White paper 1 of 4

# Governed enterprise AI: approvals, audit trails and PII masking in practice

How identity, access control, guards, approvals and an immutable audit trail fit together — and what to check before rollout.

About 7 min read · 1,493 words Print-friendly

CONTENTS



## Summary

How identity, access control, guards, approvals and an immutable audit trail fit together — and what to check before rollout.

# Why governance has to come first

Most organizations meet generative AI the same way: individuals start using assistants on their own, often by pasting work content into consumer tools. The benefit is real, but so is the gap it opens. Nobody can say which data left the company, which model saw it, or what the assistant was allowed to change.

Governance is the set of controls that closes that gap without stopping the work. Public guidance such as the NIST AI Risk Management Framework (https://www.nist.gov/itl/ai-risk-management-framework) and the OWASP Top 10 for LLM Applications (https://genai.owasp.org/llm-top-10/) describes the risks in general terms — sensitive information exposure, excessive agency, weak oversight. This paper looks at the practical side: what a governed request actually looks like on the AgenTorQ platform, and which controls a security team can verify.

AgenTorQ is built as a control plane for enterprise AI. Every AI Workmate, model call and data access runs through one policy layer that is role-based, PII-masked, guarded, approved and fully logged. The same governance applies in both editions: the standalone AgenTorQ Cloud workspace in the browser and the native AgenTorQ for Salesforce package.

# The governed path of a single request

The platform describes every turn as the same path: user, Workmate, routing, grounding, model, governed action, audit. No request skips grounding, model selection, policy or the audit log, whether it runs in the web workspace or inside Salesforce.

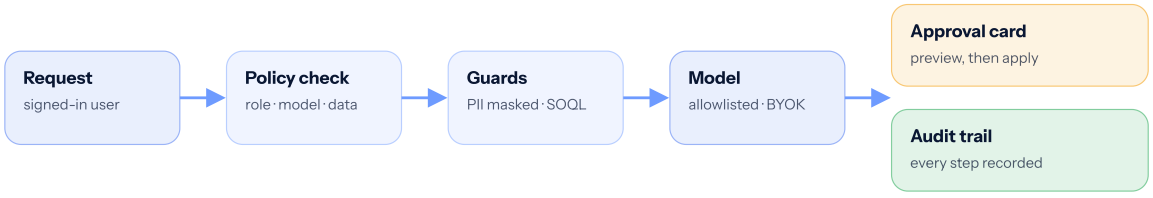


Figure 1. The order of controls on every request.

Two properties matter most. First, the order is fixed: policy and guards run *before* anything reaches a model. Second, the path is the same everywhere, so a control proven in one place does not have to be re-proven for another team or edition.

## Identity: the AI acts as a person, never above them

Every request is bound to an authenticated user and their role. The Workmate inherits that person's entitlements and cannot act beyond the human behind it. This is the single most important design choice, because it keeps AI inside the access model you already run.

- **Per-user OAuth.** Each person connects apps under their own grant, so one person's access never leaks to another — no shared service account.
- **A credential vault.** Model keys and connector credentials are held in an encrypted vault, isolated per organization, with automatic token refresh. They are kept out of prompts, logs and client storage.
- **Isolation.** Personal chats, memory and history are isolated per user. Shared organization assets — Workmates, knowledge, keys — are separated from private tokens.
- **Attribution.** Every call is attributed to the acting user in the audit trail.

## Access control: profile-style RBAC, deny-by-default

Access is defined by role and organization, not by whoever configured an agent. Admins build profiles from a feature-toggle grid, assign them with per-user overrides, and the result is enforced on the server. The client can request; only the server decides.

In practice, three kinds of people touch the platform. Administrators own the control plane: policy, keys and organization-wide guardrails. Builders design and ship Workmates within the boundaries administrators set. Users run approved Workmates against records they are already entitled to see — nothing more. Model and connector allowlists can be set per role and per workflow, and shared Workmates, knowledge and keys are isolated per tenant.

## Guards before the model: masking, query guards and USER_MODE

Between a request and a model sits an enforced pipeline. Named personal data is masked before any prompt leaves, and only the masked prompt reaches the model you chose. Masking applies to retrieval context as well as chat, so documents pulled in for an answer are treated the same way as what the user typed.

