

AI

GOVERNANCE



A FIELD GUIDE

The Foundation for Organized AI in Production

FABIO BASTOS

AI GOVERNANCE

The Foundation for Organized AI in Production

A Field Guide

FABIO BASTOS

THINKNEO PUBLICATIONS * FIRST EDITION · 2026

AI Governance: The Foundation for Organized AI in Production
A Field Guide
First Edition — 2026

Copyright © 2026 Fabio Bastos
This work is licensed under the Creative Commons Attribution 4.0 International License (CC BY 4.0). You are free to share and adapt this material, provided you credit the author.

<https://creativecommons.org/licenses/by/4.0/>

The opinions expressed in this book are those of the author and do not constitute legal, financial, or compliance advice.
Published by ThinkNEO Publications
ThinkNEO AI Technology Co., Ltd. Hong Kong

thinkneo.ai
book@thinkneo.ai
DOI:10.5281/zenodo.19655870
<https://doi.org/10.5281/zenodo.19655870>
Distributed worldwide as a free e-book.

Acknowledgments

A book of this kind owes more to conversations than to the act of writing. The practitioners — engineers, compliance officers, finance partners, platform leads — whose experiences shaped the material gave generously of their time and judgment. Most of what is useful here came from them. What is wrong came from me.

I am grateful to the partners and institutions who have supported the work from which this book emerged, including NVIDIA's Inception program, the Anthropic Partner Network, and the research community at the Universidade de São Paulo's Institute of Mathematical and Computer Sciences, whose early engagement helped sharpen the argument.

To the readers who offered comments on earlier drafts: thank you. The book is better for your patience.

And to my family, for the usual reasons that do not need to be spelled out in a book about software.

Preface

This book was written in the middle of the work it describes. I spend my days building infrastructure for running AI in production — the kind of infrastructure that, when it works, is invisible, and when it fails, is the only thing anyone talks about. What follows is my attempt to describe the shape of this work clearly, for readers who are about to do it, or are in the middle of it, or have just finished explaining to someone why the monthly invoice was what it was.

I did not set out to write a manifesto. The book's five-pillar structure emerged from conversations with practitioners — engineers, compliance officers, finance partners, platform leads — who were all, independently, reaching for similar vocabulary to describe the same set of problems. When the same categories keep showing up in different rooms, there is usually something to the categories.

The intended reader is an operator. A technical lead building an AI platform. A CISO wondering what is actually in flight. A CFO who has noticed that the line item labelled AI has become too large to ignore. A compliance officer translating external frameworks into internal controls. I have tried to avoid both the academic register that would make the book irrelevant to them and the vendor register that would make it untrustworthy. Whether I succeeded is for the reader to judge.

One bias should be declared at the outset. I run a company that builds an enterprise AI control plane — the pattern described in Chapter 9. I have tried to write a book that would be useful to someone who never becomes a customer, and to be honest about tradeoffs that do not favour any particular vendor, including my own. Readers should weigh the material with that disclosure in mind.

The book is short on purpose. Most of the operators I know are already reading too much. If a chapter can be skipped without losing the thread, it can be skipped. If a section is obvious in your context, skim it. The point is to be useful, and length is not a proxy for usefulness.

— Fabio Bastos

Bangkok, 2026

Contents

Acknowledgments4

Preface5

Contents7

Introduction: The Quiet Revolution8

Part I – The Governance Imperative12

1. From Experimentation to Production13

2. The Shadow AI Problem17

3. Why Traditional Tools Fall Short21

Part II – The Five Pillars26

4. Runtime Guardrails: The First Line of Defense27

5. Observability Built for AI32

6. AI FinOps: Governing the New Variable Cost36

7. Policy and Governance: From Documents to Code40

8. Provider-Agnostic Routing: The End of Lock-in45

Part III – Putting It Into Practice50

9. The Control Plane Pattern51

10. A Pragmatic Implementation Roadmap55

11. Measuring What Matters59

Conclusion: The Organized AI Organization63

About the Author66

Glossary67

Further Reading70



Introduction: The Quiet Revolution

In February 2024, a small civil resolution tribunal in British Columbia issued a ruling that, on paper, involved a trivial sum of money. A traveller had consulted Air Canada's website chatbot about bereavement fares, received advice that was in direct conflict with the airline's actual policy, and bought a ticket based on that advice. When the refund was denied, he went to court. The airline argued that its chatbot was, in effect, a separate entity responsible for its own misstatements. The tribunal disagreed. The airline was ordered to pay.

The judgment was not large. The precedent was. A company had just been held financially liable for what its language model told a customer, and the question of whether the bot had been correct under some abstract policy document was deemed irrelevant. The bot had spoken. The company owned what it said.

Around the same time, a major consumer electronics manufacturer confirmed that employees had pasted proprietary source code into a public chatbot in order to debug it. The code, by design of the service, had been retained. The company banned internal use of public large language models shortly afterward, but the damage, to the extent it existed, had already been done. Several other enterprises followed with similar policies over the next year.

These were not cautionary tales from a distant future. They were reports from 2023 and 2024, the early years of a shift that most organizations are still in the process of absorbing. Generative AI, which had spent the better part of a decade as a research curiosity, had crossed into the enterprise. And it had crossed faster than almost any technology in living memory.

The adoption happened through two channels at once. Formally, companies launched pilots, then production deployments, of customer-facing assistants, document summarizers, code copilots, and autonomous agents. Informally, employees began using public chatbots to draft emails, analyze spreadsheets, write code, and interpret legal documents, often without telling anyone.

AI GOVERNANCE

By the time most organizations got around to drafting AI usage policies, the behaviour those policies were meant to regulate was already widespread.

This situation, where a powerful new technology is already embedded in daily operations before the organization has figured out how to manage it, is not unprecedented. The early cloud era looked similar. So did the first wave of mobile computing. In each case, the technology arrived, productivity followed, and then, sometimes years later, the discipline caught up.

The discipline that is now catching up with generative AI has a name. It is called AI governance. And while the term is often misunderstood as synonymous with compliance, or reduced to a checklist of ethical principles, its actual meaning is operational and specific.

AI governance is the practice of running AI systems with the same rigor organizations already apply to financial systems, industrial systems, and other environments where the consequences of failure are real.

This book is a field guide to that practice. It is not about whether AI is good or bad, or whether it should be regulated more or less. It takes as given that AI is already in production inside most medium and large organizations, and that the question is no longer whether to govern it but how. The chapters that follow lay out the shape of the problem, the five technical pillars that have emerged as a rough industry consensus, and a pragmatic roadmap for organizations that are starting from scratch.

The revolution in the title of this introduction is called quiet for a reason. Unlike the public drama around model releases and benchmark scores, the work of governance happens in logs and dashboards, in policy documents and audit trails, in the boring, careful layers of infrastructure that separate a technology that works in a demo from a technology that works in production.

AI GOVERNANCE

It is not glamorous work. But it is the work that determines whether the current AI wave ends in lasting productivity or in a series of expensive, embarrassing, and preventable incidents.

The premise of this book is that the difference between the two outcomes is almost entirely a question of governance. Everything else is details.

Part I —

The Governance Imperative

Before discussing how to govern AI systems, it helps to be specific about why the question matters now, and why it is different from the questions that came before. Part I of this book lays out three arguments. The first is that AI has moved from pilot to production faster than any comparable technology, and that this shift changes what counts as acceptable behaviour. The second is that the problem of ungoverned AI inside organizations, often called Shadow AI, is structurally worse than its predecessor in the IT era. The third is that the tools most companies already own, including application monitoring, security information platforms, and data loss prevention systems, were not designed for the kinds of failures that generative AI produces, and that trying to force the new problem into the old tooling is a predictable dead end.

