The Covenant Framework

A Governance Layer for Autonomous AI Agents

Alex Salsali
Covenant Foundation
alex@covenant.foundation

May 2026

Abstract

The deployment of autonomous AI agents into real economic and operational systems has outpaced the infrastructure required to govern them. Today's approach to agent safety relies on relational oversight: trusting that a model has been trained to behave well, that a developer has prompted it correctly, and that a human will catch its mistakes downstream. This approach does not scale. As agents act on behalf of principals across organizations, jurisdictions, and timeframes, the absence of structural enforcement becomes a liability that no amount of model alignment can resolve.

Covenant is a governance layer that defines, enforces, and audits agent behavior at execution time. It treats policy as code, intervenes at the boundary between agent intent and system action, and produces an auditable record of every decision. It does not require access to model weights, does not depend on agent self-reporting, and does not assume that agents are benign. It assumes only that interactions can be observed and that observation is enough.

This paper describes the framework, the problem it addresses, and what we have learned building it.

---

1. Introduction

In the past two years, AI agents have moved from research demos into production. They write code, send email, move money, file tickets, book travel, and increasingly take action on behalf of people and companies. The agents themselves have become more capable. The systems around them have not.

Today, when an agent does something wrong, the answer is almost always the same: retrain the model, rewrite the prompt, add a human reviewer. These are relational interventions. They work at the level of the relationship between developer and model, or between user and agent. They do not work at the level of the system in which the agent operates.

This matters because agents do not act in isolation. They act inside organizations, against shared resources, with other agents, and on behalf of principals who may never see the actions taken in their name. When something goes wrong in this setting, "the model should have known better" is not a defense. It is not a defense in court, it is not a defense to a regulator, and it is not a defense to a customer.

The premise of this paper is that AI agent safety is, at its core, a governance problem rather than an alignment problem. Alignment is necessary. It is not sufficient. What is missing is the layer between the agent's decision to act and the system's commitment to that action: a layer that defines what is allowed, enforces it without trusting the agent, and produces evidence of what happened.

We call this layer Covenant.

2. Why Existing Approaches Fall Short

Three categories of solution dominate the current landscape. Each addresses a real problem. None addresses the structural one.

Model-level alignment. Constitutional methods, RLHF, and post-training safety work make individual model outputs more likely to be acceptable. They operate inside the model. They cannot enforce anything an operator did not anticipate during training, and they offer no record of why a specific decision was made in production. When an agent built on a well-aligned model takes a harmful action, the alignment work was not wrong. It was simply not the right layer.

Prompt-level guardrails. Tools like NeMo Guardrails and similar libraries let developers wrap LLM calls in input and output filters. This works for the application that owns the prompt. It does not work when the agent acts across systems the prompt author does not control, and it offers no defense against agents whose prompts have themselves been compromised.

Human-in-the-loop review. Adding a human approver is the most common production safeguard. It works until volume increases, until the human starts approving by reflex, or until the agent learns to act in ways the human does not understand well enough to evaluate. Human review is a control. It is not a system.

The common failure mode across all three is that they trust the agent, the developer, or the reviewer to do the right thing. This is relational oversight. It scales linearly with attention and breaks under load.

Structural governance is different. It does not trust. It observes, enforces, and records. The agent does not need to be honest about what it is doing because the framework can see what it is doing. The developer does not need to anticipate every misuse because the policy is enforced at execution, not at design. The reviewer does not need to be present for every action because the record is sufficient for post-hoc accountability.

There is a fourth response a careful reader is already preparing: the major model providers will eventually ship governance layers tied to their own platforms, and some already are. They will. What they cannot ship is a layer that is portable across providers, accountable to operators rather than vendors, and auditable by parties other than the vendor itself. A governance layer offered by the company whose model is being governed is structurally compromised in the same way that a financial auditor employed by the company being audited is compromised. Moving the layer outside the vendor relationship does not eliminate every conflict. Operators have their own incentives to write permissive policies. But it relocates the conflict to a party that is accountable to regulators, customers, and courts in ways the model provider is not. The independence is the product.

3. What Covenant Does

Covenant sits between an agent and the systems it acts on. Every action an agent attempts passes through it. The framework does five things, in order:

Identifies. Every actor has a verifiable identity. Agents cannot impersonate other agents, and capabilities are bound to identities rather than to sessions or tokens that can be stolen or shared.

Authorizes. Each identity has a defined set of capabilities. An agent that has not been granted the ability to send email cannot send email, regardless of what its prompt tells it to do. Capabilities are bounded the way operating system permissions are bounded: deny by default, granted explicitly, revocable at any time.

