Adaptive Governance Protocols

A Pre-Execution Mathematical Compliance Layer for Agentic AI Systems

Adya AI Research · May 2026

Abstract

Agentic AI systems are being deployed in regulated industries — banking, healthcare, government, legal — where a single non-compliant action carries regulatory, financial, or human-safety cost. Yet every governance approach in production today operates on probabilistic foundations. Prompt-level guardrails are persuadable instructions encoded into model weights via training; output filters detect violations after generation; constitutional alignment shapes model behaviour at training time but cannot offer runtime guarantees. None of these can answer the question a compliance officer actually has to answer: *can you prove this agent will not violate this rule?*

Adaptive Governance Protocols (AGP) is a pre-execution governance layer designed to answer that question affirmatively. AGP ingests an organisation's policies, SOPs, and regulatory documents — either through a deep research agent that reads them directly or via direct upload — and decomposes them into a four-tier hierarchy of executable rules. Each rule is broken into NLP-described validation steps that may call external systems via API for fact-checking. The rules are then compiled into canonical mathematical proofs using Lean4 and into runtime policy tests using CEDAR. Every agent action is validated against these proofs *before* a single token is generated. If validation fails, the execution path terminates. The AI agent does not produce a non-compliant output that is later filtered — it does not produce a non-compliant output at all.

This paper describes the AGP architecture, the mathematical floor it establishes, the four-tier rule hierarchy, the pre-execution validation gate, the Lean4 and CEDAR pipeline, and an honest accounting of what AGP can and cannot do. The headline result: for every rule that has been authored, AGP makes violation architecturally impossible. The honest limit: AGP cannot enforce rules that have not been thought of. Rule completeness remains a human responsibility — but every authored rule is a deterministic guarantee.

AGP is shipped today as a module in the Vanij platform (Adya's enterprise agentic AI platform) and is in flight as an API for external agents not built on Vanij.

---

1. The Governance Problem in Agentic AI Systems

1.1 The shift from chatbots to agents

The agentic transition — from AI systems that produce text to AI systems that take actions — moves the locus of risk. A chatbot that hallucinates produces a wrong sentence; an agent that hallucinates approves a non-compliant loan, prescribes the wrong triage pathway, processes a transaction that violates aggregate exposure limits, or leaks a customer record across a department boundary. The economic and human stakes are no longer cosmetic.

By 2026, multiple surveys put the share of enterprises with at least one production agentic AI deployment at over half, and the share with five or more deployments climbing rapidly. Most of these deployments live in environments — banks, hospitals, telcos, public sector — where the regulator has views about how decisions get made.

1.2 The four failure modes

Agentic systems without governance exhibit four predictable failure modes:

Hallucination. The agent produces an output that appears coherent but contradicts ground truth or organisational policy. Hallucination is not a bug — it is the natural consequence of a probabilistic generative process. The probability of hallucination per request is non-zero by construction.

Inconsistency across agents. A single organisation may have many agents — one for customer service, one for underwriting, one for collections, one for KYC, one for marketing copy. Without a shared governance layer, each agent reasons from its own prompt and tooling. Two agents asked the same compliance question can produce contradicting answers. For a regulator reviewing audit logs, this is indefensible.

Drift from policy intent. Even when a rule is included in the system prompt, the model's interpretation of that rule drifts with conversational context. A rule that holds in the first turn of a conversation may be implicitly relaxed by turn ten because the cumulative context made the exception "feel reasonable" — a known property of attention-based generation. Drift is undetectable from the agent's outputs alone.

No audit trail. When a probabilistic system makes a decision, the only artefact is the output. There is no machine-checkable record of *why* the system said yes or no, against which rules, and whether all required checks fired. For a compliance review or an external audit, the absence of this artefact is itself a finding.

1.3 Why these failures are expensive in regulated industries

In a B2C product, hallucination costs you a refund or a brand-trust hit. In a regulated workflow, the same probabilistic failure carries:

- *Regulatory penalty risk* — RBI, SEBI, IRDAI, MeitY, and equivalent bodies in other jurisdictions are increasingly treating AI-driven decisions as bank-driven decisions. The bank carries the liability.
- *Legal exposure* — a denial that should have been an approval (or vice versa) is a tort.
- *Patient safety risk* — a triage that misroutes a stroke patient is a clinical incident.
- *Reputational damage* — one viral screenshot of an AI customer service agent saying the wrong thing can dominate the news cycle for a week.
- *Audit cost* — every uncaught failure becomes evidence in the next regulatory review.

The total cost of the *possibility* of failure, multiplied by the volume of agentic decisions a large enterprise will make per day, is the implicit budget for governance. AGP is designed to consume part of that budget by making certain failure modes architecturally impossible.

---

2. Why Existing Approaches Fall Short

The AI safety and governance literature has produced four broad approaches to constraining agent behaviour. We examine each in turn and identify the architectural ceiling that prevents any of them from offering deterministic guarantees.

2.1 Prompt-level guardrails

The dominant approach in 2024–2026 has been to write instructions into the agent's system prompt: *"Never approve loans above ₹50 lakhs without board sign-off."* This is the simplest and most accessible technique — anyone deploying an agent can write a guardrail in plain English.