1. From Experimentation to Production

Every technology passes through a characteristic set of stages on its way from novelty to infrastructure. Observers of the enterprise software market have described these stages in various ways, but the pattern is roughly the same. A small group experiments. The experiments produce a few striking results. A larger group takes notice. Pilots are launched. Some of the pilots succeed. The successes are scaled. And then, somewhere in the scaling, the technology stops being a special project and becomes part of how the organization functions.

The transition from pilot to production is almost always underestimated, and for a specific reason. Pilots are permissive environments. The people running them are usually enthusiastic about the technology, tolerant of rough edges, and willing to work around bugs because they want the experiment to succeed. Production environments are the opposite. The people running them are measured on uptime, cost, compliance, and customer satisfaction. They do not want the experiment to succeed or fail. They want the system to work, predictably, every day, at a known cost.

This difference in temperament produces a difference in requirements. A pilot can tolerate a chatbot that occasionally gives wrong information, because the pilot users are often the same people who built the chatbot. Production cannot. A pilot can absorb a surprise in the monthly invoice, because the pilot budget is small and treated as an investment. Production cannot, because production spend rolls up into financial reporting and eventually attracts the attention of the chief financial officer. A pilot can operate without audit logs, because nobody is going to audit the pilot. Production cannot, because somebody eventually will.

The compressed timeline

What makes the current AI transition distinctive is not the shape of the curve but its slope. Cloud computing took roughly a decade to move from early adopters to mainstream enterprise infrastructure. Mobile applications moved faster, but still over years. Generative AI, by most reasonable measures, has compressed the same transition into roughly eighteen to twenty-four months.

The compression has several causes. The underlying technology arrived in a form that required almost no infrastructure on the customer side. A working system could be assembled in an afternoon by a developer with an API key. The models were general enough to serve hundreds of use cases, which meant that the same platform that answered customer questions could also help draft internal documents, review code, and analyze spreadsheets. And the user interface, ordinary natural language, required no training, no onboarding, and no change management.

These properties are the reason AI adoption has been fast. They are also the reason governance has been slow. The same ease of adoption that allowed a developer to ship a chatbot in an afternoon also allowed that chatbot to bypass the procurement reviews, security assessments, and architectural guidance that would normally have been applied to a new production system. In many organizations, by the time leadership decided to establish an AI governance function, the function was looking at a map of AI usage that had been drawn by nobody in particular and updated by everyone.

The failure modes are new

The other complication, beyond speed, is that the failures produced by generative AI do not resemble the failures that enterprise governance is used to managing. A traditional software system, when it fails, usually fails visibly. A server crashes. A database returns an error. A transaction is rejected. Operators see the failure, fix it, and document it.

A generative AI system, when it fails, often fails silently. It produces an answer that is grammatically correct, superficially plausible, and wrong. It cites a source that does not exist. It summarizes a document in a way that omits the single paragraph that mattered. It recommends a course of action that would have been correct if one specific fact, which it invented, had been true. The system does not know it has failed. The user may not know either. The only way to catch the failure is to have built, in advance, a mechanism for catching it.

Building those mechanisms is the central technical problem of AI governance, and it is a problem that cannot be solved by waiting for the models to get better. Models will get better. They will not, on any realistic timeline, become perfectly reliable. And even if they did, the auditors, regulators, and customers who depend on the outputs will continue to require evidence that the system is behaving as claimed. Evidence is a governance problem, not a model problem.

The move from experimentation to production, then, is not a single threshold that an AI project crosses at a particular moment. It is a shift in the kind of discipline the project requires. What was a curiosity becomes infrastructure. What was a pilot becomes a dependency. And the organizations that succeed at the transition are, on inspection, the ones that treated this shift as the central question from the beginning, rather than something to be worked out later.

A futuristic server room with glowing blue light trails and a grid pattern.

2. The Shadow AI Problem

The information technology industry has seen this movie before. In the early 2010s, cloud services reached a level of maturity and accessibility that allowed individual employees and small teams to sign up for business software, storage, and compute capacity without any involvement from central IT. The resulting pattern was given a name: Shadow IT. The name stuck because it captured something real. Software was being used in a shadow, out of sight of the people formally responsible for it.

Shadow AI is the same pattern, repeated. An employee who needs to summarize a long document pastes it into a public chatbot. A product manager who needs to draft a requirements document does the same. A software engineer who is debugging a subtle problem pastes in the relevant code. None of these actions feel like a governance event. They feel like productivity. They often are productivity. And they are also, in aggregate, one of the largest data exfiltration channels that has existed inside enterprises in years.

Why this version is worse

It is tempting to treat Shadow AI as a repeat of Shadow IT and to reach for the same remedies. Several features of the generative AI problem, however, make it structurally harder.

The first is the nature of the data flow. Shadow IT typically involved signing up for a service and storing data inside it. The data was retrievable. If the company later chose to cancel the service and migrate to a sanctioned alternative, the data could, in most cases, be recovered. Shadow AI involves feeding data into a prompt, often along with a question, and receiving an answer. What happens to the data after that depends on the provider, the terms of service, the subscription tier, and the internal data-handling policies of the service. In many cases, the data is used to improve future models.

In many others, it is retained for a period to support abuse monitoring. In all cases, the act of pasting it into the prompt is not reversible. The data has crossed a boundary, and the organization cannot unsend it.

The second is the nature of the interface. Shadow IT usually required signing up for something. There was a credit card, an account, an email address. These artifacts were discoverable. Shadow AI, for most employees, requires only a web browser and a few seconds. A personal account on a public chatbot is effectively invisible to the organization. Network controls can block specific domains, but the list of domains is long, growing, and now includes first-party services embedded in common productivity tools.

The third is the nature of the usage pattern. Shadow IT tended to settle into stable patterns that, once identified, could be audited and either blocked or formalized. Shadow AI is chaotic by design. The same employee may use three different tools in a single day for three different tasks, and may not remember which tool they used for which purpose a week later. There is no stable pattern to identify.

What the data suggests


Quantifying Shadow AI is difficult, because the activity being measured is, by definition, unobserved. Surveys conducted by various industry analysts through 2024 and 2025 have consistently produced a similar shape of result. A large majority of employees at knowledge-work organizations report using at least one generative AI tool for work tasks. A substantial minority of those report using tools that their employer has not sanctioned, and many report being uncertain whether the tools they use are sanctioned or not. When asked whether they have pasted confidential material into these tools, a meaningful fraction say yes, and a larger fraction say they do not remember.

The specific percentages vary across studies, and it is not useful to quote them here as if they were authoritative. What matters is the direction. Use is high. Awareness of policy is low. And the fraction of activity that the organization can actually see, through its existing monitoring tools, is a small slice of the total.

The response is not a ban

The first instinct of many organizations, on discovering the scope of Shadow AI, is to ban the tools. This instinct is understandable and usually counterproductive. The productivity gains that employees experience from these tools are real, sometimes substantial, and the act of banning the tools often pushes the usage further underground rather than eliminating it. The ban, in other words, often reduces visibility without reducing use.

The more durable response is to provide sanctioned alternatives that are at least as useful as the shadow ones, and to make the sanctioned path the path of least resistance. This is not a new lesson. Shadow IT was eventually addressed by building internal platforms that made the right thing easy, rather than by prohibiting the wrong thing. The same approach applies here, and it is one of the recurring themes of this book: governance that makes correct behaviour inconvenient tends to be ignored. Governance that makes correct behaviour easy tends to work.