Enforces. Policies are written in a domain-specific language with temporal semantics, which means they can express not just what is forbidden in a single action (“do not send confidential files to public channels”) but what is forbidden across sequences of actions (“no agent may issue more than two refunds to the same customer within twenty-four hours without human approval”). These policies compile to runtime monitors that evaluate every event and decide whether to allow it, block it, transform it, or escalate it to a human.

Sanctions. When an agent violates a policy, the response is graduated. A first violation may trigger a warning. A repeated violation throttles the agent's capabilities. A pattern of violations suspends the identity. The framing is not deterrence, since an agent has no preferences over being throttled, but capability control: each tier narrows what the agent can do per unit time, bounding the damage an erratic or compromised actor can cause while preserving room for honest mistakes to be corrected.

Records. Every decision Covenant makes, every action it allows, every intervention it imposes is logged in a format compatible with the W3C provenance standard. This means the record is not just a log file. It is a graph of causes and effects that can be queried, audited, and presented as evidence.

These five functions are not novel individually. Capability systems have existed since the 1970s. Runtime monitoring is a mature field. Provenance has a published standard. What is new is putting them together at the point where autonomous agents meet production systems, and doing so without requiring those agents to cooperate.

The five categories are not perfectly disjoint in practice. Capability bounding and policy enforcement overlap whenever a policy can be expressed as a capability restriction. In those cases the choice between them becomes an engineering decision about where enforcement is cheaper. Provenance and sanctions overlap at the boundary where a logged event becomes the trigger for a sanction tier change. These blurred edges are not weaknesses; they are where the framework is doing the most work.

4. The Architecture

Covenant is implemented as a gateway and sidecar. The gateway sits at the boundary between agents and the resources they consume: APIs, databases, messaging systems, file stores. The sidecar runs alongside individual agent processes when finer-grained observation is needed.

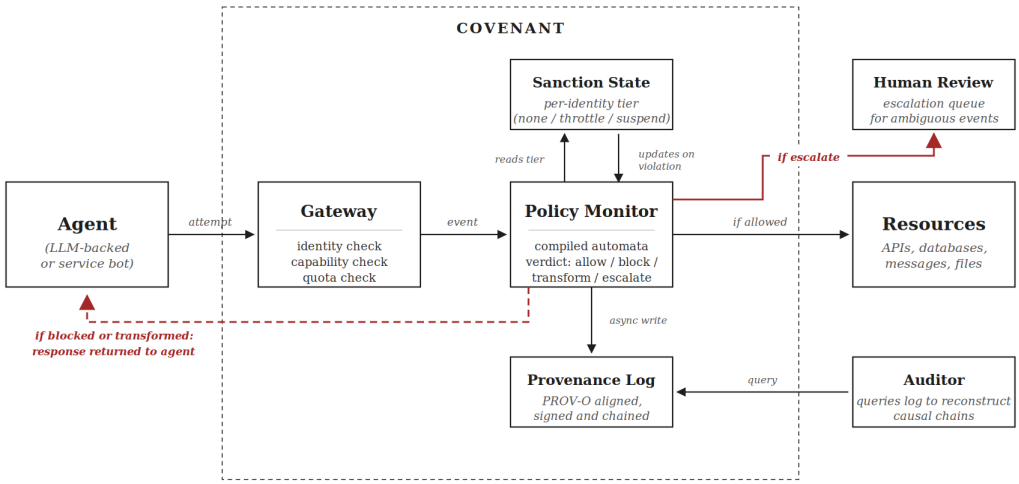

Figure 1. A request from an agent passes through the gateway (identity, capability, and quota checks), then through the policy monitor, which consults sanction state and writes provenance. Allowed actions proceed to resources. Blocked or transformed actions return to the agent; escalated actions are queued for human review. Auditors query the provenance log to reconstruct any decision after the fact.

Policies are authored in a text-based DSL and compiled ahead of time into deterministic monitor automata. This compilation step matters. It means the cost of evaluating a policy at runtime is bounded and predictable: a state transition and a verdict, not an LLM call. Enforcement does not require model inference, which is what makes it suitable for use on the hot path of high-volume agent workflows.

Before any policy is deployed, Covenant runs two pre-execution checks. The first is static: it asks whether the policy is internally consistent, whether it conflicts with other policies, and whether it can be violated given the capabilities currently granted to existing agents. The second is dynamic: it runs the policy against a sandboxed replay of recent traffic to surface any violations the static check missed. The intent is to catch policy errors at deployment rather than discover them in production.

