The Architecture Behind an Owned AI Stack

Six Entry Points, Two Doors

Whitepaper, November 2026

45degrees
CRONOS GROUP

Executive Summary

Most enterprise AI conversations start with a model. Which model, from which vendor, at which price. That is the wrong starting point, because the model is the part of the stack you will replace most often. The parts you have to own are the ones around it.

This paper describes those parts. It takes the six ways people and systems actually use AI inside an enterprise and draws all six on one architectural template. Five of them are entry points that a person or a system uses. The sixth is not a way of using AI at all. It is the loop that tells you what to switch off.

The template has two governed doors. **Door 1 is the AI Gateway**, and every model call goes through it. **Door 2 is the MCP Gateway**, and every call to a tool, a knowledge source or an internal system goes through it. Both doors log to one governance layer. What changes per use case is the entry point, the rules you set at each door, and what sits behind them. The doors themselves do not change.

That is the whole idea, and it has one practical consequence worth stating early. If you build this, your six AI use cases stop being six separate projects with six separate security reviews, six cost surprises and six vendor dependencies. They become six front doors on one platform you control.

This paper is written for a CIO who already understands the general concept and wants the architecture underneath it: what the components are, what rules live where, what each entry point adds, and what it costs you to run. It does not argue the business case. That argument is made in two companion papers, **The AI Nobody Is Selling You** and **What an Owned AI Stack Buys You**.

From the Model to the Architecture

Three years of enterprise AI have produced a familiar pattern. A department buys a seat-based assistant. A development team adopts a coding tool. A product team wires a model API into an application. Each decision is reasonable on its own. Together they produce an estate that nobody can describe, price or secure.

The reason is structural, not managerial. Each of those decisions created a direct connection from something inside your organization to something outside it. A direct connection carries its own credentials, its own log file if any, its own price list and its own roadmap. Multiply that by a few dozen and you have an estate made of point-to-point links.

Enterprise IT solved this class of problem before. You do not let every application talk to every database, so you have a data platform. You do not let every service call every other service over the public internet, so you have an API gateway and an integration layer. You do not let every team manage its own identities, so you have an identity provider. In each case the answer was the same: put a layer in the middle that you own, and make the thing behind it replaceable.

AI is the same problem with a shorter history. The layer in the middle is the gateway. The thing behind it that becomes replaceable is the model. Everything in this paper follows from that one move.

There is a second reason to start from the architecture rather than the model. Model quality changes every few months, and so do prices. Any design that depends on picking the right model will be wrong within a year. A design that assumes the model will change is stable, because changing the model becomes a configuration decision at one point in your estate instead of a migration across many applications.

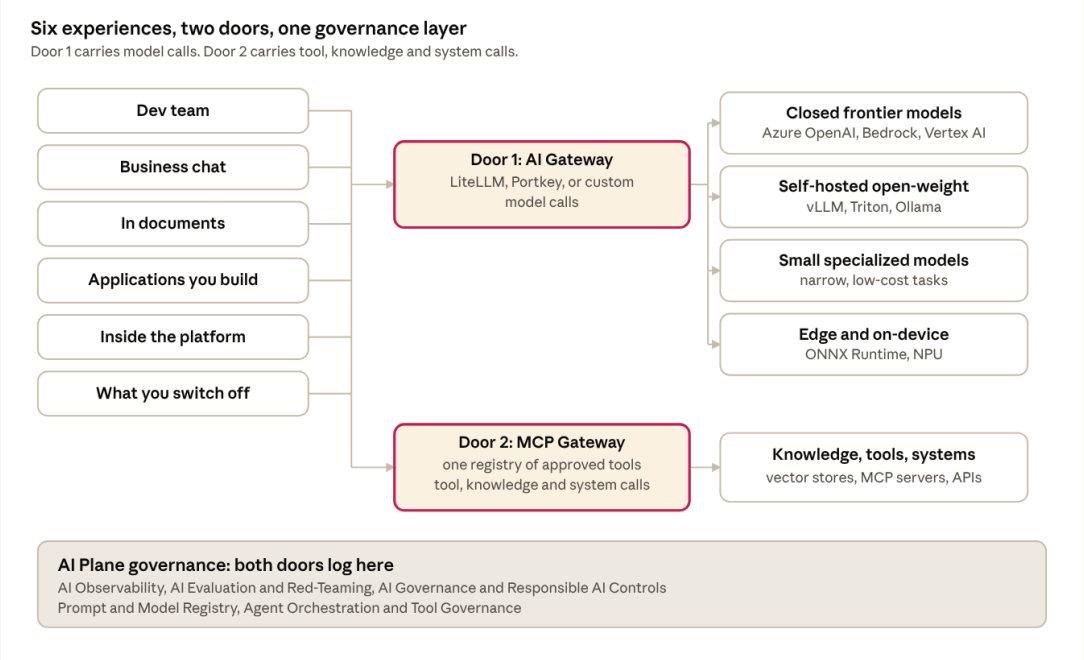
The Template: Two Doors and One Governance Layer

On stage we say this as one sentence: put all of your AI through one door. That is the right instruction, and it is worth being precise about what it means once you start drawing it.