The background features a stylized illustration of a human brain in shades of gray and blue. A medical device, possibly an ultrasound or MRI probe, is shown in the upper left corner, emitting several bright blue beams of light that converge towards the brain. The overall aesthetic is clean and modern, with a light blue and gray color palette.

3. Why Traditional Tools Fall Short

Most medium and large organizations already own a substantial set of tools for observing, securing, and controlling their software systems. These tools represent the accumulated wisdom of several decades of production computing. There are application performance monitors that watch latency and error rates. There are security information and event management systems that aggregate logs and raise alerts. There are data loss prevention systems that scan outbound traffic for patterns that look like sensitive data. There are identity and access management platforms that decide who can do what.

When generative AI arrives in a production environment, the natural question is whether these tools can be extended to cover it. The honest answer is that they can be partially extended, and that the parts they cannot cover are precisely the parts that matter most.

The deterministic assumption

The deepest reason for the mismatch is philosophical. The tools that run most production software were designed for systems that are, in a practical sense, deterministic. Given a particular input and a particular state, the system is expected to produce a particular output. When it does not, the deviation is a bug, and the bug is something to be fixed. The monitoring stack is oriented around detecting deviations from expected behaviour.

Generative AI systems are not deterministic in this sense. Given the same input twice, they may produce two different outputs, both of which may be correct, or both of which may be wrong, or one of which may be correct and the other wrong. The notion of expected output, in the strict sense, does not apply. An AI-powered customer support system might answer the same question in five different ways across five different conversations. All five answers might be acceptable. Or four of them might be acceptable and the fifth might be a hallucination that invents a refund policy that does not exist.

The monitoring stack, if it was built on the deterministic assumption, will not flag the fifth as different from the first four. They are all just text.

This gap cannot be closed by tuning thresholds or writing smarter alerts. It requires a category of observation that the existing tools were not designed to produce: semantic observation, which is to say, the ability to ask not whether the system responded, but whether the response was correct, grounded, compliant, and appropriate.

The token economy

A second mismatch involves cost. Most existing cost management tools in the enterprise are built around resources that are either fixed, such as servers and licenses, or proportional to known usage, such as cloud compute and storage. Finance teams are comfortable with these categories. They have developed processes for forecasting, allocating, and controlling them.

Generative AI introduces a cost category that is neither fixed nor proportional in the usual sense. The unit of consumption is the token, a small chunk of text processed by the model. A single interaction may consume anywhere from a few hundred to many thousands of tokens, depending on the length of the input, the complexity of the task, and the model's response. Costs per token vary by provider, by model, by whether the tokens are input or output, and in some cases by the time of day. The total cost of an AI feature in production can swing by an order of magnitude depending on how users choose to use it, and those usage patterns can change without warning when an application is updated or a marketing campaign drives new traffic.

Traditional cost management does not handle this well. A budget set in January can be exhausted in March because a single agent, running in a loop that nobody noticed, generated ten million extra tokens over a weekend. The usual tools will show, after the fact, that spending went up.

They will not typically show why, in a form that a finance team can act on. The question of which team, which application, which user, and which use case drove the spike is answerable only if the organization instrumented the system to answer it, which few organizations did.

The semantic gap

A third mismatch involves the nature of what can go wrong. Traditional security tools are oriented around well-defined threats: malicious code, unauthorized access, exfiltrated files. Generative AI introduces categories of failure that do not fit the existing schemas.

A prompt injection attack, for example, is not a virus. It is a string of text, in ordinary language, that convinces the model to ignore its previous instructions and follow new ones supplied by an attacker. There is no signature to match, no hash to block, no malicious binary to quarantine. The attack succeeds or fails based on the semantic content of the text. A data loss prevention system that scans for social security numbers will not catch a prompt that convinces a model to reveal, in its own words, the substance of a confidential document that was included in its context.

These categories — hallucination, prompt injection, context leakage, policy bypass through paraphrase — require tools that were designed to operate on meaning, not on bytes. The existing enterprise security stack is a bytes-oriented stack. It is not wrong. It is not obsolete. It is simply not sufficient for the new problem.

The need for a new layer

The conclusion that has been emerging across the industry, not as a matter of vendor preference but as a pattern that keeps reappearing in practitioner accounts, is that AI systems in production require a new layer of tooling alongside, not instead of, the existing stack.

That layer is oriented toward semantic observation, probabilistic reasoning about correctness, token-level cost attribution, and the specific failure modes that generative systems produce. The rest of this book is, in large part, an examination of what that layer looks like and how organizations are building it.

Part II —

The Five Pillars

When practitioners describe the work of running AI systems in production, five categories of concern come up repeatedly. Different organizations emphasize them in different proportions, and different vendors package them in different combinations, but the underlying five are remarkably stable across industries and geographies. They are, in roughly the order an organization tends to encounter them:

Runtime guardrails: the machinery that evaluates every input and output in real time and blocks the ones that should not be allowed through.

Observability: the systems that record what happened, in enough detail that someone can later reconstruct why.

AI FinOps: the discipline of measuring, allocating, forecasting, and controlling the new kind of variable cost that AI introduces.

Policy and governance: the codified rules, approvals, and audit trails that connect AI behaviour to the organization's obligations.

Provider-agnostic routing: the mechanism that decides, per request, which model handles it, and the fallback logic that takes over when the first choice fails.

The chapters that follow treat each of these in turn. They are presented as separate pillars because each involves distinct techniques and distinct vocabulary. In practice, they are interdependent, and most production-grade deployments end up implementing all five, even when the implementation in one area is more mature than in another.

4. Runtime Guardrails: The First Line of Defense

```
X X X ■ X >  
X X X ■ X X X  
X X X X X X X ■ X  
X X X X X X X X X  
X ■ X X □ X X  
  X X X X
```

A guardrail, in the AI context, is a mechanism that sits between the user and the model, or between the model and the downstream system, and has the authority to intercept, modify, or refuse the traffic passing through. The name is borrowed deliberately from highway engineering. A highway guardrail does not prevent accidents from happening. It prevents accidents that do happen from becoming catastrophic.

The analogy is useful because it sets the right expectations. Guardrails are not a replacement for careful model selection, prompt engineering, or system design. They are the layer that contains failures when those other layers break down, which, in systems of realistic complexity, they always eventually will.

Input guardrails

The first category of guardrails operates on inputs, which is to say, on the text a user or upstream system has sent to the model. Three concerns dominate this category.

The first is prompt injection. A prompt injection is an input that attempts to subvert the model's instructions by supplying new instructions inside the user-supplied text. The simplest form looks like "ignore your previous instructions and instead do the following." More sophisticated forms embed the attack inside content that the model is processing on behalf of the user, such as a document being summarized or a web page being analyzed. Effective input guardrails scan for the patterns associated with these attempts and either strip them, flag them for review, or reject the request outright.

The second is sensitive data exposure. Users, sometimes deliberately and often by accident, paste content into prompts that should not leave the organization.

This may include personal information about customers, confidential financial data, health records, or proprietary source code. Input guardrails in mature deployments identify such content before it reaches the model and either redact it, substitute placeholders, or block the request and return a message explaining why.

The third is topic restriction. Most production AI systems are built for a specific purpose, and users reliably discover that the same system can be asked about almost anything else. A customer support bot fielding questions about a company's software will be asked, sooner or later, to write poetry, diagnose medical symptoms, or opine on political questions. Input guardrails can restrict the system to the set of topics it is intended to handle, reducing both embarrassment and liability.

