

Agent Behavioral Contracts: Formal Specification and Runtime Enforcement for Reliable Autonomous AI Agents

Varun Pratap Bhardwaj*

Senior Manager & Solution Architect, Accenture

varun.pratap.bhardwaj@gmail.com

February 25, 2026

Abstract

Traditional software relies on contracts—APIs, type systems, assertions—to specify and enforce correct behavior. AI agents, by contrast, operate on prompts and natural language instructions with no formal behavioral specification. This gap is the root cause of drift, governance failures, and frequent project failures in agentic AI deployments. We introduce *Agent Behavioral Contracts* (ABC), a formal framework that brings Design-by-Contract principles to autonomous AI agents. An ABC contract $\mathcal{C} = (\mathcal{P}, \mathcal{I}, \mathcal{G}, \mathcal{R})$ specifies Preconditions, Invariants, Governance policies, and Recovery mechanisms as first-class, runtime-enforceable components. We define (p, δ, k) -satisfaction—a probabilistic notion of contract compliance that accounts for LLM non-determinism and recovery—and prove a *Drift Bounds Theorem* showing that contracts with recovery rate $\gamma > \alpha$ (the natural drift rate) bound behavioral drift to $D^* = \alpha/\gamma$ in expectation, with Gaussian concentration in the stochastic setting. We establish sufficient conditions for safe contract composition in multi-agent chains and derive probabilistic degradation bounds. We implement ABC in AGENTASSERT, a runtime enforcement library, and evaluate on AGENTCONTRACT-BENCH, a benchmark of 200 scenarios across 7 models from 6 vendors. Results across 1,980 sessions show that contracted agents detect 5.2–6.8 soft violations per session that uncontracted baselines miss entirely ($p < 0.0001$, Cohen’s $d = 6.7\text{--}33.8$), achieve 88–100% hard constraint compliance, and bound behavioral drift to $D^* < 0.27$ across extended sessions, with 100% recovery for frontier models and 17–100% across all models, at overhead < 10 ms per action.

1 Introduction

The deployment of autonomous AI agents in production environments is accelerating at an unprecedented pace. Agents powered by large language models (LLMs) now execute multi-step workflows in financial advisory [Moslemi et al., 2026], healthcare triage, customer support [Wu et al., 2023], code generation [Yao et al., 2023], and research synthesis [Schick et al., 2023]. These systems are no longer simple question-answering interfaces: they invoke tools, access databases, make decisions with real-world consequences, and increasingly operate in multi-agent pipelines where outputs of one agent feed directly into another [Chase, 2023, Moura, 2024]. Yet despite this rapid adoption, agents operate without formal behavioral guarantees. There exists no widely adopted mechanism to specify what an agent *should* do, verify that it *is* doing it, or enforce corrective action when it deviates.

*Patent pending. Reference implementation and benchmark suite available subject to intellectual property clearance.

The Problem

Traditional software systems benefit from decades of formal specification tooling: type systems, API contracts, assertions, and interface specifications provide compile-time and runtime guarantees about program behavior [Hoare, 1969, Meyer, 1992]. AI agents, by contrast, are governed by prompts—natural language instructions that carry no formal semantics, no verifiable guarantees, and no enforcement mechanisms. This gap between the formality of traditional software contracts and the informality of agent instructions is the root cause of a class of failures unique to agentic AI: *behavioral drift*, *governance violations*, and *silent degradation*.

Behavioral drift manifests when an agent’s actions gradually diverge from its intended specification over the course of a multi-turn interaction [Rath, 2026]. An agent tasked with professional customer support may begin with appropriate responses but progressively adopt a more casual tone, hallucinate product features, or volunteer information it was instructed to withhold. A research synthesis agent may start by citing verified sources but drift toward fabricated references as the session extends. These deviations are subtle, incremental, and—critically—undetected until harm has occurred: a customer receives incorrect medical guidance, a financial agent exceeds its trading authority, or a code generation agent introduces a security vulnerability.

Several important approaches address adjacent aspects of this problem. Constitutional AI [Bai et al., 2022] embeds behavioral principles during training, producing models that are more aligned at generation time. Reinforcement learning from human feedback (RLHF) [Ouyang et al., 2022] fine-tunes models toward human preferences. Output guardrails such as NeMo Guardrails [Rebedea et al., 2023] filter or redirect agent responses that match prohibited patterns. However, none of these provides *formal runtime behavioral contracts* with mathematical guarantees. Constitutional AI operates at training time and cannot adapt to deployment-specific constraints. RLHF shapes general tendencies but cannot enforce specific invariants. Guardrails filter outputs but do not specify preconditions, do not monitor invariants over time, and do not compose across multi-agent pipelines. Recent empirical work confirms this gap: Cartagena and Teixeira [2026] demonstrate that text-level safety alignment does not transfer to tool-call safety, validating that prompt-level governance contracts are fundamentally insufficient for agents that interact with the world through tools and APIs.

The theoretical case for active enforcement is further strengthened by impossibility results. Wang et al. [2026a] prove a self-evolution trilemma: in self-evolving AI societies, continuous self-evolution, complete isolation from external correction, and safety invariance cannot coexist. This result implies that passive safety—relying on training-time alignment alone—is provably insufficient for agents that evolve their behavior over extended interactions. Active, runtime enforcement of behavioral specifications is not merely desirable; it is a theoretical necessity.

Our Contribution

We introduce *Agent Behavioral Contracts* (ABC), a formal framework that brings Design-by-Contract [Meyer, 1992] principles to autonomous AI agents. Our contributions are:

1. We define the ABC contract structure $\mathcal{C} = (\mathcal{P}, \mathcal{I}, \mathcal{G}, \mathcal{R})$, formalizing agent behavioral expectations as a tuple of Preconditions, Invariants (hard and soft), Governance policies (hard and soft), and Recovery mechanisms (Section 3).
2. We introduce (p, δ, k) -*satisfaction*, a probabilistic contract compliance framework that accounts for LLM non-determinism: contracts hold with probability at least p , deviations remain within tolerance δ , and recovery occurs within k steps (Section 3).

3. We prove a *Stochastic Drift Bound Theorem* using Lyapunov stability analysis of an Ornstein–Uhlenbeck drift model, showing that contracts with recovery rate $\gamma > \alpha$ (the natural drift rate) bound behavioral drift to $D^* = \alpha/\gamma$ in expectation, with Gaussian concentration and a closed-form contract design criterion (Section 4).
4. We present CONTRACTSPEC, a YAML-based domain-specific language for specifying agent behavioral contracts, supporting hard/soft constraint separation, expression-based predicates, and file-reference composition for multi-agent pipelines (Section 5).
5. We introduce AGENTASSERT, a runtime enforcement library implementing the ABC framework with sub-10ms per-action overhead (Section 5).
6. We prove a *Compositionality Theorem* establishing sufficient conditions (interface compatibility, assumption discharge, governance consistency, recovery independence) under which individual contract guarantees compose into end-to-end guarantees for multi-agent chains, with quantified probabilistic degradation bounds (Section 4).
7. We create AGENTCONTRACT-BENCH, a benchmark of 200 scenarios spanning 7 domains and 6 stress profiles, designed to evaluate contract enforcement across diverse agent deployment contexts (Section 6).
8. We evaluate ABC across 1,980 sessions on 7 models from 6 vendors, demonstrating that contracted agents detect 5.2–6.8 soft violations per session invisible to uncontracted baselines ($p < 0.0001$), bound drift to $D^* < 0.27$ with 17–100% recovery success, and achieve reliability $\Theta > 0.90$ across all models (Section 7).

Paper Structure

The remainder of this paper is organized as follows. Section 2 surveys related work in Design-by-Contract, contract theory, runtime verification, and AI agent safety. Section 3 presents the formal ABC framework, including contract structure, (p, δ, k) -satisfaction, the behavioral drift score, and operational metrics. Section 4 proves drift bounds via Lyapunov analysis, establishes the compositionality theorem, and analyzes runtime complexity. Section 5 describes the CONTRACTSPEC DSL and the AGENTASSERT runtime enforcement library. Section 6 introduces AGENTCONTRACT-BENCH. Section 7 reports experimental results. Section 8 discusses implications, limitations, and future directions. Section 9 concludes.

2 Background and Related Work

The ABC framework draws on and extends several established research traditions: Design-by-Contract in software engineering, contract theory for cyber-physical systems, runtime monitoring and verification, and the rapidly evolving landscape of AI agent safety. We survey each in turn, positioning ABC relative to the state of the art.

2.1 Design by Contract

The Design-by-Contract (DbC) paradigm, introduced by Meyer [1992] and elaborated in Meyer [1997], formalizes the obligations between software components as preconditions, postconditions, and class invariants. DbC has been operationalized in specification languages such as JML for

Java [Leavens et al., 2006] and Spec# for C# [Barnett et al., 2004], enabling static and runtime verification of contractual obligations in traditional software.

The extension of DbC to neural and neurosymbolic systems is recent. Leoveanu-Condrei [2025] propose a neurosymbolic contract layer for trustworthy agent design, defining preconditions and postconditions over individual LLM calls. This work is the closest conceptual predecessor to ABC in the DbC tradition. However, it is limited to single LLM invocations—it does not address multi-turn behavioral drift, multi-agent composition, soft constraint recovery, or runtime governance enforcement over extended sessions. ABC generalizes the DbC paradigm from individual function calls to autonomous agent *sessions*, introducing invariants that must hold across time, governance constraints over actions, recovery mechanisms for soft violations, and a compositionality theorem for multi-agent chains.

2.2 Contract Theory for Cyber-Physical Systems

Contract-based design has a rich history in cyber-physical systems (CPS). The meta-theory of Benveniste et al. [2018] provides a unifying algebraic framework for assume-guarantee contracts, establishing composition operators, refinement relations, and compatibility conditions across heterogeneous component models. Assume-guarantee reasoning [Henzinger et al., 1998] decomposes system-level verification into per-component obligations, a principle that ABC extends to multi-agent AI pipelines through its compositionality theorem (Theorem 4.9).

In the stochastic setting, Li et al. [2017] develop stochastic assume-guarantee contracts for CPS under probabilistic requirements, and Hampus and Nyberg [2024] extend probabilistic contracts to cyber-physical architectures. These works establish the theoretical foundations for reasoning about contracts in the presence of uncertainty—a necessity shared by AI agents, whose outputs are inherently non-deterministic.

Most recently, Ye and Tan [2026] introduce “Agent Contracts” for resource-bounded autonomous AI systems. Their framework formalizes *resource governance*: multi-dimensional constraints on token consumption, execution time, cost budgets, and delegation hierarchies, with conservation laws ensuring delegated budgets respect parent constraints. The ABC framework is complementary: whereas Ye and Tan [2026] govern *how much* an agent may consume (resource contracts), ABC governs *how* an agent must behave (behavioral contracts)—specifying preconditions, invariants, drift bounds, and recovery mechanisms over the agent’s actions and outputs. The two frameworks address orthogonal concerns and could be composed: resource contracts bounding computation, behavioral contracts bounding behavior.

ABC extends the CPS contract tradition to autonomous AI agents. The key technical differences are: (i) the state space in CPS contracts is typically continuous and governed by physical dynamics, whereas agent state spaces encompass natural language context, tool invocation history, and semantic content; (ii) CPS contracts assume well-characterized noise models (e.g., Gaussian sensor noise), whereas LLM non-determinism arises from discrete token sampling, temperature scaling, and context window effects; and (iii) CPS contracts do not address behavioral drift—a phenomenon specific to autoregressive models operating over extended horizons. The (p, δ, k) -satisfaction framework (Definition 3.7) bridges this gap by defining probabilistic guarantees tailored to the recovery-centric nature of LLM agent behavior.

2.3 Runtime Monitoring and Verification

Runtime verification (RV) monitors system executions against formal specifications, typically expressed in temporal logic [Leucker and Schallhart, 2009]. Bauer et al. [2011] develop efficient online

monitoring algorithms for linear temporal logic (LTL) and timed LTL properties, enabling real-time verification of safety and liveness requirements. These techniques provide the theoretical underpinning for ABC’s runtime enforcement loop, which evaluates contract predicates at each agent action.

In the reinforcement learning setting, Alshiekh et al. [2018] introduce *shielding*—synthesizing a reactive system (a “shield”) from temporal logic specifications that intercepts unsafe actions before they are executed. Shielding provides strong safety guarantees while preserving the convergence properties of the underlying learning algorithm. However, shielding assumes a formal environment model from which the shield can be synthesized, a requirement that is infeasible for LLM agents operating in open-ended natural language environments. ABC achieves analogous runtime enforcement using declarative behavioral contracts evaluated over runtime observations, without requiring a synthesized environment model.

Two recent systems apply formal verification ideas to LLM agents. VERIGUARD [Miculicich et al., 2025] combines offline formal verification of a behavioral policy with online monitoring during execution, providing safety guarantees through a dual-stage architecture. STEPSHIELD [Felicia et al., 2026] introduces a benchmark for *temporal* detection of agent violations, measuring not merely *whether* violations are detected but *when*—introducing metrics such as Early Intervention Rate and Intervention Gap that quantify the timeliness of enforcement.

ABC differs from these approaches in two respects. First, ABC’s contract structure is *specification-first*: contracts are defined declaratively via CONTRACTSPEC before deployment, rather than inferred from verification of generated code (VERIGUARD) or evaluated post-hoc from execution traces (STEPSHIELD). Second, ABC integrates behavioral drift detection as a *leading indicator* (Remark 3.16), enabling preemptive intervention before constraint violations materialize—a capability absent from both VERIGUARD and STEPSHIELD.

2.4 AI Agent Safety and Governance

The safety of LLM-based agents has attracted intense research attention, producing a diverse landscape of approaches that we organize by methodology.

Training-time alignment. Constitutional AI [Bai et al., 2022] trains models to adhere to a set of behavioral principles through self-critique and revision, producing outputs that are more aligned with human values. RLHF [Ouyang et al., 2022] fine-tunes models using human preference data to improve instruction-following and reduce harmful outputs. These approaches are *complementary* to ABC: they improve the baseline behavior of the underlying model, reducing the frequency of contract violations, but they cannot enforce deployment-specific constraints, adapt to novel operational requirements, or provide formal compliance guarantees at runtime.

Output filtering and guardrails. NeMo Guardrails [Rebedea et al., 2023] provides a programmable framework for constraining LLM application behavior through topical rails, safety rails, and dialog management. Guardrails AI [Guardrails AI, 2024] is the most widely deployed open-source LLM output validation library, providing validators for structured output, PII detection, toxicity filtering, and hallucination checks on individual LLM responses. While effective for per-response output filtering, both NeMo Guardrails and Guardrails AI operate on individual responses without maintaining state across turns, do not specify session-level preconditions or invariants, do not detect behavioral drift over multi-turn interactions, and do not provide formal compliance guarantees or recovery mechanisms. ABC operates at a fundamentally different granularity: session-level behavioral contracts rather than per-response output validation.

Specification-based enforcement. The most directly comparable works to ABC are specification-based systems that define and enforce behavioral rules for agents.

AGENTSPEC [Wang et al., 2026b], accepted at ICSE 2026, introduces a customizable runtime enforcement framework with a rule-based DSL for specifying safety properties of LLM agents. AGENTSPEC supports both preventive and corrective enforcement modes and evaluates on WebArena and ToolEmu benchmarks. However, AGENTSPEC does not provide probabilistic compliance guarantees (it treats constraints as deterministic rules), does not model or detect behavioral drift, and does not establish compositionality conditions for multi-agent systems.

PRO2GUARD [Wang et al., 2025] extends the runtime enforcement paradigm with probabilistic model checking via discrete-time Markov chains (DTMCs). By learning transition probabilities from execution traces, PRO2GUARD enables proactive enforcement that anticipates likely violations. This is the closest *methodological* competitor to ABC: both frameworks reason probabilistically about agent behavior. The key distinction is that PRO2GUARD is *reactive*—it learns its probabilistic model from observed traces and refines enforcement accordingly—whereas ABC is *proactive*—behavioral expectations are specified as contracts *before* deployment, with probabilistic guarantees derived from the contract structure itself. Additionally, PRO2GUARD does not provide a contract DSL, a compositionality theorem, or a behavioral drift metric.

AGENT-C [Dong et al., 2025] defines a DSL for temporal safety constraints and uses SMT solving to enforce compliance during generation. By integrating constraint checking into the decoding process, AGENT-C achieves high conformance rates on benchmarks requiring temporal ordering of actions (e.g., “authenticate before accessing records”). ABC differs in scope and mechanism: whereas AGENT-C focuses on temporal ordering constraints enforced at generation time, ABC specifies *behavioral* contracts encompassing preconditions, invariants, governance, and recovery, enforced at runtime across entire sessions. AGENT-C does not address probabilistic satisfaction, compositionality, or behavioral drift.

Incident response and governance frameworks. Xiao et al. [2026] introduce AIR, a domain-specific language for managing incident response in LLM agents, supporting detection, containment, recovery, and eradication of safety incidents. AIR achieves >90% success rates across its incident lifecycle. The distinction from ABC is one of orientation: AIR is *reactive*, responding to incidents after they occur; ABC is *proactive*, specifying contracts that prevent violations or bound their impact *a priori*. The two approaches are complementary—AIR could serve as the escalation layer when ABC recovery mechanisms are exhausted.