AI traffic inside an enterprise is not one kind of traffic. It is two.

- The first kind is a **model call**. Text, or an image, or an audio stream goes out, and a completion comes back. It is stateless, it is metered by tokens, and it is the traffic that appears on your invoice.
- The second kind is a **tool, knowledge or system call**. An agent reads a document store, opens a ticket, queries a database, calls an internal API. It is stateful, it touches your data, and it is the traffic that appears in your incident reports.

Those two kinds of traffic need different rules, different identities and different failure handling, so architecturally they get two doors.



The template that every section of this paper reuses: six entry points, two governed doors, four options in the model inference layer, and one governance band underneath.

Whether those two doors are two products is an implementation choice, not an architectural one. Some products cover both. LiteLLM, for example, offers model routing and MCP gateway features in one product, according to its own materials. If you write your own gateway as custom code, you can also put both behind one deployment. So the precise statement is this: **two doors in the architecture, one or two products in the build.** The sentence from the stage still holds in the sense that matters, which is that nothing reaches a model or an internal system without passing through infrastructure you own.

Door 1: The AI Gateway

Door 1 sits between every entry point and every model. It is an HTTP service that speaks to a model API, so from the caller it looks like a model provider. Behind it, it does four things that a model provider will not do for you.

- **Routing.** A logical model name, for example "general-chat" or "code-assist", is mapped to a physical model in the inference layer. The caller never names a vendor. You change the mapping in one place.
- **Policy.** An approved model list, a cost budget per team and per application, rate limits, and a response filter. Policy lives at the door, not in the client.
- **Resilience.** A fallback when a provider is slow, rate limited or down. A pinned version so a silent vendor update does not change your behaviour overnight.

- **Accounting.** Every call recorded with the caller identity, the team, the model, the token count and the cost. This is the record your finance function has never had.

Behind Door 1 sits the model inference layer, which the 45 Degrees platform engineering framework calls Model Serving and Inference. It comes in four kinds, and most organizations will end up with more than one. A closed vendor API, typically Azure OpenAI, Amazon Bedrock or Google Vertex AI, for general-purpose work. A self-hosted open-weight model served on vLLM, for high-volume, sensitive or cost-sensitive work. A small specialized model for narrow and repetitive tasks. And, where latency or privacy demands it, an on-device or edge model running through a runtime such as ONNX Runtime, sometimes on a dedicated neural processing unit.

The important property is that the choice between those four is a routing policy at Door 1. It is not a decision taken in each application, and it is not visible to the user.

Door 2: The MCP Gateway

Door 2 sits between every agent and everything that agent can act on. The Model Context Protocol has become the common way to describe a tool to a model, and it is now governed under the Linux Foundation, which matters for an architecture you intend to keep for years. Door 2 is the registry and the control point for those tools.

- **One registry of approved tools.** If a tool is not in the registry, no agent can call it. This is the AI equivalent of an approved software catalogue.
- **Non-human identity, with least privilege.** Each agent gets its own identity, not a shared service account and not the identity of the developer who deployed it. Scopes are narrow and mostly read-only by default.
- **Scoped access that follows the user.** Where a tool returns data, the result respects the access rights of the person who asked. This is the single most important rule at Door 2, and the next section explains why.
- **One audit trail.** Which agent called which system, under which identity, with which arguments. Without this, an AI incident is unreconstructable.

Products that list MCP gateway features include LiteLLM and TrueFoundry, according to their own materials. The space is young, so check current capabilities against your own requirements before you choose, and treat a custom build as a legitimate option rather than a failure.