The provenance layer emits events asynchronously to avoid adding latency to the enforcement path. Every event is signed and chained, so the log itself is tamper-evident. Auditors querying the log can reconstruct the causal chain behind any decision: which agent, acting under which identity, attempting which operation, against which resource, in response to which prior event, and with what outcome.

5. A Walkthrough

To make the framework concrete, consider a single event traced end to end.

An agent operating in a customer service workflow attempts to issue a refund to a customer it has already refunded twice in the previous twenty-four hours. The agent's prompt does not forbid this. The model has no reason to think it is doing anything wrong. From the agent's perspective, it is being helpful.

The action leaves the agent as a structured request to the refund API. Before it reaches that API, it passes through the Covenant gateway.

The gateway checks the agent's identity against the registry. The identity is valid. It checks the requested operation against the agent's capabilities. Issuing refunds is permitted. It checks the agent's quota for refund operations in the current window. Within budget. So far, the action would proceed.

Then the policy monitor evaluates the event against the active rules. One of those rules, written by the operator three weeks earlier, looks roughly like this:

```
policy  refund_velocity_limit
  on    event.type == refund_request
  when  count(refund_request
              where customer_id == event.customer_id
              within 24h) >= 2
  then  escalate(reviewer_queue)
```

The monitor walks its automaton, finds two prior refund events for this customer ID inside the window, and returns a verdict of escalate. The action is held. A review request is dispatched to a queue monitored by a human operator.

Simultaneously, the provenance layer writes an entry recording the agent's identity, the attempted operation, the customer ID, the prior refund events that triggered the rule, the policy that fired, the verdict, and the escalation target. The entry is signed and chained to the log's prior state.

Two outcomes are now possible. If the operator approves the refund, the action proceeds and the approval itself is logged, attributed to a named human, and linked to the original event. If the operator denies it, the agent receives a denial response and the sanction state for that agent is updated. A second escalation of the same type within a defined window would shift the agent into a throttled tier; a pattern would suspend the identity entirely.

None of this requires the agent to know it is being governed. The model is not retrained. The prompt is not edited. The policy lives outside the agent, was written by someone other than the agent's developer, and is enforced regardless of what the agent intends. Six weeks later, if a customer asks why their third refund attempt was delayed, the answer is in the log, with the policy that caused it, the human who approved or denied it, and the timestamp of every step.

Every element described in sections 3 and 4 is doing work in that single event.

6. What We Have Seen, and What We Are Testing

The empirical state of the framework should be described honestly. Covenant is in production inside one operator-led deployment, WedFlow, which has been running long enough to generate qualitative observations but not long enough, and not at sufficient scale, to support quantitative claims that would survive peer review. What follows is a deliberate split: what we have observed in that deployment, and what the benchmark program still needs to establish.

What we have observed

In WedFlow, the capability model has been load-bearing from day one. Agents that handle vendor communication cannot reach payment systems; agents that process payments cannot send external messages. This is a structural property of the deployment rather than a measured outcome, but it is the property that determines how large any single agent failure can be. We have not yet had a containment failure to report on; that is partly a function of the small sample size and partly the point.

The pre-execution validator has caught policy conflicts during deployment that would have produced silent failures in production. Some were subtle enough to escape our own review before being surfaced by the tooling. The runtime monitors have flagged sequences of individually-legitimate actions that combine into policy violations only when viewed as a trace: a class of failure that prompt-level guardrails miss by construction. These are anecdotes from a single deployment, not statistics across many; they suggest the architecture is doing something real but they are not a substitute for controlled measurement.

What the benchmarks still need to establish

Five questions, each tied to one of the five functions, define the research program now underway.

Detection. Across heterogeneous workloads with thirty to one hundred agents and a defined fraction attempting off-policy actions, what is the detection rate, false positive rate, and added latency of the runtime monitors?

Containment. Under controlled adversarial conditions, what fraction of a resource budget can a non-compliant agent consume before being stopped, and what overhead does enforcement impose on legitimate throughput?

Prevention. Holding everything else constant, how much of the runtime violation surface does the pre-execution validator remove relative to enforcement alone?

Sanction-tier effectiveness. Does the graduated capability-control schedule reduce repeat violations relative to a warnings-only baseline, and does it do so without degrading task success for honest agents?

Auditability. Can independent auditors, using only the provenance log, reconstruct the causal chain behind a given decision and attribute responsibility with high inter-rater agreement? This is the question we consider most open. The design supports it; the empirical work to validate it is still ahead of us.