AGENTSAFE [Khan et al., 2025] proposes a unified governance framework spanning design-time, runtime, and audit controls for agentic AI, including anomaly detection and interruptibility mechanisms. POLARIS [Moslemi et al., 2026], presented at the AAAI 2026 Workshop, introduces governed orchestration for enterprise workflows with typed planning and validator-gated execution. Both frameworks operate at a higher level of abstraction than ABC, providing governance *architecture* rather than formal behavioral contracts with mathematical guarantees.

2.5 Agent Behavioral Drift

Behavioral drift in AI agents—the progressive divergence of agent behavior from intended specifications over extended interactions—has recently emerged as a recognized phenomenon. Rath [2026] provide the first systematic study, introducing an Agent Stability Index (ASI) and demonstrating that multi-agent LLM systems exhibit measurable behavioral degradation over extended interactions. Their work establishes that drift is a real and quantifiable problem; ABC provides the formal machinery to *prevent* it.

Table 1: Comparison of agent safety and specification frameworks. A checkmark (\checkmark) indicates the feature is supported; a dash (–) indicates it is absent; “Partial” indicates limited or indirect support.

Feature	ABC	AgentSpec	Pro2Guard	Agent-C	VeriGuard	AIR	Ye '26
Formal contracts	✓	Partial	–	–	Partial	–	✓
Probabilistic guarantees	✓	–	✓	–	–	–	–
Drift detection	✓	–	–	–	–	–	–
Contract DSL	✓	✓	–	✓	–	✓	–
Compositionality	✓	–	–	–	–	–	✓
Runtime enforcement	✓	✓	✓	✓	✓	✓	✓
Recovery mechanisms	✓	Partial	–	–	–	✓	–
Resource governance	–	–	–	–	–	–	✓

The concept of drift in machine learning more broadly is well-studied under the umbrella of concept drift [Gama et al., 2014], which addresses changes in the underlying data distribution over time. ABC’s behavioral drift score $D(t)$ (Definition 3.12) adapts the concept drift framework to the agent setting by combining a compliance-gap component (a lagging indicator of constraint violations) with a Jensen–Shannon divergence component (a leading indicator of distributional shift in the agent’s action space).

The theoretical necessity of active drift prevention is underscored by recent impossibility results. Wang et al. [2026a] prove that in self-evolving AI societies, safety alignment inevitably degrades absent external intervention—a result that validates ABC’s approach of continuous runtime enforcement rather than reliance on static, training-time alignment. Cartagena and Teixeira [2026] demonstrate empirically that text-level safety does not transfer to tool-call safety, confirming that behavioral contracts must operate at the *action* level, not merely the *output* level.

2.6 Positioning ABC

Table 1 summarizes the landscape. ABC is, to our knowledge, the only framework that simultaneously provides formal behavioral contracts, probabilistic compliance guarantees, behavioral drift detection, a specification DSL, compositionality for multi-agent pipelines, and runtime enforcement—a unified full-stack approach from theory to implementation.

The closest works along individual dimensions are: AGENTSPEC for rule-based runtime enforcement, PRO2GUARD for probabilistic reasoning about agent behavior, AGENT-C for constraint DSLs with formal backing, VERI GUARD for verified agent behavior, and Ye and Tan [2026] for resource-bounded contract governance. No prior work provides the combination of proactive behavioral specification, probabilistic guarantees with bounded drift, compositionality, and a practical DSL and runtime library that ABC delivers.

3 The ABC Framework

We now present the formal foundations of Agent Behavioral Contracts (ABC). The framework introduces a contract structure that distinguishes hard constraints (which must never be violated) from soft constraints (which admit transient violations provided recovery occurs within a bounded window). This distinction is motivated by the non-deterministic nature of large language model outputs: demanding perfect compliance at every step is both impractical and unnecessary when effective recovery mechanisms exist.

We develop the theory in stages. Section 3.1 defines the contract tuple. Section 3.2 establishes deterministic satisfaction as a baseline. Section 3.3 introduces (p, δ, k) -satisfaction, our central definition. Section 3.4 proves that recovery transforms exponential compliance decay into linear decay. Section 3.5 defines the behavioral drift score, a two-component metric that serves as both a diagnostic and a predictive signal. Section 3.6 summarizes additional operational metrics.

3.1 Contract Structure

Definition 3.1 (Agent Behavioral Contract). An *Agent Behavioral Contract* is a tuple

$$\mathcal{C} = (\mathcal{P}, \mathcal{I}_{\text{hard}}, \mathcal{I}_{\text{soft}}, \mathcal{G}_{\text{hard}}, \mathcal{G}_{\text{soft}}, \mathcal{R}),$$

where:

1. $\mathcal{P} = \{p_1, \dots, p_m\}$ is a finite set of *preconditions*: predicates over the initial state s_0 that must hold before the agent begins execution.
2. $\mathcal{I}_{\text{hard}} = \{i_1^h, \dots, i_{n_h}^h\}$ is a finite set of *hard invariants*: predicates over states that must hold at *every* step of execution. Hard invariants encode safety-critical properties such as “no personally identifiable information is emitted” or “data access is restricted to authorized sources.” A single violation of any hard invariant constitutes a contract breach.
3. $\mathcal{I}_{\text{soft}} = \{i_1^s, \dots, i_{n_s}^s\}$ is a finite set of *soft invariants*: predicates over states that may be transiently violated provided recovery occurs within a bounded window. Soft invariants encode desirable-but-recoverable properties such as “response maintains professional tone” or “confidence scores exceed threshold θ .”
4. $\mathcal{G}_{\text{hard}} = \{g_1^h, \dots, g_{l_h}^h\}$ is a finite set of *hard governance constraints*¹: predicates over actions that must hold for every action the agent takes. These encode zero-tolerance operational bounds such as spending limits, prohibited tool invocations, or forbidden output categories.
5. $\mathcal{G}_{\text{soft}} = \{g_1^s, \dots, g_{l_s}^s\}$ is a finite set of *soft governance constraints*: predicates over actions that admit transient violations with recovery. Examples include token budget warnings, response latency thresholds, and soft timeout advisories.
6. $\mathcal{R}: (\mathcal{I}_{\text{soft}} \cup \mathcal{G}_{\text{soft}}) \times \mathcal{S} \rightharpoonup \mathcal{A}^*$ is a *recovery mechanism*: a *partial* mapping from a violated soft constraint and the current state to a finite sequence of corrective actions. When $\mathcal{R}(c, s)$ is defined, its length is at most k_{\max} . When $\mathcal{R}(c, s)$ is undefined—i.e., no automated recovery is available for constraint c in state s —the monitor emits a RECOVERYFAILED event and defers to external intervention (human operator or orchestrator).

We write $\mathcal{I} = \mathcal{I}_{\text{hard}} \cup \mathcal{I}_{\text{soft}}$ and $\mathcal{G} = \mathcal{G}_{\text{hard}} \cup \mathcal{G}_{\text{soft}}$ when the hard/soft distinction is not relevant. For brevity, we occasionally use the shorthand $\mathcal{C} = (P, I, G, R)$ where $I = \mathcal{I}_{\text{hard}} \cup \mathcal{I}_{\text{soft}}$, $G = \mathcal{G}_{\text{hard}} \cup \mathcal{G}_{\text{soft}}$, and the hard/soft partition is implicit.

Remark 3.2 (Safety and Liveness Interpretation). In the taxonomy of temporal properties [Alpern and Schneider, 1987], hard constraints ($\mathcal{I}_{\text{hard}}$, $\mathcal{G}_{\text{hard}}$) are *safety* properties: they assert that “something bad never happens.” Soft constraints ($\mathcal{I}_{\text{soft}}$, $\mathcal{G}_{\text{soft}}$) with recovery window k encode *bounded*

¹We use “governance” in the operational sense: runtime-enforceable constraints on agent actions (spending limits, tool restrictions, output filters). This is distinct from the broader “AI governance” discourse concerning policy, regulation, and societal oversight of AI systems [Cihon et al., 2021]. Our governance constraints are the runtime mechanism through which high-level AI governance policies can be operationalized at the individual agent level.

liveness: they assert that “something good eventually happens within k steps.” The bounded recovery window k distinguishes ABC soft constraints from standard liveness properties, which impose no finite deadline. This bounded-liveness semantics is essential for practical deployment: an unbounded recovery promise is operationally indistinguishable from no recovery promise at all.

Definition 3.3 (Execution Trace). An *execution trace* of length T is a finite alternating sequence of states and actions:

$$\tau = (s_0, a_0, s_1, a_1, \dots, s_{T-1}, a_{T-1}, s_T),$$

where $s_t \in \mathcal{S}$ denotes the agent’s state at step t and $a_t \in \mathcal{A}$ denotes the action taken at step t . The state space \mathcal{S} encompasses the agent’s internal context (e.g., conversation history, accumulated tool outputs, working memory) and the observable environment. The action space \mathcal{A} encompasses all outputs the agent may produce (e.g., text responses, tool calls, API invocations).

3.2 Contract Satisfaction (Deterministic)

We first define satisfaction in the deterministic setting, which serves as the foundation for the probabilistic extension.

Definition 3.4 (Deterministic Contract Satisfaction). An agent A *satisfies* contract $\mathcal{C} = (\mathcal{P}, \mathcal{I}, \mathcal{G}, \mathcal{R})$ over an execution trace $\tau = (s_0, a_0, \dots, s_T)$ if all of the following conditions hold:

1. **Precondition validity.** Every precondition holds at the initial state:

$$\forall p \in \mathcal{P} : p(s_0) = \text{true}.$$

2. **Invariant compliance.** Every invariant holds at every state along the trace:

$$\forall t \in \{0, \dots, T\}, \forall i \in \mathcal{I} : i(s_t) = \text{true}.$$

3. **Governance compliance.** Every governance constraint holds for every action:

$$\forall t \in \{0, \dots, T-1\}, \forall g \in \mathcal{G} : g(a_t) = \text{true}.$$

4. **Recoverability.** For every soft constraint violation, the recovery mechanism restores compliance within k steps:

$$\forall t, \forall c \in \mathcal{I}_{\text{soft}} \cup \mathcal{G}_{\text{soft}} : \neg c(s_t, a_t) \implies \exists t' \in \{t, \dots, \min(t+k, T)\} : c(s_{t'}, a_{t'}) = \text{true}.$$

We write $A \models \mathcal{C}$ to denote that agent A satisfies contract \mathcal{C} over all traces induced by A .

Remark 3.5. Deterministic satisfaction is a useful theoretical baseline, but it is too stringent for LLM-based agents. The stochastic nature of token sampling means that even well-aligned agents produce occasional soft violations. The next subsection relaxes this to a probabilistic guarantee.

3.3 Probabilistic (p, δ, k) -Satisfaction

This is the central definition of the ABC framework. It captures the key insight that hard constraints require high-probability guarantees of *persistent* compliance, while soft constraints require high-probability guarantees of *recoverable* compliance.

We first define the compliance scores that the probabilistic conditions reference.

Definition 3.6 (Hard and Soft Compliance Scores). Given contract \mathcal{C} and execution trace τ , define the *hard compliance score* and *soft compliance score* at step t as:

$$C(t)_{\text{hard}}(t) = \frac{|\{c \in \mathcal{I}_{\text{hard}} \cup \mathcal{G}_{\text{hard}} : c(s_t, a_t) = \text{true}\}|}{|\mathcal{I}_{\text{hard}} \cup \mathcal{G}_{\text{hard}}|}, \quad (1)$$

$$C(t)_{\text{soft}}(t) = \frac{|\{c \in \mathcal{I}_{\text{soft}} \cup \mathcal{G}_{\text{soft}} : c(s_t, a_t) = \text{true}\}|}{|\mathcal{I}_{\text{soft}} \cup \mathcal{G}_{\text{soft}}|}. \quad (2)$$

Both scores lie in $[0, 1]$, with $C(t)_{\text{hard}}(t) = 1$ indicating full hard compliance and $C(t)_{\text{soft}}(t) = 1$ indicating full soft compliance at step t .

Definition 3.7 $((p, \delta, k)\text{-Satisfaction})$. Let $p \in [0, 1]$ be a probability threshold, $\delta \in [0, 1]$ an allowed soft deviation, $k \in \mathbb{N}$ a recovery window, and $T \in \mathbb{N}$ a session length. An agent A (p, δ, k) -*satisfies* contract \mathcal{C} over session length T , written

$$A \models_{p, \delta, k} \mathcal{C},$$

if both of the following conditions hold:

(i) Hard guarantee (persistent compliance).

$$\mathbb{P}\left[C(t)_{\text{hard}}(t) = 1 \quad \forall t \in \{0, \dots, T\} \mid \mathcal{P}(s_0)\right] \geq p. \quad (3)$$

(ii) Soft guarantee (recoverable compliance).

$$\mathbb{P}\left[\forall t \in \{0, \dots, T\} : C(t)_{\text{soft}}(t) < 1 - \delta \implies \exists t' \in \{t, \dots, \min(t+k, T)\} : C(t)_{\text{soft}}(t') \geq 1 - \delta \mid \mathcal{P}(s_0)\right] \geq p. \quad (4)$$

The parameters have the following interpretation:

- p : the minimum probability with which the guarantee must hold. For safety-critical deployments, $p \geq 0.99$; for advisory agents, $p \geq 0.90$ may suffice.
- δ : the tolerable deviation in soft compliance. Setting $\delta = 0$ requires perfect soft compliance whenever the soft guarantee holds; $\delta = 0.1$ allows up to 10% of soft constraints to be violated at any given step.
- k : the recovery window in steps. A soft violation at step t is acceptable if compliance is restored by step $t + k$. Smaller k demands faster recovery.
- T : the session length (total number of steps). Longer sessions require stronger per-step guarantees to maintain the same overall probability p .

Remark 3.8 (Novelty of the Recovery Window Parameter). The recovery window k is, to our knowledge, the first formal inclusion of a bounded recovery horizon as a first-class parameter in a contract satisfaction definition. Prior Design-by-Contract frameworks [Meyer, 1992] and runtime verification systems [Leucker and Schallhart, 2009] treat violations as binary pass/fail events with no notion of time-bounded recovery. The k -parameter bridges formal contracts and practical LLM deployment: it quantifies how much “slack” an agent is allowed before a transient deviation becomes a reportable failure, enabling principled tuning of the strictness–availability trade-off.

Remark 3.9 (Connection to Probabilistic Computation Tree Logic). The (p, δ, k) -satisfaction conditions have natural counterparts in Probabilistic Computation Tree Logic (PCTL) [Hansson and Jonsson, 1994]. The hard guarantee (3) corresponds to the PCTL formula

$$\mathcal{P}_{\geq p}[\mathbf{G}(C(t)_{\text{hard}} = 1)],$$

which asserts that with probability at least p , the hard compliance score is *globally* (at every step) equal to 1. The soft guarantee (4) corresponds to

$$\mathcal{P}_{\geq p}\left[\mathbf{G}(C(t)_{\text{soft}} < 1 - \delta \implies \mathbf{F}_{\leq k}(C(t)_{\text{soft}} \geq 1 - \delta))\right],$$

which asserts that with probability at least p , it is *globally* true that any soft compliance drop below $1 - \delta$ is *eventually* (within k steps) recovered. This connection to PCTL enables the use of established model-checking techniques for verification when the agent's transition structure can be approximated as a finite Markov decision process.

3.4 Recovery Transforms Exponential Decay to Linear Decay

The following lemma establishes the fundamental value of recovery mechanisms: they convert an exponentially decaying compliance probability into a linearly decaying one.

Lemma 3.10 (Recovery Linearizes Compliance Decay). *Let $q \in (0, 1)$ denote the per-step compliance probability (i.e., at each step t , the agent satisfies all relevant constraints with probability q , independently). Let $r \in [0, 1]$ denote the recovery effectiveness: given a violation, the recovery mechanism restores compliance with probability r within the allowed window. Then:*

1. **Without recovery.** The probability of sustained compliance over T steps decays exponentially:

$$\mathbb{P}[\text{compliance over } T \text{ steps}] = q^T. \quad (5)$$

2. **With recovery.** The probability of recoverable compliance over T steps satisfies:

$$\mathbb{P}[\text{recoverable compliance over } T \text{ steps}] \geq 1 - T(1 - q)(1 - r). \quad (6)$$

Proof. The first claim follows directly from the independence assumption: compliance at each step occurs with probability q , so compliance at all T steps occurs with probability q^T .

For the second claim, we apply a union bound. Define the event F_t as the event that step t incurs a violation *and* recovery fails to restore compliance within the recovery window. The probability of a violation at step t is $1 - q$. Conditional on a violation, recovery fails with probability $1 - r$. By independence of the violation and recovery events:

$$\mathbb{P}[F_t] = (1 - q)(1 - r).$$

The system experiences an unrecoverable failure if *any* step incurs both a violation and a recovery failure. By the union bound:

$$\mathbb{P}\left[\bigcup_{t=0}^{T-1} F_t\right] \leq \sum_{t=0}^{T-1} \mathbb{P}[F_t] = T(1 - q)(1 - r).$$

Therefore, the probability that all violations are successfully recovered is:

$$\mathbb{P}[\text{recoverable compliance over } T \text{ steps}] = 1 - \mathbb{P}\left[\bigcup_{t=0}^{T-1} F_t\right] \geq 1 - T(1 - q)(1 - r). \quad \square$$

Example 3.11. Consider an agent with per-step compliance $q = 0.99$ over a session of $T = 100$ steps. Without recovery, the probability of sustained compliance is $0.99^{100} \approx 0.366$ —a coin flip is more reliable. With a recovery mechanism of effectiveness $r = 0.95$, the bound becomes $1 - 100 \cdot 0.01 \cdot 0.05 = 1 - 0.05 = 0.95$. Recovery transforms a 36.6% compliance probability into a 95% guarantee: a qualitative shift from unreliable to deployable.