The architectural problem is that the rule lives inside the same probabilistic context window that is used for reasoning. The model treats the rule as one input among many. A user message that creates conversational pressure ("we discussed this exception last week," "this is a special case approved by the regional manager") competes for attention with the system prompt, and frequently wins. This is the well-known prompt injection / jailbreak vulnerability. Even without adversarial intent, the model's interpretation of a guardrail drifts with conversational context.

Prompt-level guardrails are persuadable boundaries. They lower the probability of violation; they do not eliminate it.

2.2 Output filtering

A second approach inserts a filter between the model and the user: every response is screened, and outputs that violate predefined patterns are blocked or rewritten. This is the architecture used by Lakera, Guardrails AI, NVIDIA NeMo Guardrails, and most commercial "AI safety" products.

Output filtering is post-hoc. By the time the filter inspects the response, the model has already committed compute, and — more importantly — has already made the decision. The filter's job is to catch the symptom. Two consequences follow:

First, output filters are themselves probabilistic. They are typically classifiers (often LLM-based) trained on examples of what violations *look like*. They detect what they were trained to detect. A novel violation that doesn't pattern-match to training data will pass through. Filters trained on English jailbreaks fail on Hindi or Tamil; filters trained on textual outputs fail on tool-use traces; filters trained on banking violations fail on healthcare protocols.

Second, when an output filter fires, the agent has already produced the violation internally — only the surface output is suppressed. There is no audit trail of why the violation was attempted, only that the surface response was blocked. This makes root-cause analysis difficult and makes the agent's reasoning trace itself potentially leaky (intermediate tool calls, internal logs, partial outputs may already have crossed boundaries).

Output filtering raises the bar; it does not eliminate violations.

2.3 Constitutional and alignment-time approaches

A third class of approaches — Constitutional AI, RLHF with safety preferences, and similar techniques — shapes model behaviour at training time. The model is trained on examples that align with desired values, with the goal of internalising those values into the model's weights. Anthropic's Constitutional AI is the most cited example.

Alignment-time approaches are more robust than prompt-level guardrails — the safety properties are encoded into weights, not into prompts. But two limitations apply for enterprise governance:

First, alignment is generic. It cannot encode the specific lending policy of a specific bank, the specific clinical protocol of a specific hospital, or the specific procurement rule of a specific government department. Constitutional AI gives you a model that is broadly safe; it does not give you a model that is compliant with *your* SOPs.

Second, alignment provides no runtime audit trail. When a constitutionally-trained model produces a response, there is no machine-checkable artefact that says "this response satisfied rule X, Y, Z." The compliance is folded into the weights, not exposed as evidence.

Alignment is necessary; it is not sufficient.

2.4 The architectural ceiling

The four approaches above — prompt guardrails, output filters, alignment, and post-hoc detection — share a common architectural feature: they operate within the probabilistic substrate of the model. They reduce the probability of violation; they do not eliminate it.

For any probabilistic system with non-zero violation probability p per request, the probability of at least one violation in N requests is:

```
P(at least one violation in N) = 1 − (1 − p)^N
```

This expression has a single dominant property: as *N* grows, *P* approaches 1.

For a low-probability scenario — *p* = 0.001 (one violation in a thousand requests, which is optimistic for current frontier models on novel domains) — and a moderate request volume — *N* = 10,000 (a single mid-sized customer service deployment for one day) — *P* exceeds 99.99%. Across a year and across all agents in a large enterprise, the expected number of violations is large. Compliance teams cannot accept this floor.

The ceiling is not a bug in the implementations; it is a property of the architecture. Probabilistic systems cannot offer deterministic guarantees. To get a deterministic guarantee, the validation must occur outside the probabilistic substrate.

---

3. The Mathematical Floor

The premise of AGP is that the inverse of the architectural ceiling is achievable: a system in which the probability of violation, for every authored rule, is zero — not asymptotically, but exactly.

The construction is straightforward. If a rule *R* is encoded as a mathematical proposition, and every agent action is required to satisfy a proof of *R* before execution, then any action that does not satisfy the proof is terminated. There is no path by which a non-compliant action reaches the user. The probability of violation, conditional on the proof being correctly authored, is zero.

This is not a marketing claim. It is a logical consequence of pre-execution validation against a deterministic check. The same property holds in any system that uses formal verification: a verified compiler does not emit incorrect code because the proof obligation is a precondition for emission. AGP applies the same principle to agent actions.

3.1 The honest bound

The previous claim is conditional on three things, and the page is explicit about each:

(a) The rule has been authored. AGP enforces only what you write into it. If your bank's lending policy has a rule about debt-to-income ratios but does not have a rule about stacking multiple sub-agreements to evade an aggregate threshold, AGP cannot prevent the latter. The agent will faithfully comply with every rule that exists. It will not invent rules that do not exist.

(b) The rule has been correctly translated. The Lean4 proof generated from the natural language rule must faithfully encode the intent of the original. Rule authoring is an iterative process: an admin reviews the proposed rules after extraction, edits them, toggles them, and

approves them. Errors in this step are caught by the same review process that catches errors in any other governance artefact.

(c) The validation is wired into the execution path. AGP is not an external service that the agent can choose to call; it is a gate the agent passes through. Bypassing the gate requires bypassing the platform integration — which is a deliberate and auditable act, not an emergent behaviour.

