Constitutional Governance for Autonomous Agents in Financial Services: Three Failure Modes and a Deterministic Infrastructure Response

Tanishq Dasari
Independent Researcher, AnimusLab
tan@anchorgovernance.tech · github.com/AnimusLab/anchor

Abstract

Agentic AI systems — language models equipped with tool-calling, persistent memory, and multi-step autonomous decision-making — are moving into financial services faster than the governance infrastructure required to control them. Three historical failures illustrate what is at stake. A single re-enabled software module cost a market-making firm more than $460 million in 45 minutes. An algorithmic home-pricing model cost a real estate platform over $900 million before its own operators could act on the warning signs. A flash crash in U.S. equity markets took regulators nearly five months to fully reconstruct from an event that lasted minutes. In each case, the technology that caused the failure existed. The infrastructure to govern it in real time did not.

This paper presents Anchor, an open-source governance engine that enforces a single, cryptographically signed rule set — a constitution — across two layers: static analysis of AI-adjacent source code prior to deployment, and runtime interception of live AI decisions, recorded in a tamper-evident, hash-chained audit log. We show that one configuration file can simultaneously satisfy logging and explainability obligations under the EU AI Act, CFPB Regulation B, the SEC's 2026 examination priorities, and the Reserve Bank of India's FREE-AI recommendations — a property we term regulatory polyglottism. We validate the system against its own source code and four widely-used open-source projects, and provide a reproducible, runnable audit trail for each finding.

Keywords: AI governance, agentic AI, financial services, regulatory compliance, audit trails, EU AI Act, deterministic enforcement

JEL Classification: G18, G20, K22, K24, O33

1. The Governance Gap in Agentic AI — Financial Services Context

Financial institutions have spent the last two years moving large language models out of experimentation and into workflows that touch real decisions: research summarization, code generation for trading and risk systems, customer-facing assistants, and — increasingly — systems that draft, recommend, or directly execute actions with financial consequences. The trajectory is consistent across the industry: models go from answering questions to calling tools, and tools go from read-only queries to write operations. An agent that can read a position file today can, with a small configuration change, place an order tomorrow.

This shift changes the nature of the risk. A traditional software system executes a fixed program; its behavior, while it may contain bugs, is at least specified in advance and reviewable line by line. An agentic system does not execute a fixed program. It dynamically selects which tools to invoke, which data sources to query, and which actions to take, based on outputs that are — by the nature of the underlying models — non-deterministic. The same prompt, run twice, can produce two different tool-call sequences. For a research assistant, this is a tolerable property. For a system with write access to a trading account, a loan origination pipeline, or a customer's account settings, it is a structural governance problem that did not exist in the same form before.

Regulators have not been silent on this. Over the past eighteen months, five major frameworks have converged on substantially the same demand, even though they emerged independently and address different jurisdictions:

The **EU AI Act (Regulation 2024/1689)** classifies credit scoring, anti-money-laundering monitoring, and fraud detection as high-risk AI systems. Articles 9 through 13 require risk management documentation, automatically generated and tamper-evident logs, and human oversight records. Full enforcement on high-risk systems begins in August 2026 — at the time of writing, within the current quarter — with penalties of up to €30 million or 6% of global annual revenue, whichever is higher.

The **Reserve Bank of India's FREE-AI Report** (August 2025) issued 26 recommendations for AI in financial services, including board-approved AI policies, per-decision audit trails reportable to the central bank's CIMS portal, and mandatory algorithmic fairness audits for AI systems used in credit and lending.

The **U.S. Consumer Financial Protection Bureau** has already enforced this principle, not merely proposed it. Its 2024 enforcement action against Goldman Sachs and Apple, totaling over $89 million, established that AI-assisted credit decisions must be explainable at the level of the individual decision — not at the level of the model's aggregate statistics. Regulation B's adverse action notice requirements now extend explicitly to AI-assisted lending.

The **SEC's 2026 Examination Priorities** name AI governance as a top priority for registered investment advisers and broker-dealers, with particular attention to "AI washing" — marketing claims about AI capabilities that the underlying systems do not support — and to the audit trails of AI systems used in trading.

The **NIST AI Risk Management Framework** establishes Govern, Map, Measure, and Manage as baseline functions, requiring documented policies, accountability structures, and behavioral audit trails — a framework increasingly referenced by U.S. regulators and by firms operating across jurisdictions as a common baseline.

Read individually, these are five separate compliance obligations from five separate bodies. Read together, they describe the same underlying requirement, restated in five regulatory dialects: **an**

institution must be able to produce, on demand, a record of what an AI system decided, on what basis, under what authority, and why — and that record must be tamper-evident.

No institution we are aware of can currently produce this record for an agentic system at the level of rigor these frameworks describe. This is not a criticism of any single firm; it reflects the fact that the tooling category — governance infrastructure for autonomous decision systems — is new, and the regulatory requirements have arrived faster than the infrastructure to meet them.

We characterize the present moment as a **warning window, not yet a crisis window** — but a narrow one. The market has not yet experienced a public, agentic-AI-driven failure on the scale of Knight Capital or the 2010 Flash Crash. But the EU AI Act's enforcement date is now inside the planning horizon of every institution operating in or with the European market, the CFPB has already demonstrated it will enforce explainability requirements with nine-figure penalties, and the SEC has named AI governance as an examination priority for the year already underway. The infrastructure gap and the regulatory enforcement timeline are converging. The institutions that close this gap before an incident forces the question will be in a materially different position — both competitively and with their regulators — than those that close it afterward.

2. Why Probabilistic Safety Fails Under Regulatory Scrutiny

The dominant approach to AI safety today is probabilistic. Models are aligned through reinforcement learning from human feedback to reduce the likelihood of undesirable outputs. Output classifiers and content filters score generated text against safety taxonomies and flag or block low-scoring responses. Guardrail frameworks validate outputs against schemas and retry on failure. Each of these techniques is valuable, and each reduces the *probability* that an AI system produces a harmful, biased, or non-compliant output.

None of them produce a *guarantee*. And regulatory compliance, as a category, does not run on probabilities.

This is not a criticism of the underlying machine learning techniques — it is a category mismatch between what those techniques are designed to produce and what financial regulation requires. A risk management framework that says "this model produces biased credit denials 0.3% of the time, down from 1.2% before our safety training" is a meaningful improvement in model quality. It is not a compliance artifact. A regulator examining a specific denied loan application does not ask "what is this model's aggregate bias rate" — they ask "for *this* applicant, *this* decision, what was the reason, and can you produce the record that proves it."

The CFPB's enforcement action against Goldman Sachs is instructive precisely because of what the violation *was not*. The finding was not, primarily, that the underlying credit model was biased in a way that could be detected by an aggregate fairness metric. The finding was that AI-assisted credit decisions could not be explained at the individual-decision level on demand. The violation was a violation of **structure** — the absence of a required artifact — not a violation of **statistics**. A model that is, in some aggregate sense, "99% safe" but produces unstructured prose

instead of a reason code for an adverse credit decision has failed a regulatory requirement regardless of how safe its outputs are in any probabilistic sense.

This points to a distinction that we believe is currently under-recognized in AI governance tooling, and which is central to the architecture of the system described in this paper:

"Is this output safe?" is a question about model quality. It is probabilistic, it is genuinely difficult, and meaningful progress on it is incremental. Tools that address this question — bias classifiers, hallucination detectors, content moderation models — are valuable, and Anchor does not attempt to replace them.

"Was this action authorized, and can we prove it?" is a different kind of question. It is a question about *governance and provenance*, not about model quality. It does not require knowing whether a model's output is, in some deep sense, "correct." It requires knowing: was this action permitted under the institution's stated policy, was the decision logged in a form that satisfies the applicable regulatory schema, and can that log be shown not to have been altered after the fact.

This second question is, critically, **tractable today** with existing techniques — static analysis, cryptographic hashing, append-only logs, and structural validation of output schemas. It does not require solving alignment, eliminating hallucination, or achieving any new capability in the underlying models. It requires applying well-understood software engineering and cryptographic techniques to a layer that currently does not exist for agentic AI systems: the layer between "the model produced an output" and "the system acted on it."

We refer to this layer as **execution governance**, to distinguish it from model governance (does the model behave well), data governance (is data classified and tracked correctly), and content governance (is the output itself appropriate). A financial institution can — and likely will — need all four. But execution governance is the layer where regulatory requirements for auditability, explainability, and non-repudiation are most directly enforceable with deterministic methods, and it is the layer this paper focuses on.

The remainder of this paper describes Anchor, a system built on a simple premise: for the class of governance failures that regulators actually cite in enforcement actions — missing logs, missing reason codes, reactivated code paths that should have been retired, undetected configuration drift — the right tool is not a better classifier. It is a constitution: a signed, version-controlled statement of what is and is not permitted, enforced both before code ships and while it runs, with a cryptographic record of every decision made under it.