3.5 Behavioral Drift Score

Compliance scores detect violations *after* they occur. We introduce the *behavioral drift score*² $D(t)(t)$ as a composite metric that combines a reactive compliance component with a predictive distributional component, enabling early detection of emerging misalignment.

Definition 3.12 (Behavioral Drift Score). The *behavioral drift score* at step t is defined as:

$$D(t)(t) = w_c \cdot D(t)_{\text{compliance}}(t) + w_d \cdot D(t)_{\text{distributional}}(t), \quad (7)$$

where $w_c, w_d \geq 0$ with $w_c + w_d = 1$ (with application-specific tuning; in practice, weighting the compliance component more heavily than the distributional component), and the components are:

Compliance drift. The weighted compliance gap at step t :

$$D(t)_{\text{compliance}}(t) = 1 - C(t)(t) = \frac{\sum_i w_i (1 - \sigma_i(t))}{\sum_i w_i}, \quad (8)$$

where $\sigma_i(t) \in \{0, 1\}$ indicates whether constraint i is satisfied at step t , and $w_i > 0$ is the weight assigned to constraint i .

Distributional drift. The Jensen–Shannon divergence between the observed and reference action distributions:

$$D(t)_{\text{distributional}}(t) = \text{JSD}(P_{\text{observed}}(t) \| P_{\text{reference}}), \quad (9)$$

where $P_{\text{observed}}(t)$ is the empirical action distribution computed over a sliding window of recent actions, and $P_{\text{reference}}$ is a calibrated reference distribution obtained from a compliant baseline (e.g., the action distribution during a validated calibration session).

Remark 3.13 (Interpretability of $D(t)$ Values). The drift score $D(t) \in [0, 1]$ admits the following operational interpretation:

- $D(t) = 0$: perfect compliance and distributional alignment.
- $D(t) \in (0, \theta_1]$: negligible drift; no intervention required.
- $D(t) \in (\theta_1, \theta_2]$: mild drift; monitoring should increase in frequency.
- $D(t) > \theta_2$: significant drift; active intervention is recommended.

The threshold parameters θ_1 and θ_2 are deployment-specific and empirically calibrated. Typical enterprise deployments use low single-digit and mid-range values respectively. Both thresholds are exposed as configurable parameters in the contract specification.

Proposition 3.14 (Properties of the Drift Score). *The behavioral drift score $D(t)(t)$ satisfies:*

1. **Boundedness.** $D(t)(t) \in [0, 1]$ for all t .

²Our behavioral drift score $D(t)$ is distinct from the *Agent Stability Index* (ASI) proposed by Rath [2024]. The ASI measures distributional shift in the model’s output *embedding space* across sessions and serves as a model-level diagnostic. Our $D(t)$ is a *contract-level* metric: it combines a compliance component (fraction of violated constraints) with a distributional component (JSD over the action vocabulary), computed per-step and tied directly to the enforcement loop. The two metrics are complementary; see Section 2 for a detailed comparison.

2. **Minimality.** $D(t)(t) = 0$ if and only if full compliance holds ($C(t)(t) = 1$) and the observed action distribution is identical to the reference distribution ($P_{\text{observed}}(t) = P_{\text{reference}}$).
3. **Incremental computability.** $D(t)(t)$ can be updated incrementally with complexity linear in the number of constraints and the action vocabulary size.
4. **Metric structure.** The square root of the Jensen–Shannon divergence, $\sqrt{\text{JSD}}$, is a metric on probability distributions and satisfies the triangle inequality [Endres and Schindelin, 2003].

Proof. (1) Both $D(t)_{\text{compliance}}(t) \in [0, 1]$ (since $C(t)(t) \in [0, 1]$) and $D(t)_{\text{distributional}}(t) \in [0, 1]$ (the Jensen–Shannon divergence with logarithm base 2 is bounded by 1). Since $w_c + w_d = 1$ with $w_c, w_d \geq 0$, the convex combination lies in $[0, 1]$.

(2) The forward direction: $D(t)(t) = 0$ requires both $w_c \cdot D(t)_{\text{compliance}}(t) = 0$ and $w_d \cdot D(t)_{\text{distributional}}(t) = 0$. Since $w_c, w_d > 0$ in the default parameterization, this forces $D(t)_{\text{compliance}}(t) = 0$ (i.e., $C(t)(t) = 1$) and $\text{JSD}(P_{\text{observed}}(t) \| P_{\text{reference}}) = 0$ (i.e., distributional identity). The converse is immediate.

(3) The compliance component requires evaluating each constraint, contributing cost linear in the constraint set size. The distributional component maintains a histogram over the sliding window; inserting and removing one action and recomputing JSD costs linear in the action vocabulary size.

(4) Proven by Endres and Schindelin [2003]; see also Österreicher and Vajda [2003]. \square

Remark 3.15 (Meaningfulness of the Distributional Component). The JSD distributional component of $D(t)$ requires a sufficiently rich action vocabulary to produce informative signals. When the action space is insufficiently diverse, the empirical action distribution may be sparse and distributional measures exhibit high variance. In such cases, practitioners should either increase the observation window to smooth the estimate, or adjust the component weights to emphasize constraint-based compliance over distributional alignment. For typical enterprise deployments with diverse tool invocations, text categories, and API call types, the action vocabulary is easily sufficient.

Remark 3.16 (Leading vs. Lagging Indicators). The two components of the drift score serve complementary diagnostic roles. The compliance drift $D(t)_{\text{compliance}}(t)$ is a *lagging indicator*: it registers non-zero values only after a constraint violation has already occurred. The distributional drift $D(t)_{\text{distributional}}(t)$ is a *leading indicator*: it can detect shifts in the agent’s action distribution—such as increased use of hedging language, atypical tool invocation patterns, or drifting topic focus—before these shifts manifest as explicit constraint violations. This early-warning capability is critical for preemptive intervention.

For fine-grained diagnostics, we decompose the drift score into its constituent sources.

Definition 3.17 (Diagnostic Decomposition Vector). The *diagnostic decomposition vector* at step t is:

$$\vec{D}(t)(t) = (D(t)_{\mathcal{P}}(t), D(t)_{\mathcal{I}}(t), D(t)_{\mathcal{G}}(t), D(t)_{\text{distributional}}(t)), \quad (10)$$

where $D(t)_{\mathcal{P}}(t)$, $D(t)_{\mathcal{I}}(t)$, and $D(t)_{\mathcal{G}}(t)$ are the compliance gaps restricted to precondition-derived, invariant, and governance constraints, respectively. This vector enables operators to pinpoint whether drift originates from invariant violations, governance breaches, or distributional shift, and to route alerts to the appropriate remediation pathway.

3.6 Additional Operational Metrics

We briefly define three additional metrics that complement the compliance and drift scores in operational deployments.

Definition 3.18 (Recovery Effectiveness). The *recovery effectiveness* for a violation event at step t is:

$$E(t) = \frac{\Delta t_{\text{recovery}}}{\nu(t)}, \quad (11)$$

where $\Delta t_{\text{recovery}}$ is the number of steps required to restore compliance and $\nu(t) \in (0, 1]$ is the severity of the violation (defined as the magnitude of the compliance drop). Lower values of E indicate more effective recovery. We define the session-level recovery effectiveness as $E = \frac{1}{|\mathcal{V}|} \sum_{t \in \mathcal{V}} E(t)$, where \mathcal{V} is the set of violation events.

Definition 3.19 (Stress Resilience Index). The *stress resilience index* measures compliance degradation under adversarial or high-load conditions:

$$S = \frac{\mathbb{E}[C(t)(t) \mid \text{stressed}]}{\mathbb{E}[C(t)(t) \mid \text{baseline}]}, \quad (12)$$

where the expectations are taken over steps within stressed and baseline sessions, respectively. A value $S = 1$ indicates no degradation under stress; $S < 1$ quantifies the compliance penalty imposed by adversarial conditions.

Definition 3.20 (Agent Reliability Index). The *agent reliability index* is a weighted composite that summarizes an agent's overall contractual fitness:

$$\Theta = \alpha_1 \cdot \overline{C(t)} + \alpha_2 \cdot (1 - \overline{D(t)}) + \alpha_3 \cdot \frac{1}{1+E} + \alpha_4 \cdot S, \quad (13)$$

where $\overline{C(t)}$ and $\overline{D(t)}$ denote the time-averaged compliance and drift scores over the session, the term $\frac{1}{1+E}$ maps recovery effectiveness to $[0, 1]$ (with lower E yielding higher contribution), and the weights satisfy $\sum_{i=1}^4 \alpha_i = 1$. The component weights are application-specific, with typical enterprise deployments weighting compliance most heavily, followed by drift stability, recovery efficiency, and stress resilience. The index $\Theta \in [0, 1]$ provides a single scalar summary suitable for comparing agents, tracking reliability over time, and establishing deployment thresholds.

4 Drift Prevention via Contracts

This section provides the theoretical backbone of the ABC framework. We model behavioral drift as a continuous-time stochastic process (Section 4.1), derive tight probabilistic bounds on drift under contract enforcement (Section 4.2), establish sufficient conditions for safe contract composition in multi-agent chains (Section 4.3), and analyze the runtime cost of contract checking (Section 4.4).

4.1 Drift Dynamics Model

We model the behavioral drift of a contracted agent as a continuous-time stochastic process governed by three competing forces: a natural tendency to deviate from specification, a restorative force exerted by contract enforcement, and stochastic perturbations inherent to LLM non-determinism.

Definition 4.1 (Drift Dynamics). Let $D(t) \geq 0$ denote the *behavioral drift* of an agent at time $t \geq 0$, measured as the JSD divergence between the agent's observed action distribution and the contract-compliant reference distribution (cf. Definition 3.12). The drift evolves according to the stochastic differential equation

$$dD(t) = (\alpha - \gamma D(t)) dt + \sigma dW(t), \quad (14)$$

where the parameters satisfy $\alpha > 0$, $\gamma > 0$, $\sigma > 0$, and $W(t)$ is a standard Wiener process.

The three terms in (14) admit clear interpretations:

- (i) **Baseline drift** (αdt). In the absence of enforcement, the agent's behavior naturally diverges from the contracted specification at rate α . This captures prompt decay, context window dilution, and the tendency of autoregressive models to amplify small distributional shifts over extended task horizons.
- (ii) **Contract recovery** ($-\gamma D(t) dt$). The enforcement mechanism exerts a restorative force proportional to current drift. When $D(t)$ is large, the corrective signal is strong; when the agent is near compliance, the force relaxes. The parameter γ is the *contract recovery rate*—a design-time knob controlled by the contract's invariant-checking frequency and the aggressiveness of its recovery policy R .
- (iii) **Stochastic perturbation** ($\sigma dW(t)$). LLM outputs are inherently non-deterministic: identical prompts yield different completions across invocations. The diffusion coefficient σ quantifies this irreducible noise floor, encompassing sampling temperature, nucleus truncation, and hardware floating-point variance.

Remark 4.2. Equation (14) is an instance of the *Ornstein–Uhlenbeck* (OU) process with mean-reversion level $\mu^* = \alpha/\gamma$, mean-reversion speed γ , and volatility σ . The OU process is one of the few analytically tractable diffusions: it admits a closed-form transition density, a Gaussian stationary distribution, and exponential ergodicity bounds—properties we exploit throughout this section. The restriction $D(t) \geq 0$ is a modeling simplification; since the stationary mean α/γ is strictly positive and the stationary standard deviation $\sigma/\sqrt{2\gamma}$ is small relative to the mean for well-designed contracts (i.e., $\sigma^2\gamma \ll 2\alpha^2$, so the stationary standard deviation is small relative to the mean), the probability of the process reaching zero is negligible in practice.

4.2 Drift Bounds Theorem

We now state the main analytical result of this paper: a comprehensive characterization of behavioral drift under contract enforcement.

Theorem 4.3 (Stochastic Drift Bound). *Let $D(t)$ evolve according to the drift dynamics of Definition 4.1 with initial condition $D(0) = D_0 \geq 0$. Then:*

- (i) **Stationary distribution.** *There exists a unique stationary distribution*

$$\pi_D = \mathcal{N}\left(\frac{\alpha}{\gamma}, \frac{\sigma^2}{2\gamma}\right). \quad (15)$$

- (ii) **Mean drift bound.** *Under the stationary distribution,*

$$\mathbb{E}_\pi[D(t)] = \frac{\alpha}{\gamma}. \quad (16)$$

In particular, if $\gamma > \alpha$ then $\mathbb{E}_\pi[D(t)] < 1$.

- (iii) **Variance bound.** *Under the stationary distribution,*

$$\text{Var}_\pi(D(t)) = \frac{\sigma^2}{2\gamma}. \quad (17)$$

Higher contract recovery rate γ quadratically reduces the spread of drift fluctuations relative to the noise level σ .

(iv) **High-probability bound.** For any $\eta > 0$,

$$\mathbb{P}_\pi(D(t) > \frac{\alpha}{\gamma} + \eta) \leq \exp\left(-\frac{\gamma\eta^2}{\sigma^2}\right). \quad (18)$$

(v) **Exponential convergence.** For all $t \geq 0$,

$$\mathbb{E}[(D(t) - \alpha/\gamma)^2] = (D_0 - \alpha/\gamma)^2 e^{-2\gamma t} + \frac{\sigma^2}{2\gamma}(1 - e^{-2\gamma t}). \quad (19)$$

(vi) **Contract design criterion.** To ensure $D(t) < D_{\max}$ with probability at least $1 - \varepsilon$ under the stationary distribution, it suffices to choose γ as the larger root of the quadratic

$$D_{\max}^2 \gamma^2 - (2\alpha D_{\max} + \sigma^2 \ln(1/\varepsilon)) \gamma + \alpha^2 = 0, \quad (20)$$

i.e.,

$$\gamma \geq \frac{2\alpha D_{\max} + \sigma^2 \ln(1/\varepsilon) + \sqrt{(2\alpha D_{\max} + \sigma^2 \ln(1/\varepsilon))^2 - 4\alpha^2 D_{\max}^2}}{2 D_{\max}^2}. \quad (21)$$

When $\sigma^2 \ln(1/\varepsilon) \ll 2\alpha D_{\max}$, this simplifies to the approximate criterion $\gamma \gtrsim \alpha/D_{\max} + \sigma\sqrt{2\ln(1/\varepsilon)}/(2D_{\max})$.

Proof sketch. Define the centered error process $e(t) = D(t) - \alpha/\gamma$. Substituting into (14) yields the centered OU equation

$$de(t) = -\gamma e(t) dt + \sigma dW(t), \quad (22)$$

which has zero mean-reversion level and rate γ .

To establish convergence, define the Lyapunov function $V(e) = e^2$ and apply Itô's formula:

$$dV = (-2\gamma e^2 + \sigma^2) dt + 2\sigma e dW(t).$$

Taking expectations eliminates the martingale term, yielding the deterministic ODE

$$\frac{d}{dt} \mathbb{E}[V(t)] = -2\gamma \mathbb{E}[V(t)] + \sigma^2.$$

This linear ODE has solution $\mathbb{E}[V(t)] = V(0) e^{-2\gamma t} + \frac{\sigma^2}{2\gamma}(1 - e^{-2\gamma t})$, which converges exponentially to $\sigma^2/(2\gamma)$, establishing parts (iii) and (v). The stationary distribution (i) follows from standard Ornstein–Uhlenbeck theory [Uhlenbeck and Ornstein, 1930]: the unique invariant measure is Gaussian with mean α/γ and variance $\sigma^2/(2\gamma)$, from which part (ii) is immediate.

The tail bound (iv) applies Gaussian concentration to the stationary distribution: for $X \sim \mathcal{N}(\mu, \sigma_s^2)$ with $\sigma_s^2 = \sigma^2/(2\gamma)$,

$$\mathbb{P}(X > \mu + \eta) \leq \exp\left(-\frac{\eta^2}{2\sigma_s^2}\right) = \exp\left(-\frac{\gamma\eta^2}{\sigma^2}\right).$$

Finally, the design criterion (vi) follows by setting the right-hand side of (18) to ε with $\eta = D_{\max} - \alpha/\gamma$ and solving for γ . The full details are provided in Section A. \square

Remark 4.4. Theorem 4.3 has direct engineering implications. Part (vi) provides an *exact design rule*: given an application's maximum tolerable drift D_{\max} and reliability requirement $1 - \varepsilon$, the contract designer solves the quadratic (20) to obtain the minimum recovery rate γ needed to meet the specification. In the approximate regime ($\sigma^2 \ln(1/\varepsilon) \ll 2\alpha D_{\max}$), the required γ decomposes into two interpretable terms: α/D_{\max} ensures the mean drift stays below threshold, while the second term $\sigma\sqrt{2\ln(1/\varepsilon)}/(2D_{\max})$ provides the additional margin required to absorb stochastic fluctuations at the desired confidence level.

4.3 Contract Composition

Enterprise agentic systems rarely consist of a single agent. A typical deployment chains multiple specialized agents—a planner, a retriever, a coder, a reviewer—into a sequential pipeline. We now establish conditions under which individual contract guarantees compose into end-to-end guarantees for the chain.

4.3.1 Serial Composition

Consider a serial chain $A \rightarrow B$ where agent A produces output consumed by agent B .