Quantitative results from this program will appear in the accompanying research paper, Governance Under Load. The reason this whitepaper does not report numbers is not that there are none, but that the numbers that exist are not yet at the scale or under the methodological controls required to defend them.

7. What This Is Not

Covenant is not an alignment technique. It does not make models better. It does not reduce hallucination, improve reasoning, or affect what the model would say in isolation. It governs what the model can do once it tries.

Covenant is not a replacement for human oversight. It is what makes human oversight tractable. By recording every decision and escalating ambiguous ones, it shifts the human role from approving routine actions to reviewing edge cases and audit trails. The human is still in the loop. The loop is just no longer the bottleneck.

Covenant is not a guarantee of safety. No governance framework is. What it provides is a structural defense that does not depend on the agent's cooperation, and an evidentiary record that does not depend on the agent's honesty. These are necessary properties of any system that will eventually need to defend its actions to a court, a regulator, or a customer who asks why something happened.

Covenant is not a complete defense against an agent that can act through channels Covenant does not sit on. The framework's guarantees hold over observed events; an agent that writes directly to a database the gateway does not mediate, or that exfiltrates information through a side channel, or that induces another agent to take an action it cannot itself take, falls partially or wholly outside the enforcement surface. The completeness of that surface is an engineering question that has to be answered per deployment, and it is the limit any operator deploying this framework should understand before they trust it.

8. Why Now

The argument for structural governance has been theoretically available for years. What has changed is that the cost of not having it is now concrete. Agents are deployed, they

make mistakes, and the mistakes are starting to have legal, financial, and reputational consequences. The people responsible for those consequences are starting to ask what evidence they have of what their agents actually did.

Three trends make this moment specific. First, agents are crossing organizational boundaries. An agent built by company A increasingly acts on company B's systems, with company C's data, on behalf of person D. Each of these parties has different policies, different liabilities, and different evidentiary needs. Without a common governance layer, the only way to resolve disputes is to take the agent's word for it, or the developer's. Neither is sustainable.

Second, the regulatory environment is closing in. The EU AI Act, emerging state-level regulation in the US, and existing sectoral regimes in finance and healthcare all converge on the same requirement: demonstrable controls and audit trails for automated decisions. A framework that produces this evidence as a byproduct of operation is materially cheaper than one bolted on afterward.

Third, the agent ecosystem is fragmenting. A governance layer that lives at the model is captive to one vendor. A governance layer that lives at the system is portable across all of them.

9. The Path Forward

Covenant is being built in the open. The core framework is in active development, and the first operator-led venture built on it, WedFlow, is using Covenant to govern agent behavior in the wedding industry. That environment was chosen specifically because the consequences of agent misbehavior are immediate, visible, and impossible to hide. If the framework cannot survive that, it cannot survive anywhere.

One open problem is honest to name. Policy authoring is itself a discipline. Writing good policy in a temporal DSL requires understanding both the operational domain and the language's semantics, and the operators who most need governance are often the ones least equipped to author it. The framework's value in any given deployment depends on someone capable of expressing the constraints that matter, whether that is an internal team, a third-party specialist, or eventually a more accessible authoring layer. Building the framework is the first step. Making policy authoring tractable is a separate problem that the broader effort will have to engage with.

If you operate an agent in production and the question of what your agent actually did under what authority keeps you up at night, this framework is being built for you. If you build agents and want them to be deployable in environments that take governance seriously, the same applies.

10. Conclusion

The frame that has dominated AI safety to date treats safety as a property of the model. That frame produced real progress and is not wrong. It is incomplete. As agents take action in systems they do not own, on behalf of principals they cannot fully represent, against resources whose value they do not understand, the property that matters most is no longer how well-trained the model is. It is whether the system around the agent can define, enforce, and audit what the agent is allowed to do.

That system does not exist by default. It has to be built. Covenant is our attempt to build it.

References

A full technical bibliography accompanies the research paper Governance Under Load: Why Principal-Agent Systems Require Structural Governance, Not Relational Oversight, available soon at covenant.foundation. Key references include foundational work on runtime assurance (Bloem et al., 2019; Desai et al., 2018), capability systems (Watson et al., 2015), norm enforcement in multi-agent systems (Morris-Martin et al., 2018; GAVEL, 2019), guardrails for language model applications (Rebedea et al., 2023; Dong et al., 2024), and the W3C PROV-O provenance standard (2013).

The Covenant Foundation is building governance infrastructure for autonomous AI agents.
covenant.foundation