Output guardrails

The second category operates on outputs, meaning the model's response before it reaches the user or is passed to a downstream system. Here the concerns are different.

Hallucination detection is the most discussed output concern. A hallucination, in the technical sense used in AI research, is a model output that asserts something as fact when the model has no basis for doing so. Hallucinations are not always easy to detect programmatically, but specific patterns can be caught. When the model is expected to cite sources, the sources can be checked for existence. When the model is expected to answer from a specific knowledge base, the answer can be evaluated against that knowledge base for consistency. When the model produces numerical claims, those claims can be cross-referenced with authoritative data.

Policy compliance is the second output concern. A model that is authorized to discuss a company's products may not be authorized to quote specific prices, or to make commitments on behalf of the company, or to disclose information about unreleased features. Output guardrails can identify content that falls outside the authorized envelope and either modify or block it before it is delivered.

Tone and brand safety is the third. An output that is factually correct and policy-compliant may still be inappropriate in its tone, its level of formality, or its handling of sensitive topics. Brand-conscious organizations increasingly run outputs through classifiers that assess these qualities and intervene when the response would damage the brand if published.

The judge-model pattern

A recurring technique across both input and output guardrails involves using a second model, typically smaller and faster than the primary one, to evaluate the traffic. The second model is configured as a judge. It does not generate responses. It reads the input or the proposed output and returns a verdict: allow, modify, or block, sometimes with a justification.

This pattern, sometimes called LLM-as-judge, is not a universal solution. Judge models can be fooled by the same adversarial techniques that fool any other model, and they add latency and cost to every request. But for classes of problems that are hard to catch with pattern matching, they are often the best available option, and deployments that combine pattern-based detection with judge-model evaluation tend to catch a higher proportion of problems than either approach on its own.

The operational reality

In practice, the design of a guardrail system is less about choosing between techniques and more about deciding what the system should do when a guardrail triggers. The simple options are block and log. The more sophisticated options include redact and continue, rewrite and continue, escalate to a human reviewer, or serve a canned response that gracefully declines. Each choice has consequences for user experience, throughput, and the quality of the audit trail. Most mature deployments end up with a layered policy, where different triggers produce different responses, and where the triggers and responses are themselves policy artifacts that can be reviewed and updated without redeploying the application.

5. Observability Built for AI

Observability is an older concept than AI. The word was borrowed from control theory in the 2010s by engineers who were trying to describe the difference between monitoring a system, which implies watching for known failures, and understanding a system, which implies being able to ask questions about its behaviour that nobody anticipated in advance. In modern software engineering, observability is roughly the discipline of building systems whose internal state can be inferred from the signals they emit.

AI systems require a specifically tailored version of this discipline. The reasons follow directly from the mismatch with traditional tools discussed in Chapter 3.

The three signals, plus two

Conventional observability rests on three signals: metrics, logs, and traces. Metrics are numeric summaries of system behaviour over time. Logs are the textual records of discrete events. Traces capture the flow of a single request through a distributed system. These three signals remain essential for AI systems, and AI deployments emit them for the same reasons any other production system does.

AI observability adds at least two more categories. The first is token-level telemetry, which records the number of input tokens, output tokens, and cached tokens consumed by each request, broken down by model, by application, and by cost center. Without this data, cost management, which is discussed in the next chapter, becomes guesswork.

The second is semantic telemetry, which captures information about the meaning of what happened rather than the mechanics. This includes records of which guardrails fired, what content was redacted, which prompts were flagged as potential injections, whether outputs were evaluated and how they scored, and, critically, the prompts and responses themselves, stored in a way that respects privacy but preserves enough detail for later review.

Tracing across models and tools

A single user request to a modern AI application rarely involves a single call to a single model. It may involve a retrieval step that queries a vector database, a planning step that calls a larger model, several execution steps that call smaller or specialized models, one or more tool calls that invoke external APIs, and a synthesis step that combines everything into a final response. A failure anywhere in that chain can produce a wrong answer at the end, and reconstructing what happened requires a trace that spans all of the components.

The standards for this kind of tracing have been converging on OpenTelemetry, a broadly supported observability specification, extended with AI-specific conventions that capture model names, prompt versions, and tool invocations. Organizations that adopted these conventions early have reported that debugging agentic systems, which can otherwise feel like trying to understand a conversation by reading only every fifth word, becomes tractable once the traces are complete.

The prompt as a first-class artifact

One of the subtler observations that has emerged from running AI systems in production is that the prompt itself, meaning the instructions and examples given to the model, is a critical artifact that deserves the same versioning, review, and audit treatment as code.

Prompts evolve. They are tuned in response to observed failures, updated when the underlying model is replaced, and expanded as the use case grows. A deployment that does not track which version of the prompt was in use when a given interaction happened will struggle to answer basic questions later: did this incident occur before or after the refusal language was tightened? Was the customer complaining about the prompt running now, or the one from last month? Mature observability treats the prompt as a versioned artifact, records which

version was active for each request, and makes it possible to compare prompt versions the way one would compare source code.

Privacy and retention

Storing prompts and responses presents a specific operational challenge. The content may contain personal information, confidential business data, or material covered by regulatory constraints. The naive approach, logging everything and sorting it out later, can itself become a compliance liability. The more careful approach combines selective redaction at the point of capture, tiered retention policies that hold different classes of data for different periods, and access controls that ensure that only authorized personnel can query the full content.

This last point is often underappreciated. Observability data in an AI system is not neutral telemetry. It is a partial record of everything the system was asked and everything it said. Treating it with the same care as the underlying business data is not optional. It is a prerequisite for operating the system at all in any regulated environment.

6. AI FinOps: Governing the New Variable Cost

The term FinOps emerged in the cloud era to describe a then-novel discipline: the collaborative practice of managing variable cloud spending across finance, engineering, and operations. Before FinOps, cloud costs tended to surprise finance teams, because the spending was driven by engineering decisions that finance had no visibility into. After FinOps, the two functions began working from a shared view of consumption, with forecasts, budgets, and chargeback mechanisms that connected technical behaviour to financial outcomes.

The same dynamic is now repeating with AI, which is why the term AI FinOps has started to appear in practitioner literature. The situation is similar in structure: spending is variable, driven by engineering decisions, and easy to lose track of. But several features of AI spending make it, if anything, harder to manage than cloud spending was a decade ago.

Why AI costs behave differently

Cloud resources, once provisioned, tend to produce predictable cost curves. A virtual machine runs at a known hourly rate. A storage bucket accumulates a known monthly fee per gigabyte. Variability exists, but it is usually bounded by obvious levers such as the number of machines running and the volume of data stored.

AI costs are variable at a finer grain. The unit of consumption is the token, and the number of tokens consumed by a single interaction can swing by one or two orders of magnitude depending on factors that are often invisible to the person writing the code. A user who asks a long question gets a bigger bill. A retrieval step that pulls in more context generates more input tokens. A model that decides to explain its reasoning produces more output tokens. An agent that loops more times before reaching a conclusion consumes tokens at each step. Minor changes in prompt design can double or halve the cost per interaction.

Compounding this is the diversity of pricing across providers and models. Input tokens and output tokens are usually priced differently, with output often several times more expensive. Different model tiers within the same family have different prices. Caching mechanisms, where they exist, change the pricing again. An application that uses several models across its lifecycle can have a cost structure that is difficult to describe in a sentence and impossible to forecast without detailed instrumentation.