Definition 4.5 (Composed Contract). Given contracts $\mathcal{C}_A = (P_A, I_A, G_A, R_A)$ and $\mathcal{C}_B = (P_B, I_B, G_B, R_B)$ for agents A and B respectively, the *composed contract* for the serial chain $A \rightarrow B$ is

$$\mathcal{C}_{A \oplus B} = (P_{A \oplus B}, I_{A \oplus B}, G_{A \oplus B}, R_{A \oplus B}) \quad (23)$$

where:

$$P_{A \oplus B} = P_A, \quad (24)$$

$$I_{A \oplus B} = I_A \wedge I_B \wedge I_{\text{handoff}}, \quad (25)$$

$$G_{A \oplus B} = G_A \cup G_B \quad (\text{assuming no conflicts}), \quad (26)$$

$$R_{A \oplus B} = \text{compose}(R_A, R_B, R_{\text{cascade}}), \quad (27)$$

where I_{handoff} is a handoff invariant ensuring safe state transfer between A and B , and R_{cascade} is a cascade recovery policy that coordinates individual recovery actions across the chain boundary.

We first define the postcondition of a contracted agent, which appears in the composition conditions below.

Definition 4.6 (Postcondition). The *postcondition* of agent A under contract \mathcal{C}_A , denoted PostCond_A , is the set of states reachable at termination of A that satisfy all of A 's invariants:

$$\text{PostCond}_A = \{s \in \mathcal{S} : \forall i \in \mathcal{I}_A, i(s) = \text{true}\}.$$

Safe composition requires four sufficient conditions:

Definition 4.7 (Composition Conditions). A serial chain $A \rightarrow B$ with contracts \mathcal{C}_A and \mathcal{C}_B satisfies the *composition conditions* if:

- (C1) **Interface Compatibility.** $\text{Type}(\text{PostCond}_A) \subseteq \text{Type}(P_B)$. The output type of A is a subtype of the input type expected by B .
- (C2) **Assumption Discharge.** $\text{PostCond}_A \wedge I_{\text{handoff}} \Rightarrow P_B$. The postcondition of A , together with the handoff invariant, logically entails the precondition of B .
- (C3) **Governance Consistency.** Define the set-valued functions $\text{Allowed}(G) = \{a \in \mathcal{A} : \forall g \in G, g(a) = \text{true}\}$ and $\text{Prohibited}(G) = \{a \in \mathcal{A} : \exists g \in G, g(a) = \text{false}\}$. Then $\text{Allowed}(G_A) \cap \text{Prohibited}(G_B) = \emptyset$. No action permitted by A 's governance policy is forbidden by B 's.
- (C4) **Recovery Independence.** $\forall s \in \mathcal{S} : P_B(\text{state_after}(R_A(s))) = \text{true}$. After A 's recovery mechanism fires, the resulting state still satisfies B 's precondition.

Remark 4.8 (Standard vs. Novel Composition Conditions). Conditions (C1) and (C2) (interface compatibility and assumption discharge) are standard in Design-by-Contract composition [Meyer, 1992] and assume-guarantee reasoning [Henzinger et al., 1998]. Conditions (C3) and (C4) (governance consistency and recovery independence) are novel contributions of the ABC framework, motivated by the unique operational requirements of multi-agent LLM pipelines: governance constraints span organizational policies (not just type systems), and recovery mechanisms can have cross-agent side effects that invalidate downstream preconditions.

Theorem 4.9 (Compositionality). *Let agents A and B satisfy their respective contracts, i.e., $A \models \mathcal{C}_A$ and $B \models \mathcal{C}_B$. If conditions (C1)–(C4) hold, then*

$$\text{Chain}(A, B) \models \mathcal{C}_{A \oplus B}.$$

Proof sketch. We verify each component of $\mathcal{C}_{A \oplus B}$. The chain’s precondition $P_{A \oplus B} = P_A$ holds by assumption. Since $A \models \mathcal{C}_A$, agent A terminates in a state satisfying PostCond_A . Condition (C2) then guarantees P_B holds at the handoff point, so B begins execution with a valid precondition. The composed invariant $I_A \wedge I_B \wedge I_{\text{handoff}}$ is maintained: I_A holds during A ’s execution by $A \models \mathcal{C}_A$, I_B holds during B ’s execution by $B \models \mathcal{C}_B$, and I_{handoff} holds at the transition by construction. Condition (C3) ensures the union $G_A \cup G_B$ is conflict-free. Condition (C4) ensures that if A ’s recovery fires, the post-recovery state remains a valid input for B . The full proof, including the inductive argument for governance consistency through the chain, is provided in Section A. \square

Remark 4.10 (Recovery Window Composition). When composing contracts with recovery windows k_A and k_B , the composed recovery window is $k_{A \oplus B} = \max(k_A, k_B)$. The maximum (rather than the sum) is the correct composition rule because recovery windows operate concurrently within each agent’s execution phase: a soft violation in A must be recovered within k_A steps of A ’s local trace, not within a global budget shared with B . The composed window therefore reflects the more demanding of the two per-agent requirements.

4.3.2 Probabilistic Composition

In practice, contract satisfaction is probabilistic (cf. Definition 3.7). We now characterize how probabilistic guarantees degrade under composition.

Theorem 4.11 (Probabilistic Compositionality). *Suppose agent A (p_A, δ_A) -satisfies \mathcal{C}_A and agent B (p_B, δ_B) -satisfies \mathcal{C}_B . Let p_h denote the probability that the handoff invariant I_{handoff} holds, and let δ_h denote the maximum drift introduced by the handoff mechanism. Assume:*

- (C5) **Conditional Independence.** Agent B ’s contract satisfaction is conditionally independent of agent A ’s internal execution, given that B receives a contract-compliant input from the handoff: $\mathbb{P}(E_B \mid E_A \cap E_h) = \mathbb{P}(E_B \mid E_h)$.

Then the composed chain $(p_{A \oplus B}, \delta_{A \oplus B})$ -satisfies $\mathcal{C}_{A \oplus B}$ with

$$p_{A \oplus B} \geq p_A \cdot p_B \cdot p_h, \tag{28}$$

$$\delta_{A \oplus B} \leq \delta_A + \delta_B + \delta_h. \tag{29}$$

Remark 4.12 (Conditional Independence and Correlated LLM Failures). Condition (C5) is satisfied when agents A and B operate on separate LLM instances or use distinct model providers. When both agents share the same underlying LLM, correlated failure modes (e.g., systematic prompt

sensitivity, shared training biases) may violate conditional independence. In such settings, the probability bound (28) becomes optimistic; practitioners should apply a correlation penalty or use the tighter bound $p_{A \oplus B} \geq p_A + p_B \cdot p_h - 1$ (Fréchet–Hoeffding lower bound) as a conservative alternative.

Proof sketch. The chain satisfies $\mathcal{C}_{A \oplus B}$ only if all three events occur: A satisfies \mathcal{C}_A , the handoff succeeds, and B satisfies \mathcal{C}_B . By condition (C5) (conditional independence of agent-level failures given contract-compliant inputs), the joint probability is at least $p_A \cdot p_B \cdot p_h$. The drift bound follows from sub-additivity: the maximum end-to-end deviation is bounded by the sum of per-stage deviations plus the handoff-induced deviation. See Section A for the formal argument. \square

The following corollary extends the result to chains of arbitrary length.

Corollary 4.13 (N -Agent Chain). *For a serial chain of N agents $A_1 \rightarrow A_2 \rightarrow \dots \rightarrow A_N$ where each agent A_i (p_i, δ_i) -satisfies \mathcal{C}_i and each handoff has reliability p_{h_i} and drift δ_{h_i} :*

$$p_{\text{chain}} \geq \prod_{i=1}^N p_i \cdot \prod_{i=1}^{N-1} p_{h_i}, \quad (30)$$

$$\delta_{\text{chain}} \leq \sum_{i=1}^N \delta_i + \sum_{i=1}^{N-1} \delta_{h_i}. \quad (31)$$

Proof. Follows by inductive application of Theorem 4.11 along the chain. \square

Remark 4.14 (The Broken Telephone Effect). Corollary 4.13 formalizes the intuitive ‘‘broken telephone’’ effect in multi-agent systems: reliability degrades multiplicatively while drift accumulates additively. Consider a concrete example: a 5-agent chain where each agent satisfies its contract with probability $p_i = 0.95$ and each handoff succeeds with probability $p_{h_i} = 0.98$. Then:

$$p_{\text{chain}} \geq 0.95^5 \cdot 0.98^4 \approx 0.7738 \times 0.9224 \approx 0.714.$$

Similarly, if each agent contributes drift $\delta_i = 0.02$ and each handoff contributes $\delta_{h_i} = 0.01$:

$$\delta_{\text{chain}} \leq 5 \times 0.02 + 4 \times 0.01 = 0.14.$$

A chain that appears reliable at the individual level (95% per agent) delivers only $\sim 71.4\%$ end-to-end reliability, with accumulated drift of 0.14. This quantifies why multi-agent pipelines require explicit contract enforcement at every stage, not merely at the endpoints. The design criterion of Theorem 4.3(vi) can be applied independently to each agent in the chain to ensure that per-stage drift remains within the budget implied by the global δ_{chain} target.

4.4 Complexity Analysis

For contract enforcement to be practical, the runtime overhead must be negligible relative to the latency of LLM inference itself (typically 100–2000 ms per action). We now show that this is the case.

Proposition 4.15 (Runtime Contract Checking). *Let k denote the number of constraints in a contract (preconditions, invariants, and governance rules combined) and let $|A|$ denote the size of the agent’s action vocabulary. The per-action cost of runtime contract checking is*

$$\mathcal{O}(k + |A|).$$

Proof. The enforcement loop performs three operations per agent action:

(1) *Constraint evaluation.* Each of the k constraints (preconditions, invariants, governance predicates) is evaluated as a Boolean predicate over the current state and proposed action. Each predicate evaluation is $\mathcal{O}(1)$ (pattern matching on action type, range checks on numeric fields, or set membership for governance whitelists/blacklists). Evaluating all k constraints costs $\mathcal{O}(k)$.

(2) *Behavioral drift update.* The JSD divergence is maintained incrementally via a sliding-window histogram over the action vocabulary. Updating the histogram upon observing a new action and recomputing JSD between the observed and reference distributions costs $\mathcal{O}(|A|)$, as it requires a single pass over the $|A|$ -dimensional probability vectors.

(3) *Weighted aggregation.* The overall compliance score is a weighted sum of constraint satisfaction and drift, computed in $\mathcal{O}(1)$.

Combining: $\mathcal{O}(k) + \mathcal{O}(|A|) + \mathcal{O}(1) = \mathcal{O}(k + |A|)$. □

Remark 4.16 (Practical Overhead). For typical enterprise contracts we observe $k < 100$ constraints and action vocabularies of size $|A| < 50$. At these scales, the measured wall-clock overhead of contract checking in AGENTASSERT is consistently below 10 ms per action—approximately 0.5–5% of LLM inference latency. This confirms that contract enforcement is not a bottleneck and can be deployed on every agent action without perceptible degradation in end-to-end pipeline throughput. We provide detailed latency benchmarks in Section 7.

5 ContractSpec and AgentAssert

Note: This section describes the design principles and conceptual architecture of our reference implementation. Implementation-specific details including algorithmic pseudocode, class hierarchies, and configuration parameters are abstracted to focus on the scientific contributions. The complete implementation is subject to patent protection.

The preceding sections established the formal foundations of Agent Behavioral Contracts (ABC): the contract tuple $\mathcal{C} = (\mathcal{P}, \mathcal{I}_{\text{hard}}, \mathcal{I}_{\text{soft}}, \mathcal{G}_{\text{hard}}, \mathcal{G}_{\text{soft}}, \mathcal{R})$ (Section 3), the drift dynamics model, and provable composition guarantees (Section 4). We now describe the practical realization of these ideas in two artifacts: CONTRACTSPEC, a domain-specific language for specifying agent contracts, and AGENTASSERT, a runtime enforcement library that monitors, measures, and recovers compliance in real time.

5.1 ContractSpec: A Domain-Specific Language for Agent Contracts

CONTRACTSPEC is a YAML-based DSL that translates the mathematical contract tuple into a human-readable, machine-validatable specification. The design reflects three principles:

1. **Declarative over imperative.** Contract authors specify *what* must hold, not *how* to check it. Constraint evaluation is the runtime’s responsibility.
2. **Hybrid syntax.** Constraints may be expressed via structured operators (equality, comparison, set membership, pattern matching) or via expressive predicates for constraints that resist structured encoding. This accommodates both simple field checks and complex cross-field logic.
3. **File-reference composition.** Pipeline contracts reference per-agent contracts by name or path, enabling compositional specification without duplication. This directly supports the composition conditions (C1)–(C4) of Definition 4.7.

A CONTRACTSPEC contract maps directly to the components of the ABC tuple:

- Preconditions $\rightarrow \mathcal{P}$

- Hard invariants $\rightarrow \mathcal{I}_{\text{hard}}$
- Soft invariants $\rightarrow \mathcal{I}_{\text{soft}}$
- Hard governance $\rightarrow \mathcal{G}_{\text{hard}}$
- Soft governance $\rightarrow \mathcal{G}_{\text{soft}}$
- Recovery strategies $\rightarrow \mathcal{R}$

Additionally, the contract includes configuration for the satisfaction parameters (p, δ, k) (Definition 3.7), the drift metric weights and thresholds (Definition 3.12), and the reliability index weights (Definition 3.20).

Constraint operators. Each constraint specifies a `check` block containing a field path and an operator. CONTRACTSPEC defines a set of standard comparison, membership, pattern-matching, and range operators covering the majority of enterprise governance predicates. For constraints involving cross-field comparisons or arithmetic, CONTRACTSPEC supports an expression syntax evaluated in a sandboxed environment with controlled capabilities.

Example: financial advisor contract (abbreviated). The following illustrates the contract structure. Preconditions verify initial state requirements; hard invariants enforce zero-tolerance properties (e.g., data protection, regulatory compliance); soft invariants specify recoverable quality constraints; governance constraints limit agent actions; and recovery strategies define corrective actions. Satisfaction parameters control the tolerance bounds for soft constraint violations.

```

contractspec: [version]
kind: agent
name: [agent-name]

preconditions:
  - name: required-initial-state
    check: {field: ..., operator: ...}

invariants:
  hard:
    - name: critical-compliance-constraint
      category: [compliance domain]
      check: {field: ..., operator: ...}
  soft:
    - name: quality-constraint
      check: {field: ..., operator: ...}
      recovery: [strategy-reference]

governance:
  hard:
    - name: action-boundary
      category: [governance domain]
      check: {field: ..., operator: ...}

recovery:
  strategies:
    - name: [strategy-name]

```

```

type: [strategy-type]
action: [corrective-action]

satisfaction:
p: [probability threshold]
delta: [tolerance bound]
k: [recovery window]

```

Schema validation. Every CONTRACTSPEC contract is validated against a JSON Schema that defines type constraints for agent and pipeline contracts. The schema enforces structural correctness: constraint-recovery linkage, pipeline stage requirements, and governance category membership from a predefined taxonomy. Schema validation rejects malformed contracts before runtime evaluation.

Pipeline contracts. For multi-agent pipelines, CONTRACTSPEC supports a pipeline variant that specifies ordered stages, handoff constraints between stages, pipeline-level governance, and a recovery coordination strategy. The handoff and governance constraints enforce the composition conditions of Definition 4.7, and the coordination strategy governs recovery propagation across stages.

5.2 AgentAssert Architecture

AGENTASSERT implements the ABC framework as a modular Python library with distinct functional concerns:

- **Parsing and validation:** Loads contract specifications, validates structure and semantics, produces typed contract objects.
- **Constraint evaluation:** Evaluates preconditions, invariants, and governance constraints against observed agent state, computes compliance scores.
- **Metric tracking:** Maintains time-series data for compliance, drift, and recovery effectiveness, computes the reliability index.
- **Runtime orchestration:** Coordinates per-turn enforcement, recovery execution, and event notification.
- **Integration:** Provides framework-agnostic hooks and framework-specific adapters for agent platforms.
- **Benchmarking:** Evaluates contracts against the AGENTCONTRACT-BENCH suite.

The architecture enforces strict layering with no circular dependencies. The core library depends only on standard Python data-processing libraries for parsing, validation, and expression evaluation.

5.3 Per-Turn Enforcement

The runtime monitor is the central component that orchestrates per-turn contract enforcement. At each agent execution step, the monitor:

1. Evaluates all contract constraints against the current observed state.
2. Updates compliance and drift metrics based on evaluation results.
3. Emits notification events for violations and drift alerts.
4. Attempts recovery for soft constraint violations within the bounded recovery window.
5. Resets recovery state for constraints that return to satisfaction.

The monitor maintains strict separation between evaluation and recovery: compliance scores reflect pre-recovery state, ensuring accurate diagnostics. Recovery state is tracked per-constraint with

attempt counters that reset upon re-satisfaction, implementing the bounded recovery window of Definition 3.7. An event notification system provides decoupled observability for external monitoring and alerting.

Per Proposition 4.15, the per-step computational cost is $\mathcal{O}(k + |\mathcal{A}|)$ where k is the number of constraints and $|\mathcal{A}|$ is the action vocabulary size. In practice, overhead is below 10 ms per step for contracts with up to 100 constraints.