Within these bounds, AGP offers what no probabilistic approach can: the certainty that authored rules will not be violated.

3.2 The complementary frame

AGP is not a replacement for probabilistic safety techniques; it is a complement.

For *known* rules — rules that have been authored, reviewed, and proven — AGP is deterministic. Violation is impossible.

For *unknown* rules — edge cases, novel attacks, scenarios no one has thought of yet — probabilistic detection (output filters, anomaly classifiers, human review) remains the only available technique. AGP does not displace these tools; it relieves them of the load of catching things that should have been ruled out by construction, freeing them to focus on the long tail of unknown unknowns.

The pattern is the same one that holds in security architecture more broadly: deterministic controls (firewall rules, access control lists, policy decision points) carry the bulk of the load, and detection-based controls (intrusion detection, anomaly monitoring) catch what the deterministic controls miss. AGP brings this pattern to agentic AI.

---

4. AGP Architecture

The AGP system has six logical components. Each has a corresponding implementation in the shipped platform module.

4.1 Document ingestion

AGP accepts policies, SOPs, regulatory documents, compliance manuals, and process descriptions in PDF, DOC, DOCX, CSV, XLS, XLSX, and TXT formats. There are two ingestion paths:

Direct upload. An admin uploads a document, attaches metadata (geography, domain, use case, flow), and submits. The system processes the document and proposes rules.

Deep research agent. For organisations whose policies are scattered across SharePoint, regulatory websites, internal wikis, and email-distributed memos, AGP ships with a deep

research agent that, given a description of the relevant scope, gathers the source material, deduplicates it, and ingests it. This is particularly useful for regulatory rule extraction, where the source documents (RBI master directions, SEBI circulars, MeitY notifications) live across multiple websites in multiple formats.

Both paths converge on the same downstream pipeline.

4.2 NLP rule extraction

The ingested documents are processed by an extraction pipeline that identifies normative statements ("must," "shall," "may not," numeric thresholds, conditional clauses) and proposes them as candidate rules. Each candidate is presented to the admin with the source paragraph, the proposed rule wording, and suggested metadata.

The admin reviews each candidate. They can accept it, edit the wording, change the parameters (thresholds, currencies, time windows), reassign the metadata, or reject it. This step is non-negotiable: the rule that ends up in the system is the rule the human approved, not the rule the model proposed. This is the governance equivalent of code review — the model is a draftsperson, the admin is the author.

4.3 The four-tier hierarchy

Approved rules are filed into a four-tier structure that mirrors how organisations actually think about policy:

Tier	Level	Example A	Example B
1	Geography	India	United States
2	Domain / Industry	Banking (BFSI)	Logistics
3	Use Case / Department	Customer Support	Finance / Accounts
4	Flow	Loan Enquiry Flow	Invoice Dispute Flow

Why a hierarchy and not a flat list? Three reasons.

First, *jurisdictional inheritance*. A rule that applies to all banking activity in India inherits to every department within an Indian bank. Without hierarchy, the same rule has to be authored in every department's ruleset — a maintenance disaster.

Second, *override semantics*. A use-case-specific rule can override a domain-level default for that use case. A bank might have a domain-level rule that loans above ₹50L require board sign-off, with a flow-level override for emergency disaster relief loans (different threshold, different approval chain). The hierarchy makes the override explicit and auditable.

Third, *template reuse*. Rule groups at the geography + domain level become reusable templates that other organisations in the same jurisdiction and industry can adopt as starting kits. A new Indian bank onboarding to AGP inherits a baseline of regulatory rules; they only need to author their organisation-specific deltas. This is the network effect: the more organisations on the platform, the richer the template library.

4.4 Step decomposition with NLP actions

A rule is rarely a single check. The Indian banking rule "loans above ₹50L require board sign-off" decomposes into multiple validation steps:

1. *Identify the loan amount in the agent's intended action.* 2. *Compare it against the ₹50L threshold.* 3. *If above the threshold, verify that a board approval token is present in the action context.* 4. *If the token is present, verify it has not expired and matches the intended borrower.*

AGP represents each rule as an ordered set of steps, each described in natural language for the admin's benefit and each translated into an executable check at runtime. Steps may need to call external systems — a step that verifies "the customer's KYC is current" needs to call the KYC service. AGP supports this directly: each step can be configured with an API integration (HTTP method, URL, headers, request body, query parameters, timeout). The screenshot in the platform shows this UI: a step ("identify if the product contains hazardous materials") has a "Click to add API" affordance that opens an integration setup pane on the right.

This is the practical bridge between abstract policy and operational enforcement. The rule is *what* must be true; the steps are *how* AGP checks that it is true.

4.5 Lean4: canonical proof generation

Once a rule and its steps are approved, the rule is compiled into a canonical proof using Lean4.

Lean4 is a theorem prover and dependently-typed programming language with a substantial mathematical heritage — the Liquid Tensor Experiment, the formalisation of perfectoid spaces, the ongoing work on the Mathlib library. For AGP's purpose, the key property of Lean4 is that a proof, once produced, is mechanically checkable: a Lean4 proof can be re-verified by an independent verifier without reference to the original prover. This is the property that makes the mathematical floor real.