The Governance Layer

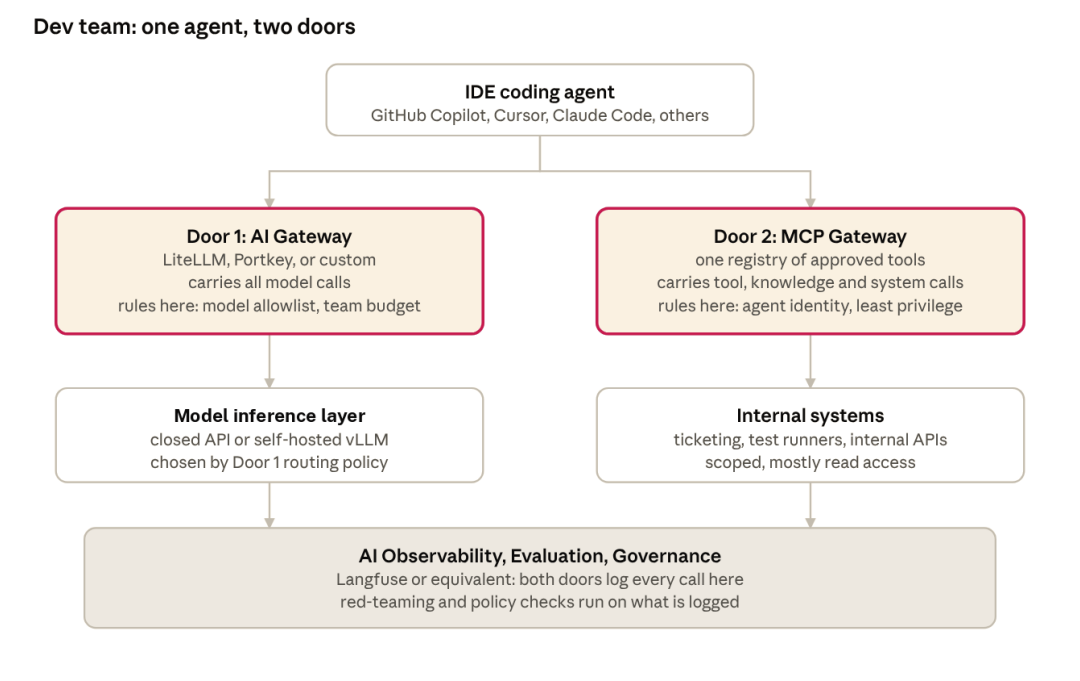
Both doors log to one place. In the 45 Degrees platform engineering framework this is the AI Observability component, and Langfuse is a common implementation, although the choice matters less than the discipline of having exactly one instance. Next to it sit two components that only work if observability is already there: AI Evaluation and Red-Teaming, which gates

a new agent or model version before wide rollout, and the Prompt and Model Registry, which versions prompts and model configurations the way you version source code.

One instance is the point. Six use cases logging to six dashboards is the same estate you started with, just better instrumented.

Entry Point One: Dev Teams

One coding agent, two jobs. It asks a model for code through Door 1, and it acts on your real systems through Door 2. Neither path is a direct, ungoverned connection.



Dev team: an IDE agent, the two doors, and the governance layer.

This is usually the first entry point an enterprise builds, for two reasons. It has the clearest value, and developers will adopt a governed path if it is not slower than the ungoverned one. The building blocks, in order:

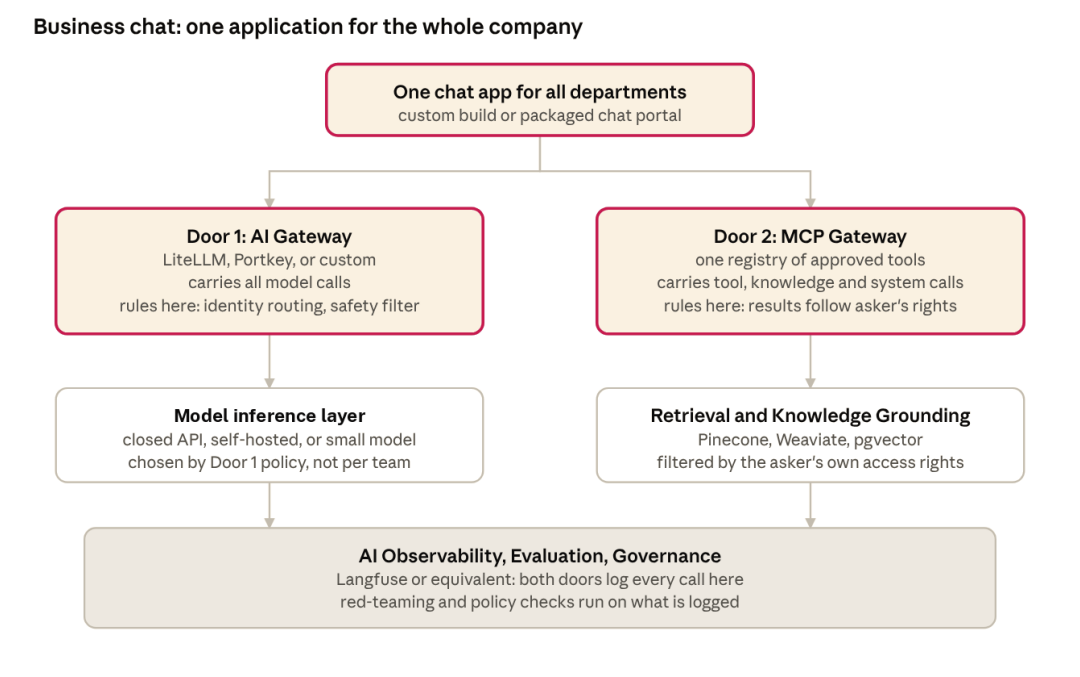
1. **Entry point.** The IDE coding agent. GitHub Copilot, Cursor, Claude Code, JetBrains AI Assistant, Amazon Q Developer and Tabnine are all in this category. Teams choose from an approved catalogue. You do not build this part.
2. **Door 1: AI Gateway.** Rules that matter here are the approved model list, a cost budget per team, and a fallback when a provider is down. A development team that loses its assistant for an afternoon loses an afternoon.
3. **Model inference layer.** A closed API, or a self-hosted open-weight model on vLLM for teams working on sensitive code. The Door 1 routing policy decides this, not the IDE.