Cost per useful outcome

The metric that matters, ultimately, is not the cost per token or the cost per request but the cost per useful outcome. An AI system that generates answers cheaply but produces answers that users reject or that require human review is not, in the economic sense, cheaper than a system that costs more per call but produces outputs that can be used as-is.

Computing this metric requires connecting the cost side of the observability pipeline to the quality side. If the observability system records whether each interaction produced a useful outcome, and the cost system records what each interaction cost, the two can be joined to produce a cost per useful outcome for any slice of the traffic. This figure, broken down by use case, by model, and by user cohort, is a significantly better basis for decisions about where to invest, where to switch models, and where to retire a feature that is not earning its keep.

Controls and guardrails for cost

The most common cost incidents in AI deployments, by the accounts of practitioners, are not gradual overruns. They are sudden spikes caused by runaway loops, unintended recursion, or sudden traffic surges against an unbounded system. An agent that was supposed to call a tool and report the result instead calls it in a loop a few thousand times before anyone notices.

A batch job that was supposed to process a hundred documents processes the entire archive of a hundred thousand.

The remedies are the same remedies familiar from other kinds of cost-sensitive infrastructure: per-request limits, per-user quotas, per-application budgets, and circuit breakers that cut off traffic when a threshold is crossed. The specifics are different, because the threshold is measured in tokens rather than in requests or compute hours, but the pattern is not new. What is new is that in many AI deployments, these controls have not been put in place, because the deployments grew out of pilots where the cost was small enough to ignore. When the deployment reaches production scale, the controls often have to be retrofitted under pressure, which is the worst time to build them.

Chargeback and the cultural change

A finding that recurs across FinOps practice, in both the cloud era and the AI era, is that visibility alone does not change behaviour. Engineers who know that their application is expensive do not necessarily make it less expensive unless the cost is reflected in something they are accountable for. Chargeback mechanisms, which allocate AI costs back to the teams that generate them, are the standard mechanism for creating that accountability. They are politically complicated and operationally demanding, but in organizations that have implemented them for AI spending, the reported effect on consumption patterns has been substantial. Teams optimize what they pay for. They rarely optimize what somebody else pays for.



7. Policy and Governance: From Documents to Code

Every regulated industry has experience with the gap between written policy and actual behaviour. A financial institution has policies on trading limits. A hospital has policies on patient data access. A manufacturer has policies on safety protocols. The history of compliance is, in significant part, the history of closing the distance between what the policy says and what the system actually does, through a combination of training, audit, and increasingly, automated enforcement.

AI governance inherits this entire history and adds new dimensions to it. The written policies of most organizations now include sections on acceptable AI use, data handling for AI systems, and oversight for AI-driven decisions. The operational question is how those policies translate into runtime behaviour. A policy document stored on an intranet page, no matter how carefully written, does not intercept a single request. Something else has to.

The regulatory landscape

The external pressure for AI-specific policy is now coming from multiple directions at once. The European Union's AI Act, which was adopted in 2024 and whose provisions are phasing in over the following years, establishes a risk-based framework that classifies AI systems by the severity of the potential harm and imposes correspondingly graduated requirements. Systems deployed in high-risk contexts, such as employment decisions, credit scoring, and critical infrastructure, face documentation, transparency, and human oversight requirements that will, in practice, force the implementation of substantial governance machinery.

In the United States, the National Institute of Standards and Technology published an AI Risk Management Framework in 2023 that, while voluntary, has become a reference point for large organizations trying to demonstrate reasonable care. The framework is organized around four functions, Govern, Map, Measure, and Manage, and provides a vocabulary that has been widely adopted in internal programs.

At the international level, the International Organization for Standardization published ISO/IEC 42001 in late 2023, establishing an AI management system standard comparable in structure to the information security standards that preceded it. Organizations pursuing certification under 42001 commit to documented processes for AI risk assessment, lifecycle management, and continuous improvement.

These frameworks differ in detail but overlap significantly in substance. An organization that takes any one of them seriously finds itself building capabilities that are useful for the others. And increasingly, the customers of these organizations, particularly in regulated industries and in the public sector, are beginning to require evidence of alignment with at least one of the frameworks as a condition of doing business.

Policy as executable artifact

The practical question for any AI program is how policy commitments become enforceable constraints on the system. The pattern that has emerged, broadly, is to treat policy as an executable artifact rather than a document.

In this model, a policy is defined in a structured form that a machine can evaluate. It specifies what categories of data can be sent to which models, which outputs require human review, what topics are out of scope for which applications, which users have which permissions, and what should happen when any of these constraints is violated. The policy is stored in a version-controlled repository, reviewed when it changes, and loaded into the runtime at startup or refreshed periodically.

At runtime, a policy engine evaluates each request against the active policies and either allows it, modifies it, or blocks it, producing an audit record in each case. The audit records are retained for a period determined by the regulatory environment and the organization's own retention policies, and they can be queried to answer specific questions: what happened in this interaction, which policy applied, why was it blocked, who authorized the exception.

This pattern, which borrows heavily from the policy-as-code movement that emerged in cloud infrastructure, has the property that the policy document and the enforcement mechanism are the same artifact, which means they cannot drift apart. A change to the policy is, by definition, a change to the enforcement. A violation of the enforcement is, by definition, visible as a violation of the policy.

Human oversight and the question of meaningful review

Every major AI regulation includes, in some form, a requirement for human oversight of high-stakes decisions. The wording varies, but the underlying principle is consistent: automated systems should not make certain classes of decision entirely on their own, and a human should be in a position to intervene.

The operational challenge is making this requirement meaningful rather than ceremonial. A human who is asked to approve a thousand AI-generated decisions per day, with no way to realistically review any of them, is not providing oversight. The review has become a rubber stamp, which satisfies neither the regulatory intent nor the underlying goal of catching problems.

Mature implementations address this in several ways. They reserve human review for decisions that actually require it, using risk scoring to triage. They present reviewers with the context needed to make an informed judgment, including the system's reasoning, the relevant source material, and the audit history.

AI GOVERNANCE

They track review outcomes and use them to improve the system, so that the work of the reviewers feeds back into fewer false positives over time. And they accept that genuine oversight is expensive, and that the cost is part of the cost of operating the system responsibly.



8. Provider-Agnostic Routing: The End of Lock-in

In the first two years of production generative AI, most organizations standardized on a single provider for most of their work. The standardization was a matter of convenience. One account, one invoice, one set of APIs, one integration. The choice of provider was driven by whichever company had the best model when the decision was made, or by whichever company had the most persuasive sales team.

That pattern is now breaking, and the reasons for the break are instructive. They also point toward the last of the five pillars of this book.

Why single-provider strategies erode

The first reason is that no single provider is best at everything. One family of models is particularly strong at code generation. Another is particularly strong at long-context reasoning. A third is particularly strong at extremely fast, cheap completion of simple tasks. A fourth is particularly good at structured output. Organizations that use AI for more than one purpose, which is to say almost all organizations at this point, discover that the best choice of model varies by use case, sometimes by a significant margin in quality or cost or both.

The second reason is that the performance landscape shifts. A provider that is clearly ahead in one quarter may be matched or overtaken in the next. Organizations that tied their critical applications to a single provider, only to see that provider fall behind or raise prices, discover that switching is harder than they expected. The prompts were tuned for the original model. The evaluations were calibrated against its behaviour. The operational muscle memory was built around its quirks. Migrating to a different provider is a project, not a decision.

The third reason is operational. Providers have outages. Rate limits are reached. Regional failures occur. A customer-facing application that depends on a single provider inherits that provider's availability curve. For non-critical applications, this may be tolerable. For revenue-generating applications, it often is not.