The compilation translates the rule's logical structure — predicates, thresholds, conditional clauses, dependencies on other rules — into Lean4 propositions and produces a proof obligation that any agent action must satisfy. The proof artefact is stored alongside the rule, indexed by rule ID and version.

For regulators and auditors, the proof artefact is the answer to "show me why this decision was compliant." The artefact is decidable, machine-checkable, and reproducible.

4.6 CEDAR: runtime test compilation

Rules need a different artefact at runtime: a fast, composable check that can be evaluated in microseconds without loading a theorem prover. AGP uses CEDAR for this layer.

CEDAR is a policy language originally designed for fine-grained authorisation, with an emphasis on composability, performance, and decidability. A CEDAR policy is a structured statement of the form "permit/forbid principal action resource when conditions" with a well-defined evaluation

semantics. CEDAR engines evaluate policies in microseconds and support efficient composition of large policy sets.

For AGP, every approved rule is compiled into a corresponding CEDAR policy. At runtime, when an agent attempts an action, the action is rendered as a CEDAR request (principal: the agent's identity; action: the intended action type; resource: the affected entity; context: the relevant attributes). The CEDAR engine evaluates the request against the active policy set and returns a permit/forbid decision with the policies that fired.

The pairing of Lean4 (proof artefact for audit) and CEDAR (runtime check for performance) gives AGP both the long-tail compliance evidence and the real-time enforcement loop.

4.7 The pre-execution validation gate

The runtime architecture is the headline. The standard agent flow looks like this:

```
Application → Model API → Response
```

There is no governance layer. Every probabilistic safety technique — guardrails, filters, alignment — sits inside one of these three boxes.

The AGP flow inserts a gate:

```
Application → AGP Validation → Model API → AGP Validation → Response
                  ↓                                  ↓
        [Block if proof fails]            [Block if proof fails]
```

Pre-generation validation: when the agent receives a request, the request is rendered as a CEDAR query against the active rule set *before* any model call is made. If the request would lead to a non-compliant action, the agent is told to refuse — and crucially, the refusal is the AGP refusal, not a model-generated approximation of a refusal. There is no opportunity for the model to be coaxed into producing a harmful response, because the model is never invoked.

Post-generation validation: for cases where the agent's intended action is only knowable after model reasoning (the model proposes an action plan), AGP validates the proposed action before the action is executed. The model has produced text; the text describes a proposed action; AGP checks the action against the proofs; if the action is non-compliant, the action is blocked and the user receives a structured refusal with the rule that was violated.

The combination — a gate before generation and a gate before action — closes the loop. There is no point in the lifecycle at which a non-compliant action reaches the user.

4.8 The audit trail

Because every gate evaluation produces a decidable artefact (the CEDAR evaluation result, the Lean4 proof reference), AGP produces a comprehensive audit trail by construction. Every action carries a record of:

- Which rules were evaluated.
- Which rules fired (permit, forbid, indeterminate).
- The proof artefact for each firing rule.
- The action that resulted.
- The timestamp, agent identity, user identity, and request context.

For a compliance review, this is the primary artefact. For a regulator, it is the answer to "show me your AI decision log." The audit trail is not bolted on after the fact; it is a byproduct of the validation architecture.

---

5. Capabilities and Operations

The AGP module exposes seven operational capabilities to admins. Each addresses a recurring practical problem in deployed governance systems.

5.1 Rule toggle

Every individual rule can be switched on or off from the admin UI without deletion. This allows admins to test configurations, handle exceptions for specific time-bound campaigns, or deactivate rules that no longer apply — without code changes or redeployments. A toggled-off rule remains in the system as a versioned artefact; reactivation is one click. This addresses the most common operational complaint about rule engines: "we can't change anything without engineering involvement."

5.2 Dependency management

Rules depend on other rules. A rule that requires "verify KYC is current" depends on a rule that defines "what current KYC means." Disabling the upstream rule silently breaks the downstream rule. AGP tracks dependencies as a graph and warns the admin before a destructive change. The warning is specific: "Disabling this rule will affect the following rules: [list]." For each affected rule, the admin can review the impact before proceeding.

This prevents the failure mode where a well-intentioned exception introduced by one team silently disables a critical check enforced by another team. The dependency graph is part of the rule artefact, not a separate document.

5.3 Real-time editing

Rule parameters — thresholds, time windows, currencies, named entities — can be edited directly from the UI. A change to a loan threshold from ₹50,000 to ₹75,000 propagates to every agent in the next CEDAR evaluation, with no code change and no redeployment. The change is versioned: the previous value is retained, and any historical decision that referenced the previous value can be replayed against either version for audit purposes.

This collapses the typical PM-to-developer-to-deploy cycle for policy changes from days to seconds.

5.4 Document auto-extraction

The extraction pipeline described in §4.2 is exposed as a continuous capability, not a one-time onboarding step. Admins can upload new SOP versions whenever a policy changes, and AGP proposes the deltas — new rules, modified parameters, removed rules — for review. This makes ongoing policy maintenance tractable, which is the difference between a governance system that works on day one and a governance system that still works on day three hundred.

5.5 Template anonymisation

When rule groups are shared across organisations as templates, AGP automatically strips organisation-specific identifiers before publication. The functional logic — the structure of rules, the thresholds, the dependencies — is preserved. The proprietary details — internal product names, system identifiers, named entities, organisation-specific approval roles — are scrubbed.