4. **Door 2: MCP Gateway.** This is where a coding model becomes an agent. It reads tickets, runs tests, opens pull requests. Each agent has its own non-human identity with least privilege.
5. **Internal systems.** Ticketing, test runners and internal APIs, each exposed through Door 2 with scoped and mostly read access.
6. **Observability and governance.** Both doors log to the same instance. A new agent or model version passes AI Evaluation and Red-Teaming before it goes wide.

The design question that decides whether this entry point succeeds is not which IDE tool you approve. It is how much write access you give an agent at Door 2. Read-heavy scopes with a human approving every change are slower but safe, and they are the right place to start. Widening them is a later decision, taken per system, with the audit trail from the first phase as evidence.

Entry Point Two: Business Chat

Every department uses the same chat application. It is not one app for legal and another for HR. Division-specific behaviour happens behind that single front door, through access control, not through separate tools. That single front door is also your control point. Without it, staff use a public AI tool and your data leaves the organization.



Business chat: one application, the same two doors, scoped knowledge behind Door 2.

1. **Entry point, one instance.** One chat application for all employees, either custom built or a packaged chat portal.
2. **Door 1: AI Gateway.** It reads the identity and the department of the person asking from your identity provider. Rules here are identity-based routing, rate limits, full logging and

a safety filter on the response. This is what makes a public AI tool unnecessary, and detectable if someone uses one anyway.

3. **Model inference layer.** A closed API for general questions, or a smaller self-hosted model for narrow, high-volume or sensitive ones. Door 1 policy decides, not the team.
4. **Door 2: MCP Gateway.** In this use case it fronts the knowledge sources. Behind it sits Retrieval and Knowledge Grounding, implemented on a vector store such as Pinecone, Weaviate or pgvector. The rule here is that results follow the access rights of the person asking.
5. **Observability and governance.** The same instance as every other use case.

The department-specific assistant that business stakeholders ask for is step 4, not a separate interface. One front door, many scoped answers behind it. Someone in legal retrieves from legal documents, and someone in HR does not, from the same application.

How those rights are enforced is a design decision, and it is not a trivial one. In the ideal case the retrieval path acts on behalf of the user and inherits the rights of the underlying system. A document the user cannot open in SharePoint is a document the assistant cannot return, because one permission model decides both. Most gateways also allow you to define the rights in the gateway itself. That is quicker to build, and it gives you a second copy of your permission model which then has to be kept in step with the first. This paper does not try to settle that question. It flags it, because it is the choice in this entry point that is most often made by accident rather than on purpose.

Putting retrieval behind Door 2 is a design choice worth making explicitly. The alternative is a retrieval service that the chat backend calls directly, which is simpler to build. Behind Door 2, retrieval gets the same identity, the same logging and the same rules as every other tool call, which is worth the extra hop.