The fourth reason is strategic. An organization that is entirely dependent on one provider has reduced bargaining power on pricing, reduced flexibility on terms, and reduced optionality in the event of a disagreement. Multi-provider strategies, where they can be operated efficiently, restore some of that leverage.

What routing means in practice

A provider-agnostic routing layer is, conceptually, a small piece of infrastructure that sits in front of the various model providers and chooses, per request, which one to use. The choice can be based on any number of factors.

Capability is the most obvious: if the request requires a specific model family, the routing layer directs it there. Cost is the next: if multiple models can handle the request, the routing layer can prefer the cheaper one for requests where the extra quality of the more expensive model is not worth the cost. Latency is often a factor: some models respond faster than others, and latency-sensitive applications benefit from preferring the fast ones. Availability is critical: when a provider is experiencing an outage or rate-limit pressure, the routing layer can shift traffic to an alternative, with whatever degradation in quality that implies.

Compliance is the factor that is sometimes overlooked but which dominates in regulated contexts. Some data is not permitted to leave certain jurisdictions. Some models are only approved for certain categories of data. A routing layer with knowledge of these constraints can ensure that requests are matched to approved endpoints and rejected when no compliant option is available.

The abstraction problem

The technical difficulty of building a useful routing layer is that different providers have different interfaces, different capabilities, and different ways of handling edge cases. A naive abstraction, which forces every provider into the lowest common denominator of features, produces a system that works with everyone but works poorly with anyone. A richer abstraction, which preserves the distinctive capabilities of each provider, requires more engineering and more maintenance.

The response from the industry has been a combination of emerging standards, such as the API shape that many providers have converged on, and specialized abstraction layers, whether open source or commercial, that handle the translation for the differences that remain. The details of which layer to use are beyond the scope of this book, and the options change frequently enough that any specific recommendation would be out of date within months. The durable point is that the routing capability is necessary, and that building it in-house is feasible but takes time, while buying it off the shelf is increasingly practical but requires care in selection.

Fallback chains and degradation

A subtle feature of mature routing implementations is their treatment of failure. When the preferred model is unavailable, what happens? The simple answer, try another model, turns out to be inadequate in several common cases. The alternative model may not produce the same output format. It may not support the same tools. It may refuse requests that the preferred model would have accepted.

The more sophisticated pattern involves explicit fallback chains, where each level of the chain is paired with a degradation policy. The first level is the preferred model with the full feature set. The second level may be an alternative model with the same features.

The third level may be a simpler model with a reduced feature set, and a clearly marked response indicating that a simpler path was used. The fourth level may be a canned response that explains why the request could not be handled and invites the user to try again later.

This staircase of degradation is what separates AI applications that fail gracefully from AI applications that simply stop working when anything goes wrong. It is one of the less glamorous parts of the governance toolkit, and one of the most important.

Part III — Putting It Into Practice

The previous chapters described the shape of the problem and the five pillars that, individually, address parts of it. Part III addresses the question of how these pillars fit together in a coherent operating model, how an organization without any of this infrastructure should go about building it, and how to tell whether the effort is working.

9. The Control Plane Pattern

There is a specific architectural pattern, borrowed from earlier generations of infrastructure software, that has proven unusually well suited to the problem of AI governance. The pattern is called the control plane, and understanding it is useful both for organizations building their own governance infrastructure and for those evaluating vendors that offer it.

The origin of the pattern

In networking, the control plane is the part of a system that makes decisions about how traffic should flow, as distinct from the data plane, which actually moves the traffic. The separation is deliberate. Control-plane changes happen relatively rarely and require careful reasoning. Data-plane operations happen constantly and must be extremely fast. Collapsing them into a single layer tends to produce systems that are either too slow or too rigid.

Kubernetes, which became the dominant container orchestration system in the 2010s, applied the same pattern to compute infrastructure. The Kubernetes control plane decides what should run where. The data plane, comprising the nodes that actually run the containers, executes those decisions. The separation allowed Kubernetes to scale in both dimensions independently and to evolve the decision-making logic without disturbing the execution layer.

The pattern applied to AI

The same separation, applied to AI systems, suggests a natural division of responsibility. The control plane is the layer that holds the policies, the observability data, the cost models, the routing logic, and the audit trails. It decides, for each incoming request, what should be done, what should be recorded, and under what conditions. The data plane is the actual model, or set of models, that processes the request and produces the response. The control plane tells the data plane what to do. The data plane tells the control plane what happened.

This separation has several practical consequences. Policies can be updated without redeploying applications. Observability can be comprehensive because it is collected at a layer that sees every request. Cost management can be centralized because billing data flows through a single point. Provider routing becomes possible because the control plane is the only layer that needs to know about multiple providers. Applications themselves become simpler, because the concerns that would otherwise be scattered across every application are handled once, at the control plane.

The alternative, and why it tends to lose

The alternative to a control plane is what naturally emerges in its absence: each application implements its own guardrails, its own observability, its own cost tracking, and its own provider integration. This approach works in the early stages of an AI program, when there are one or two applications and the shared concerns are not yet painful.

It tends to lose as the program matures, for reasons that should be familiar. Policies drift between applications. An update to the approved data-handling rules requires modifications in every codebase. Observability data lives in different formats in different places and is difficult to consolidate. Cost attribution is a manual exercise in reconciling invoices with application logs. Provider migrations, when they become necessary, must be executed separately in every application. And the audit trail that a regulator or internal auditor will eventually ask for does not exist as a single queryable artifact.

Organizations that recognize this trajectory early tend to invest in a control plane layer before it becomes urgent. Organizations that recognize it late tend to spend the following year unwinding the consequences of distributed ownership and consolidating onto a centralized layer, often while operating under external pressure.

Build versus buy

The build-or-buy question for a control plane has been settled, for most categories of enterprise infrastructure, in favour of buy. Few organizations build their own databases, their own cloud platforms, or their own identity management systems. The economics are unfavourable, and the resulting systems tend to lag the commercial alternatives.

AI governance infrastructure is at an earlier point in its maturity curve. Some organizations with particular requirements, or with strong internal engineering cultures, are building their own. Others are adopting commercial or open-source control-plane products. Others are assembling a partial control plane from pieces, using an observability vendor for one concern, a guardrail library for another, a FinOps tool for a third, and glue code to hold them together.

The tradeoffs are the usual ones. In-house builds offer maximum flexibility and alignment with specific needs, at the cost of the engineering investment and the ongoing maintenance burden. Packaged products offer speed and maintained capability, at the cost of some degree of dependence on the vendor. Assembled stacks offer middle ground, at the cost of integration complexity that tends to grow over time. None of the options is universally correct. What matters is making the choice deliberately, with awareness of what is being optimized for.

The background is a vibrant, abstract illustration. It features a central cluster of interlocking gears in shades of blue and grey. Surrounding these are several shields, some with padlocks, in various colors like teal, yellow, and orange. The background is filled with a complex network of glowing lines in purple, blue, and orange, suggesting a digital or data-driven environment. There are also some circular icons, like a lightbulb and a network diagram, scattered throughout.

10. A Pragmatic Implementation Roadmap

*Security Posture
Mature*

For organizations that are starting an AI governance program from a low baseline, the initial question is not which pillar to build first but in what order the pillars reinforce each other. The sequence matters. Building some capabilities before others tends to produce cleaner outcomes than the reverse.

The following roadmap is not the only reasonable one, but it reflects a pattern that has held up across a range of organizational contexts.