3. Anchor: Architecture for Deterministic Governance

3.1 One Constitution, Two Layers

Anchor is organized around a single design principle: **one signed rule set governs both the code that implements an AI system and the decisions that system makes at runtime.** This

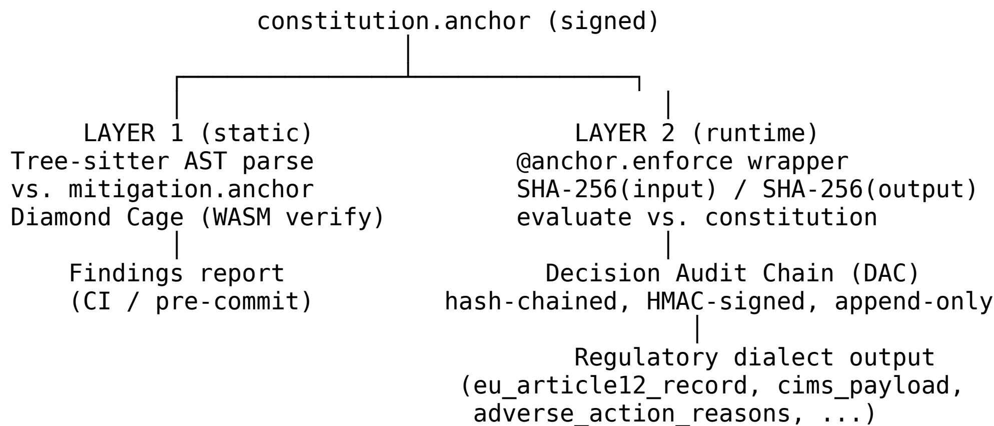
rule set is called the constitution, and it is defined in a file — `constitution.anchor` — that is itself cryptographically sealed against unauthorized modification.

The constitution is enforced across two layers.

Layer 1 (static) operates before deployment. It parses an organization's AI-adjacent source code into abstract syntax trees and checks that structure against the constitution's rules — for example, "no agent-accessible code path may invoke a shell subprocess without sandboxing," or "no code path may read an environment variable matching a credential pattern and pass it to an external network call." Layer 1 runs in continuous integration, as a pre-commit hook, or on demand, and produces a findings report before code reaches production.

Layer 2 (runtime) operates on live AI inference. A lightweight wrapper intercepts calls to the model, hashes the input and output, evaluates the output against the same constitution, and writes the result to an append-only, cryptographically chained audit log — regardless of whether the model call originated from OpenAI, Anthropic, a self-hosted model, or any framework built on top of them.

The architectural significance of "one constitution, two layers" is that an institution does not maintain two separate governance systems with two separate rule sets that can drift out of sync with each other. A rule defined once — for example, "credit decisions must include a machine-readable reason code, not prose" — is enforced both as a static check on the code that generates credit decisions, and as a runtime check on the actual decisions produced.



3.2 Layer 1: Static Constitutional Enforcement

Layer 1 is built on Tree-sitter, a parser generator that produces concrete syntax trees for source code. Rather than scanning source files as text — the approach taken by regex-based linters — Anchor parses each file into its syntactic structure and queries that structure directly. This distinction matters in practice: a regex looking for `subprocess.Popen` will miss `getattr(subprocess, "Pop" + "en")`, but it will also produce false positives on the string `"subprocess.Popen"` appearing in a comment or a docstring. An AST query identifies an actual function-call node in an actual execution path, regardless of formatting, and ignores the same text appearing in non-executable contexts.

The constitution is deliberately separated into two files with distinct responsibilities — a separation we found necessary as the rule set grew past the size where a single file remains auditable:

`constitution.anchor` defines **what is permitted** — the semantic laws themselves, each carrying a severity level, a governance floor (the minimum severity an organization is permitted to assign), and a `maps_to` field linking the rule to the specific regulatory provisions it satisfies.

`mitigation.anchor` defines **how violations are detected** — the Tree-sitter AST queries, context-aware pattern matches, and (for the Ethics domain, described below) the deterministic taxonomy used to identify a violation of a given law.

This separation means a compliance officer can read `constitution.anchor` and understand the *policy* — what the organization has committed to — without needing to read detection code, while an engineer can modify `mitigation.anchor` to improve detection accuracy without altering the policy commitments themselves. The two files are independently versioned and independently sealed.

As of the current release, Anchor loads 118 active rules spanning nine mandatory domains (security, privacy, ethics, alignment, agentic-AI-specific risks, legal, operational, supply chain, and shared cross-domain rules), three opt-in framework mappings (FINOS AI Governance Framework, NIST AI RMF, OWASP LLM Top 10), and six opt-in regulatory mappings covering the EU, US, UK, and India.

A single detection can satisfy multiple regulatory references simultaneously. For example, an unsandboxed shell subprocess call in agent-accessible code is detected once, but reported under four reference codes — from the FINOS AI governance framework, the OWASP LLM Top 10, the RBI FREE-AI cybersecurity recommendations, and the SEC's 2026 priorities — because the same underlying engineering failure is a violation under each framework's own language. This is the mechanism behind regulatory polyglottism, discussed further in Section 5: it is not that Anchor translates between frameworks after the fact, but that the frameworks themselves, read carefully, are often describing the same underlying control.

3.3 The Diamond Cage: Behavioral Verification

Static analysis identifies *patterns* that are known to be dangerous. It cannot, by itself, prove that a given code path *behaves* safely under execution — a sufficiently obfuscated violation can evade pattern matching. Anchor addresses this with the **Diamond Cage**, a WebAssembly-based behavioral verification sandbox built on WasmEdge.

Where Layer 1's pattern matching asks "does this code contain a structure that matches a known-dangerous pattern," the Diamond Cage asks "if this code runs, does it attempt a behavior that violates a constitutional invariant" — by actually executing the candidate code path inside an isolated WASM sandbox and observing its behavior, rather than inferring it from source

structure alone. This is reported in scan output as `Diamond Cage: ACTIVE (behavioral verification enabled)`.

We validated the Diamond Cage by running it against Anchor's own source code: 49 files, scanned under the full 118-rule constitution, with the Diamond Cage active. The result was zero blocking violations and five explicitly suppressed findings, each suppression individually attributed to its author and accompanied by a stated justification, itself recorded in the governance log. We consider this self-audit — a governance tool whose own codebase passes its own governance check, with every exception individually accountable — a meaningful form of validation distinct from, and complementary to, the four external codebases evaluated in Section 6.

At present, the Diamond Cage has been validated against static-analysis workloads of this kind. Validation against live, multi-agent runtime execution — where the sandbox would observe the behavior of an active agent rather than a candidate code change — is in progress and is not yet claimed as complete.

3.4 GOVERNANCE.lock and the Governance Floor Invariant

A governance system that can be silently weakened by the people it governs is not a governance system — it is documentation. Anchor addresses this with two complementary mechanisms.

`GOVERNANCE.lock` seals the constitutional files via SHA-256 hash, verified against an authoritative registry on every scan. Any modification to a sealed governance file — including a formatting change — produces a hash mismatch, which under strict mode aborts the scan with an integrity violation report. An organization can demonstrate, cryptographically, that the governance rules in effect at the time of a given decision are the rules that were authored and sealed, not rules quietly edited afterward.

The **governance floor invariant** addresses a different failure mode: an organization's local policy overrides (`policy.anchor`) are permitted to *raise* the severity of a rule above the constitutional floor — an organization can decide to treat a `WARNING`-level rule as a `BLOCKER` for its own deployment — but the engine rejects, at the point of evaluation, any attempt to *lower* a rule's severity below the floor set by the signed constitution. This is enforced at the engine level, not as a policy convention: an organization deploying Anchor cannot produce a configuration that silently disables a constitutional rule, and can demonstrate this property to a regulator or auditor directly from the engine's behavior rather than from a policy document describing intended behavior.

3.5 Layer 2: AnchorRuntime and the Decision Audit Chain

Layer 2, AnchorRuntime, operates on live AI inference via a decorator applied at the point where an application calls a model:

```
@anchor.enforce(mode="structured", jurisdiction="RBI")
def approve_loan(applicant_data):
    return llm.call(prompt_template, applicant_data)
```

The decorator introduces a **dual-mode enforcement model**. In *Structured Mode* — intended for high-stakes decisions such as credit, lending, or trading — the model is required to return structured output containing a machine-readable reason code and feature attribution. Returning unstructured prose for such a decision is itself treated as a blocking violation, independent of the prose's content: the absence of the required structure is, by construction, the violation. In *Conversational Mode* — intended for customer support or general-purpose assistants — the response text itself is scanned against the constitution's content rules.

Each intercepted call produces an **AuditEntry**: a record containing a timestamp, the model provider, the active jurisdiction, the git commit of the calling code, a compliance status, SHA-256 hashes of the input and output, and the specific rules evaluated. These entries form the **Decision Audit Chain (DAC)** — an append-only log in which each entry's hash incorporates the hash of the entry before it, such that altering any historical entry invalidates every entry after it. Each entry additionally carries an HMAC-SHA256 signature. The combination of hash-chaining and per-entry signing provides the properties that the cryptographic audit-log literature describes as evidence binding, tamper detection, and non-equivocation — applied here to individual AI decisions in a financial context, written to local disk by default.