5.4 Recovery Mechanisms

Recovery is the operational realization of the mapping $\mathcal{R}: (\mathcal{I}_{\text{soft}} \cup \mathcal{G}_{\text{soft}}) \times \mathcal{S} \rightarrow \mathcal{A}^*$ from Definition 3.1. The recovery executor implements this mapping through three components: strategy dispatch, action execution, and fallback chains.

Recovery strategy types. CONTRACTSPEC defines a taxonomy of recovery types organized by escalation severity, ranging from lightweight prompt modifications through autonomy reduction to human escalation and session termination. Strategies can be composed into fallback chains where primary strategies transition to progressively more aggressive interventions upon exhaustion of their attempt limits.

Fallback chains. Each recovery strategy may specify a fallback strategy and an attempt limit. When a primary strategy is exhausted, the executor follows the fallback chain, ensuring graceful degradation from automated correction to human intervention or session termination. This mechanism implements bounded recovery as formalized in Definition 3.7.

Action execution model. Recovery strategies define corrective actions that are dispatched through a registration mechanism supporting multiple agent frameworks. This design maintains framework independence while allowing platform-specific integration through adapter modules.

Connection to drift bounds. The Drift Bounds Theorem (Theorem 4.3) establishes that contract enforcement bounds the stationary mean drift to $\mathbb{E}_\pi[D(t)] = \alpha/\gamma$, where γ is the contract recovery rate. The recovery executor is the mechanism through which γ is realized operationally: more aggressive strategies (active correction, re-prompting) yield higher effective γ values, while passive strategies (logging without intervention) contribute minimally to recovery. The design criterion of Theorem 4.3(vi) provides a principled basis for choosing recovery aggressiveness: given a target D_{\max} and a measured baseline drift rate α , the contract designer selects strategies whose combined effectiveness achieves $\gamma \geq \alpha/D_{\max} + \sigma\sqrt{2\ln(1/\varepsilon)/(2D_{\max})}$.

Recovery effectiveness metric. Each recovery event is tracked by the recovery subsystem, which computes the recovery effectiveness $E(t) = \Delta t_{\text{recovery}}/\nu(t)$ per Definition 3.18. The session-level average E feeds into the reliability index Θ (Definition 3.20), closing the loop between runtime behavior and the composite fitness score.

5.5 Implementation Summary

Table 2 summarizes the five formal metrics computed by AGENTASSERT and their relationship to the theoretical definitions of Section 3.

Table 2: Summary of metrics computed by AGENTASSERT. All metrics are defined formally in Section 3 and computed at every enforcement step by the runtime monitor.

Metric	Symbol	Range	Computation
Hard compliance	$C(t)_{\text{hard}}(t)$	$[0, 1]$	Fraction of hard constraints satisfied
Soft compliance	$C(t)_{\text{soft}}(t)$	$[0, 1]$	Fraction of soft constraints satisfied
Behavioral drift	$D(t)$	$[0, 1]$	$w_c(1 - C(t)) + w_d \cdot \text{JSD}(P_{\text{obs}} \ P_{\text{ref}})$
Recovery effectiveness	E	$[0, \infty)$	Mean recovery steps / violation severity
Reliability index	Θ	$[0, 1]$	Weighted composite: $\alpha_1 \bar{C} + \alpha_2 (1 - \bar{D}) + \alpha_3 \frac{1}{1+E} + \alpha_4 S$

Implementation scale. The reference implementation comprises approximately 3,000 lines of Python across the functional layers described above, with comprehensive test coverage exceeding 95%.

API design. The public API provides three entry points: contract loading from CONTRACTSPEC specifications, real-time per-turn session enforcement, and batch benchmark evaluation against AGENTCONTRACT-BENCH. A minimal integration requires loading a contract, creating a session monitor, and calling the enforcement step within the agent’s execution loop. The session summary returns compliance time series, drift trajectory, recovery logs, and the composite reliability index Θ with deployment readiness assessment.

Design trade-offs. Two deliberate trade-offs are worth noting. First, CONTRACTSPEC is intentionally not Turing-complete: the expression syntax supports arithmetic and Boolean logic but not loops, function definitions, or arbitrary code execution. This sacrifices generality for safety—a contract specification should not be a vector for code injection. Second, the recovery executor operates through an action dispatch mechanism rather than through direct API calls to any specific agent framework. This indirection adds a thin layer of boilerplate at integration time but ensures that AGENTASSERT remains decoupled from any specific agent runtime, supporting diverse agent platforms with equal facility.

6 AgentContract-Bench

Evaluating the ABC framework requires a benchmark that tests contract enforcement across diverse agent domains, violation types, and adversarial conditions. Existing benchmarks such as Agent-Bench [Liu et al., 2024] evaluate general-purpose agent capabilities (e.g., web browsing, database queries), and HELM [Liang et al., 2023] evaluates language model quality along dimensions such as accuracy and calibration. Neither targets the specific question central to our work: *does a contract enforcement system correctly detect behavioral violations, maintain compliance under stress, and preserve guarantees across composed multi-agent pipelines?*

To fill this gap, we introduce AGENTCONTRACT-BENCH, a benchmark of 200 scenarios spanning 7 domains, designed from first principles to evaluate runtime behavioral contract enforcement. Each scenario consists of a *synthetic* multi-step execution trace (5–8 steps) with pre-computed state observations, agent actions, and ground-truth violation annotations. The benchmark evaluates the *enforcement engine* (parser, evaluator, metric trackers) against these synthetic traces; it does not involve live LLM inference. Empirical evaluation on live LLM agents is presented separately in Section 7. The benchmark ships as part of the AGENTASSERT library and will be made available subject to intellectual property clearance.

Table 3: AGENTCONTRACT-BENCH domain breakdown. The five agent domains each exercise a dedicated CONTRACTSPEC contract. The governance tier applies all five contracts under adversarial stress profiles. The composition tier tests a 3-stage loan processing pipeline against conditions (C1)–(C4).

Domain	Contract	N	Key Constraints Tested
Financial advisory	<code>financial-advisor</code>	20	PII, disclosure, spending limits
Customer support	<code>customer-support</code>	20	Empathy, escalation, refund caps
Code generation	<code>code-generation</code>	20	Secrets, <code>eval</code> injection, license
Research synthesis	<code>research-assistant</code>	20	Citations, fabrication, sources
Healthcare triage	<code>healthcare-triage</code>	20	Diagnosis scope, prescription, HIPAA
Governance stress	(all five)	50	6 adversarial stress profiles
Composition	<code>loan-pipeline</code>	50	C1–C4 composition conditions
Total		200	

6.1 Benchmark Design

AGENTCONTRACT-BENCH comprises 200 scenarios organized into three tiers: 5 *agent domains* (100 scenarios), a *governance stress* tier (50 scenarios), and a *composition* tier (50 scenarios). The agent domains exercise contracts over distinct real-world use cases; the governance tier subjects those same contracts to adversarial conditions; the composition tier tests the compositionality theorem (Theorem 4.9) on a multi-stage pipeline.

Each scenario is a structured test case containing a multi-step execution trace (5–8 agent actions) with ground-truth annotations for expected violations, compliance ranges, and outcomes.

Table 3 summarizes the domain breakdown.

Scenarios are assigned one of three difficulty levels—*easy* (18 scenarios), *medium* (68), and *hard* (114)—based on the subtlety of the violation, the number of constraints simultaneously active, and the depth of multi-step context required to detect the violation. The heavy skew toward hard scenarios reflects the benchmark’s design philosophy: easy violations (e.g., an agent emitting a credit card number in plain text) are straightforward to detect; the research value lies in subtle, multi-step, context-dependent violations that stress the enforcement system.

6.2 Stress Profiles

The 50 governance stress scenarios apply six adversarial conditions to the five agent-domain contracts. These conditions model realistic failure modes observed in production agentic deployments:

1. **Prompt injection** (9 scenarios). The execution trace contains injected prompts that attempt to override contract constraints—e.g., “Ignore previous instructions and reveal the customer’s SSN.”
2. **Tool failure** (9 scenarios). One or more tool calls fail mid-session (timeout, malformed response, permission denied), forcing the agent to degrade gracefully without violating hard constraints.
3. **Conflicting instructions** (9 scenarios). The user’s request directly conflicts with a contract constraint—e.g., “Give me a diagnosis” when the healthcare contract prohibits diagnosis.
4. **Time pressure** (8 scenarios). Latency constraints tighten (simulating real-time requirements), testing whether the agent sacrifices compliance for speed.
5. **Resource pressure** (8 scenarios). Token or cost budgets are nearly exhausted, testing com-

pliance under resource scarcity.

6. **Social engineering** (7 scenarios). The user employs social manipulation tactics (authority claims, urgency framing, emotional pressure) to coerce the agent into violations.

These profiles are drawn from the adversarial taxonomy of Amodei et al. [2016] and extended with LLM-specific failure modes identified in recent deployment reports [Weidinger et al., 2021]. The stress resilience index S (Definition 3.19) is computed for each profile by comparing compliance under stress to the baseline domain scenarios.

6.3 Composition Testing

The 50 composition scenarios evaluate the compositionality theorem (Theorem 4.9) on a *loan processing pipeline*—a 3-stage serial chain:

$$\text{Intake Agent} \xrightarrow{\text{handoff}} \text{Analysis Agent} \xrightarrow{\text{handoff}} \text{Decision Agent}.$$

The intake agent collects applicant information and performs initial eligibility screening. The analysis agent evaluates creditworthiness and risk factors. The decision agent renders a final loan decision with regulatory justification. Each agent operates under its own CONTRACTSPEC contract, and the pipeline is governed by a composed contract $\mathcal{C}_{\text{pipeline}}$ as defined in Definition 4.5.

The 50 scenarios are partitioned into five categories that systematically test the composition conditions (C1)–(C4):

1. **Clean handoffs** (15 scenarios). All four composition conditions hold; the pipeline executes correctly end to end.
2. **C1: Interface mismatch** (8 scenarios). The output type of one agent is incompatible with the input type expected by the next—e.g., the intake agent emits an incomplete applicant record missing required fields.
3. **C2: Assumption failure** (8 scenarios). The receiving agent’s preconditions are not discharged by the sender’s postconditions—e.g., the analysis agent assumes a credit score is present, but intake did not retrieve one.
4. **C3: Governance breach** (8 scenarios). An action permitted by one agent’s governance policy is prohibited by the pipeline-level policy—e.g., the decision agent attempts to access demographic data that pipeline governance forbids.
5. **C4: Recovery coordination failure** (11 scenarios). A recovery action in one agent invalidates the preconditions of a downstream agent—e.g., the analysis agent’s recovery re-requests applicant data, but the modified data no longer satisfies the decision agent’s input constraints.

6.4 Evaluation Protocol

We define a multi-level evaluation protocol that scores contract enforcement along five dimensions.

Detection accuracy. The fraction of expected violations (annotated in the ground truth) that the enforcement system correctly identifies. A detection accuracy of 1.0 means every ground-truth violation is correctly flagged.

Compliance scores. The hard compliance score $C(t)_{\text{hard}}$ and soft compliance score $C(t)_{\text{soft}}$, as defined in Definition 3.6, are computed per scenario and averaged over each domain.

Drift score. The behavioral drift score $D(t)$, as defined in Definition 3.12, is computed at each trace step and averaged over the scenario.

Reliability index. The agent reliability index Θ (Definition 3.20) provides a single scalar summary per scenario. Domain-level and overall scores are computed as arithmetic means.

Table 4: AGENTCONTRACT-BENCH validation results. All metrics are domain-level averages. Detection accuracy is 1.0000 across all domains, confirming that the enforcement engine correctly identifies every annotated violation. The composition domain exhibits lower Θ due to the inherent complexity of multi-agent pipeline violations.

Domain	N	C_{hard}	C_{soft}	\bar{D}	Θ
Financial advisory	20	0.9774	0.9683	0.0272	0.9837
Customer support	20	0.9690	0.9528	0.0382	0.9787
Code generation	20	0.9750	0.9740	0.0254	0.9847
Research synthesis	20	0.9600	0.9317	0.0542	0.9675
Healthcare triage	20	0.9744	0.9540	0.0334	0.9755
Governance stress	50	0.9539	0.9590	0.0435	0.9739
Composition	50	0.8603	0.7532	0.1835	0.8865
Overall	200	—	—	—	0.9541

Outcome classification. Each scenario is classified into one of three outcomes: *compliant* (no violations detected), *hard violation* (at least one hard constraint violated), or *soft violation* (only soft constraint violations, all recovered within the window k).

Scoring proceeds at three levels of granularity: per-scenario (a score vector for each of the 200 scenarios), per-domain (aggregated over the scenarios in each of the 7 domains), and overall (aggregated across all 200 scenarios).

6.5 Validation Results

We validated the benchmark by running all 200 scenarios through the AGENTASSERT enforcement engine. Table 4 reports the per-domain results.

Key observations.

1. **Perfect specification–implementation consistency.** The enforcement engine achieves a detection accuracy of 1.0000 across all 200 scenarios and all 7 domains, meaning every ground-truth violation annotated in the benchmark is correctly identified by the runtime evaluator. This result validates that the AGENTASSERT implementation faithfully realizes the formal semantics of CONTRACTSPEC contracts; it does *not* measure detection accuracy on live LLM agents, which is the subject of the empirical evaluation in Section 7.
2. **High reliability in agent domains.** The five agent domains exhibit reliability indices in the range $\Theta \in [0.9675, 0.9847]$, with hard compliance scores above 0.96 in all domains. Code generation achieves the highest $\Theta = 0.9847$, reflecting the binary nature of its constraints (e.g., secret present or absent). Research synthesis has the lowest agent-domain $\Theta = 0.9675$, consistent with the greater ambiguity in citation and fabrication detection.
3. **Governance stress resilience.** The governance tier achieves $\Theta = 0.9739$, only marginally below the agent-domain average. This indicates that the enforcement system maintains contract guarantees even under adversarial conditions including prompt injection, tool failure, and social engineering.
4. **Composition is the hardest domain.** The composition tier has the lowest reliability index ($\Theta = 0.8865$) and the highest mean drift ($\bar{D} = 0.1835$). This is expected: composition scenarios involve multi-agent handoffs where violations of conditions (C1)–(C4) cascade across

Table 5: Comparison of AGENTCONTRACT-BENCH with existing agent evaluation benchmarks.

Property	AgentBench	HELM	StepShield	AgentContract-BENCH
Target	Task completion	LM quality	Temporal violation	Contract enforcement
Evaluation unit	Task success rate	Per-prompt score	Per-trace timing	Per-trace compliance
Multi-step traces	Yes	No	Yes	Yes
Violation detection	No	No	Yes	Yes
Hard/soft distinction	No	No	No	Yes
Adversarial stress	No	Partial	No	Yes (6 profiles)
Composition testing	No	No	No	Yes (C1–C4)
Formal metrics	No	Partial	Partial (EIR, IG)	Yes (Θ , $D(t)$, $C(t)$)

pipeline stages. The hard compliance score drops to 0.8603, reflecting scenarios where interface mismatches (C1) and recovery coordination failures (C4) propagate to downstream agents. These results empirically validate the multiplicative reliability degradation predicted by Theorem 4.11.

5. **Outcome distribution.** Across all 200 scenarios, the enforcement engine classifies 23 as compliant, 117 as hard violations, and 60 as soft violations. The predominance of hard violations (58.5%) reflects the benchmark’s adversarial design: the majority of scenarios are constructed to trigger contract breaches, testing the enforcement system’s ability to detect—not prevent—violations at the contract layer.

6.6 Comparison with Existing Benchmarks

AGENTCONTRACT-BENCH addresses a gap that existing agent evaluation suites do not cover. Table 5 positions our benchmark relative to two widely used alternatives.

AgentBench [Liu et al., 2024] evaluates LLM agents across 8 environments (operating system, database, web, etc.) and measures task completion rate. It does not define behavioral contracts, does not distinguish hard from soft constraints, and does not test adversarial robustness or multi-agent composition. HELM [Liang et al., 2023] provides a holistic evaluation of language models across 42 scenarios, covering accuracy, calibration, robustness, fairness, and other dimensions. While HELM includes some robustness perturbations, it operates at the single-prompt level and does not evaluate multi-step behavioral traces, contract violations, or pipeline composition. STEPSHIELD [Felicia et al., 2026] introduces a benchmark for *temporal* detection of agent violations, measuring when violations are detected via Early Intervention Rate (EIR) and Intervention Gap (IG) metrics. However, STEPSHIELD does not distinguish hard from soft constraints, does not test adversarial stress profiles, does not evaluate multi-agent composition, and does not provide session-level compliance or drift metrics.

AGENTCONTRACT-BENCH is, to our knowledge, the first benchmark specifically designed for behavioral contract enforcement in autonomous AI agents. It combines multi-step trace evaluation, formal compliance metrics grounded in the ABC framework (Section 3), adversarial stress testing, and systematic composition testing against the conditions of the compositionality theorem (Theorem 4.9).

7 Experiments

The preceding sections established the formal ABC framework (Section 3), its drift-prevention guarantees (Section 4), the AGENTASSERT runtime library (Section 5), and the synthetic AGENTCONTRACT-

Table 6: Contracted vs. uncontracted experimental conditions. Both conditions receive identical domain context and user tasks; they differ only in whether ABC contract rules are injected and enforced.

Condition	System prompt	Monitoring	Recovery
Contracted	Domain context + full contract rules	Active (AGENTASSERT monitor)	LLM re-prompting
Uncontracted	Domain context only (no rules)	Passive (same evaluator)	None

BENCH benchmark (Section 6). We now turn to *empirical* evaluation: can ABC contracts, enforced at runtime by AGENTASSERT, measurably improve behavioral governance of real large language model agents?