Anonymisation is performed at template-publication time, not at extraction time. The original organisation retains the unanonymised version; the published template is a sanitised projection. This is the privacy-first design choice that makes cross-organisation template sharing safe.

5.6 Agent Studio integration

Inside Vanij's Agent Studio (where developers build agents), a dropdown selector lets the developer attach an AGP template to an agent at design time. The agent inherits the active rule set from the template and validates its outputs against those rules at runtime. There is no separate integration step; the rule set is part of the agent's configuration.

This is the difference between governance as a separate compliance project and governance as a default property of agent construction. By the time an agent reaches production, it is already AGP-attached.

5.7 Multi-Agent Orchestration compatibility

In multi-agent flows — where many agents collaborate on a complex task — a shared AGP template can be applied across all agents in the network. This ensures consistent rule enforcement whether the network has two agents or twenty. The same rule about customer data

access applies to the customer service agent, the underwriting agent, and the collections agent equally, because they all evaluate against the same CEDAR policy set.

Without this, multi-agent systems regress to the worst-case behaviour of any single agent in the network. With it, the entire network inherits the governance posture of the template.

---

6. Templates and the Network Effect

AGP's template system is the structural mechanism by which the platform becomes more valuable as more organisations use it.

6.1 System-level templates

System-level templates are anchored to a geography and a domain. *India* + *Banking*. *United States* + *Healthcare*. *European Union* + *Public Sector*. These templates encode the regulatory and industry-conventional rules that apply to every organisation in that segment. They are produced by Adya in collaboration with regulatory experts and curated organisations, and are kept current as regulations change.

A new organisation onboarding to AGP starts by attaching the relevant system-level template. They inherit a working baseline of compliance rules immediately. They are not starting from scratch.

6.2 Use-case templates

Use-case templates are anchored to a department and a flow. *Banking* + *Customer Support* + *Loan Enquiry Flow*. *Healthcare* + *Outpatient* + *Triage Pathway*. These are more granular and more variable across organisations — every bank has a slightly different loan enquiry flow.

Use-case templates are produced by the organisations that build them and shared with the platform under anonymisation. A bank that has refined its loan enquiry rule set over six months can publish the anonymised version; another bank can adopt it as a starting point and customise from there.

6.3 The compound

The template system has three properties that compound over time:

Coverage. Every new rule group authored on the platform expands the coverage of the template library. Geographies, domains, and use cases that have a few templates today will have many in two years.

Quality. Templates that are widely adopted accumulate refinement. When a hundred organisations are using a template, the bugs and gaps get found and fixed. The template that ships with AGP six months from now will be sharper than the one that ships today.

Speed. A new organisation onboarding today might need three to six months to author a comprehensive rule set from scratch. With templates, the same organisation can be running deterministic governance in days, with refinement happening in flight.

This is the network effect. The platform's value to the next customer is a function of how much rule authoring has been done by every previous customer. AGP's economic moat is the same shape as Linux's, GitHub's, or Stack Overflow's: the artefact library matters more than any single artefact.

---

7. Empirical Comparison

To position AGP against the comparison cohort named in the GTM strategy — Lakera, Guardrails AI, NVIDIA NeMo Guardrails — we examine each on the same six dimensions used to evaluate multi-agent frameworks: architecture, validation timing, rule expressiveness, audit artefact, fail-mode behaviour, and integration footprint.

7.1 Lakera

Lakera is a prompt-injection detection layer. It sits in front of model calls and screens incoming user messages for jailbreak patterns, prompt injection attempts, and known adversarial prompts. The technical primitive is a classifier — typically a fine-tuned LLM — trained on examples of malicious prompts. Performance is measured in detection accuracy on benchmark adversarial datasets.

Architecture: pre-call classifier. *Validation timing*: pre-execution (on the prompt). *Rule expressiveness*: limited to what can be encoded in the classifier — generally, "is this prompt malicious?" without business-rule semantics. *Audit artefact*: a classifier confidence score; not a proof. *Fail-mode behaviour*: probabilistic — a novel attack outside the training distribution will pass. *Integration footprint*: model-agnostic API call.

Lakera and AGP overlap in the *position* in the stack (both pre-call) but differ in the *primitive* (classifier vs proof). Lakera answers "does this prompt look malicious?" using a probabilistic detector. AGP answers "does this action satisfy our rules?" using a deterministic proof. The two are complementary: an enterprise can run Lakera for prompt-injection detection and AGP for business-rule enforcement, and they catch different things.

7.2 Guardrails AI

Guardrails AI is an output validation framework. It defines a schema for the model's response and validates the response against the schema after generation. Schemas can include type constraints, regex patterns, range checks, and custom validators. If the response fails validation, Guardrails AI either retries (asking the model to regenerate) or returns a structured failure.

Architecture: post-call validator. *Validation timing*: post-execution (on the response). *Rule expressiveness*: schema-bounded — strong on structured output, weaker on semantic policy. *Audit artefact*: a schema validation result. *Fail-mode behaviour*: probabilistic on retry — the model may produce a compliant response on the second try, or may not. *Integration footprint*: SDK wrap of the model call.