The Permission Lesson This Entry Point Has to Learn

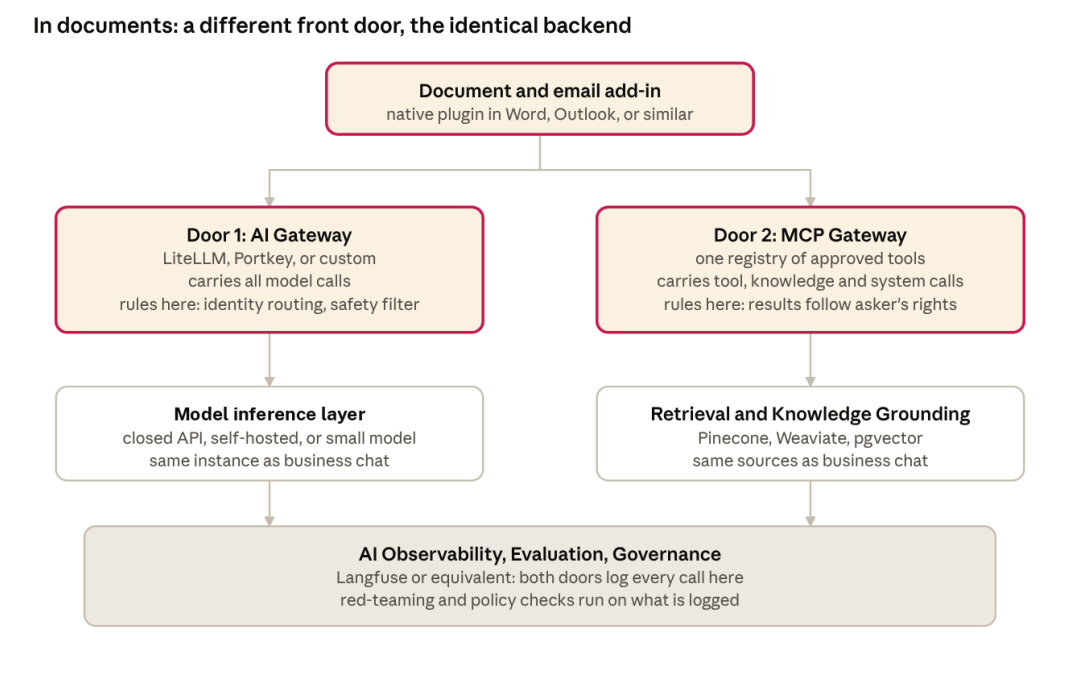
There is an older lesson in enterprise IT that applies here exactly. A decade ago Microsoft shipped a product called Delve, which surfaced documents a colleague had access to but had never thought about. The technology worked as designed. The problem was that it made years of over-broad file permissions visible all at once, and the organizational reaction was severe enough that many enterprises switched it off rather than fix the permissions underneath.

A retrieval-backed assistant does the same thing, faster and in natural language. It will find the salary spreadsheet in the open folder. It will summarize the reorganization document that was shared with the wrong group. It does not create the exposure, it reveals it.

Architecturally the answer is the rule in step 4, enforced at Door 2 rather than inside the chat application. Operationally, it means a permissions review is part of the project plan for this entry point and not a follow-up item. Teams that skip it discover the same thing enterprises discovered with Delve, which is that the fastest available fix is to switch the assistant off.

Entry Point Three: In Documents

AI appears inside Word or Outlook, where people already work, instead of in a separate window. It is a different front door on the same backend as the business chat. It is not a second platform to build or secure.



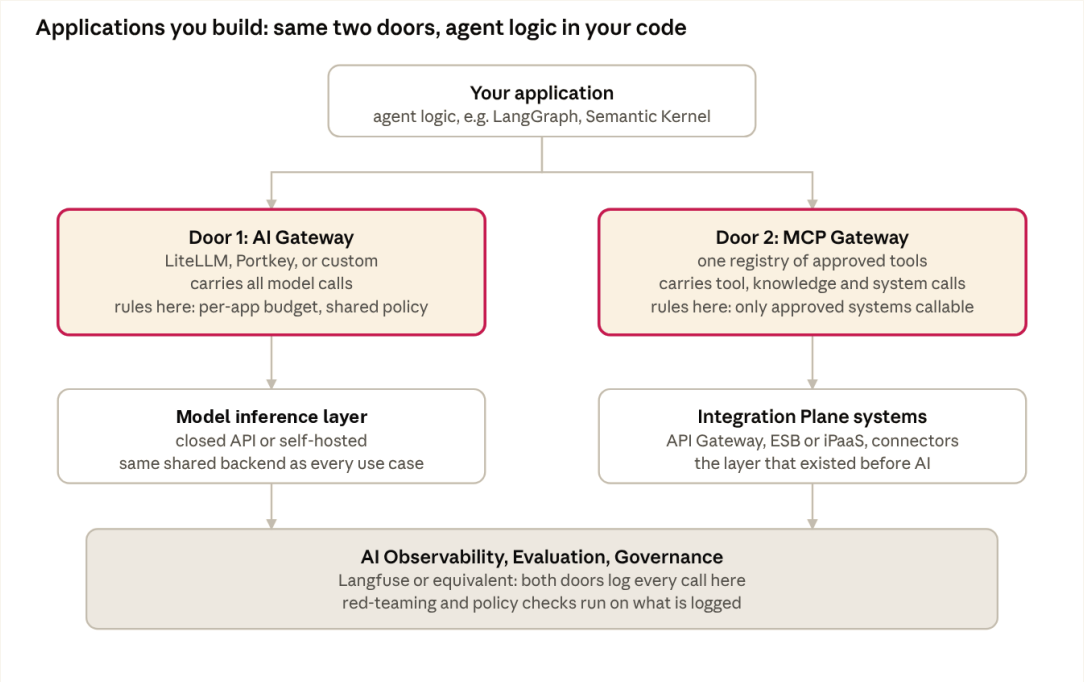
In documents: a different entry point, identical doors and backends.

1. **Entry point.** A native add-in in Word, Outlook or a similar tool. It is not a standalone application.
2. **Both doors, unchanged.** The add-in calls the same Door 1 and the same Door 2 as business chat, with the same rules and the same knowledge sources. One backend, several front doors.
3. **One risk specific to this front door.** A prompt inside an add-in is easy to forget. The Prompt and Model Registry versions it like source code, so every change is reviewable and you know what is running in production.