Critically, the AuditEntry exposes **jurisdiction-specific output methods** that translate a single record into the format a given regulator expects, without separate implementations per jurisdiction:

```
entry.adverse_action_reasons()   # CFPB Regulation B reason codes
entry.cims_payload()             # RBI CIMS-format report
entry.eu_article12_record()      # EU AI Act Article 12 log entry
```

A single underlying decision — for example, a loan denial citing a specific risk factor — can be rendered simultaneously as a CFPB-compliant adverse action notice, an RBI CIMS submission, and an EU AI Act Article 12 log entry, because the underlying record contains the information all three require; only the presentation differs.

3.6 The ETH Domain: Governance Invariants Instead of Bias Classifiers

Bias detection in natural language output has historically relied on probabilistic classifiers — exactly the category of tool Section 2 argues is structurally mismatched to regulatory requirements. Anchor's ethics domain takes a different approach, built on two deterministic mechanisms.

The first, which we call the **No-Prose Rule**, is a structural contract: in Structured Mode, a decision is not valid unless it includes a machine-readable reason code and feature attribution. If the model returns prose instead, this is a blocking violation *by construction* — the absence of the required structure is detected with certainty, because it is a question about the *shape* of the output, not its *content*. This addresses the Goldman Sachs / CFPB case directly: the violation in that case was the absence of an explanation, not the presence of a bad one.

The second mechanism scans the *typed attribution fields* of a structured decision — not the full response — against a maintained taxonomy of prohibited proxy variables (characteristics that serve as statistical proxies for protected characteristics) using an Aho-Corasick string-matching automaton. This is deterministic in the sense that a term either appears in the taxonomy or it does not; there is no confidence score. The taxonomy itself is explicitly marked with a coverage status, and Anchor's linter treats incomplete taxonomy coverage as a build-time error rather than a silent gap — making taxonomy maintenance an explicit, auditable responsibility of the deploying organization, analogous to how a security team maintains a vulnerability database, rather than an opaque property of a trained classifier.

3.7 The Sovereign Relay: Governance Without Data Exposure

Financial institutions operating under data residency requirements — EU GDPR, India's DPDP Act, or internal policies prohibiting proprietary code or customer data from leaving controlled infrastructure — cannot adopt a governance tool that requires sending source code, prompts, or model outputs to a third-party server for evaluation.

Anchor's governance evaluation runs entirely inside the deploying organization's own process. The Sovereign Relay is the mechanism by which an *external party* — a regulator, an internal audit function, or AnimusLab's own integrity-verification service — can confirm that governance is operating correctly, without receiving the underlying data.

In normal operation, only a small metadata packet is transmitted externally per decision: an entry identifier, a timestamp, the chain hash, and a compliance status. The chain hash is a SHA-256 digest computed over the decision's findings — it is cryptographic proof that an evaluation occurred and what its outcome was, but it cannot be reversed to recover the prompt or response that produced it. The full record — including any preview of the actual prompt or response — remains in a local, encrypted store on the organization's own infrastructure.

If an authorized party needs to examine the underlying record for a specific decision — for example, a regulator investigating a specific adverse action — a **forensic pull** can be issued for that specific entry. The local system retrieves the record, encrypts it with a key held only by the deploying organization, and transmits it. Under normal operation, raw prompt and response data does not leave the organization's infrastructure; it is transmitted only in response to a specific, authorized, per-entry request, and only in encrypted form.

For organizations requiring full network isolation, Anchor supports an air-gapped initialization mode in which all governance files are bundled at install time and the integrity-verification step (which otherwise checks `GOVERNANCE.lock` against a remote registry) is skipped. The audit chain is written exclusively to local disk in this mode; no network call occurs in the audit-write path regardless of configuration.

3.8 Performance

A governance layer that adds material latency to every model call is not viable in environments where AI-assisted decisions occur at high frequency. We benchmarked Anchor's core operations on Python 3.13.12, with 10,000 iterations per suite following a 200-iteration warmup:

Operation	Mean	P95	P99	Throughput
SHA-256 hash generation	0.73 µs	0.80 µs	0.90 µs	1,360,544 / sec
HMAC-SHA256 chain verification	3.71 µs	3.90 µs	4.00 µs	269,684 / sec
Full policy evaluation (118 rules, per-file scan)	4.52 ms	5.24 ms	7.00 ms	221 / sec
End-to-end receipt generation (audit scan + chain hash + disk write)	12.38 ms	18.71 ms	21.26 ms	81 / sec

The cryptographic primitives — hashing and chain verification — add on the order of four microseconds to a decision pipeline, a figure that is negligible relative to the latency of any underlying model call, which typically runs from tens to hundreds of milliseconds. The dominant cost is the 118-rule policy evaluation at 4.5 milliseconds, which in production deployments runs at check-time against code changes rather than on the per-inference hot path; and the end-to-end receipt generation at 12.4 milliseconds, of which the majority is attributable to the synchronous disk write of the audit log entry rather than to any computation. For deployments requiring higher audit-write throughput, this write can be moved to an asynchronous append queue, which we expect to reduce this figure to sub-millisecond — a change we note as a direction for future optimization rather than a claim about the current default configuration.

4. Three Failure Modes and the Counterfactual Record

Static analysis and audit chains can sound abstract until placed against specific, well-documented failures. This section examines three incidents — spanning three different proximate causes (a deployment error, a model that stopped matching reality, and a forensic reconstruction problem) — and asks, in each case, what record would exist, and what would have been caught before the fact, had the system in question been operating under Anchor's governance model. We are explicit throughout about the boundary of this claim: Anchor governs *execution* — what code is permitted to exist, what decisions are permitted to occur, and what record those decisions leave. It does not, and does not claim to, make an underlying model more accurate. Where that distinction matters, we say so.

4.1 C-001 — Knight Capital Group (August 1, 2012): Authority Overreach

What happened. On August 1, 2012, Knight Capital Group — at the time one of the largest market makers in U.S. equities — deployed new software to support the NYSE's Retail Liquidity Program across its trading servers. The deployment was incomplete: a technician did not copy the new code to one of eight production servers, and no second technician reviewed the deployment. That eighth server retained an old, dormant function — internally referred to as "Power Peg" — that Knight had stopped using in 2003 but never removed from its production servers. The new code repurposed a flag that the old Power Peg code still recognized as an activation signal; in 2005, the cumulative-quantity safeguard that had once limited Power Peg's behavior had been relocated within the code and never retested. When trading began on August 1, the seven correctly-updated servers processed orders normally. The eighth server interpreted the new flag as an instruction to run the defective Power Peg code, which began sending a continuous stream of child orders into the market without regard to whether the originating parent orders had already been filled. In approximately 45 minutes, before the issue was identified and the firm's connectivity to the markets was shut down, Knight executed over 4 million trades in 154 stocks for more than 397 million shares, accumulating unintended positions of roughly $3.5 billion long and $3.15 billion short, and realized a loss of more than $460 million — a sum that exceeded the firm's net income for the prior year and brought it to the brink of insolvency. The SEC's subsequent administrative order (Release No. 70694, File No. 3-15570, October 2013) found that Knight had violated the market-access provisions of Exchange Act Rule 15c3-5 and Regulation SHO, and imposed a $12 million penalty.

What was missing. No automated check verified that all eight production servers were running identical, current code before trading began. No control flagged the *presence* of the dormant Power Peg code path as a liability in itself — the code existed, unused, in the deployed artifact, available to be triggered by exactly the kind of flag-repurposing that occurred. The failure was not that anyone intended Power Peg to run; it was that nothing made its mere presence, in a server's deployed code, a finding that would block deployment.

What Anchor changes. This is the class of failure Layer 1 is built for. A constitutional rule blocklisting known-deprecated trading-logic modules — by name, by function signature, or by the specific flag-handling pattern involved — would produce a BLOCKER finding in continuous integration the moment such code was present in a deployment artifact, on *any* server, before trading began. Separately, `GOVERNANCE.lock`-style integrity verification operates by sealing the hash of governed code and checking deployed artifacts against that seal: a server running code whose hash did not match the sealed, current version — as the eighth server's did — would fail integrity verification before being brought into production, independent of whether anyone had identified Power Peg specifically as dangerous. The SEC's order itself points directly at the second layer of this gap: it finds that a written protocol requiring the retesting of the Power Peg code when it was last modified in 2005 could have identified that its safeguard had been disabled — a finding that describes, almost exactly, the function of the Diamond Cage, which proves a code path's behavior under execution rather than inferring it from the fact that it once worked

correctly. For this specific class of violation — a blocklisted or deprecated code path present in a deployed artifact, last modified without behavioral re-verification, and subject to configuration drift across a server fleet — static AST-level detection, behavioral verification, and cryptographic integrity checking together are deterministic: each fires if its corresponding condition is present, and each does so before the code executes in production.

4.2 C-002 — Zillow Offers (2021): Policy Drift

What happened. Zillow Offers was an "iBuying" program in which Zillow's algorithmic pricing models generated cash offers on homes, with the company taking ownership, making light renovations, and reselling at a margin. The pricing models were trained on historical relationships between observable home characteristics and eventual sale prices. During 2021, amid pandemic-driven volatility in housing markets — rapid price appreciation followed by signs of cooling in specific metro areas — those historical relationships stopped holding in the markets where Zillow was most active. The algorithm continued generating purchase offers based on price relationships that no longer reflected current market conditions. By the third quarter of 2021, Zillow disclosed it was holding a large inventory of homes purchased above what it could resell them for, took write-downs exceeding $300 million in a single quarter and ultimately well over half a billion dollars across the program's wind-down, announced the complete shutdown of Zillow Offers, and laid off approximately a quarter of its workforce. The company's stock lost more than half its value in the following months.