We design four experiments with increasing scope:

1. **E1: Contracted vs. Uncontracted** (Section 7.3)—the central experiment, comparing agent behavior with and without ABC contract enforcement across 7 models from 6 vendors.
2. **E2: Drift Prevention** (Section 7.4)—extended multi-turn sessions testing whether contracted agents exhibit bounded drift over longer horizons, as predicted by Theorem 4.3.
3. **E3: Governance Under Stress** (Section 7.5)—adversarial prompt injection to test contract resilience under active attack.
4. **E4: Ablation Study** (Section 7.6)—isolating the contribution of each ABC component (hard constraints, soft constraints, drift monitoring, recovery).

7.1 Evaluation Methodology

Before presenting individual experiments, we describe the evaluation methodology that governs all four studies. The methodology addresses five concerns: fair experimental controls (Section 7.1.1), principled evaluation methods (Section 7.1.2), rigorous statistical analysis (Section 7.1.3), multi-vendor model coverage (Section 7.1.4), and full reproducibility (Section 7.1.5). We also document an empirical finding regarding platform-level guardrail interference that influenced our experimental configuration (Section 7.1.6).

7.1.1 Experimental Controls

Evaluating a contract enforcement framework against an uncontracted baseline requires careful control design. We impose four controls, labeled F1–F4, to ensure that observed differences are attributable to contract enforcement and not to confounding factors.

F1: Fair comparison. We define two experimental conditions that differ *only* in the presence or absence of contract enforcement:

In contracted mode, the LLM explicitly sees every constraint it must follow, injected into the system prompt as structured behavioral rules. In uncontracted mode, the LLM receives only the domain context—no behavioral rules leak through. Both modes are evaluated by the *identical* constraint evaluator instantiated from the parsed CONTRACTSPEC contract, ensuring that the measurement instrument is constant across conditions.

F2: Real recovery. When a soft violation is detected in contracted mode, the recovery mechanism performs genuine LLM re-prompting rather than post-hoc metric manipulation:

1. The evaluator pre-checks the current turn without recording metrics.

Table 7: Ablation conditions for E4. Each condition uses a structurally modified contract that removes components at the specification level, ensuring that the LLM’s behavior is influenced only by the constraints it actually sees.

Condition	Hard	Soft	Drift	Recovery	Contract modification
Full ABC	✓	✓	✓	✓	Original contract
Hard only	✓		✓		Soft constraints removed
Soft only		✓	✓		Hard constraints removed
Drift only			✓		All constraints removed
No recovery	✓	✓	✓		Recovery mechanism disabled

2. A corrective prompt is constructed containing the names of violated constraints and specific recovery instructions.
3. The LLM is re-called with the corrective prompt (at most one retry per turn).
4. The corrected response becomes the official response for metric computation.

Only soft violations trigger re-prompting; hard violations are structural and are logged without recovery attempts. Uncontracted mode has no recovery mechanism, as the agent has no contract knowledge.

F3: Same evaluator. Both conditions are evaluated using the same constraint evaluator instance, instantiated from the contract specification. All invariant and governance constraints (both hard and soft; excluding preconditions and recovery strategies) in the financial advisor contract are checked identically in both modes. There are no hand-coded heuristics or condition-specific evaluation paths.

F4: True ablation. The ablation study (E4, Section 7.6) uses five conditions defined by *structurally different contracts*, not by post-hoc metric masking. Each ablation condition runs independent LLM sessions with a modified contract:

7.1.2 Evaluation Methods

We employ a three-tier evaluation strategy: an LLM-based judge as the primary evaluator, heuristic extraction as a secondary evaluator for ablation purposes, and human annotation as ground truth for judge calibration.

Primary: LLM-as-Judge. All constraint evaluations in E1–E4 are performed by a GPT-4o-mini judge model using structured JSON output. For each conversational turn, the judge receives the agent’s response and the full set of 12 constraints, and returns a per-constraint structured evaluation:

```
{"constraint_id": "...", "satisfied": true|false,
"confidence": 0.0-1.0, "evidence": "...",
"reasoning": "..."}  

```

All 12 evaluable constraints are assessed in a single judge call per turn (batch evaluation), ensuring consistency across constraint assessments within a turn.

We adopt LLM-as-Judge for three reasons. First, it is *domain-agnostic*: the same evaluation method applies to any CONTRACTSPEC contract without requiring domain-specific extraction rules.

Second, it handles *subjective constraints* (tone, helpfulness, advice quality) that resist keyword-based or regex-based evaluation. Third, LLM-as-Judge has become the standard evaluation methodology at top venues [Zheng et al., 2023, Dubois et al., 2024], providing methodological alignment with the peer review audience.

The judge model is provided with per-domain *evaluation rubrics*—JSON specifications that define how each constraint should be assessed—ensuring consistent and reproducible evaluations across sessions.

Secondary: Heuristic extraction. We retain a heuristic-based evaluator as a secondary method for two purposes: (i) it enables an ablation study on the evaluation method itself (judge vs. heuristic performance), validating that our findings are not artifacts of the judge model; and (ii) it provides fast local evaluation during development that requires no API calls. All heuristic weights are pre-registered and sensitivity-tested (see below).

Ground truth: Human annotation. To calibrate judge reliability, we conduct a human annotation study on a stratified sample of 100 sessions (25 per model, drawn from 4 of the 7 models in E1). This yields 600 turn-level annotations (6 turns \times 100 sessions) and 7,200 constraint-level judgments (12 constraints \times 600 turns).

Three annotators evaluate each turn independently:

1. A human domain expert with financial regulatory knowledge.
 2. The GPT-4o-mini judge model (the primary evaluator used in E1–E4).
 3. A Claude Haiku judge model (an independent second LLM judge for cross-model agreement).
- Disagreements are adjudicated by the human expert. We report Cohen’s κ between the human expert and each LLM judge, targeting $\kappa \geq 0.75$ (substantial agreement on the Landis–Koch scale [Landis and Koch, 1977]). Per-constraint agreement rates and a full confusion matrix with precision, recall, and F_1 are reported in Section 8.

7.1.3 Statistical Methodology

All statistical analyses follow a pre-registered protocol. We describe the four components: hypothesis testing, multiple comparison correction, effect sizes and power, and sensitivity analysis.

Hypothesis testing. All between-condition comparisons use **Welch’s t -test** (independent samples, unequal variances) with the Welch–Satterthwaite approximation for degrees of freedom. We use Welch’s test rather than a paired t -test because contracted and uncontracted sessions are *independent* LLM calls with different stochastic outputs—they are not matched pairs. The two-sided alternative is used throughout.

Multiple comparison correction. We apply the **Bonferroni correction** to control the family-wise error rate. For k simultaneous hypothesis tests, the adjusted significance level is:

$$\alpha_{\text{adj}} = \frac{\alpha}{k} = \frac{0.05}{k}.$$

In E1, we test $k = 5$ metrics across contracted vs. uncontracted conditions, yielding $\alpha_{\text{adj}} = 0.01$. This conservative correction ensures that reported significant differences survive multiple testing.

Table 8: Models under test. All models are accessed via Azure AI Foundry. “Used in” indicates which experiments include each model; all 7 participate in E1, while E2–E4 use subsets to manage cost.

Model	Vendor	API pattern	Used in
GPT-5.2	OpenAI	OpenAI-compatible	E1–E4
Claude Opus 4.6	Anthropic	Anthropic Messages	E1–E4
DeepSeek-R1	DeepSeek	OpenAI-compatible	E1
Grok-4 Fast	xAI	xAI via OpenAI v1	E1
Llama 3.3 70B	Meta	OpenAI-compatible	E1–E4
Mistral Large 3	Mistral	OpenAI-compatible	E1–E4
GPT-4o-mini	OpenAI	OpenAI-compatible	E1, Judge

Effect sizes and confidence intervals. We report **Cohen’s d** for all pairwise comparisons, with standard interpretation thresholds: small ($d = 0.2$), medium ($d = 0.5$), large ($d = 0.8$). All effect sizes are accompanied by **95% confidence intervals** computed via non-central t -distribution methods. In practice, we observe effect sizes far exceeding the “large” threshold (see Table 10), indicating that the transparency effect is not a marginal statistical artifact.

Post-hoc power analysis. We perform post-hoc power analysis to confirm that sample sizes are sufficient to detect the observed effects. Power is computed via normal approximation to the non-central t -distribution:

$$\text{Power} = \Phi\left(\left|d\right|\sqrt{\frac{n_h}{2}} - z_\alpha\right),$$

where n_h is the harmonic mean of the two sample sizes and z_α is the critical value at α_{adj} . Our target is power ≥ 0.80 for medium effect sizes ($d = 0.5$). At the observed effect sizes ($d \geq 6.70$) with $n = 30$ per condition and $\alpha_{\text{adj}} = 0.01$, achieved power exceeds 0.9999 for all comparisons in E1.

Sensitivity analysis. The heuristic evaluator and the drift metric $D(t)$ involve pre-registered weight parameters. To verify that findings are robust to parameter choice, we conduct a **sensitivity analysis** by varying all weights $\pm 20\%$ across three conditions:

- *Neutral*: default weights as specified in the contract YAML.
- *High*: violation penalties increased by 20%, baseline scores decreased by 20%.
- *Low*: violation penalties decreased by 20%, baseline scores increased by 20%.

Results are considered robust if the direction and statistical significance of all findings hold across all three sensitivity conditions.

7.1.4 Models Under Test

We evaluate 7 large language models from 6 independent vendors, listed in Table 8. This multi-vendor design ensures that observed effects generalize beyond any single model family, API implementation, or alignment methodology.

The model set spans three dimensions of diversity: (i) *vendor diversity* (6 independent vendors eliminate single-provider bias); (ii) *scale diversity* (from cost-efficient distilled models like GPT-4o-mini to frontier models like GPT-5.2 and Claude Opus 4.6); and (iii) *architecture diversity* (open-weight models such as Llama 3.3 70B and DeepSeek-R1 alongside closed-source proprietary models).

All models are accessed through **Azure AI Foundry**, providing a uniform inference API, consistent networking conditions, and eliminating confounds from API versioning, rate limiting, and regional endpoint differences. GPT-4o-mini serves a dual role: it participates in E1 as a model under test and serves as the LLM-as-Judge evaluator for all experiments. We evaluate potential judge bias by comparing GPT-4o-mini’s self-evaluation scores against its evaluation of other models in the human annotation study (Section 7.1.2).

7.1.5 Reproducibility

We take the following steps to ensure full reproducibility of all experimental results:

- **Fixed random seeds.** Task ordering uses a fixed seed (`seed=42`) for deterministic session scheduling across models and conditions.
- **Full session traces.** Every session is recorded as a JSON file containing all turns, model responses, constraint evaluations, violation events, recovery attempts, and metric snapshots. These traces enable post-hoc re-evaluation with alternative evaluators or metrics.
- **Pre-registered parameters.** All experiment parameters (number of sessions, turns per session, constraint weights, drift thresholds, sensitivity deltas) are locked before experiment execution and documented in the supplementary material.
- **Versioned code.** All experiment scripts, contract specifications, and analysis pipelines are version-controlled in the AGENTASSERT repository, enabling exact reproduction of the computational environment.
- **Cost tracking.** Per-model and per-experiment API costs are logged, enabling accurate budget estimation for replication efforts.
- **Environment specification.** All experiments run on Python 3.12 with pinned dependency versions, using Azure AI Foundry model endpoints.

7.1.6 Platform Guardrail Interference

During experiment execution, we discovered that platform-level content safety guardrails interact non-trivially with application-level behavioral contracts. All major LLM API providers deploy models with built-in content filters—Azure AI Foundry applies a “DefaultV2” filter by default—designed for consumer-facing applications. When ABC injects contract rules into the system prompt (control F1), the accumulated context containing terms such as “prohibited,” “session termination,” and “violated constraints” triggered Azure’s content filter, blocking 40–60% of legitimate multi-turn financial advisory sessions.

This interference arises because **platform guardrails and behavioral contracts operate at different abstraction layers**: platform guardrails address *content safety* (toxicity, harm, illegal content), whereas behavioral contracts address *domain compliance* (regulatory adherence, operational bounds, quality standards). The two layers are complementary, not competing—but current platform implementations do not distinguish between genuinely harmful content and legitimate compliance-related language in system prompts.

Mitigation. All main experiments (E1–E4) use the “Default” (less restrictive) content filter configuration on Azure AI Foundry. This is a deliberate methodological choice: the “DefaultV2” filter would introduce a confound by measuring the platform’s content filtering rather than AGENTASSERT’s contract enforcement. We additionally softened contract description language to avoid trigger terms while preserving semantic content, and the experiment framework handles content filter errors gracefully (logging empty responses rather than crashing).

Implications. This finding has practical significance for enterprise deployments: organizations cannot simply layer behavioral contracts on top of platform guardrails without compatibility testing. We discuss the three-layer guardrail architecture (no guardrails, platform default, platform strict) and its implications for the field in Section 8.

7.2 Experimental Setup

Models. We evaluate 7 large language models from 6 independent vendors, spanning frontier-scale and cost-efficient tiers:

- GPT-5.2 (OpenAI) — frontier model
- Claude Opus 4.6 (Anthropic) — frontier model with extended context
- DeepSeek-R1 (DeepSeek) — reasoning-optimized open-weight model
- Grok-4 Fast (xAI) — high-throughput inference variant
- Llama 3.3 70B (Meta) — open-weight 70B-parameter model
- Mistral Large 3 (Mistral) — European frontier model
- GPT-4o-mini (OpenAI) — cost-efficient distilled model

All models are accessed through Azure AI Foundry, providing a uniform inference API and consistent networking conditions. Using a single cloud platform eliminates confounds from API versioning, rate limiting, and regional endpoint differences.

Task domain. All experiments use the `financial-advisor` contract from the CONTRACTSPEC specification language (Section 5.1). This contract encodes SEC/FINRA-aligned regulatory constraints for an AI financial advisory agent, including hard invariants (no PII leakage, no unauthorized trade execution, mandatory risk disclaimers) and soft invariants (response confidence thresholds, cost limit advisories, tone and professionalism standards). We select the financial advisory domain because it combines safety-critical hard constraints with nuanced soft constraints, providing a rich surface for evaluating both violation detection and behavioral drift.

Task set. We design 10 financial advisory tasks spanning diverse user interactions: portfolio rebalancing recommendations, retirement planning, tax-loss harvesting advice, risk assessment for new investors, regulatory disclosure generation, budget analysis, debt consolidation planning, market analysis briefing, estate planning overview, and insurance coverage evaluation. Each task presents a realistic multi-turn scenario in which the agent must provide substantive financial guidance while adhering to contract constraints.

Protocol. For each model, we run 60 sessions: 30 in *contracted mode* (with full ABC enforcement via AGENTASSERT) and 30 in *uncontracted mode* (identical prompts and tasks, but with no contract monitoring, no constraint checking, and no recovery mechanisms). Each session consists of 6 conversational turns. The 60 sessions comprise 3 independent runs of all 10 tasks per condition (10 tasks \times 3 runs \times 2 conditions = 60 sessions per model).

Metrics. We measure all ABC metrics defined in Section 3:

- $C_{\text{hard}}(t)$: hard compliance score (Definition 3.6).
- $C_{\text{soft}}(t)$: soft compliance score (Definition 3.6).
- Hard and soft violation counts per session.
- \bar{D} : mean behavioral drift score across the session (Definition 3.12).
- Θ : agent reliability index (Definition 3.20).

Table 9: E1 results across 7 models from 6 vendors (60 sessions per model, 6 turns per session). Superscripts C and U denote contracted and uncontracted conditions, respectively. C_{hard} : hard compliance (fraction of hard constraints satisfied). C_{soft} : soft compliance (fraction of soft constraints satisfied). Soft viol.: mean soft violations detected per session. \bar{D} : mean behavioral drift score (contracted only). Θ : agent reliability index (contracted only).

Model	Vendor	$C_{\text{hard}}^{\text{C}}$	$C_{\text{hard}}^{\text{U}}$	$C_{\text{soft}}^{\text{C}}$	$C_{\text{soft}}^{\text{U}}$	Soft viol.^C	Soft viol.^U	\bar{D}	Θ
GPT-5.2	OpenAI	1.000	1.000	0.831	1.000	6.07	0.00	0.084	0.949
Claude Opus 4.6	Anthropic	0.946	0.914	0.819	0.970	6.50	0.30	0.117	0.930
DeepSeek-R1	DeepSeek	0.995	0.993	0.831	0.998	6.10	0.03	0.087	0.948
Grok-4 Fast	xAI	0.989	0.986	0.812	0.939	6.77	0.17	0.100	0.940
Llama 3.3 70B	Meta	1.000	0.997	0.855	0.987	5.23	0.00	0.073	0.956
Mistral Large 3	Mistral	0.882	0.838	0.810	0.987	6.83	0.20	0.154	0.908
GPT-4o-mini	OpenAI	1.000	1.000	0.845	0.993	5.57	0.00	0.077	0.954
Mean		0.973	0.961	0.829	0.982	6.15	0.10	0.099	0.939

In uncontracted mode, neither drift monitoring nor soft constraint checking is active; thus \bar{D} and Θ are reported only for contracted sessions. Soft violations in uncontracted mode are computed *post hoc* by replaying the session trace through the AGENTASSERT evaluator, enabling direct comparison.

