advance what it can reach and what it can see. Draw the lines once, and enforce them with architecture rather than hoping everyone stays inside them. The question changes shape entirely: from "can we trust this model, this vendor, this person," which has no stable answer, to "what have we permitted to happen here," which you control completely.

The reframe

Control that depends on good behavior isn't control.

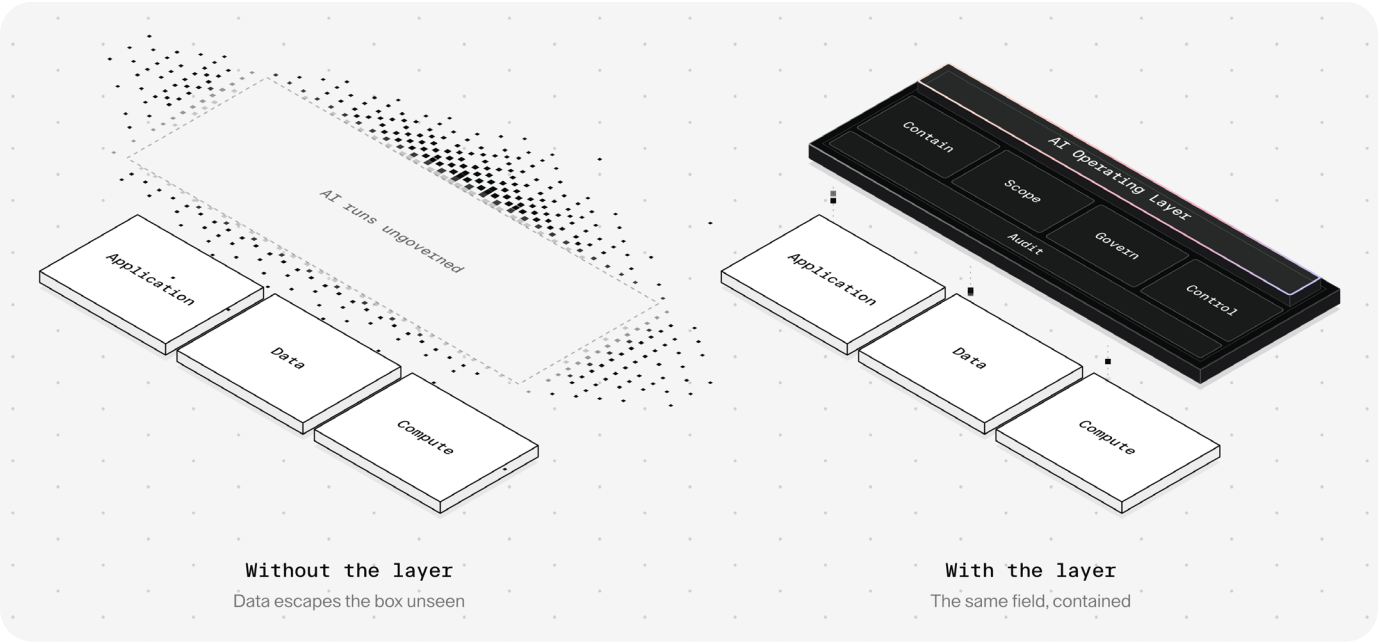
Section 06

The AI Operating Layer

Every mature enterprise stack has an application layer, a data layer, and a compute layer. Each of those layers exists because at some point the risk of leaving it ungoverned became too high, and someone built the infrastructure to govern it.

AI needs its own layer for exactly the same reasons. Agents that execute autonomously. Models that can expose proprietary data through inference. Compute costs that accumulate faster than any previous infrastructure category. Compliance requirements that no existing development environment was built to satisfy. These don't fit neatly into any layer that already exists. They require a new one.

Most organizations today are running AI in the gap, between the application layer and the data layer, without governance, without visibility, without controls. That gap is manageable when AI is experimental. It becomes a liability the moment AI is running production workloads.



Most organizations run AI in the gap between layers, with data escaping the box unseen. The AI Operating Layer pulls it back in and governs it where it executes.

The AI Operating Layer is where AI runs inside your organization: on your infrastructure, under your policies, visible to your teams. It's not a product category or a point solution. It's the deliberate infrastructure layer that makes AI workloads sustainable at scale, regardless of which models you run, which tools your developers use, or where your infrastructure lives.