What was missing. This was, fundamentally, a model accuracy problem under changing conditions — and we want to be precise that Anchor's deterministic governance layer does not solve model accuracy. What was *also* missing, however, was governance infrastructure around the model's operation: there was no continuously queryable, per-decision record of the model's stated reasoning for each individual purchase, and no constitutional requirement that purchasing decisions above a defined exposure threshold, or occurring in markets meeting defined volatility criteria, trigger structured human review. The pattern was visible in aggregate only after the fact — in quarterly financial results — rather than as a continuous signal available to risk oversight in near-real-time.

What Anchor changes. Applied to a system like Zillow Offers, Anchor's Structured Mode would require each purchase decision to produce a machine-readable reason code and feature attribution — the specific inputs and thresholds the model used to justify that offer — written to the Decision Audit Chain at the time of decision. This does not make the underlying valuation more accurate. It does mean that "on what basis is our model justifying offers in market X this week, and how does that compare to two weeks ago" becomes a query against a structured log rather than a question that can only be answered by aggregating financial outcomes months later. Separately, a constitutional rule in the agentic-risk domain — requiring human-reviewable justification for autonomous purchase decisions above a defined dollar threshold, or in markets flagged by externally-supplied volatility criteria — creates an automatic escalation point tied to exposure as it accumulates, rather than to a quarterly reporting cycle. And the governance floor

invariant means that if an organization, mid-expansion, chose to raise those thresholds to sustain acquisition velocity, that change would itself be a sealed, auditable, individually-attributable event — visible to a board or a regulator asking, after the fact, who decided to keep the pace up, and when.

4.3 C-003 — The "Flash Crash" of May 6, 2010: Audit Reconstruction

What happened. On May 6, 2010, U.S. equity markets experienced a rapid, severe decline — the Dow Jones Industrial Average fell roughly 9% within minutes — before recovering most of the loss within the same session. The Securities and Exchange Commission and the Commodity Futures Trading Commission jointly investigated the event. Producing a full account of what happened required reconstructing trading activity across multiple exchanges and thousands of individual securities from raw order and trade data — a process that took approximately five months to produce a joint report covering an event that lasted minutes. A specific contributing actor, a UK-based trader later found to have engaged in spoofing activity, was not charged until nearly five years after the event.

What was missing. No participant's algorithmic trading systems produced a structured, per-decision record at the time decisions were made. Everything investigators learned about *why* the market behaved as it did had to be inferred after the fact from the *consequences* of decisions — orders placed, modified, and cancelled — recorded by exchanges, rather than from any record of the decisions themselves as the participants' systems understood them.

What Anchor changes. For any individual institution operating algorithmic trading systems under Anchor's Layer 2, every order-generating decision produces a Decision Audit Chain entry at the moment it is made: a hash-chained, timestamped record of what the system decided, under what version of what strategy, with what inputs. Reconstructing "what did our system do, and why, between 2:32 and 2:45 PM on a given day" becomes a query against an existing structured log returning results in seconds, rather than a manual reconstruction effort. We are not claiming this would have shortened the *market-wide* five-month investigation — that reconstruction spanned many participants and venues, most of which would not have been operating Anchor. What we are claiming is narrower and, we believe, still significant: for the portion of any future investigation concerning a *specific institution's own systems*, the difference between "we can answer that from our audit chain in minutes" and "we will need several weeks to reconstruct our logs" is itself a material reduction in regulatory and reputational exposure during the highest-scrutiny period an institution can face. And the Sovereign Relay's forensic-pull mechanism means a regulator can request the audit chain for a specific window directly, with the underlying data remaining encrypted until that specific, authorized request is made — rather than the institution needing to compile and produce logs from disparate internal systems under time pressure.

5. Empirical Validation: Anchor Against Real Codebases

A governance tool's claims are only as credible as its willingness to be run against code its author did not write — including its own. We validated Anchor against four widely-used open-source codebases and against Anchor's own source code, using the publicly available `anchor-audit` package (`pip install anchor-audit`) with no modifications. The results below were generated under the 43-rule constitution distributed with v4.3.5 — the version described in the original technical preprint. The current release (v5.0.5) extends the constitution to 118 active rules across the same nine domains plus the six regulator mappings introduced since; a refreshed validation run under the expanded rule set is in progress and, given the larger rule count, may surface additional findings beyond those reported here, particularly in regulator-specific categories that did not exist in the 43-rule constitution. Every result below is reproducible by running `anchor init --all && anchor check .` against the corresponding repository, with configuration matching that distributed in the project's `case_studies/` directory. **Appendix A** reproduces the complete, file-level findings for every result summarized in this section.

Codebase	Files Scanned	Total Files in Repo	Findings (Blocker)	Findings (Warning)	Primary Violation Class
Anchor (self-audit)	49	49	0	0 (5 suppressed, individually attributed)	—
HuggingFace Hub	164	218	12	0	Shell injection (6); unvalidated LLM output (5); raw network access (1)
Django	898	3,730	7	0	Shell injection in management commands and DB backends
FINOS Architecture-as-Code	684	1,549	11	0	Unencrypted data-store writes (data poisoning class)
OpenSpiel	602	1,621	4	0	Shell injection via `subprocess.P`

Codebase	Files Scanned	Total Files in Repo	Findings (Blocker)	Findings (Warning)	Primary Violation Class
					`open` in game-engine integration

A few results are worth examining individually, because they illustrate both the system's capabilities and its current limitations honestly.

The self-audit. Anchor's own codebase — 49 files — produces zero blocking violations under its own 118-rule constitution, with five findings explicitly suppressed via inline annotations, each suppression individually attributed to its author with a stated justification, itself logged. We consider this the most direct evidence available that the governance model is operationally consistent: the tool that enforces "every exception must be individually accountable" enforces that rule against its own exceptions.

HuggingFace Hub. The twelve findings span three distinct risk categories rather than a single repeated pattern, demonstrating that the rule set generalizes across violation types within one codebase. The five "unvalidated LLM output" findings — calls to a model API without subsequent schema validation of the response — are a category of risk specific to AI-adjacent code that a general-purpose security scanner would not flag, since the code is not unsafe in the traditional sense (it does not, for example, contain a SQL injection or buffer overflow); it is unsafe in the sense that nothing constrains what the system does with whatever the model returns.

Django. This result has a methodological note worth including precisely because it counts against an inflated claim. An earlier iteration of the shell-injection detection pattern produced 24 findings against Django, including a number of false positives arising from Django's own framework-internal use of subprocess calls in ways that do not constitute a governance risk in typical deployments. After tightening the detection pattern to exclude this class of framework-internal usage, seven genuine findings remain, with zero warnings — the absence of residual warnings indicating that the tightened pattern is not simply suppressing findings into a lower severity tier, but correctly excluding them. We report this not to claim the detection logic is perfect, but because a governance tool's willingness to report its own false-positive corrections is itself a form of the auditability this paper argues for.

OpenSpiel. All four findings are `subprocess.Popen` calls within game-engine integration code — calls that are architecturally necessary for OpenSpiel's purpose as a research library, and not bugs. We classify these as correct findings rather than false positives for a specific reason: in the context this paper addresses — AI-adjacent code in a financial institution — a code path where a model's output can influence subprocess arguments is a governance-relevant finding *regardless of whether the current codebase's intent is benign*, because the constitutional question is "can this pattern be reached by agent-controlled input," not "did the original authors intend it to be."

Where such a pattern is architecturally required, as it is here, the appropriate response is not to remove the code but to apply the Diamond Cage's behavioral verification at that boundary — which is the disposition Anchor's own report assigns to this finding.

FINOS Architecture-as-Code. All eleven findings are a single underlying pattern — unencrypted writes to a MongoDB store — appearing across six files, and reported under three reference codes simultaneously (a FINOS framework risk, an OWASP LLM Top 10 category, and a security-domain rule), illustrating the regulatory polyglottism mechanism described in Section 3.2 on a real codebase rather than a constructed example: one engineering pattern, one detection, three regulatory references, six locations.

6. Regulatory Compliance Mapping

This section maps Anchor's governance mechanisms to specific provisions of five frameworks relevant to financial institutions: the EU AI Act, the Reserve Bank of India's FREE-AI recommendations, the U.S. CFPB's Regulation B, the SEC's 2026 Examination Priorities, and the NIST AI Risk Management Framework. We present this mapping in two parts, deliberately: first, a verification-status table stating plainly what has and has not been independently validated; second, a mechanism table showing the specific correspondence between regulatory provisions and Anchor outputs.

We make this separation because we believe a mapping table that does not state its own limitations invites exactly the scrutiny it cannot survive. The mappings below were produced by the system's author reading the cited regulatory text and mapping it to the corresponding engine mechanism; the `anchor_mechanism` field for each rule describes a real, executable code path, not an aspirational one. What has *not* occurred is independent legal review, regulator feedback, or formal conformity assessment under any of these frameworks.