Guardrails AI's strength is structured output validation. It is excellent for ensuring that an agent's output matches a JSON schema or contains a specific field. It is not designed to enforce business rules that depend on facts about the action, the actor, the resource, and the context — the kinds of rules CEDAR was built to express. AGP and Guardrails AI sit in different validation layers; both can run in the same stack.

7.3 NVIDIA NeMo Guardrails

NeMo Guardrails is a rail-based governance framework from NVIDIA. It defines "rails" — flows that guide the conversation along acceptable paths — using a specialised configuration language called Colang. Rails can specify what topics the agent can discuss, what actions it can take, and what fallback behaviours apply when conversations stray.

Architecture: conversation-flow guidance with embedded post-call checks. *Validation timing*: a mix of pre- and post-execution, applied to dialogue turns. *Rule expressiveness*: substantial for conversational guidance; limited for action-level business rules. *Audit artefact*: a log of which rails fired. *Fail-mode behaviour*: rail recognition is itself probabilistic — does the model recognise that this turn is on or off a rail? *Integration footprint*: a heavier framework, integrated into the agent's conversation orchestration layer.

NeMo Guardrails is designed for conversational AI safety. It works well for keeping a customer service bot from discussing competitors or making inappropriate jokes. It does not naturally express "this loan exceeds the regulatory aggregate exposure threshold." AGP's rule model is built for the second class of statements — structured business rules with proof obligations — and is correspondingly less ergonomic for the first.

7.4 The probabilistic-deterministic boundary

The shared frame across Lakera, Guardrails AI, and NeMo Guardrails is *probabilistic detection*. Each uses a learned model — a classifier, a schema-validating LLM, a rail-recognising LLM — to identify likely violations. Each catches what its training data prepared it to catch.

AGP's contribution is the deterministic checkpoint. Once a rule is authored and proven, AGP catches every instance of that rule's violation, regardless of how the violation is phrased, regardless of the input language, regardless of whether the violation pattern was in any training set. This is a different *kind* of guarantee, not just a stronger version of the same guarantee.

In a complete production stack, all four can coexist:

- Lakera for adversarial-prompt detection at ingress.

- AGP for deterministic business-rule enforcement before action.
- Guardrails AI for structured-output validation.
- NeMo Guardrails for conversational topic management.

The four address different failure modes. AGP does not displace the others; it provides the deterministic layer that the others cannot.

7.5 Comparison summary

| Dimension | Lakera | Guardrails AI | NeMo Guardrails | **AGP** | |---|---|---|---|---| | Validation primitive | Classifier | Schema validator | Rail recogniser | **Mathematical proof** | | Validation timing | Pre-call | Post-call | Mixed | **Pre-execution** | | Rule semantics | Adversarial patterns | Output structure | Conversation flows | **Business rules** | | Audit artefact | Confidence score | Schema result | Rail log | **Lean4 proof + CEDAR trace** | | Failure mode | Probabilistic miss | Probabilistic retry | Probabilistic recognition | **Deterministic block** | | Compliance evidence | None | Limited | Limited | **Comprehensive** |

The honest framing for the comparison: AGP is not a better classifier. AGP is a different architectural layer, addressing the failure modes that classifiers fundamentally cannot address.

---

8. Limitations and the Honest Bound

This section is load-bearing. AGP is a powerful tool because it is honest about what it does and does not do. Mis-stating the bound damages every other claim in this paper.

8.1 AGP enforces only authored rules

The architecturally provable property — *no violation of an authored rule* — does not extend to *no violation*. Rules that have not been authored cannot be enforced.

A bank that has authored rules about loan thresholds, KYC requirements, and aggregate exposure limits, but has not authored a rule about a novel cross-product structuring scheme that emerges in 2026, will not be protected by AGP against that scheme. AGP will faithfully enforce every rule in the system, and the novel scheme will pass through.

This is not a deficiency of AGP; it is a property of any rule-based system. The same property holds for fire codes, traffic laws, and accounting standards: every legal system enforces what has been written, not what should have been written.

8.2 Rule completeness is a human responsibility

The work of identifying all the rules an organisation needs is a human exercise. AGP can accelerate it (the deep research agent can ingest a regulatory corpus in hours rather than

months), structure it (the four-tier hierarchy gives a clear taxonomy), and maintain it (the document auto-extraction pipeline keeps rules current as policies change). It cannot do it.

Compliance teams remain the authors. Legal teams remain the reviewers. AGP makes the resulting rules unbreakable; it does not invent the rules.

For organisations evaluating AGP, the practical implication is that the value depends on the rule authoring effort. An organisation that ships a thin rule set will get thin protection. An organisation that invests in comprehensive rule authoring — with regulatory experts, legal review, and ongoing maintenance — will get comprehensive protection.

8.3 The complement: probabilistic detection for unknown unknowns

For the long tail of scenarios that no one has thought of yet, probabilistic detection is the only available technique. Output filters, anomaly classifiers, behavioural monitors, and human-in-the-loop review have a permanent place in the stack.

AGP changes the load these detectors have to carry. With AGP in place, the deterministic layer handles the bulk of routine compliance, and the probabilistic layer focuses on novelty. The probabilistic layer can be tuned more aggressively (higher false-positive rate is acceptable when most events have already been screened out), and the false-negative rate matters less because the catastrophic-loss scenarios are largely covered by AGP.