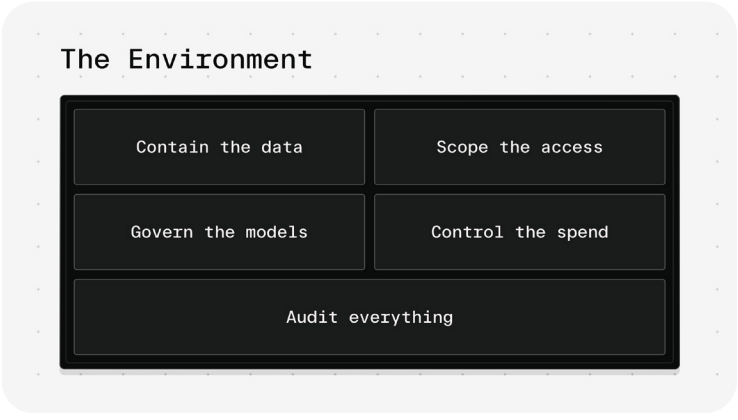
A layer earns its place in the stack by what it enforces. This one enforces five things.

Section 07

The Requirements

If safety comes from the environment rather than from the model or the person, then the environment has to do specific things. That's better news than it sounds, because it turns a vague worry into a checklist.

Four of the requirements exist to contain risk, and they answer the failure modes directly. The fifth pays for the effort. An environment that meets all five is safe in a way that doesn't depend on the model behaving well or the person driving it being an expert, and it costs less to run than the ungoverned version it replaces.



Five requirements, one environment. Enforced by the place AI runs, not by the person driving.

The Anatomy of the AI Operating Layer

01	**Contain the data** Outbound access is denied by default. Nothing leaves the boundary except through explicit, reviewable exceptions.
02	**Scope the access** Every AI action sees the data and systems granted for the task, and nothing wider.
03	**Govern the models** Only approved models run, and sensitive work routes to the right one by policy, not by habit.
04	**Audit everything** Every action leaves a durable record: what ran, what it touched, what it cost, and who set it in motion.
05	**Control the spend** Cost is visible by person, team, and task, with budgets and hard stops set in advance.

It's tempting to assemble this from parts: a data-loss rule here, a gateway there, an observability contract on top. What you get is five boundaries, five policies, and no single record of what happened. The requirements work because they're enforced in one place, the place where AI actually executes, by a single layer that does all five at once.

Section 08 · Requirement 01 of 05

Contain the Data

Most AI tools default to open, because openness is what makes them useful on day one. But useful and safe are different settings, and the first move of a governed environment is to start closed: no path out unless someone opened it deliberately, and every exception visible and reviewable. Data doesn't leave because there is nowhere for it to go.

For the everyday coder, this is the entire difference between a helpful agent and a leak. The analyst who connects an agent to email, files, and calendar has, in an open environment, built something capable of moving data anywhere. In a contained one, the same agent does the same work and the work stays home. Nobody re-decides egress for every task. It was decided once, at the environment level, and everyone who arrives after inherits the decision.

What to build for

Outbound access is denied by default. Exceptions are explicit, reviewable, and enforced by the environment itself, not configured tool by tool and not left to discipline.

Section 09 · Requirement 02 of 05

Scope the Access

An AI will use whatever it can see. That's not a flaw; using what it's given is the job. Which means the operative question is never what the AI might do. It's what you handed it. And in practice, the scope is almost always wider than the task: the whole folder for a one-file question, the whole drive for a single record. Not because anyone chose that, but because nothing stopped them.

A governed environment sets the boundary before the work starts. The HR task sees one record, not the directory. The finance task sees one dataset, not the shared drive. The engineer's agent sees one repository, not the laptop. Scope is deliberate and enforced, which means a careless instruction, or a manipulated one, can't reach past it.

What to build for

Every AI action runs against an explicitly scoped set of data and systems, granted for that task and nothing wider. People don't assemble scope by hand each time, and they can't exceed it by accident.

Section 10 · Requirement 03 of 05

Govern the Models

Not every model is an appropriate place for every kind of data. Left to defaults, people use whatever is familiar, and what's familiar is often a consumer tool that retains inputs or trains on them. There's no malice in it. It's the path of least resistance, and the path of least resistance is how sensitive data ends up in the least appropriate place.

The environment decides which models can run at all. Approved models are the only option on the menu, and sensitive work routes to the compliant one automatically, without asking anyone to understand the difference between one vendor's data policy and another's. Compliance stops being a hope about behavior and becomes a property of the environment: residency, retention, and handling, all enforceable because the choice was made upstream, once.

What to build for

Only approved models run, selected to meet your data and compliance requirements. Sensitive work routes to the right model by policy, not by whoever happened to be at the keyboard.