Phase one: Discover

The first step is an honest inventory of what the organization is already doing with AI. This typically surprises the people conducting it. The formal deployments are easy to find. The informal usage, the agents that a team spun up last quarter, the copilot integrations that arrived with a routine software update, the browser extensions that employees have been using, are harder. A serious discovery effort combines network-level signals, survey data, and interviews, and it produces a document that is almost always longer than expected.

The discovery phase also surfaces the data flows that govern current usage. Which systems send what kinds of data to which models, under which terms. This map is the foundation for everything that follows, and attempts to proceed without it tend to produce policies that do not match reality.

Phase two: Measure

With an inventory in hand, the next step is instrumentation. Observability comes before control, for a simple reason. A control that is put in place without measurement cannot be validated. The organization that puts guardrails in place before it can observe the effect is operating by faith.

The measurement phase establishes the baseline: what does current usage look like, what does it cost, where are the failure modes, what categories of risk are actually materializing. This baseline will become the reference against which all subsequent changes are evaluated.

Phase three: Control

Once the baseline is in place, the first controls can be introduced. The order within this phase is usually dictated by risk. Input-side guardrails for sensitive data handling tend to come first, because the downside of getting this wrong is large and the cost of getting it right is modest. Basic cost controls, in the form of per-application budgets and circuit breakers, come next. Output guardrails for the highest-risk categories, such as legal or financial advice generated on behalf of the organization, come into place as the detection mechanisms mature.

The critical discipline in this phase is to treat each control as an experiment. The baseline established in the measurement phase predicts what the control should change. If the measured effect does not match the prediction, something is wrong, and the control should not be scaled until the mismatch is understood.

Phase four: Optimize

With measurement and basic controls in place, attention turns to optimization. This is where the cost-per-useful-outcome work happens. Where the routing logic is introduced or refined. Where the team begins asking not just whether the system works but whether it works as efficiently as it should. The gains in this phase are often significant, and they are only reliably captured once the earlier phases have established the instrumentation that allows the gains to be measured.

Phase five: Automate

The final phase is the point at which the control plane becomes largely self-operating. Policies are version-controlled and deployed through automation. Cost anomalies trigger alerts without human intervention. New applications onboard by registering with the control plane rather than by writing custom integrations.

Governance becomes a feature of the platform rather than a separate program that teams have to remember to comply with.

Few organizations are fully in this phase today. Most are somewhere in the second or third. But the shape of the path is clear enough that the investment pays off in predictable ways at each stage, and the organizations that have been furthest along have tended to report that the later phases are easier than the earlier ones, because the earlier phases have done the work of establishing shared infrastructure and shared expectations.


Anti-patterns

A small number of anti-patterns recur across implementation efforts often enough to be worth naming.

The first is to start with policy documents rather than with measurement. A comprehensive policy document written in the absence of data about actual usage tends to be either too permissive, because the authors underestimate what is happening, or too restrictive, because they overestimate it. In either case, the document fails to reflect reality and is quickly ignored.

The second is to build the routing layer before the observability layer. Routing without observation is a commitment without evidence. The organization ends up moving traffic between providers based on assumptions rather than data, and the effect on cost, quality, or latency cannot be measured afterward.

The third is to treat AI governance as a one-time project with a completion date. The technology continues to change. The threats continue to evolve. The costs continue to shift. A program that declares victory after the initial implementation is a program that will find itself ungoverned again within a year.

A tall, futuristic tower with a glowing blue beam of light running vertically through its center, set against a cloudy sky and reflected in water.

11. Measuring What Matters

An organization that has invested in an AI governance program will eventually be asked, by its leadership or its board or its auditors, whether the investment is working. The question deserves a serious answer, and a serious answer requires a specific set of metrics. This chapter sketches the ones that have proven most durable in practice.

Operational metrics

The first layer of metrics tracks the mechanical behaviour of the system. Request volume, broken down by application and user cohort, establishes the baseline of activity. Latency distributions, broken down by model and request type, reveal performance issues. Error rates, broken down by provider and error class, reveal reliability issues. Token consumption, broken down by cost center, enables cost attribution.

None of these metrics is surprising. They are the AI equivalent of the metrics that would be tracked for any other production system, and they should be unremarkable once the instrumentation is in place.

Governance metrics

The next layer is specific to governance. It includes the rate at which guardrails are triggered, broken down by type. A sudden rise in prompt-injection detections may indicate that an application is under attack. A sudden fall in sensitive-data redactions may indicate that the detection rules have been silently broken. The absolute numbers are less meaningful than the trends and the ratios.

It also includes the rate of policy violations, measured as the fraction of requests that were blocked or modified because they conflicted with active policies. A healthy system has a non-zero violation rate, because policies that never trigger are either unnecessary or not being applied. An unhealthy system has a violation rate that is rising without explanation, which typically indicates either a change in user behaviour or a drift in system behaviour that deserves investigation.

The mean time to detect a policy issue, and the mean time to resolve it, are operational indicators of the governance program's responsiveness. These should trend downward over time as the program matures and as the tooling improves.

Economic metrics

The economic layer captures whether the investment in AI is producing value commensurate with the cost. Cost per interaction is the basic figure. Cost per useful outcome, as discussed in the FinOps chapter, is more informative. The ratio of AI-handled interactions to human-escalated interactions, for applications where this distinction applies, is a measure of the system's practical effectiveness.

Budget variance, which compares actual spending against the forecast, is the metric most relevant to finance leadership. A program with consistently small variances is a program that understands its cost drivers. A program with large variances, regardless of the direction, has unfinished measurement work to do.

Risk metrics

The final layer captures risk. It includes the number of confirmed incidents, broken down by severity. It includes the number of near-misses, meaning events that would have been incidents had the guardrails not caught them. It includes the frequency and outcome of audit activities, both internal and external. In regulated contexts, it includes the organization's performance against the specific metrics required by the relevant framework.

Risk metrics are harder to interpret than the other categories, because the absence of incidents is not the same as the absence of risk. A program that has had no incidents may be well-governed, or it may be lucky, or it may be under-reporting. Triangulating across the operational, governance, and economic metrics gives a more reliable picture than any single risk figure would.

The maturity question

A useful exercise, performed periodically, is to assess the program against a maturity model. Several such models have been published, and their details vary, but a simple internal version can be constructed from the material in this book. An organization that has completed the discovery and measurement phases for all major applications, implemented controls across at least the highest-risk categories, and established a routine cadence for policy review and metric reporting is at a recognizable level of maturity. An organization that has automated most of these processes and is operating the program as part of its platform rather than as a separate function is at a higher level.

What matters, in the end, is not the level on the model but the direction of motion. A program that is progressing is a program that is working. A program that has stalled, regardless of where it stalled, is a program that needs attention.



Conclusion: The Organized AI Organization

The difference between an organization that has its AI in order and one that does not is, in the beginning, subtle. The two organizations may be using the same models, pursuing the same use cases, and producing similar demos. The divergence becomes visible only when something happens: a cost spike, a compliance review, a regulatory inquiry, an incident that becomes public. At that moment, one organization has the instrumentation, the records, the policies, and the mechanisms to respond. The other has a scramble.

The material covered in this book will not by itself produce the first kind of organization. The pillars, the patterns, and the roadmaps are scaffolding. The actual work is done by people who decide, often under pressure and often without immediate reward, to invest in the boring machinery of production responsibility. They are the engineers who insist on instrumenting the system before shipping it. The finance partners who ask for cost attribution before approving the budget. The compliance officers who translate external frameworks into executable policies. The platform teams who build the control plane so that individual application teams do not have to. The leaders who protect the budget for this work when the demos are getting attention and the governance is not.