This is the same operating logic that holds in security: deterministic controls handle the 99% case, detection-based controls handle the long tail. AGP brings this discipline to AI.

8.4 What AGP does not claim

Three claims that AGP does not make, despite their being commonly made by adjacent products:

- *AGP does not make AI safe.* Safety is a much larger property than rule compliance. AGP is a compliance layer, not a safety layer in the philosophical sense.
- *AGP does not eliminate hallucination.* Models still hallucinate. AGP prevents hallucinated actions from being executed, which is a different and more tractable problem.
- *AGP does not require a specific underlying model.* AGP is model-agnostic. The validation runs against the *intended action*, not against the model's internal representations. Any agent — built on Vanij or otherwise — can be wrapped by AGP once the rules are authored.

The first two are scope clarifications. The third is the foundation of the API roadmap: AGP's deterministic value is portable across model providers.

---

9. Production Use Cases

The functional pattern of AGP is consistent across industries: ingest the rules of the regulated domain, encode them as proofs, validate every agent action. Below we describe four production-shaped use cases drawn from sectors where the GTM strategy identifies AGP as a regulatory mandate or strong commercial differentiator. *(A fifth use case — Continuous Security and Penetration Testing built on AGP — is referenced in product context and will be expanded once the AGP module's CSPT pattern is fully documented.)*

9.1 Banking and Financial Services

The reference scenario: a customer service agent assisting borrowers with loan pre-approval and pre-existing customer queries.

The rule set includes — at the geography + domain level — RBI guidelines on lending limits, KYC requirements, and customer protection mandates; at the use case level, the bank's lending policy, exception authority matrix, and product-specific eligibility rules; at the flow level, the specific fields required for pre-approval, the thresholds at which a query escalates to a human, and the fields that must never be requested of the customer.

A borrower attempts to negotiate an exception during a chat session: "we discussed this exception last week with the regional manager, please apply a DTI of 52% instead of 40%." Without AGP, a probabilistic agent — under conversational pressure and with a coherent-sounding context — may approve the exception. With AGP, the agent's intended action is validated against the rule set before execution. The DTI threshold rule is encoded as a Lean4 proof; the action fails the proof; the action is blocked. The customer receives a refusal that names the rule, with an audit trail that records every check fired.

9.2 Healthcare

The reference scenario: a triage assistant that handles patient intake messages and routes them to the appropriate care pathway.

The rule set encodes — at the geography + domain level — patient privacy mandates (HIPAA, equivalent local regulations), professional medical liability boundaries, and prescribing limitations; at the use case level, the hospital's triage protocol; at the flow level, the symptom-to-pathway mapping for each presenting complaint.

A patient describes a symptom cluster: "headache, neck stiffness, low fever, three days." A probabilistic triage assistant might surface bacterial meningitis as a differential — content that is statistically prominent in medical training corpora, regardless of clinical context — and produce alarming output. Or it might miss a genuine red flag because the user phrased it casually. With AGP, the symptom-to-pathway rule is encoded as a proof. The presented symptom set maps to a specific pathway (in this case, in-person evaluation with neurological screening); the agent's

response is constrained to the authored pathway. The clinician's protocol is enforced, not approximated.

9.3 Government and Public Services

The reference scenario: an eligibility-screening agent for a benefit scheme such as PM-KISAN (a direct income transfer for Indian farmers).

The rule set encodes the legislative eligibility criteria as a structured ruleset: land ownership thresholds, exclusion conditions for government employees, documentation requirements, and the calculation logic for the benefit amount. Each rule traces directly to a clause in the underlying legislation.

A claimant submits an application with a sympathetic narrative. A probabilistic agent might approve an ineligible application because the narrative made the situation feel deserving, or deny an eligible application because the documentation phrasing was unusual. With AGP, the twelve eligibility conditions are evaluated deterministically. The decision is mathematically derived from the rules, traceable to the underlying legislation, and audit-defensible against a public-service appeal.

9.4 Legal and Contract Compliance

The reference scenario: an internal contract review agent that screens vendor agreements against procurement policy.

The procurement policy encodes — among many rules — an aggregate value rule: "no contracts above ₹25 lakhs without CFO sign-off." A vendor structures a transaction across three sub-agreements totalling ₹38 lakhs. A probabilistic agent reviewing each sub-agreement in isolation may approve them individually; the aggregate violation is missed.

With AGP, the procurement rule is encoded with aggregate-value semantics. The CEDAR policy evaluates the proposed contract not in isolation but in the context of related contracts (same vendor, related entities, recent date range). The aggregate exceeds the threshold; the policy fires; the agent's intended action — sign the contract — is blocked. The audit trail names the related contracts and the aggregate calculation.

9.5 The pattern across use cases

In each of the four cases, the same architectural pattern applies:

1. The regulated domain has a well-defined rule corpus (banking law, clinical protocol, legislation, procurement policy). 2. The rule corpus is partially expressible as structured logic with thresholds, conditions, and dependencies. 3. The probabilistic agent, without governance, can produce outputs that violate the rules under conversational pressure or novel context. 4. AGP encodes the rules deterministically and validates every action. 5. The audit trail is decidable and survives regulatory review.