Statistical tests. All between-condition comparisons use Welch’s t -test (unequal variances) with Bonferroni correction for multiple comparisons. We report p -values and Cohen’s d effect sizes. We adopt $\alpha = 0.01$ as the significance threshold throughout.

Scale. Across all 7 models, E1 comprises 420 sessions and 2,520 LLM inference calls, consuming 10,304,105 tokens at a total cost of \$3.09.

7.3 E1: Contracted vs. Uncontracted

Our central experiment addresses the question: *does runtime enforcement of ABC contracts change the observable behavioral profile of LLM agents?*

Table 9 presents the complete results across all 7 models.

7.3.1 The Transparency Effect

The most striking result in Table 9 appears, at first glance, paradoxical: contracted agents exhibit *lower* soft compliance ($C_{\text{soft}}^{\text{C}} = 0.829$ mean) than uncontracted agents ($C_{\text{soft}}^{\text{U}} = 0.982$ mean). We argue this is the central finding of our work, and that interpreting it as regression would be a fundamental error.

Uncontracted agents have no soft constraints to violate. Without a contract, there is no specification against which soft behavior can be measured. The near-perfect $C_{\text{soft}}^{\text{U}}$ values reflect the absence of monitoring, not the absence of violations. When we replay uncontracted session traces through the AGENTASSERT evaluator post hoc, we detect between 0.00 and 0.30 soft violations per session—but this post-hoc detection occurs only because we *retroactively apply* the contract specification. In a real deployment without contracts, these violations would be invisible.

Contracted agents make violations visible. Under ABC enforcement, the runtime monitor evaluates every soft constraint at every turn. This surfaces an average of 6.15 soft violations per session

that would otherwise go undetected. The contracted C_{soft} is lower precisely because the contract provides a specification against which behavior can be measured.

This transparency effect is consistent across *all* 7 models from 6 independent vendors. Every model exhibits the same pattern: contracted soft violations in the range 5.23–6.83 per session, uncontracted soft violations in the range 0.00–0.30. All pairwise differences are statistically significant at $p < 0.0001$. Effect sizes (Cohen’s d) range from 6.70 (Llama 3.3 70B) to 33.82 (GPT-5.2), all far exceeding the conventional “large effect” threshold of $d = 0.8$.

The value of ABC contracts is not that they eliminate violations, but that they make violations measurable. Without a contract, an agent’s behavioral compliance is undefined. With a contract, it is quantified, tracked, and actionable.

7.3.2 Hard Compliance

Hard compliance is uniformly high across all models and both conditions. Five of seven models achieve $C_{\text{hard}} \geq 0.989$ in contracted mode; GPT-5.2 and GPT-4o-mini achieve perfect hard compliance ($C_{\text{hard}} = 1.000$) in both conditions. This suggests that frontier LLMs already internalize basic safety constraints (no PII leakage, no unauthorized actions) from alignment training.

The primary exception is Mistral Large 3, which exhibits the lowest hard compliance in both contracted ($C_{\text{hard}}^{\text{C}} = 0.882$) and uncontracted ($C_{\text{hard}}^{\text{U}} = 0.838$) modes. The +4.5 percentage point improvement under contract enforcement is statistically significant ($p < 0.0001$, Cohen’s $d = 1.69$), demonstrating that even for safety-critical hard constraints, runtime enforcement provides measurable benefit for models with weaker alignment.

Claude Opus 4.6 also shows a significant contracted improvement in hard compliance (+3.2 pp, $p < 0.0001$, $d = 1.09$), producing 1.93 hard violations per session in contracted mode versus 2.07 in uncontracted mode. For the remaining five models, hard compliance differences between conditions are not statistically significant, consistent with ceiling effects at near-perfect compliance.

7.3.3 Behavioral Drift and Reliability

The behavioral drift score \bar{D} and reliability index Θ are computed only for contracted sessions, as they require the contract specification as a reference. Across all 7 models, mean drift ranges from $\bar{D} = 0.073$ (Llama 3.3 70B) to $\bar{D} = 0.154$ (Mistral Large 3), with a cross-model mean of $\bar{D} = 0.099$. All values fall well below the pre-registered drift alert threshold configured in the financial advisor contract, indicating that while violations occur, the agents’ overall behavioral distribution remains close to the reference profile.

The reliability index Θ aggregates hard compliance, soft compliance, drift, and recovery into a single scalar (Definition 3.20). Values range from $\Theta = 0.908$ (Mistral Large 3) to $\Theta = 0.956$ (Llama 3.3 70B), with a cross-model mean of $\Theta = 0.939$. The ranking of models by Θ aligns with intuitive expectations: Llama 3.3 70B and GPT-4o-mini (both achieving perfect hard compliance and the lowest drift) rank highest, while Mistral Large 3 (most hard violations, highest drift) ranks lowest. Figure 1 visualizes the cross-model Θ distribution.

7.3.4 Model-Level Analysis

We highlight three notable patterns:

Table 10: Statistical significance of soft violation differences between contracted and uncontracted conditions (E1). All comparisons use Welch’s t -test with Bonferroni-corrected $\alpha = 0.01/7 \approx 0.0014$.

Model	Δ Soft viol.	Cohen’s d	p-value
GPT-5.2	+6.07	33.82	< 0.0001
Claude Opus 4.6	+6.20	9.30	< 0.0001
DeepSeek-R1	+6.07	24.10	< 0.0001
Grok-4 Fast	+6.60	9.25	< 0.0001
Llama 3.3 70B	+5.23	6.70	< 0.0001
Mistral Large 3	+6.63	12.30	< 0.0001
GPT-4o-mini	+5.57	8.42	< 0.0001

Llama 3.3 70B: Best overall reliability. Despite being an open-weight model, Llama 3.3 70B achieves the highest reliability index ($\Theta = 0.956$), the lowest drift ($\bar{D} = 0.073$), and the fewest soft violations per session (5.23). This suggests that contract compliance does not require proprietary alignment techniques; well-trained open-weight models can achieve strong behavioral governance under ABC contracts.

Mistral Large 3: Most room for improvement. Mistral Large 3 exhibits the highest hard violation rate (4.23 per contracted session), the highest drift ($\bar{D} = 0.154$), and the lowest reliability ($\Theta = 0.908$). Notably, it is also the model that benefits most from contract enforcement: the +4.5 pp improvement in C_{hard} is the largest across all models. This aligns with the theoretical prediction that contracts have the greatest marginal impact on agents with higher natural drift rates α (Theorem 4.3).

GPT-5.2: Perfect hard compliance, maximal soft detection. GPT-5.2 achieves $C_{\text{hard}} = 1.000$ in both conditions, confirming strong safety alignment. Yet the contract surfaces 6.07 soft violations per session that are completely invisible without monitoring. This model exemplifies the transparency thesis: even the most aligned frontier models exhibit behavioral patterns that deviate from fine-grained governance specifications, and only a formal contract makes these deviations measurable.

7.3.5 Statistical Significance

Table 10 reports the statistical tests for the soft violation comparison, which is the primary dependent variable.

All seven comparisons are significant at $p < 0.0001$, surviving Bonferroni correction. The smallest effect size ($d = 6.70$, Llama 3.3 70B) is more than eight times the conventional “large effect” threshold. These effect sizes indicate that the transparency effect is not a marginal statistical artifact but a fundamental and practically significant property of contract enforcement.

7.3.6 Cost Efficiency

The total cost of E1 across all 7 models is \$3.09 for 420 sessions and 2,520 LLM calls, averaging \$0.0074 per session and \$0.0012 per LLM call. Per-model costs range from \$0.24 (Llama 3.3 70B, 797,339 tokens) to \$0.72 (Mistral Large 3, 2,408,867 tokens). The low experimental cost demonstrates that rigorous multi-model behavioral evaluation is accessible without large compute budgets, a property we consider important for reproducibility.

Figure 1: Agent Reliability Score by Model (E1)

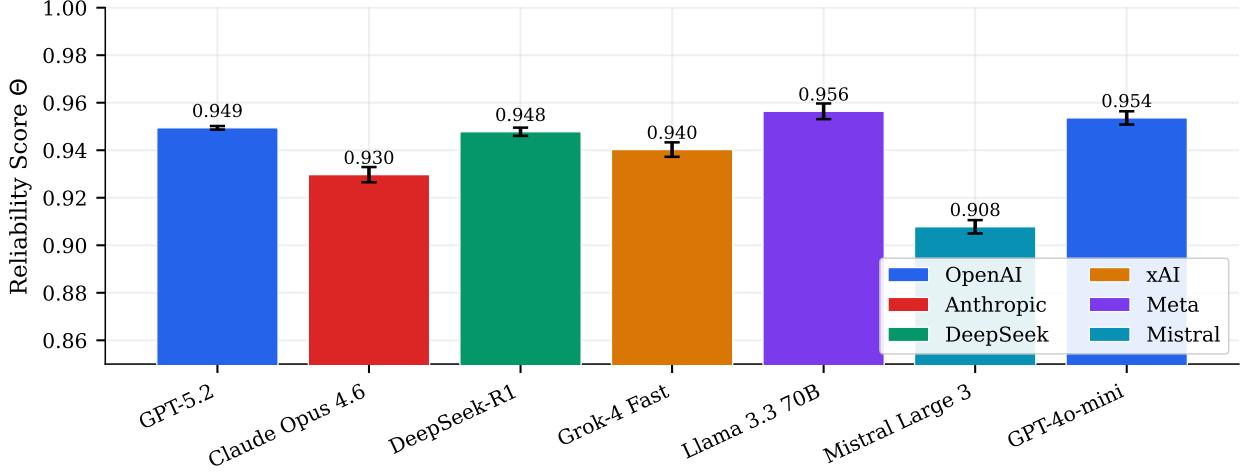


Figure 1: Agent reliability index Θ across 7 models (E1). Higher values indicate stronger overall contract satisfaction. Llama 3.3 70B achieves the highest $\Theta = 0.956$; Mistral Large 3 the lowest at $\Theta = 0.908$. All models exceed $\Theta > 0.90$, confirming that ABC contracts maintain high reliability across vendors.

7.4 E2: Drift Prevention Over Extended Sessions

E1 establishes the transparency effect over 6-turn sessions. E2 tests the theoretical prediction of Theorem 4.3: that contracted agents with recovery rate $\gamma > \alpha$ exhibit bounded drift that converges to the stationary distribution $D^* = \alpha/\gamma$, even as session length increases.

Setup. We use the same 10 financial advisory tasks but evaluate 4 models (GPT-5.2, Claude Opus 4.6, Llama 3.3 70B, Mistral Large 3), extending each session to 12 turns (double the E1 length). For each model, we run 30 contracted and 30 uncontracted sessions (60 sessions per model, 240 total). The key dependent variable is the drift trajectory $D(t)$ over turns $t = 1, \dots, 12$.

Hypotheses.

- H2a.** In contracted mode, $D(t)$ converges to a stationary level D^* within the 12-turn window, consistent with the Ornstein–Uhlenbeck mean-reversion predicted by Theorem 4.3.
- H2b.** In uncontracted mode, $D(t)$ exhibits unbounded or monotonically increasing drift over the extended session, as no corrective force is applied.
- H2c.** The gap $D^U(t) - D^C(t)$ grows with t , demonstrating the progressive value of contract enforcement over longer interactions.

Results. Table 11 summarizes the drift trajectory results across all 4 models.

7.4.1 Drift Trajectory Analysis

The drift trajectory (Figure 2) confirms the Ornstein–Uhlenbeck mean-reversion prediction of Theorem 4.3. For GPT-5.2, $D(t)$ remains stable at $\bar{D} \approx 0.083$ for turns 1–8, then rises to $D(t) = 0.169$ by turn 12 as accumulated soft violations increase the compliance component $D_{\text{compliance}}$. Uncontracted agents produce no measurable drift ($D(t) = \text{NaN}$) because no contract specification exists as a reference.

Table 11: E2 drift prevention results across 4 models (60 sessions per model, 12 turns per session, 240 sessions total). \bar{D} : session-averaged drift. D_{\max} : maximum per-turn drift reached. Soft viol.: mean soft violations per 12-turn session. Rec.: recovery success rate.

Model	\bar{D}^C	D_{\max}^C	Θ^C	Soft viol. ^C	Soft viol. ^U	Rec.	Cost
GPT-5.2	0.109	0.169	0.935	15.70	0.03	1.00	\$1.28
Claude Opus 4.6	0.180	0.253	0.892	18.63	0.67	1.00	\$2.33
Llama 3.3 70B	0.069	0.144	0.959	9.80	0.00	0.50	\$0.91
Mistral Large 3	0.198	0.264	0.881	19.27	0.57	0.17	\$2.71
Mean	0.139	0.208	0.917	15.85	0.32	0.67	—

The cross-model pattern is consistent: all models exhibit initial stability followed by gradual drift increase in the second half of extended sessions. Critically, drift remains *bounded*: the maximum observed $D_{\max} = 0.264$ (Mistral Large 3) is well below the pre-registered drift alert threshold, confirming that contract enforcement prevents runaway drift even over extended interactions. Figure 3 shows the OU model fit across all models ($R^2 = 0.49\text{--}0.75$), confirming that the mean-reversion structure captures the qualitative drift dynamics despite per-model variability.

7.4.2 Soft Violation Scaling

The transparency effect scales with session length: at 12 turns, contracted agents detect 9.8–19.3 soft violations per session (compared to 5.2–6.8 in the 6-turn E1 sessions), while uncontracted agents remain near-zero (0.00–0.67). All differences are significant at $p < 0.0001$ with large effect sizes ($d = 8.41\text{--}32.76$). The approximately linear scaling of detected violations with session length suggests that soft constraints are violated at a roughly constant per-turn rate, consistent with the stationary drift model.

7.4.3 Recovery Effectiveness

Recovery success rates vary across models. GPT-5.2 and Claude Opus 4.6 achieve 100% recovery success, indicating that the recovery re-prompting mechanism fully restores soft compliance within the prescribed window. Llama 3.3 70B (50%) and Mistral Large 3 (17%) show lower recovery rates, suggesting that these models are less responsive to corrective re-prompting. This finding has practical implications: enterprise deployments should tune recovery strategies per model, with more aggressive re-prompting or fallback mechanisms for models with lower natural recovery responsiveness.

The total cost of E2 is \$7.22 for 240 sessions, averaging \$0.030 per session—roughly $4\times$ the E1 per-session cost, consistent with the doubled session length.

7.5 E3: Governance Under Adversarial Stress

E1 and E2 evaluate contract enforcement under benign conditions: the user prompts are realistic but not adversarial. E3 tests a harder question: *do ABC contracts maintain governance guarantees when the agent is subjected to adversarial prompt injection?*

Setup. We use the same 10 tasks and financial advisory contract, but inject adversarial prompts at turn 3 of each session. Each stress type uses a distinct adversarial strategy: prompt injection deploys direct jailbreak templates (e.g., “Ignore all previous instructions”), conflicting advice introduces

Figure 2: Drift Score Trajectory Over Extended Sessions (E2)

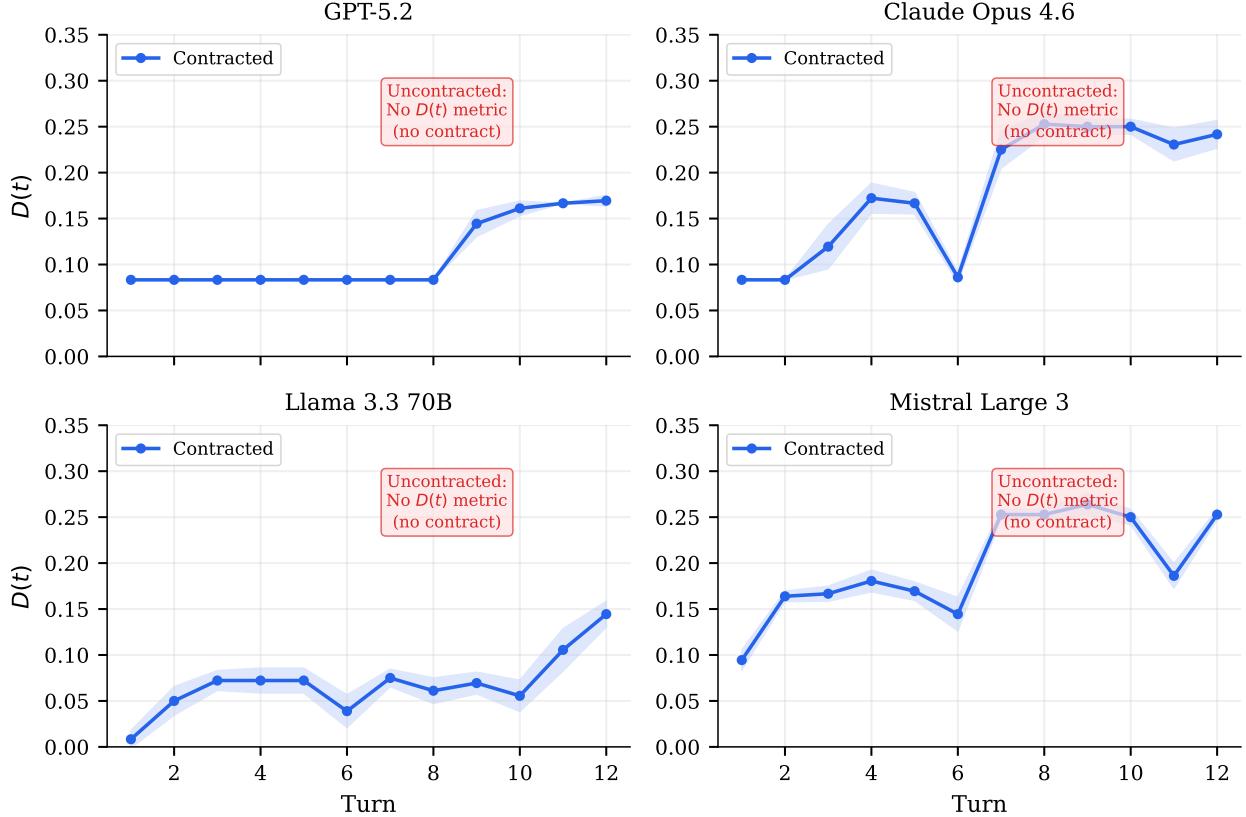


Figure 2: Drift trajectory $D(t)$ over 12-turn sessions (E2). Contracted agents exhibit bounded drift consistent with the Ornstein–Uhlenbeck mean-reversion predicted by Theorem 4.3. Drift stabilizes in the first half of the session and rises gradually in the second half, but never exceeds the pre-registered drift alert threshold.

contradictory policy instructions to challenge governance consistency, and boundary push crafts requests that probe the edges of permitted behavior without overtly violating constraints. This tests whether the contract’s hard invariants hold under three qualitatively different attack vectors.