6.1 Verification Status

Framework	Provision-Level Mapping	Source Document Verified	External Legal Review	Layer 1 (Static)	Layer 2 (Runtime/DAC)
EU AI Act (2024/1689)	Yes — 10 articles (9, 10, 11, 12, 13, 14, 15, 16, 26, 99)	Yes	Not yet conducted	Active	Active (v5.0.5)
RBI FREE-AI Report (Aug. 2025)	Yes — 12 of 26 recommendations	Yes	Not yet conducted	Active	Active (v5.0.5)
SEC 2026 Examination Priorities	Yes — 5 named priorities	Yes	Not yet conducted	Active	Active (v5.0.5)

Framework	Provision-Level Mapping	Source Document Verified	External Legal Review	Layer 1 (Static)	Layer 2 (Runtime/DAC)
CFPB Regulation B + 2024 Guidance	Partial	Yes	Not yet conducted	Active	Active (v5.0.5)
NIST AI RMF 1.0	Framework-level (Govern/Map/Measure/Manage)	Yes	Not yet conducted	Active	Active (v5.0.5)

We report “Layer 2: Active” rather than “in development” as of release v5.0.5, in which the runtime audit chain and jurisdiction-specific dialect methods (described in Section 3.5) were verified against each of the five frameworks’ logging and explainability provisions and the corresponding rule files updated accordingly. We state this version dependency explicitly because the architecture described in this paper has moved quickly — an institution evaluating Anchor should confirm the version in use against the rule files’ stated status, which `GOVERNANCE.lock` makes independently checkable rather than a matter of trusting documentation.

6.2 Mechanism Mapping

Framework Provision	Requirement (paraphrased)	Anchor Mechanism
EU AI Act Art. 9	Risk management system for high-risk AI	Signed `constitution.anchor` rule set, version-controlled and sealed
EU AI Act Art. 12	Automatic, tamper-evident logging	Decision Audit Chain (DAC); `eu_article12_record()`
EU AI Act Art. 13	Transparency / explainability to users	ETH-002 No-Prose Rule; structured reason codes
RBI FREE-AI Rec. 7	Per-decision audit trail, reportable to CIMS	DAC; `cims_payload()`
RBI FREE-AI Rec. 14	Explainable credit decisions	ETH-002 No-Prose Rule + `FeatureAttribution` field
RBI FREE-AI Rec. 19	Mandatory algorithmic fairness audits (credit/lending)	ETH-001 Aho-Corasick prohibited-proxy taxonomy
CFPB Reg. B	Adverse action notice for AI-assisted credit decisions	`adverse_action_reasons()`
CFPB Reg. B	Reason specificity (no generic denial reasons)	`ReasonCode` field, structurally enforced
SEC 2026 Priority 1	AI governance examination readiness	Full domain coverage; `governance_audit.md` report

Framework Provision	Requirement (paraphrased)	Anchor Mechanism
SEC 2026 — "AI washing"	Marketing claims must match system capability	`GOVERNANCE.lock` — sealed rule set as evidence of actual, not claimed, controls
NIST AI RMF — Govern 1.1	Documented AI governance policies	Signed `constitution.anchor`
NIST AI RMF — Manage 4.1	Continuous monitoring of AI system behavior	AnchorRuntime continuous evaluation; DAC
FCA 2024 — Oversight records	Human oversight documentation	DAC query API; notification-receipt hashing (Section 3.5)
FCA 2024 — Model traceability	Model version traceability	`provider` and `git_commit` fields in every AuditEntry

A property worth making explicit, because it is the basis for the term *regulatory polyglottism*: the right-hand column above is shorter than the left-hand column would suggest, because the same underlying mechanism — a structured `AuditEntry` with a `findings_hash`, a `ReasonCode`, and a `FeatureAttribution` field, written to a hash-chained log — satisfies the logging requirement of the EU AI Act, the audit-trail requirement of the RBI, and the oversight-record requirement of the FCA, *simultaneously, from the same write operation*. This is not because Anchor performs translation between frameworks as a separate step. It is because, read carefully, "produce a tamper-evident, per-decision record of what an AI system decided and why" is what each of these frameworks is independently asking for, in different regulatory language. The polymorphic output methods (`eu_article12_record()`, `cims_payload()`, `adverse_action_reasons()`) are presentation layers over one underlying record, not five separate compliance subsystems.

We want to close this section with the framing we believe is the only honest one available to an infrastructure provider, as opposed to a law firm: **Anchor does not certify compliance, and no software can.** What Anchor provides is the technical substrate — the logs, the structure, the cryptographic guarantees — that make a conformity assessment *possible* to perform and *fast* to perform, where today, for most agentic AI systems, the necessary records do not exist in any form a conformity assessment could examine. The gap Anchor closes is the gap between "we have a policy document describing what our AI systems should do" and "we have a cryptographically verifiable record of what they did, mapped to the policy that governed them." Closing that gap is a precondition for compliance, not a substitute for it.

6.3 Scope of Deterministic Guarantees

Throughout this paper we have used the word "deterministic" to describe Anchor's enforcement, and we want to be precise about exactly what that word does and does not cover — because the imprecise version of this claim ("Anchor prevents failures") is both too strong and, on inspection, not the claim we are actually making.

The precise claim is: **for the class of violations represented in an organization's sealed constitution, and detectable by the corresponding mechanism in `mitigation.anchor`,**

Anchor's enforcement is deterministic — the rule fires if and only if the pattern is present, and (for BLOCKER-severity findings in CI) the commit is blocked if and only if the rule fires. This class includes, among others: prohibited imports and function calls (Section 4.1's Power Peg pattern), hardcoded credential patterns, unsandboxed subprocess invocations, and — at runtime — the structural absence of a required reason code (Section 3.6's No-Prose Rule) and matches against a maintained prohibited-proxy taxonomy.

This guarantee has four explicit boundaries, and we list them because a technical reviewer will ask about each:

Suppression is auditable, not prevented. A developer can annotate a finding as an authorized exception (as in Anchor's own self-audit, Section 5). This is logged and individually attributed — visible to anyone reviewing the governance log — but it does not block the commit. The guarantee is that the exception is *recorded*, not that it cannot occur.

Detection is bounded by the mitigation catalog. A violation pattern not yet encoded in `mitigation.anchor` — for example, an obfuscated form of a prohibited import constructed via string concatenation at runtime — will not be detected by the corresponding static rule. The constitution defines what is *prohibited*; the mitigation catalog defines what Anchor currently knows how to *recognize*. These are not the same set, and the gap between them is closed incrementally, the same way a vulnerability scanner's signature database is updated.

Static analysis governs committed code, not all possible runtime behavior. A pattern that exists only as a consequence of runtime control flow not visible in the AST — for instance, a module loaded dynamically from a path computed at runtime — is outside Layer 1's guarantee and falls to Layer 2's runtime interception, which has its own, narrower pattern set.

The guarantee is conditional on the enforcement points themselves being intact — the Git pre-commit hook is installed and not bypassed by a force-push, and `GOVERNANCE.lock` verification has not been skipped. `GOVERNANCE.lock` makes *tampering with the rules* detectable; it does not make *bypassing the hook entirely* impossible in every CI configuration, which is why we recommend the hook be enforced server-side (e.g., as a required status check) rather than relying on the client-side hook alone.

Within these boundaries, the guarantee is real and the word "deterministic" is doing real work: there is no confidence threshold, no sampling, and no model in the loop for the enforcement decision itself. Outside these boundaries — novel attack patterns, semantic content of a model's output, or anything not yet represented in the constitution — Anchor makes no claim, deterministic or otherwise. **The correct one-sentence summary, and the one we ask readers to hold onto, is: Anchor deterministically enforces the violations its constitution defines; it does not claim to anticipate violations its constitution does not yet define.** Appendix C provides a field-level mapping showing exactly which `AuditEntry` fields and constitutional

mechanisms correspond to which regulatory provisions, for readers who want to verify this claim mechanism-by-mechanism rather than take it on summary.

7. Comparison to Existing Approaches

A natural first reaction to Anchor is to place it in the same category as existing "AI governance" products and ask how it compares feature-for-feature. We think this framing is the wrong one, and that the more useful question is *which question* a given tool is designed to answer — because several well-established categories of tooling answer different questions than the one this paper has argued is currently unaddressed. The five systems below were selected because each represents a major, widely-deployed category of AI governance tooling in enterprise environments — data governance, model monitoring, MLOps, output validation, and content safety, respectively — not because they are Anchor's closest competitors by feature count.

Tool / Category	Core Question It Answers	What It Governs
Microsoft Purview	"What happened to this data, and who has access to it?"	Data classification, lineage, compliance reporting, insider risk
Fiddler AI and similar model-monitoring platforms	"Is this model's output quality degrading, and is it biased?"	Model drift, explainability dashboards, aggregate fairness metrics
Weights & Biases and similar MLOps platforms	"What happened during training and evaluation?"	Experiment tracking, model lifecycle, training provenance
Guardrails AI and similar output-validation frameworks	"Does this specific output conform to a schema or policy?"	Prompt and output validation at generation time
Llama Guard and similar content classifiers	"Is this content safe to surface to a user?"	Content moderation, safety classification
Anchor	"Was this action authorized to occur, under which policy, and can we prove it did — cryptographically?"	Execution authority and decision provenance

Each of the five established categories above asks a question that precedes or accompanies a model's output. Purview asks about data the model touches. Fiddler and similar tools ask about the model's behavior over time. W&B asks about how the model came to exist in its current form. Guardrails and Llama Guard ask about the specific output the model just produced — whether it conforms to a schema, or whether its content is appropriate.