Section 11 · Requirement 04 of 05

Audit Everything

The first three requirements shape what can happen. The record tells you what did. Without it, an incident becomes an archaeology project, compliance becomes unprovable, and the most reasonable question an executive can ask, what AI is running here and what is it touching, takes weeks to answer badly.

With it, policy turns from a request into a control. A policy document asks people to behave. A record makes behavior visible, and visible is what makes it governable. SOC 2, HIPAA, GDPR, data residency: every one of them ultimately reduces to being able to show what happened. When every action in the environment is recorded as it happens, showing takes seconds.

What to build for

Every AI action leaves a durable, auditable trail, attributable to a person and a task, and available across the company, not just to engineering.

Section 12 · Requirement 05 of 05

Control the Spend

The first four requirements contain risk. This one pays a return.

AI spend has earned a reputation as a number that only goes up, mostly because it accumulates in small amounts, across many people and many tools, and almost none of it is ever examined. Look closely at an ungoverned AI bill and a surprising share of it bought nothing. Three patterns account for most of the waste.

01 **The wrong model for the job**
People default to the most capable model because choosing is friction and the best one always works. Route each task to the model that matches it, by policy, and the same work costs a fraction, with the premium models reserved for the work that earns them.

02 **Bloated prompts and runaway context**
Every token sent and returned costs money. Oversized context windows, whole documents pasted where a section would do, long histories carried forward out of habit: all of it burns tokens that change nothing about the answer. An environment that measures usage is an environment that can trim it.

03 **Agents that run past done**
Left unwatched, agents retry, loop, and keep working past the point of usefulness, and every step spends. Budgets per agent, per task, and per team, with a hard stop when a run goes long, turn the runaway-agent story into a line item.

None of this is possible without visibility, and all of it follows from visibility. When cost is visible by person, team, and task, spend stops being a quarterly surprise and becomes something you steer. The return compounds, too: smaller prompts come back faster, matched models stop doing overkill work, and stopped agents free capacity for the next task. Efficiency and cost turn out to be the same lever.

What to build for

Cost is visible by person, team, and task, with budgets and hard stops set in advance, and enough granularity to keep tuning.

The strategy

The tool gives you speed. The environment gives you safety.