This section is short because the architecture is short. That is the point of the template. Once Door 1 and Door 2 exist, a new entry point is an integration job of days or weeks, not a platform project. The business reads it as a new capability. Your architecture reads it as one more client of something that already runs.

Entry Point Four: Applications You Build

AI inside a product your own teams build uses the same two doors as everything else. There is no separate path because the request comes from your own code.



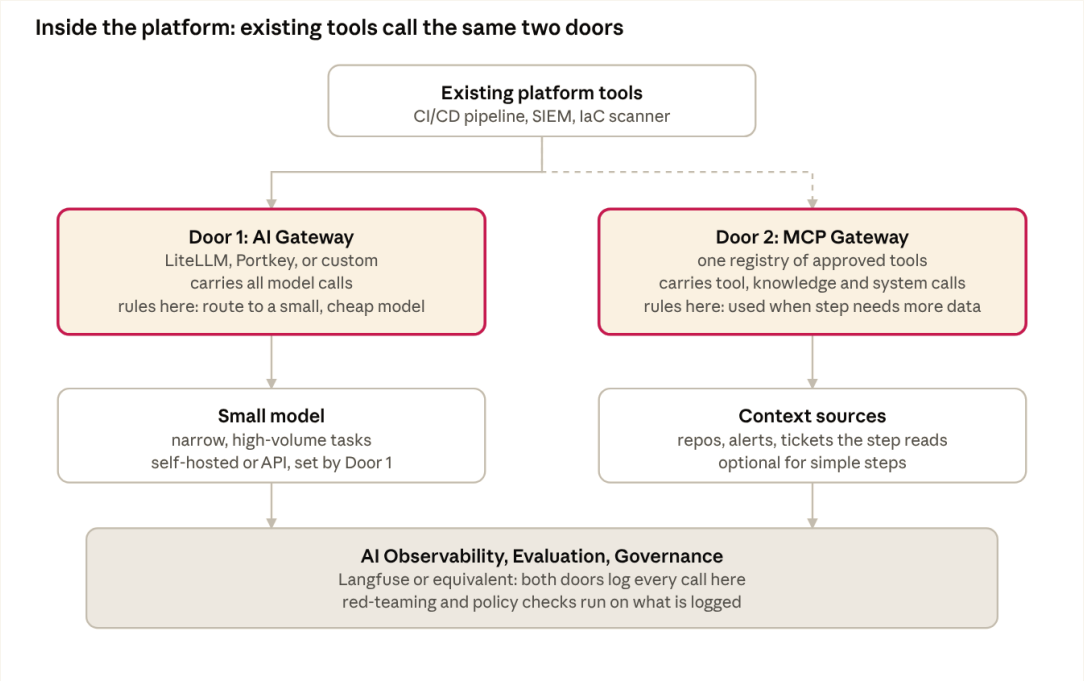
Applications you build: the same template as the dev team, with your own agent logic as the entry point.

1. **Entry point.** Your application, with its own agent logic built on a framework such as LangGraph, Semantic Kernel or AutoGen. This logic belongs inside your application. It is not a separate platform layer, and treating it as one is a common and expensive mistake.
2. **Door 1: AI Gateway.** The same instance as every other use case. Rules here are a budget per application and the shared model policy.
3. **Model inference layer.** The same shared layer. A closed API or a self-hosted model, decided by routing policy.
4. **Door 2: MCP Gateway.** The same registry as every other use case. The rule here is that the application can only call approved systems.
5. **Integration Plane systems.** Behind Door 2 sit your API gateway, your ESB or iPaaS, and your connectors. This layer existed before AI. Door 2 is what makes it callable by an agent, safely and with an audit trail.
6. **Optional: Fine-Tuning and Model Customization.** Only when prompt engineering and retrieval are not enough. It stays thin against a closed API and becomes a real training pipeline with a self-hosted model, with the cost and the team that implies.
7. **Observability and governance.** Both doors log to the same instance. One dashboard shows all AI in the enterprise, including the AI inside your own products.

Two architectural notes. First, the agent logic in step 1 is application code and should be owned, versioned and tested by the application team. Centralizing it creates a bottleneck that will be bypassed. Second, step 5 is where most of the real value of this entry point lives. An AI feature that can only generate text is a demonstration. An AI feature that can read your order book and act on it is a product, and the difference between the two is entirely a Door 2 question.

Entry Point Five: Inside the Platform

This is AI speeding up tools you already run, such as a build pipeline or a security console. It is not a new product with its own interface. It uses the same two doors, called by a machine instead of a person.



Inside the platform: existing tools call the same doors, with Door 2 used only when the step needs more data than the tool already sends.