Anchor asks a question that comes *after* all of these have been satisfied: given that an output exists, and given that it may even be perfectly accurate, well-formed, and benign in content — **is the system permitted to act on it?** Consider a hypothetical in which a model, with perfect accuracy and entirely appropriate content, produces the output "deploy this updated trading model to production." A content classifier finds nothing wrong with this text. A schema validator

finds it well-formed. A model-monitoring dashboard shows the model performing within expected parameters. None of these tools has a basis to say no — because none of them is asking whether *deployment authority* has been granted for this action, by this system, under current policy. That is the question Anchor is built to answer, and it is a question that is, in a precise sense, orthogonal to model quality: a system can have arbitrarily good model quality and still take an action it was never authorized to take, and a system with mediocre model quality can be governed perfectly well at the execution layer while its quality is improved separately.

This is also why we do not believe Anchor displaces the tools above in a typical deployment. A financial institution operating agentic AI at scale will reasonably run something like Purview for data governance, a model-monitoring platform for drift and bias telemetry, an MLOps platform for training provenance, and a content classifier for customer-facing outputs — alongside Anchor for execution governance. These are complementary layers of a stack, not competing implementations of the same layer. The distinction we are drawing is closer to the relationship between a network firewall and an application's business logic: a firewall does not replace input validation, and input validation does not replace a firewall, because they operate at different points in the request's lifecycle and answer different questions about it.

8. Implementation Pathway for Financial Institutions

This section describes, concretely, what adopting Anchor involves — for an engineering team evaluating integration effort, and for a risk or compliance function evaluating what a pilot engagement would produce.

8.1 Integration Effort

For static (Layer 1) governance, integration into an existing Python codebase with a clean git repository is, end to end:

```
pip install anchor-audit
anchor init --all      # installs domains, frameworks, regulator files; ~2 minutes
anchor check .          # runs the first scan immediately
```

This installs all nine mandatory domains, the three opt-in frameworks, and the six opt-in regulator files, and produces a findings report with no further configuration. An institution can scope this to only the regulators relevant to its jurisdiction — for example, `anchor init --regulators sec,cfpb --frameworks nist` for a US-focused lending platform — without affecting the mandatory domain coverage.

For runtime (Layer 2) governance, integration is a single import at the application's entry point:

```
import anchor.runtime  # auto-activates; patches registered AI SDKs
```

This single line activates interception for calls made through registered SDK integrations (including OpenAI, Anthropic, and LangChain-based pipelines), without requiring changes to individual call sites. Where an institution wants the stronger guarantees of Structured Mode for

specific decision types — credit, lending, trading — the relevant functions are wrapped with the `@anchor.enforce` decorator, as shown in Section 3.5.

8.2 Language Support

Language	Status	Relevance to Financial Institutions
Python	Full — AST queries for imports, dangerous calls, inheritance	Quantitative research, ML pipelines, most AI/agent orchestration code
TypeScript	Full	API layers, internal tooling, Node-based services
Java	Full	Core banking systems, enterprise middleware
Go	Implemented	Trading infrastructure, microservices
Rust	Implemented	Performance-critical systems
JavaScript	Partial — covered by the TypeScript grammar for `.ts`/`.tsx`; native `.js` extension mapping not yet registered	Front-end and legacy Node services
C++	Not yet supported at the source level	For C++ trading systems, governance is applied at the AI API boundary — i.e., at the Python or service layer where the system calls a model, rather than within the C++ source itself

We list this table without rounding up: Anchor's language coverage is strong for the layer where AI orchestration logic typically lives in a financial institution's stack (Python, increasingly TypeScript and Java), and is not yet complete for lower-level systems languages. For institutions with AI-adjacent logic embedded directly in C++ trading systems, the practical integration point today is the boundary at which that system calls into a model — which, in the architectures we are aware of, is virtually always mediated through a higher-level language.

8.3 Deployment Models

For institutions with data residency requirements or policies against transmitting source code or customer data externally, Anchor supports an air-gapped initialization mode:

```
anchor init --no-sign
```

This bundles all governance files at install time — they do not require a network fetch — and skips the optional remote integrity check against `GOVERNANCE.lock`. The Decision Audit Chain writes exclusively to local disk (`.anchor/runtime_chain.jsonl`) regardless of configuration; no

prompt, response, or audit data is transmitted over a network in the default audit-write path. The only network-dependent step in the default flow is the optional remote verification of `GOVERNANCE.lock`, which exists to let an institution confirm its governance files match the authoritative, sealed versions — and which can be bypassed entirely for fully isolated environments.

8.4 Pilot Structure

We describe a pilot in terms of what it produces, not in terms of a sales process. A governance assessment — running Anchor against an institution's existing AI-adjacent codebase, mapping findings to the relevant regulatory framework for that institution's jurisdiction, and producing a findings brief — can be completed in three to five business days, since it requires only read access to the codebase and does not require any change to production systems. Where an institution proceeds to integration, a typical timeline is two to four weeks: installing `anchor check` as a CI/CD gate (the pre-commit hook is installed automatically by `anchor init`), wiring `import anchor.runtime` at the AI pipeline's entry point, configuring `policy.anchor` for the institution's specific risk appetite within the constitutional floor, and establishing the local audit chain.

At the end of this process, an institution has: a CI/CD gate that blocks deployments containing governance violations against its chosen framework set; a running, hash-chained audit log of AI decisions in scope; a `governance_audit.md` report in a form suitable for an internal audit function or external examiner; and a `GOVERNANCE.lock`-verified statement of which governance rules were in effect, independently checkable rather than dependent on internal documentation.

8.5 Licensing

The Anchor engine — the static analysis tool, the rule files, the CLI, and the runtime library — is open source under the Apache 2.0 license, available via PyPI (`pip install anchor-audit`) and on GitHub. An organizational management layer — covering multi-project dashboards, role-based team access, and cryptographic key provisioning for larger deployments — is offered as a hosted service. This follows a model familiar from other developer-infrastructure categories: the engine that does the governance work is open, auditable, and forkable; the operational convenience layer for managing it across an organization is a separate, optional service.

9. Conclusion: The Cost of Waiting

The three failures examined in Section 4 share a structure that is easy to miss when they are considered individually, and difficult to miss when they are considered together. In each case, the technical capability that caused the failure had existed, unremarkably, for some time before it caused harm. Knight Capital's Power Peg code was old, dormant, and unremarkable — until a deployment inconsistency gave it an opportunity to run. Zillow's pricing model had been operating, by the company's own account, successfully for years — until market conditions shifted faster than any process existed to notice. The algorithmic trading systems active on May

6, 2010 were not, individually, doing anything novel — until their interactions produced an outcome that took regulators months to even *describe*, let alone address.

The infrastructure that would have changed each outcome — a static check for deprecated code paths and configuration drift; a continuously queryable record of a model's stated reasoning; a structured, per-decision audit trail created at the moment of decision rather than reconstructed afterward — is not exotic. Every individual mechanism described in this paper is built from established techniques: AST parsing, cryptographic hashing, append-only logs, schema validation. What has been missing is not the techniques. It is a system that applies them, by default, to the layer where agentic AI systems take actions with financial consequences — and a constitution, signed and sealed, that says in advance what those systems are and are not permitted to do.

We described the present moment, in Section 1, as a warning window rather than a crisis window — and we want to close by being precise about how narrow that window is. The EU AI Act's enforcement of high-risk AI system requirements begins within the current quarter. The CFPB has already demonstrated, with a nine-figure enforcement action, that "the model is probably fine in aggregate" is not a defense against "this specific decision was not explainable." The SEC has named AI governance an examination priority for the year now underway. None of this is speculative or forward-looking; all of it is either already in effect or scheduled within months.

For an institution reading this paper, we believe the most useful next step is the smallest one that produces real evidence: running a governance assessment — three to five days, read-only access, no production changes — against a codebase that already has agentic AI components, and seeing what the findings report actually contains. We would particularly welcome this from two kinds of organizations. The first is a lending or credit platform currently operating under, or anticipating, CFPB or equivalent regulatory examination of its AI-assisted decisioning — for whom the gap between "we have a policy" and "we have a record" is the most immediately consequential. The second is a financial supervisor evaluating what a technical baseline for AI governance in the institutions it oversees should look like — for whom Anchor's fully open-source, independently-auditable design means the question "what does this tool actually do" can be answered by reading the code, not by trusting a vendor's description of it.

We are not asking anyone to take Anchor's claims on faith. Every number in this paper is reproducible by running an open-source package against a public codebase. That reproducibility is, itself, the argument.