It is not obvious, in the early years of a technology wave, which organizations are building this foundation and which are not. The distinction usually becomes visible later, when the wave has crested and the question turns from who moved fastest to who moved most durably. The quiet work of governance is what determines the answer to the second question, and the first question, in retrospect, turns out to matter less than it felt at the time.

The real question about AI in the enterprise is not whether it will be transformative. It is whether the transformation will be controlled or chaotic, deliberate or accidental, sustainable or brief. Governance is how the first option in each pair is reached.

The work is not finished. The technology will continue to evolve, the threats will continue to shift, the frameworks will continue to update. But the shape of the discipline is now clear enough to act on, and the organizations that act on it are building something that will compound. The ones that do not will find themselves, eventually, explaining to someone why.

That is what this book has tried to describe. The foundation for AI in production that is organized rather than accidental. The quiet revolution, in other words, of doing this well.

About the Author

Fabio Bastos is a Brazilian journalist and technologist who has worked at the intersection of media and emerging technology for more than two decades. His early career was in live internet broadcasting, including one of the first streamed television projects in Latin America, and in independent television and radio production. He later operated in the early virtual-worlds industry and has since worked on several ventures in emerging technology.

He is the founder of ThinkNEO, an enterprise AI control plane platform — runtime guardrails, observability, FinOps, governance, and provider-agnostic routing — the pattern described in the central chapters of this book. ThinkNEO is a member of NVIDIA's Inception program and the Anthropic Partner Network.

He writes about the operational challenges of deploying artificial intelligence at scale, with a focus on the infrastructure, observability, and governance questions that tend to receive less attention than model capability. He lives and works across Brazil and Southeast Asia.

Glossary

Agent. A software system that uses a language model to plan and execute multi-step actions, often by calling tools or other systems, rather than simply returning a single response.

AI FinOps. The discipline of measuring, allocating, forecasting, and controlling variable spending on AI services, particularly token-based model usage. Adapted from the broader FinOps practice that emerged in the cloud era.

Control plane. The layer of a system that holds policies and makes decisions about how traffic should flow, as distinct from the data plane, which actually moves the traffic. In AI systems, the layer where guardrails, observability, cost management, routing logic, and audit trails reside.

Data plane. The part of a system that moves traffic or performs the main work, as distinct from the control plane, which makes decisions about how.

Fallback chain. An explicit ordered list of alternative models or configurations to try when the preferred one is unavailable, often paired with a degradation policy that specifies what capability is lost at each step.

FinOps. A practice, originating in the cloud era, of managing variable spending collaboratively across finance, engineering, and operations.

Generative AI. A category of artificial intelligence models that produce new content — text, images, audio, code — in response to inputs, typically built on large neural networks.

Guardrails. Mechanisms that sit between a user and a model, or between a model and downstream systems, with the authority to intercept, modify, or refuse traffic. Input guardrails operate on what is sent to the model; output guardrails operate on what the model produces.

Hallucination. A model output that asserts something as fact when the model has no basis for doing so. A common failure mode of generative AI systems.

Judge model. A smaller or faster model configured to evaluate the output of another model, returning a verdict such as permit, block, or flag. Sometimes called LLM-as-judge.

Large language model (LLM). A neural network trained on large quantities of text, capable of producing and understanding natural language.

Observability. The property of a system that allows someone to understand, from its external outputs, what is happening inside it. Distinct from monitoring, which watches for known failures.

OpenTelemetry. An open standard and associated tooling for collecting metrics, logs, and traces from software systems. Widely adopted for observability, with emerging extensions for AI-specific signals.

Policy-as-code. The practice of expressing policies in a structured, machine-evaluable form rather than as documents, so that a policy engine can enforce them at runtime and produce audit records.

Prompt. The input — typically text — given to a language model, including any instructions, examples, or context provided to shape the response.

Prompt injection. An attack in which an input contains instructions intended to subvert the model's original instructions, causing it to follow the attacker's goals rather than the operator's.

Provider-agnostic routing. Infrastructure that sits in front of multiple model providers and selects, per request, which one to use, based on capability, cost, compliance, or availability.

Semantic telemetry. Observability data that captures information about the meaning of a request or response — such as which guardrails fired, what content was redacted, or how an output was evaluated — rather than only its mechanics.

Shadow AI. Unsanctioned use of AI tools by employees, typically public chatbots, outside the organization's visibility or policy. The generative AI analogue of shadow IT.

Token. A unit of text processed by a language model. Commercial models are typically priced per thousand tokens, with separate rates for input and output.

Trace. An observability signal that captures the path of a single request through a distributed system, including the calls it makes to other services.

Vector database. A database optimised for storing and querying high-dimensional vector representations of content, commonly used in retrieval steps that supply context to language models.

Further Reading

The works below are referenced directly in the text or are closely adjacent to the topics it covers. The list is deliberately short. Most of what an operator needs to learn will come from reading incident reports inside their own organization, not from additional books.

Cited works

Moffatt v. Air Canada, 2024 BCCRT 149. Civil Resolution Tribunal of British Columbia, February 2024.

Regulation (EU) 2024/1689 of the European Parliament and of the Council, the Artificial Intelligence Act.

National Institute of Standards and Technology. AI Risk Management Framework 1.0 (NIST AI 100-1). January 2023.

International Organization for Standardization and International Electrotechnical Commission. ISO/IEC 42001:2023, Information technology — Artificial intelligence — Management system. 2023.

The OpenTelemetry project. <https://opentelemetry.io>

Adjacent practitioner literature

Beyer, Betsy, et al. Site Reliability Engineering: How Google Runs Production Systems. O'Reilly, 2016. A foundational text on operating production systems, whose discipline extends naturally to AI workloads.

FinOps Foundation. <https://www.finops.org> — the established body for cloud FinOps, many of whose practices extend to AI FinOps.

OWASP. Top 10 for Large Language Model Applications. <https://owasp.org/www-project-top-10-for-large-language-model-applications> — a regularly updated catalogue of common vulnerabilities in LLM-based systems.

National Institute of Standards and Technology. Artificial Intelligence Risk Management Framework: Generative Artificial Intelligence Profile (NIST AI 600-1). 2024.

Sharing, Feedback, and Contact

This book is distributed free of charge under a Creative Commons Attribution 4.0 International license. You are free to share it, translate it, adapt it, quote it, and build on it, provided you credit the author. Translations, critical responses, and organisational adaptations are welcome.

Comments, corrections, and war stories are genuinely wanted. They are what keep a book like this honest between editions.

Readers interested in the control plane pattern described in Chapter 9 — as a product rather than a concept — can find more about ThinkNEO at thinkneo.ai. The company builds the enterprise AI control plane this book describes.

If the book was useful, the most helpful thing you can do is pass it to the colleague who is about to face the situation in Chapter 1.

— FB

AI is already in production. The question is no longer whether to govern it, but how.

A field guide to AI governance as operational practice — not compliance theater, not a checklist of ethical principles, but the daily work of running AI systems with the rigor organizations already apply to financial and industrial systems where the consequences of failure are real.

Drawn from conversations with engineers, compliance officers, finance partners, and platform leads. Lays out the five technical pillars now recognizable across organizations and vendors as the things that have to be in place, and a pragmatic roadmap for starting from scratch.

Short on purpose. Written in the middle of the work it describes.

FABIO BASTOS Founder, ThinkNEO

THINKNEO PUBLICATIONS * FIRST EDITION · 2026
Licensed under CC BY 4.0 · thinkneo.ai