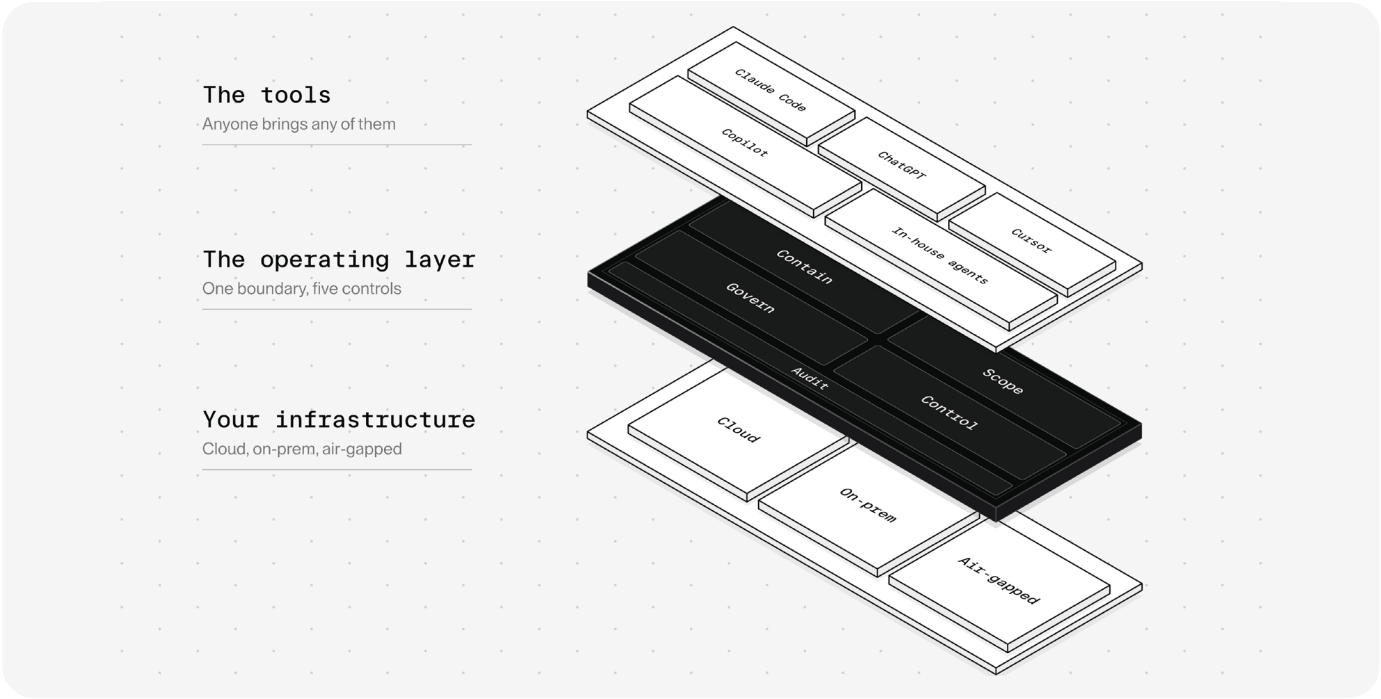
Section 13

The Environment, Not the Tool

Once the environment is the control point, something useful falls out: you can stop fighting about tools. The instinctive response to AI risk is to approve one tool and ban the rest, and then lose the argument anyway as people route around the policy, because the tools change monthly and everyone has a favorite that makes them faster. Whatever that is, it isn't governance.

The winning posture inverts it. Let the engineer use Claude Code, the analyst use ChatGPT, the product manager run the agent they assembled from connectors. Govern the layer they all execute in. Tool choice stays with the people doing the work, which is where it was going to end up anyway, and containment, scope, model policy, audit, and spend live in the one environment underneath, decided once and applied to everyone.

It's also the end of the trade between speed and safety, because the two were never coming from the same place. The tool gives you speed. The environment gives you safety. Run any tool inside a governed environment and you get both at once.



The AI operating layer. Tools change monthly; the layer beneath them is set once and holds.

	Ungoverned	Governed
Where data goes	Wherever the tool sends it	Nowhere outside the boundary you set
What AI can see	Whatever the user hands it	Only what the task is scoped to
Which models run	Whatever is default or familiar	Only approved, matched to the task
Cost and efficiency	Invisible, and mostly waste	Seen, capped, and tuned by person and team
The record	None	Every action logged and answerable
Who's protected	The people who already knew the risk	Everyone, including those who didn't

This is the posture that fits the moment: not locking AI down until it's useless, and not letting it run loose until something breaks, but building the place where AI runs and letting the whole company go fast inside it.

Section 13

The Operating Model

The five requirements describe a layer: a set of capabilities, and what the environment can enforce. That's not the same thing as an operating model, and the gap between the two is where most AI programs quietly stall. Every layer in a stack depends on someone owning it and a discipline that keeps it current. The application and data layers have platform and governance teams behind them. Without that, a layer is right on the day it ships and drifting by the next quarter, because the tools it governs never hold still.

What closes the gap is the layer plus two things a layer can't supply on its own: a clear owner, and a cadence. That's what keeps five well-built requirements governing the same way a year from now.

Who Owns the Layer

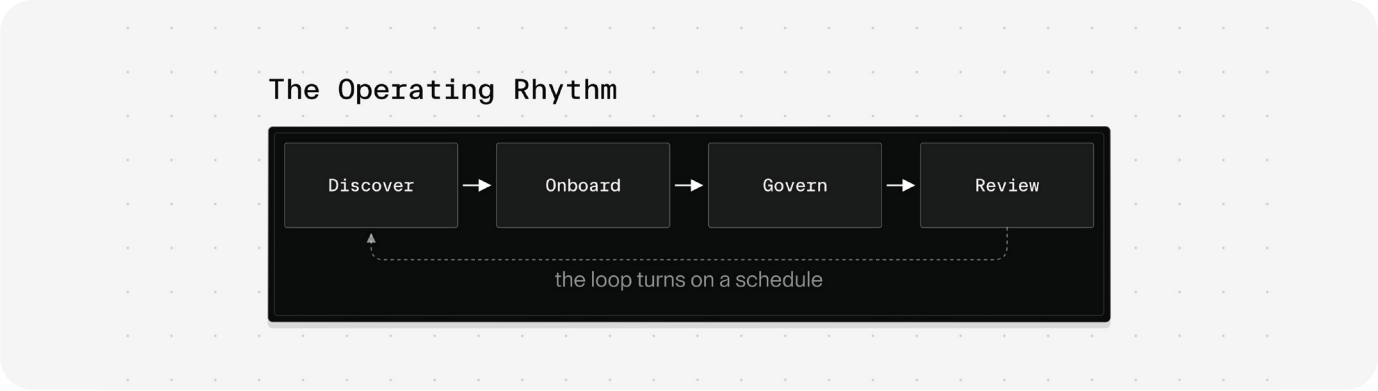
Look at who already does this work, and the owner picks itself. Platform teams provision the environments other people build in, define the policies those environments enforce, and set the permissions inside them: the same team this paper keeps returning to. The AI Operating Layer is that job widened, from developers to everyone who now touches AI. That's a change in scope rather than a new function, which is what makes it something an organization can start on rather than reorganize around.