1. **Entry points, plural.** Existing tools in the other eleven planes of your platform that add an AI step. A CI/CD pipeline reviewing a pull request, a SIEM triaging an alert, an infrastructure-as-code scanner checking a template. There is no new list of tools to buy.
2. **Door 1: AI Gateway.** The same instance as every other use case. The rule that matters here is to route to a small, cheap model.
3. **Small model.** Most of this work is narrow and high volume: flag a file, summarize a diff, correlate alerts. A small model fits, and Door 1 routes it there by policy rather than by a decision inside each tool.
4. **Door 2: MCP Gateway, optional here.** It is used when the step needs more data than the tool sends with its request. Same registry, same rules.

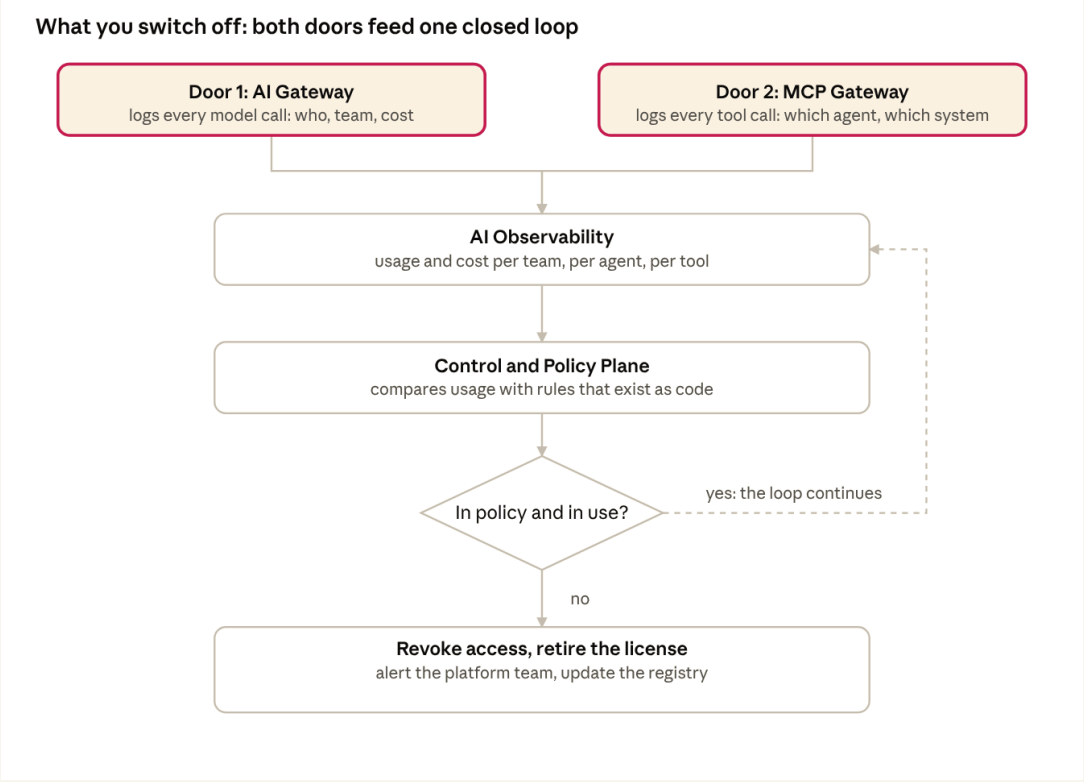
5. **Result returns to the tool that asked.** A pull request gets its review comment. A security queue gets its triage note. Nobody waits in a chat window.
6. **Observability and governance.** Both doors log to the same instance as every other use case.

This entry point is the one most often missed in an AI strategy, because it has no user interface and no vendor selling it to a business sponsor. It is also the one that most often pays for itself, because the volume is high, the tasks are narrow and a small model does them at a fraction of the cost of a frontier model.

It carries one specific risk. These calls are machine-to-machine, so they scale without anyone noticing. A pipeline that runs a thousand times a day will make a thousand model calls a day, and if it is routed to an expensive model by mistake, nobody will complain until the invoice arrives. This is a budget and alert rule at Door 1, and it should be set before the first pipeline is connected.

The Sixth: What You Switch Off

This is not a sixth way to use AI. It is a loop that runs continuously under the other five. It tells you which licence or workload to retire, based on evidence and not on a guess.



What you switch off: both doors feed one governance loop that ends in a decision.

It belongs in this paper for a simple reason. Every other section adds something to your estate. Without this loop, nothing is ever removed from it, and in three years you will be in the same position you are in today, with a larger bill and a better dashboard.