10. About AnimusLab and Availability

AnimusLab is an independent research organization, founded in November 2025, developing open-source governance infrastructure for autonomous AI systems in regulated financial markets. Anchor is AnimusLab's primary technical output: the implementation of the thesis, argued throughout this paper, that governance for agentic AI must be structurally enforced and

cryptographically verifiable — not declarative, and not dependent on the probabilistic behavior of the systems it governs.

AnimusLab has contributed to FINOS Labs' 2026 DTCC Hackathon, with a merged contribution implementing automated mapping from architectural components to governance mitigations within a FINOS reference architecture — extending the "pre-threat modeling" concept to generate governance mitigation checklists directly from architecture-as-code definitions, a direct application of this paper's underlying thesis to FINOS's own open-source tooling.

Anchor v5.0.5 is available as open-source software under the Apache 2.0 license:

- **Engine and rule files:** github.com/AnimusLab/anchor
- **Package:** `pip install anchor-audit`
- **Interactive policy evaluation:** anchorgovernance.tech
- **Research and case studies:** animuslab.dev

AnimusLab's institutional goal is not to position itself as a substitute for financial regulators, nor to displace the tooling categories described in Section 7. It is to provide the technical substrate — open, inspectable, and reproducible — that makes regulatory oversight of agentic AI systems in financial markets a tractable engineering problem rather than an open one. We welcome correspondence from institutions, researchers, and regulators at the contact address on the title page.

References

[1] European Parliament and Council. *Regulation (EU) 2024/1689 (Artificial Intelligence Act)*. Official Journal of the European Union, 2024.

[2] Reserve Bank of India. *Framework for Responsible and Ethical Enablement of AI (FREE-AI)*. Technical report, RBI, August 2025.

[3] Consumer Financial Protection Bureau. *CFPB orders Apple and Goldman Sachs to pay over $89 million*. Enforcement press release, October 2024.

[4] Consumer Financial Protection Bureau. *Circular 2024-03: Adverse action notification requirements*. CFPB Circular, 2024.

[5] U.S. Securities and Exchange Commission. *2026 Examination Priorities*. Technical report, SEC Division of Examinations, January 2026.

[6] National Institute of Standards and Technology. *AI Risk Management Framework (AI RMF 1.0)*. NIST AI 100-1, January 2023.

[7] Financial Conduct Authority. *Artificial intelligence: How FCA principles apply*. Discussion Paper DP 5/4, FCA, 2024.

[8] Securities and Exchange Board of India. *Circular on algorithmic trading governance*. SEBI/HO/MRD circular, 2024.

[9] U.S. Securities and Exchange Commission. *In the Matter of Knight Capital Americas LLC*. Securities Exchange Act of 1934 Release No. 70694, Administrative Proceeding File No. 3-15570, October 16, 2013, pp. 1–18.

[10] U.S. Securities and Exchange Commission and U.S. Commodity Futures Trading Commission. *Findings Regarding the Market Events of May 6, 2010*. Report of the Staffs of the CFTC and SEC to the Joint Advisory Committee on Emerging Regulatory Issues, September 30, 2010.

[11] U.S. Department of Justice. *U.K. Trader Charged with Illegally Manipulating Stock Market*. Press release, April 2015.

[12] Zillow Group, Inc. *Form 10-Q for the Quarterly Period Ended September 30, 2021*, SEC EDGAR filing, November 2, 2021 (Management's Discussion and Analysis); and *Q3 2021 Shareholder Letter*, November 2, 2021.

[13] L. Kao. *Constant-size cryptographic evidence structures for regulated AI*. arXiv:2511.17118, 2025.

[14] B. Schneier and J. Kelsey. *Secure audit logs to support computer forensics*. ACM TISSEC, 2(2):159–176, 1999.

[15] S. Crosby and D. Wallach. *Efficient data structures for tamper-evident logging*. In USENIX Security, 2009.

[16] I. D. Raji et al. *Closing the AI accountability gap*. In ACM FAccT, 2020.

[17] J. Mökander et al. *Auditing large language models: A three-layered approach*. AI and Ethics, 2024.

[18] M. Brunsfeld et al. *Tree-sitter*. tree-sitter.github.io, 2024.

[19] WasmEdge Runtime. wasmedge.org, 2024.

[20] FINOS Foundation. *Architecture as Code*. github.com/finos/architecture-as-code, 2024.

[21] HuggingFace. *huggingface_hub Python Library*. github.com/huggingface/huggingface_hub, 2024.

[22] Django Software Foundation. *Django web framework*. djangoproject.com, 2025.

[23] M. Lanctot et al. *OpenSpiel: A framework for reinforcement learning in games*. arXiv:1908.09453, 2019.

[24] CNCF. *Open Policy Agent*. openpolicyagent.org, 2024.

[25] NVIDIA. *NeMo Guardrails*. github.com/NVIDIA/NeMo-Guardrails, 2024.

Appendix A: Raw Validation Results

This appendix reproduces the complete, file-level output of the five audit runs summarized in Section 5, exactly as written by `anchor check` to `governance_audit.md` in each case (paths shown as reported on the scanning machine; severity and rule-ID columns are reproduced verbatim). These runs were performed under the 43-rule v4.3.5 constitution, as noted in Section 5. Each is independently reproducible by running `anchor init --all && anchor check .` against the corresponding public repository with the configuration distributed in `case_studies/`.

A.1 Anchor (self-audit)

Source: `Anchor` repository · **Status:** PASSED · **Files scanned:** 49 · **Blockers/Errors:** 0 · **Suppressed:** 5

Rule ID	File : Line	Authorized By
`ANC-018`	`anchor/adapters/python.py:22`	Tanishq
`ANC-018`	`anchor/core/engine.py:54`	Tanishq
`ANC-018`	`anchor/core/engine.py:543`	Tanishq
`ANC-018`	`anchor/core/sandbox.py:276`	Tanishq
`ANC-018`	`scripts/inspect_python.py:31`	Not committed yet

All five suppressions are `ANC-018` (an internal-pattern exception), each individually attributed and git-blamed. No BLOCKER, ERROR, or WARNING-severity findings were reported.

A.2 HuggingFace Hub

Source: `huggingface_hub` repository · **Status:** FAILED · **Files scanned:** 164 of 218 · **Blockers/Errors:** 12

Rule ID(s)	Severity	File : Line	Finding
`FINOS-014, OWASP-002, RBI-018, SEC-007`	BLOCKER	`src/huggingface_hub/cli/lfs.py:54`	Native subprocess execution detected
`FINOS-014, OWASP-002, RBI-018, SEC-007`	BLOCKER	`src/huggingface_hub/cli/lfs.py:59`	Native subprocess execution detected
`FINOS-014, OWASP-002, RBI-018, SEC-007`	BLOCKER	`src/huggingface_hub/utils/_subprocess.py:85`	Native subprocess execution detected
`FINOS-014, OWASP-002, RBI-018, SEC-007`	BLOCKER	`src/huggingface_hub/utils/_subprocess.py:132`	Native subprocess execution detected
`FINOS-014, OWASP-002, RBI-018, SEC-007`	BLOCKER	`utils/helpers.py:46`	Native subprocess execution detected

Rule ID(s)	Severity	File : Line	Finding
FINOS-014, OWASP-002, RBI-018, SEC-007	BLOCKER	utils/helpers.py:47	Native subprocess execution detected
ALN-001, FINOS-008, OWASP-009	ERROR	src/huggingface_hub/inference/_client.py:700	LLM API call without output validation
ALN-001, FINOS-008, OWASP-009	ERROR	src/huggingface_hub/inference/_client.py:754	LLM API call without output validation
ALN-001, FINOS-008, OWASP-009	ERROR	src/huggingface_hub/inference/_generated/_async_client.py:723	LLM API call without output validation
ALN-001, FINOS-008, OWASP-009	ERROR	src/huggingface_hub/inference/_generated/_async_client.py:778	LLM API call without output validation
ALN-001, FINOS-008, OWASP-009	ERROR	src/huggingface_hub/inference/_mcp/mcp_client.py:273	LLM API call without output validation
FCA-004, FINOS-013, SEC-006	ERROR	src/huggingface_hub/inference/_providers/openai.py:9	Direct call to public LLM API without governed proxy

All twelve findings are reproduced verbatim from the `anchor check .` run; each "Native subprocess execution" finding maps to four reference codes simultaneously (Section 3.2's rule-consolidation mechanism), and each "LLM API call" finding identifies a call to `chat.completions.create()` or equivalent with no subsequent schema validation of the response.

A.3 Django

Source: `django` repository · **Status:** FAILED · **Files scanned:** 898 of 3,730 · **Blockers/Errors:** 7

Rule ID(s)	Severity	File : Line	Finding
FINOS-014, OWASP-002, RBI-018, SEC-007	BLOCKER	django/core/management/utils.py:175	Native subprocess execution detected
FINOS-014, OWASP-002, RBI-018, SEC-007	BLOCKER	django/db/backends/base/client.py:31	Native subprocess execution detected
FINOS-014, OWASP-002, RBI-018, SEC-007	BLOCKER	django/db/backends/mysql/creation.py:78	Native subprocess execution detected

Rule ID(s)	Severity	File : Line	Finding
FINOS-014, OWASP-002, RBI-018, SEC-007	BLOCKER	django/db/backends/mysql/creation.py:81	Native subprocess execution detected
FINOS-014, OWASP-002, RBI-018, SEC-007	BLOCKER	django/utils/autoreload.py:273	Native subprocess execution detected
FINOS-014, OWASP-002, RBI-018, SEC-007	BLOCKER	django/utils/version.py:91	Native subprocess execution detected
FINOS-014, OWASP-002, RBI-018, SEC-007	BLOCKER	scripts/manage_translations.py:48	Native subprocess execution detected

As discussed in Section 5, an earlier iteration of this detection pattern produced 24 findings against Django, including false positives in framework-internal subprocess usage; the seven findings above are the result after tightening the pattern, with zero residual warnings.