Day to day, the job comes down to a few concrete things. They run the environment where AI executes and decide what it can reach. They bring new tools and models inside quickly, because that speed is what keeps people in the layer instead of routing around it. They hold the record of what ran, who ran it, and what it cost. And they own the spend, with budgets by team and task and the authority to cut waste. What they don't own stays where it already sits: policy, risk appetite, and the budget itself belong to the functions that already hold them.

The Operating Rhythm

Ownership only counts if something makes it happen on a schedule. Left to good intentions, the review that keeps the layer current is the first thing to slip the week the quarter gets busy, so the model runs on a loop instead, one that turns whether or not anything is on fire.

It starts by discovering what's running outside the layer, pulled from expense reports, SSO logs, and a direct question to team leads, and done in the open, because the moment it reads as enforcement, the tools vanish from view. What surfaces gets onboarded: scoped, routed to an approved model, given a budget, handed back. Then it gets governed, with policy set once at the environment level, where it carries to everyone who arrives after. And it gets reviewed against the record, what cost more than it returned, what reached past its scope, what's still outside, before the next pass begins. How often the loop turns depends on how fast tools arrive, and that pace isn't easing.



The operating rhythm. Discover → Onboard → Govern → Review — whether or not anything is on fire.

What good looks like

- A named team owns the AI Operating Layer, with the authority to bring new tools inside quickly, not only to refuse them.
- A new tool goes from request to governed availability in days, so the fast path and the safe path are the same one.
- Policy set once at the environment level applies to every workspace that follows, without per-team reconfiguration
- What's running outside the layer is found on a schedule, deliberately, rather than discovered in the middle of an incident.

Section 15

Enabling the AI Operating Layer with Coder

Most enterprises have a model strategy. Many have a roadmap of AI use cases. Very few have deliberately built the layer where AI actually runs: the governed, secured, cost-attributed infrastructure that makes AI workloads sustainable at scale rather than manageable at prototype. That's the gap the AI Operating Layer fills. That's what Coder is built to provide.

Coder is self-hosted, cloud-agnostic, and purpose-built for organizations running AI in production. The five requirements, contained data, scoped access, governed models, a complete audit trail, and controlled spend, aren't features layered onto a developer tooling platform. They're the architecture. Every workspace Coder provisions, whether for a developer or an autonomous agent, inherits the policies, permissions, and observability the platform team has defined.

The requirements are most effective together. Scoped access without containment still leaks: the task sees only its own data, but that data can still leave. An audit trail without spend visibility says what ran but not what it cost. Model governance without a record is a policy nobody can verify. Each depends on the others, which is why they belong in a single layer, not assembled from tools that weren't designed to work together.

Capability	Without an AI Operating Layer	With Coder
Environment provisioning	Manual, inconsistent, days to weeks	Template-driven self-service, minutes
Model access	Ad hoc, no central catalog or approval process	Approved catalog, governed access
Agent execution	Uncontrolled, broad permissions, no audit trail	Sandboxed, ephemeral, fully auditable
Cost visibility	Cloud bill with no team or project attribution	Per-workspace, per-team, per-project
Security posture	Third-party tools, data leaving the perimeter	Self-hosted, air-gap capable
Compliance	Manual audit reconstruction, incomplete trails	Continuous audit trail, policy-enforced
Platform team load	Every new environment requires manual work	Self-service within defined guardrails

The bottom line

Momentum is easy to get with AI. The AI Operating Layer is how you never lose it.

Every ungoverned environment is harder to govern after the fact. Every unaudited agent run is harder to reconstruct retroactively. Every month without cost attribution is spend that can't be explained or optimized. The cost of building the AI Operating Layer deliberately is a fraction of the cost of building it under pressure, or not building it at all.

Cloud taught us that the organizations that move fastest aren't always the ones that started first. They're the ones that built the foundation before they needed it. The same is true of AI.