Reads and writes are guarded separately. In the Salesforce edition every read passes a SOQL guard and every write passes a DML guard, and the package runs in USER_MODE, so CRUD, field-level security and sharing are always enforced by the platform. Object-level include and

exclude controls decide what the AI may ground on at all. The Cloud edition enforces the same intent through per-connector scopes and masking.

> **What to verify:** ask for a masked prompt next to its original in a test environment, and confirm that a user without access to a field cannot get an answer that uses it.

## Approvals: preview, then apply

Reading is one thing; writing is another. Any action that changes data or triggers a downstream system is proposed as a plain-language action card and held behind an explicit approval, with a preview of exactly what will happen. The approver's identity and decision are recorded.

- Preview-then-apply on every write and deploy, with the option to reject or edit before anything reaches your systems.
- Operating modes — Ask, Edit, Plan, Auto and Bypass — enforced per turn. Plan never deploys; deletes always confirm.
- Approval behaviour configurable per action type, object and workflow, with bulk approve for batches.

Approvals are what turn an assistant into something a business can let act. They also create the evidence a reviewer needs: who asked, what was proposed, who said yes.

## The audit trail: what to capture

If you cannot audit it, you cannot approve it. AgenTorQ records a structured, immutable trail of every interaction, exportable via API for SIEM and compliance tooling.

| Field | Why it matters |
|---|---|
| Actor, role and organization | Ties every event to a person and their entitlements. |
| Prompt, tool call and model response | Shows what the AI saw and said. |
| Model routed to | Proves the allowlist held for that workflow. |
| Records and fields read or written | Answers "what data did it touch?" |

| Field | Why it matters |
| --- | --- |
| Approval decision and approver | Evidence of human oversight. |
| Result and traceable ID | Supports security review and cost accounting. |

The same record feeds analytics for adoption, spend and model performance, with model, token and cost tracking and per-user and per-team attribution, plus a live health console and call logs.

## Model policy is a security control

Multi-model is a security decision, not only a cost one. Routing each workflow to an approved model, with allowlists, fallback and data-handling rules enforced centrally, stops teams from being silently locked to a single provider or from sending sensitive work to an unapproved one. With bring-your-own-key, provider credentials stay under your control, and you can swap models without rebuilding Workmates. The companion paper on multi-model routing covers this in depth.

## Deployment choices and certification posture

The platform meets environments where they are: a managed multi-tenant service with strict per-organization isolation; a dedicated, network-isolated deployment for regulated environments; and a Salesforce-native install where data stays in Salesforce. A self-hosted connector runtime keeps sensitive integrations and credentials under your control.

Be precise about certifications. AgenTorQ describes a compliance-ready architecture and does not claim SOC 2, ISO 27001, HIPAA or GDPR certification; it is engineered to support your program. Enterprise data is not used as training data by default and never crosses tenants.

## Governance for developer work

AI that writes code needs the same controls as AI that writes records, plus a few of its own. In DevSpace Studio, the Developer Workmate reads, edits and ships Apex, LWC and Flows and drives real git on GitHub, but every write and deploy follows preview-then-apply. The operating mode is enforced per turn: in Plan the Workmate can explore and propose but never deploy, and deletes always confirm regardless of mode.

The practical effect is that a platform team can decide how much autonomy to give per task. Exploration and review can run in Plan or Ask; a change the developer has read can move to

Edit; and every tool call, diff and deploy is written to the same audit trail as any other action.

## Memory and personal data

Assistants that remember context raise a fair question: where does that memory live, and who can see it? Each Workmate keeps its own memory — concepts, entities, workstreams and notes — on top of cross-session recall, and that memory is isolated per user and private by default. People can view and manage what a Workmate remembers. The server is the source of truth; local storage is only a cache.

For a governance program this means memory is treated like any other personal context: scoped to its owner, never shared across tenants, and subject to the same masking rules when it is used to ground an answer.

## A practical rollout checklist

1. Map roles to profiles in the feature-toggle grid and start deny-by-default.
2. Decide the model allowlist per role and workflow, and add your own provider keys.
3. Decide which objects, fields and sources the AI may ground on, and confirm masking.
4. Set approval rules per action type, and choose operating modes for developer work.
5. Connect the audit export to your existing tooling before the first team goes live.
6. Start with one team, review the trail together, then widen access.

---

## Sources

Product statements in this paper restate the AgenTorQ website:

- agentorq.com/security
- agentorq.com/platform
- agentorq.com/standalone
- agentorq.com/salesforce

General context links point to the public pages named in the text. This paper contains no market statistics.