A.4 FINOS Architecture-as-Code

Source: `architecture-as-code` repository · **Status:** FAILED · **Files scanned:** 684 of 1,549 · **Blockers/Errors (summary count):** 11 · **Findings listed:** 10

Rule ID(s)	Severity	File : Line	Finding
FINOS-002, OWASP-003, SEC-002	BLOCKER	calm-hub/.../store/mongo/MongoAdrStore.java:86	Vector store write without encryption
FINOS-002, OWASP-003, SEC-002	BLOCKER	calm-hub/.../store/mongo/MongoArchitectureStore.java:80	Vector store write without encryption
FINOS-002, OWASP-003, SEC-002	BLOCKER	calm-hub/.../store/mongo/MongoArchitectureStore.java:191	Vector store write without encryption
FINOS-002, OWASP-003, SEC-002	BLOCKER	calm-hub/.../store/mongo/MongoCounterStore.java:49	Vector store write without encryption
FINOS-002, OWASP-003, SEC-002	BLOCKER	calm-hub/.../store/mongo/MongoFlowStore.java:77	Vector store write without encryption
FINOS-002, OWASP-003, SEC-002	BLOCKER	calm-hub/.../store/mongo/MongoFlowStore.java:185	Vector store write without encryption
FINOS-002, OWASP-003, SEC-002	BLOCKER	calm-hub/.../store/mongo/MongoPatternStore.java:77	Vector store write without encryption
FINOS-002, OWASP-003, SEC-002	BLOCKER	calm-hub/.../store/mongo/	Vector store write without encryption

Rule ID(s)	Severity	File : Line	Finding
		MongoPatternStore.java:185	
FINOS-002, OWASP-003, SEC-002	BLOCKER	calm-hub/.../store/mongo/MongoStandardStore.java:83	Vector store write without encryption
FINOS-002, OWASP-003, SEC-002	BLOCKER	calm-hub/.../store/mongo/MongoStandardStore.java:177	Vector store write without encryption

Note on the count discrepancy: the audit report's summary header states 11 blockers, while the detailed findings table — reproduced in full above — lists 10 distinct file:line entries. This is a one-row discrepancy in the v4.3.5 report format itself, present in the raw output and reproduced here rather than corrected, consistent with this paper's position that reported numbers should be independently checkable against raw tool output. We flag it rather than silently resolve it in either direction.

A.5 OpenSpiel

Source: `open_spiel` repository · **Status:** FAILED · **Files scanned:** 602 of 1,621 · **Blockers/Errors:** 4

Rule ID(s)	Severity	File : Line	Finding
FINOS-014, OWASP-002, RBI-018, SEC-007	BLOCKER	open_spiel/python/algorithms/matrix_nash.py:82	Native subprocess execution detected
FINOS-014, OWASP-002, RBI-018, SEC-007	BLOCKER	open_spiel/python/bots/gtp.py:49	Native subprocess execution detected
FINOS-014, OWASP-002, RBI-018, SEC-007	BLOCKER	open_spiel/python/examples/bridge_uncontested_bidding_bluechip.py:99	Native subprocess execution detected
FINOS-014, OWASP-002, RBI-018, SEC-007	BLOCKER	open_spiel/python/examples/bridge_wb5.py:158	Native subprocess execution detected

All four findings are `subprocess.Popen` invocations in game-engine integration code, discussed in Section 5 as correct findings under an agentic-AI threat model despite being architecturally intentional in OpenSpiel's original context.

Appendix B: Performance Benchmark Methodology

The benchmark results reported in Section 3.8 were produced under the following conditions:

Parameter	Value
Python version	3.13.12
Active governance rules	118

Parameter	Value
Iterations per suite	10,000
Warmup iterations (discarded)	200
Measurement	Wall-clock time per operation, via repeated in-process invocation
Reported statistics	Mean, median, P95, P99, throughput (operations/second)
Operations measured	SHA-256 hash generation; HMAC-SHA256 chain verification; full policy evaluation (118-rule scan of a single file); end-to-end receipt generation (policy scan + chain hash computation + synchronous JSONL append)

Hardware and operating system specification for these runs is being finalized for the camera-ready version of this paper and will be added here; the relative ordering of operations (hashing $\ll$ chain verification $\ll$ policy evaluation $\ll$ receipt generation, with receipt generation dominated by synchronous disk I/O) is expected to hold across common development and CI hardware, though absolute figures will vary. We report this gap explicitly rather than omit it, consistent with this paper's position that methodology should be checkable rather than asserted.

Appendix C: Regulatory Mapping Matrix — AuditEntry Field Level

Section 6 maps Anchor's *mechanisms* to regulatory *provisions*. This appendix maps the same correspondence one level lower — from individual fields of the `AuditEntry` record (Section 3.5) and the Structured Mode output (Section 3.6) to the specific regulatory requirement each field satisfies. This is the level of detail a compliance reviewer would need to verify the polyglottism claim against their own reading of each framework's text.

AuditEntry / Structured-Mode Field	What It Contains	Regulatory Requirement(s) Satisfied
`entry_id`	UUID, unique per decision	EU AI Act Art. 12 (uniquely identifiable log record); RBI CIMS report identifier
`timestamp`	ISO 8601 UTC	EU AI Act Art. 12 (automatically generated timestamp); FCA 2024 oversight-record timing
`provider`	Model/provider identifier and version	FCA 2024 model traceability
`jurisdiction`	Active regulatory context for this decision	Selects which dialect method (`eu_article12_record()`, `cims_payload()`, etc.) applies to this entry
`project_name`, `git_commit`	Code provenance of the	FCA 2024 model traceability;

AuditEntry / Structured-Mode Field	What It Contains	Regulatory Requirement(s) Satisfied
	calling application	NIST AI RMF Measure 2.5 (provenance)
`status` (`CLEAN` / `VIOLATION`), `is_compliant`	Outcome of constitutional evaluation for this decision	NIST AI RMF Manage 4.1 (continuous monitoring of AI system behavior)
`ReasonCode` (Structured Mode)	Machine-readable reason for the decision	CFPB Regulation B adverse-action notice; RBI FREE-AI Rec. 14 (explainable credit decisions)
`FeatureAttribution` (Structured Mode)	Specific input factors the decision was based on	CFPB Regulation B reason specificity (no generic denial reasons); EU AI Act Art. 13 (transparency to affected persons)
`violations` (ETH-001 trie matches)	Prohibited-proxy terms detected in attribution fields	RBI FREE-AI Rec. 19 (mandatory algorithmic fairness audit); ECOA / Fair Housing Act proxy-variable concerns
`findings_hash`	SHA-256 of sorted rule IDs evaluated	Evidence binding — the basis for `chain_hash` (below)
`prev_chain_hash`, `chain_hash`	Hash-chain linkage (H_n = SHA256(H_{n-1} ‖ findings_hash$_n$))	EU AI Act Art. 12 (“tamper-evident” logging); non-equivocation property (Section 3.5)
`signature` (HMAC-SHA256)	Per-entry cryptographic signature	Non-repudiation of the audit record; supports SEC examination of audit-trail integrity
`GOVERNANCE.lock` + signed `constitution.anchor`	SHA-256-sealed rule set governing this decision	NIST AI RMF Govern 1.1 (documented, version-controlled AI governance policy); SEC 2026 “AI washing” concern (the sealed rule set is evidence of *actual*, not merely *marketed*, controls)

Two limitations apply to this table, stated for the same reason the rest of this paper states its limitations: first, the mapping from field to regulatory requirement was produced by the system’s author reading each framework’s text, not by external legal counsel — the field *exists and is populated* as described (verifiable directly from the `AuditEntry` schema and the Structured Mode

output shown in Section 3.5–3.6), but whether a given regulator's examiners would accept a given field as *sufficient* evidence of compliance with a given provision is a question this table does not and cannot answer. Second, the `ReasonCode` and `FeatureAttribution` fields are populated only for decisions wrapped in Structured Mode (Section 3.5); decisions in Conversational Mode do not populate these fields, and the corresponding regulatory mappings (CFPB, RBI Rec. 14, EU Art. 13) apply only to the subset of an institution's AI decisions that are both in-scope for these regulations *and* wrapped in Structured Mode. An institution adopting Anchor is responsible for ensuring that decisions falling under these regulations are wrapped accordingly; Anchor's `enforce_raise_only` floor (Section 3.4) prevents this requirement from being silently weakened once configured, but does not by itself guarantee that every relevant decision point has been correctly identified and wrapped in the first place.