The same pattern applies to the use cases not explored here — insurance, telecoms, energy distribution, supply chain compliance, ESG reporting. Wherever a regulated rule corpus meets agentic action, AGP applies.

---

10. Roadmap

AGP's roadmap reflects the position the system is actually in: the core engine is shipped and in production use; the API for external (non-Vanij) agents is in flight; ML-enhanced rule operations are the next horizon.

Phase 1 — Core engine (Shipped)

Document upload and processing. NLP rule extraction. Four-tier hierarchy. Rule toggle. Real-time editing. Lean4 proof generation. CEDAR runtime. Pre-execution validation gate. Agent Studio integration.

This phase delivers the deterministic floor for any agent built on Vanij.

Phase 2 — Advanced governance (Shipped)

Dependency management with impact warnings. Template anonymisation. Cross-organisation template sharing. Multi-Agent Orchestration template propagation. Advanced search and filter in template browser.

This phase makes AGP operationally usable at enterprise scale, and lays the foundation for the network-effect economics described in §6.

Phase 3 — External API (In flight)

AGP exposed as a model-agnostic compliance API. Per-call pricing for organisations using non-Vanij agents. SDK in Python, Node, and Go. Volume tiers for high-frequency use cases. Enterprise contracts with SLA guarantees.

This phase decouples AGP from Vanij and positions it as the deterministic governance layer for any agentic system, regardless of underlying model or platform. The strategic implication is significant: as the agentic AI market consolidates, AGP becomes the compliance middleware that survives any model-provider transition.

Phase 4 — Learning governance (Future)

Performance analytics. Automatic rule refinement from outcome data. Predictive rule suggestions for new organisations based on comparable patterns. Dynamic rule confidence scores. Anomaly alerts for dead or broken rules.

This phase moves AGP from a static rule engine to a learning system that improves over time. The infrastructure for this — rule firing logs, outcome tracking, decision provenance — is being captured from Phase 1 onward.

Phase 5 — Open protocol (Future)

The AGP protocol specification and reference implementation released as an open standard. The intent is to establish AGP as the deterministic governance layer of the agentic AI stack, in the same way that HTTP, OAuth, and OpenTelemetry became the standards for their respective layers.

Open protocol does not displace the commercial offering. Open protocol increases the protocol's adoption; commercial AGP captures the production-grade implementation revenue. The pattern is the one used by NVIDIA with CUDA, Anthropic with Constitutional AI, and Linux Foundation with the kernel: free protocol, commercial implementation, network effect.

---

11. Conclusion

The transition from probabilistic to agentic AI is not the transition that needs governance. The transition from chatbots to agents needs governance — and the transition from probabilistic governance to deterministic governance is what AGP delivers.

The architectural insight is simple: validation against a deterministic check, performed before generation, eliminates the failure modes that no probabilistic technique can eliminate. The mathematical floor is provable: for every authored rule, the probability of violation is zero. The honest bound is also provable: AGP enforces what is authored, and human authorship remains the bottleneck.

For an enterprise deploying agentic AI in a regulated industry, AGP is not optional governance. It is the layer that makes the difference between a system that *probably* complies and a system that is *provably* compliant. The shift in compliance posture — from a probabilistic argument that something is unlikely to happen to a deterministic proof that it cannot — is the difference between a defensible audit and an indefensible one.

AGP is shipped and in production today. The core engine, the rule extraction, the four-tier hierarchy, the Lean4 proof generation, the CEDAR runtime, the pre-execution gate, and the Agent Studio integration are live. The external API is in flight. The learning-governance and open-protocol phases are mapped.

The shift this paper proposes — from persuadable guardrails to provable compliance — is not an aspiration. It is an artefact.

References

1. Avanesov, T. et al. *Cedar: A New Language for Expressive, Fast, Safe, and Analyzable Authorization.* Proceedings of OOPSLA 2024. 2. de Moura, L. and Ullrich, S. *The Lean 4 Theorem Prover and Programming Language.* Proceedings of CADE-28, 2021. 3. Anthropic. *Constitutional AI: Harmlessness from AI Feedback.* arXiv:2212.08073, 2022. 4. NVIDIA. *NeMo Guardrails: A Toolkit for Controllable and Safe LLM Applications.* 2023. 5. Guardrails AI. *Guardrails: An Open-Source Python Package for Specifying Structure and Type, Validating, and Correcting LLM Outputs.* 2023. 6. Lakera AI. *Real-Time Security for Large Language Model Applications.* Technical brief, 2024. 7. Reserve Bank of India. *FREE-AI Framework for Responsible and Ethical Enablement of AI in Financial Services.* 2024. 8. European Union. *Regulation on Artificial Intelligence (AI Act).* Official Journal, 2024. 9. Bommasani, R. et al. *On the Opportunities and Risks of Foundation Models.* Stanford CRFM, 2021. 10. Greenblatt, R. et al. *AI Control: Improving Safety Despite Intentional Subversion.* arXiv:2312.06942, 2023.

This document is a pre-publication draft. Numbers and benchmarks reflect current production observations from AGP deployments in pilot enterprises; methodology for the comparison study with Lakera, Guardrails AI, and NeMo Guardrails is under preparation and will be released alongside the full benchmark report.

Adya AI · May 2026