1. **Door 1 logs every model call.** Who called, which team, which model, what it cost. This is the gateway doing its normal job, not an extra reporting project.
2. **Door 2 logs every tool and data call.** Which agent, which system, under which identity.
3. **AI Observability combines both logs.** Usage and cost per team, per agent and per tool, in the same instance every other use case writes to.
4. **The Control and Policy Plane compares usage with the rules.** This step only works when policy exists as code. A rule written only in a slide or a PDF cannot be checked automatically, and will not be.
5. **A decision.** In policy and in use: nothing happens and the loop continues. Out of policy, or a seat nobody has opened for months: the action step starts.
6. **Revoke, retire, alert.** An agent loses access, an unused licence is flagged to the platform team, and the registry is updated for the next pass.

Step 4 is where most organizations stop, and it is worth being honest about why. Writing policy as code means deciding, in precise terms, what a reasonable amount of AI spend per team is, which models are allowed for which data classification, and how long an unused seat may stay unused. Those are governance decisions, not engineering ones, and the engineering cannot proceed without them.

Step 6 is where it becomes political. Retiring a licence that a vocal department asked for is a conversation, even with evidence. The loop does not remove that conversation. It changes it from an argument about opinions into a review of measured usage, which is a conversation you can win.

What This Costs You to Run

This architecture is not free, and a paper that implied otherwise would not be useful. The costs are predictable, which is the actual benefit, but they are real.

The doors are platform services. They need an owner, a budget, an availability target and an on-call rotation, like your API gateway or your identity provider. Both doors are also on the critical path of every AI feature you run, which makes them single points of failure. That is not an argument against them. Your identity provider and your primary database are single points of failure too, and you manage them with redundancy, monitoring and a tested failover rather than by avoiding them.

The governance layer needs discipline, not headcount. One observability instance, one registry and an evaluation gate are small to build and easy to let rot. The failure mode is not an outage. It is a dashboard nobody reads and a registry that no longer matches reality, which quietly returns you to an ungoverned estate while the architecture diagram still says otherwise.

Self-hosting is a separate decision with a separate cost. Door 1 lets you serve open-weight models on your own infrastructure, which is where the significant cost reductions are

available for high-volume work. It also means capacity planning, GPU procurement or reservation, model lifecycle management and a team that can operate vLLM. Build Door 1 first and route everything to a closed API. Self-host later, when your own logs tell you which workloads justify it.

The skills question is real. This is platform engineering work: networking, identity, policy as code, observability. It is not data science. Most enterprises already have the people, in the platform team rather than in an AI team, and the common mistake is to staff this from the wrong place.

Where to Start

The order matters more than the speed, because each step makes the next one cheaper.

1. **Build Door 1 with one entry point.** Pick the dev teams, because they adopt fast and give direct feedback. Route everything to a closed API. Do not self-host anything yet.
2. **Turn on observability on day one.** Not in phase two. The logs from the first weeks are what justify every later decision, and you cannot reconstruct them afterwards.
3. **Move the business chat behind Door 1.** This is where shadow AI stops, and where you find out how much of it there was.
4. **Add Door 2, starting read-only.** One registry, narrow scopes, a human approving writes. Widen per system, with the audit trail as evidence.
5. **Write the policies as code.** Budgets, approved models per data classification, unused-seat thresholds. Until this exists, the switch-off loop cannot run.
6. **Then widen.** Documents, your own applications, platform tooling. Each one is now an integration, not a project.

A reasonable first milestone is one door, one entry point and one dashboard, in production, with real traffic. Everything in this paper is reachable from there.

Conclusion

The six entry points in this paper are not a roadmap of products to buy. They are the six places where AI already enters your organization, whether or not you have designed for it. The choice in front of you is not whether those entry points exist. It is whether they pass through something you own.

Two doors and one governance layer is a small architecture by enterprise standards. It is less work than a data platform and far less work than an ERP programme. What makes it hard is not the engineering. It is that it requires deciding, in advance and in writing, what your organization is allowed to do with AI, and then enforcing that decision in infrastructure rather than in a policy document.

That is the work. It is also the only version of AI governance that survives contact with a model price change, a vendor acquisition or a regulator asking what your systems actually do.

This Deserves a Conversation

This paper is deliberately architectural, and architecture only becomes useful against a specific estate. Which entry points you already have, what your identity and integration layers look like, where your data sits and what you are allowed to send out of the organization all change the sequence and the shape of the design.

If this is interesting enough that you want to apply it to your own situation, that is a conversation worth having properly rather than in a follow-up email. We are happy to sit down with you and your architects and work through it, with no obligation and no deck.

Diederik Wyffels, Managing Partner, 45 Degrees. diederik.wyffels@45degrees.be

Bert Teuwen, Managing Partner, 45 Degrees. bert.teuwen@45degrees.be

Companion papers: **The AI Nobody Is Selling You**, on why the structural alternative exists and what it is worth, and **What an Owned AI Stack Buys You**, on the benefits and the risk management, across the six areas a CIO is accountable for.