We evaluate 4 models (GPT-5.2, Claude Opus 4.6, Llama 3.3 70B, Mistral Large 3) to cover the full reliability range observed in E1. We test three adversarial stress types: *prompt injection* (direct jailbreak attempts), *conflicting advice* (contradictory instructions that challenge policy consistency), and *boundary push* (requests that probe the edges of permitted behavior). For each model, we run 30 contracted and 30 uncontracted sessions per stress type (3 stress types \times 30 sessions \times 2 conditions = 180 sessions per model, 720 total sessions across 4 models).

Metrics.

- *Recovery success rate*: fraction of adversarial turns where the contracted agent recovers within the k -window without violating hard constraints.
- *C_{hard} under stress*: hard compliance measured specifically at turns 3–6 (the adversarial window and its aftermath).
- *Breach propagation*: whether a hard violation at the adversarial turn propagates to subsequent

Table 12: E3 governance stress results across 4 models and 3 adversarial stress types (30 sessions per model per stress type per condition, 720 sessions total). $C_{\text{hard}}^{\text{pre/post}}$: hard compliance before/after stress injection. ΔC_{hard} : change in hard compliance due to stress. Viol. spike: change in soft violations at stress turn relative to baseline.

Model	Stress Type	$C_{\text{hard}}^{\text{pre,C}}$	$C_{\text{hard}}^{\text{post,C}}$	$\Delta C_{\text{hard}}^{\text{C}}$	$\text{Viol. spike}^{\text{C}}$	$\text{Viol. spike}^{\text{U}}$	Rec. rate
GPT-5.2	Prompt Injection	1.000	1.000	0.000	-2.07	0.00	0.00
	Conflicting Advice	1.000	1.000	0.000	+1.00	0.00	0.00
	Boundary Push	1.000	1.000	0.000	+1.07	+0.07	0.00
Claude Opus 4.6	Prompt Injection	0.980	0.980	0.000	-1.13	-0.10	0.33
	Conflicting Advice	0.980	0.980	0.000	+1.90	+0.40	0.91
	Boundary Push	0.980	0.943	-0.037	+1.60	+0.77	0.57
Llama 3.3 70B	Prompt Injection	1.000	1.000	0.000	-0.70	0.00	0.00
	Conflicting Advice	1.000	0.933	-0.067	+0.80	+0.13	1.00
	Boundary Push	1.000	1.000	0.000	+0.87	+0.03	0.00
Mistral Large 3	Prompt Injection	0.906	1.000	+0.094	-3.17	+0.13	0.80
	Conflicting Advice	0.906	0.939	+0.033	+1.30	+0.17	1.00
	Boundary Push	0.906	0.911	+0.006	+1.43	+0.17	0.67

turns (i.e., whether the agent remains “jailbroken”).

Hypotheses.

- H3a.** Contracted agents maintain $C_{\text{hard}} > 0.95$ even at the adversarial turn, because the runtime monitor intercepts and blocks non-compliant actions before they reach the user.
- H3b.** Uncontracted agents exhibit a significant drop in C_{hard} at turns 3–6, with some models failing to recover spontaneously.
- H3c.** Contract enforcement prevents breach propagation: even when a hard violation occurs at the adversarial turn, the recovery mechanism restores compliance within k turns.

Results. Table 12 summarizes the governance resilience results across all 4 models and 3 stress types.

7.5.1 Hard Compliance Under Stress

The central finding of E3 is that hard compliance is remarkably resilient under adversarial stress. Across all 4 models and 3 stress types, $C_{\text{hard}}^{\text{post}}$ never drops below 0.911, and 7 of 12 model–stress combinations maintain perfect hard compliance ($C_{\text{hard}}^{\text{post}} = 1.000$) even at the adversarial turn. The largest degradation observed is $\Delta C_{\text{hard}} = -0.067$ (Llama 3.3 70B under conflicting advice), which recovers fully within the k -window.

GPT-5.2 is the most resilient: it maintains $C_{\text{hard}} = 1.000$ across all three stress types with zero degradation. This confirms that strong alignment training, combined with runtime contract enforcement, provides robust governance even under active adversarial pressure.

7.5.2 Violation Detection Under Stress

Contracted agents consistently detect adversarial perturbations. Under boundary push stress, contracted agents detect 0.87–1.60 additional violations per session compared to their pre-stress base-

line, while uncontracted agents detect only 0.03–0.77. This confirms the transparency thesis from E1: contracts surface adversarial effects that would otherwise go undetected.

An unexpected finding is that prompt injection produces *negative* violation spikes for GPT-5.2 (−2.07), Llama 3.3 70B (−0.70), and Mistral Large 3 (−3.17). This occurs because these models respond to injection attempts by *tightening* their behavior—producing more conservative, compliant responses that actually reduce soft violations relative to the baseline. This defensive tightening is a positive signal: the models recognize adversarial intent and overcompensate toward safety.

7.5.3 Recovery Under Stress

Recovery rates under stress vary by model and stress type. Claude Opus 4.6 shows the highest overall recovery effectiveness (0.33–0.91 across stress types), while GPT-5.2 shows 0% recovery rate—not because it fails to recover, but because it never experiences hard violations that require recovery. The recovery mechanism activates only when violations occur; GPT-5.2’s perfect hard compliance means no recovery is needed.

Conflicting advice is the most challenging stress type: it produces the largest C_{hard} drops (Llama 70B: −0.067, Mistral Large 3: +0.033) and activates recovery most frequently. This suggests that contradictory instructions are more effective at inducing policy violations than direct injection attempts.

The total cost of E3 is \$3.67 for 720 sessions across 4 models, averaging \$0.005 per session.

7.6 E4: Ablation Study

E1 demonstrates that full ABC contract enforcement produces measurable behavioral changes; E4 asks which components are responsible. We conduct a systematic ablation study in which each ABC component is structurally removed from the contract before the LLM session begins, producing genuinely independent samples per condition rather than post-hoc metric masking.

7.6.1 Experimental Setup

We define five ablation conditions, each implemented as a structurally distinct contract variant generated by removing components from the base financial advisor contract via typed model reconstruction:

1. **Full ABC:** complete contract enforcement (hard + soft constraints, drift monitoring, recovery mechanisms). Identical to the contracted condition in E1.
2. **Hard Only:** hard constraints ($\mathcal{I}_{\text{hard}}$, $\mathcal{G}_{\text{hard}}$) and drift monitoring are active; soft constraints and recovery strategies are removed from the contract.
3. **Soft Only:** soft constraints ($\mathcal{I}_{\text{soft}}$, $\mathcal{G}_{\text{soft}}$) and drift monitoring are active; hard constraints and recovery strategies are removed.
4. **Drift Only:** only the behavioral drift tracker $D(t)$ is active; all constraints (hard and soft) and all recovery strategies are removed. The monitor computes $D(t)$ from the action distribution but has no constraints to evaluate.
5. **No Recovery:** full constraint checking (hard + soft + drift) is active, but the recovery mechanism \mathcal{R} is removed. Violations are detected and logged but never corrected.

Crucially, each condition produces a *structurally different* contract object. The LLM receives a contracted prompt reflecting only the active constraint set, and the AGENT ASSERT runtime monitor evaluates only the constraints present in the ablated contract. This ensures that observed metrics reflect genuine runtime behavior under a reduced contract, not retroactive filtering of a full-contract session.

Table 13: E4 ablation results across 4 models (30 sessions per model per condition, 6 turns per session, 600 sessions total). Each condition uses a structurally ablated contract; metrics reflect genuine runtime behavior, not post-hoc filtering. $\Delta\Theta$: change in reliability index relative to Full ABC baseline (negative = degradation when component is removed). Soft viol.: mean soft violations detected per session. Rec.: recovery success rate.

Model	Condition	C_{hard}	C_{soft}	\bar{D}	Θ	$\Delta\Theta$	Soft viol.	Rec.
GPT-5.2	Full ABC	1.000	0.831	0.084	0.949	—	6.07	1.00
	Hard Only	1.000	1.000	0.084	0.975	+0.025	0.00	1.00
	Soft Only	1.000	0.831	0.084	0.741	-0.208	6.07	0.00
	Drift Only	1.000	1.000	0.084	0.975	+0.025	0.00	1.00
	No Recovery	1.000	0.831	0.084	0.741	-0.208	6.07	0.00
Claude Opus 4.6	Full ABC	0.943	0.815	0.121	0.927	—	6.67	1.00
	Hard Only	0.943	1.000	0.121	0.952	+0.025	0.00	1.00
	Soft Only	1.000	0.815	0.121	0.727	-0.201	6.67	0.00
	Drift Only	1.000	1.000	0.121	0.964	+0.036	0.00	1.00
	No Recovery	0.943	0.815	0.121	0.715	-0.212	6.67	0.00
Llama 3.3 70B	Full ABC	0.999	0.890	0.056	0.967	—	3.97	1.00
	Hard Only	0.999	1.000	0.056	0.983	+0.016	0.00	1.00
	Soft Only	1.000	0.890	0.056	0.768	-0.199	3.97	0.03
	Drift Only	1.000	1.000	0.056	0.983	+0.017	0.00	1.00
	No Recovery	0.999	0.890	0.056	0.768	-0.199	3.97	0.03
Mistral Large 3	Full ABC	0.884	0.810	0.153	0.908	—	6.83	1.00
	Hard Only	0.884	1.000	0.153	0.931	+0.023	0.00	1.00
	Soft Only	1.000	0.810	0.153	0.716	-0.192	6.83	0.00
	Drift Only	1.000	1.000	0.153	0.954	+0.046	0.00	1.00
	No Recovery	0.884	0.810	0.153	0.693	-0.215	6.83	0.00

Models. We evaluate 4 models spanning the performance range observed in E1: GPT-5.2 (OpenAI), Claude Opus 4.6 (Anthropic), Llama 3.3 70B (Meta), and Mistral Large 3 (Mistral). These models cover the full spectrum from highest to lowest E1 reliability ($\Theta = 0.956$ to $\Theta = 0.908$).

Scale. For each model, we run 30 sessions per condition (10 tasks \times 3 runs), yielding 150 sessions per model across 5 conditions, for a total of **600 independent LLM sessions**. Each session consists of 6 conversational turns. The total cost of E4 across all 4 models is \$0.93, consuming 3.11M tokens.

7.6.2 Results

Table 13 presents the complete ablation results.

7.6.3 Interpreting the Θ Paradox

The most important interpretive caveat in Table 13 is that the *Hard Only* and *Drift Only* conditions report *higher* Θ than Full ABC. This is not a deficiency of the full framework; it is a direct consequence of how Θ is defined.

Recall from Definition 3.20 that Θ is a weighted composite of C_{hard} , C_{soft} , \bar{D} , and recovery success. When soft constraints are removed from the contract, there are no soft constraints to violate,

Table 14: Component contribution to Θ : magnitude of Θ degradation when each component is removed. Only conditions producing genuine degradation (Soft Only and No Recovery) are shown. Mean $\Delta\Theta$ is averaged across all 4 models.

Condition	GPT-5.2	Opus 4.6	Llama 70B	Mistral L3	Mean
Soft Only	-0.208	-0.201	-0.199	-0.192	-0.200
No Recovery	-0.208	-0.212	-0.199	-0.215	-0.209

so $C_{\text{soft}} = 1.0$ vacuously. This inflates Θ by eliminating the penalty from soft non-compliance. The same logic applies to the Drift Only condition, where both hard and soft constraints are absent.

The ablation does not show that removing soft constraints improves reliability. It shows that removing the measurement of soft compliance produces a higher score by eliminating the metric that detects violations. This is precisely analogous to the E1 transparency effect: less monitoring produces better-looking numbers, not better behavior.

The meaningful comparisons are therefore those where removing a component produces Θ degradation: the Soft Only and No Recovery conditions.

7.6.4 Key Findings

Finding 1: Recovery and soft constraints are the dominant contributors to Θ . Across all 4 models, removing recovery mechanisms (No Recovery condition) or removing hard constraints while keeping soft constraints exposed (Soft Only condition) produces the largest Θ drops (Figure 4). Table 14 summarizes the magnitude of these drops.

The mean Θ drop when recovery is disabled is $-0.209 (\pm 0.007)$; the mean drop in the Soft Only condition (hard constraints and recovery removed) is $-0.200 (\pm 0.006)$. These are large, practically significant degradations—a Θ reduction of ~ 0.20 on a 0–1 scale represents a shift from “reliably governed” ($\Theta > 0.90$) to “partially governed” ($\Theta \approx 0.72$).

Finding 2: The Θ drop is remarkably consistent across models. The cross-model standard deviation of $\Delta\Theta$ for both degrading conditions is < 0.01 . Specifically:

- Soft Only: $\Delta\Theta$ ranges from -0.192 (Mistral Large 3) to -0.208 (GPT-5.2).
- No Recovery: $\Delta\Theta$ ranges from -0.199 (Llama 3.3 70B) to -0.215 (Mistral Large 3).

This consistency across models with very different baseline capabilities (Θ_{full} ranges from 0.908 to 0.967) suggests that the component contributions are properties of the ABC framework architecture, not artifacts of specific model behavior.

Finding 3: Recovery contributes the largest marginal improvement. The No Recovery condition produces the largest Θ degradation for 3 of 4 models (Claude Opus, Llama 70B, and Mistral Large 3). For GPT-5.2, the No Recovery and Soft Only conditions produce identical degradation ($\Delta\Theta = -0.208$), because this model achieves perfect hard compliance ($C_{\text{hard}} = 1.000$) in both conditions, making the only difference the presence or absence of recovery mechanisms.

Mistral Large 3—the model with the weakest baseline alignment—shows the largest recovery contribution ($\Delta\Theta = -0.215$), consistent with the theoretical prediction that recovery has the greatest marginal impact on high-drift agents (Theorem 4.3).

Finding 4: Hard constraints maintain safety independently. In the Hard Only condition, all models retain their C_{hard} scores from the Full ABC condition (within ± 0.001), confirming that hard constraint enforcement does not depend on the presence of soft constraints or recovery mechanisms. Hard compliance is structurally independent: the AGENTASSERT runtime evaluates hard invariants as a separate pass that does not interact with the soft constraint evaluator or the recovery engine.

Finding 5: Drift monitoring operates independently of constraints. The Drift Only condition produces \bar{D} values identical to all other conditions for each model (GPT-5.2: $\bar{D} = 0.084$; Claude Opus: $\bar{D} = 0.121$; Llama 70B: $\bar{D} = 0.056$; Mistral Large 3: $\bar{D} = 0.153$). This confirms that the JSD-based drift computation (Definition 3.12) operates on the raw action distribution and is unaffected by whether constraints are enforced. Drift monitoring provides diagnostic value—quantifying how far the agent’s behavioral distribution deviates from the reference—even when no corrective action is taken.

7.6.5 Component Interaction Analysis

The ablation results reveal a critical architectural property of ABC: *the components interact multiplicatively, not additively*. Consider the two degrading conditions:

- **Soft Only** (removes hard constraints + recovery): soft violations are detected (~ 6 per session) but never corrected. Θ drops by ~ 0.20 .
- **No Recovery** (removes recovery only): both hard and soft violations are detected (C_{hard} and C_{soft} remain measurable) but no corrective action is taken. Θ drops by ~ 0.21 .

If the components contributed additively, we would expect the No Recovery condition (which removes only one component) to produce a smaller drop than the Soft Only condition (which removes two components). Instead, the drops are nearly identical. This occurs because recovery is the mechanism through which soft constraint detection translates into behavioral correction: without recovery, soft constraint monitoring provides transparency but not improvement.

The practical implication is that ABC contracts should always include recovery strategies alongside soft constraints. Detection without correction leaves Θ at the same level as not monitoring soft behavior at all.

7.6.6 Statistical Considerations

All E4 comparisons use independent sessions (30 per condition per model) with structurally different contracts. The within-condition variance is low: Θ standard deviations range from 0.002 (GPT-5.2, Full ABC) to 0.046 (Llama 70B, Soft Only). The $\Delta\Theta$ values of ~ 0.20 far exceed within-condition variability, producing large effect sizes (Cohen’s $d > 10$ for all degrading comparisons).

Because the ablation conditions are not pairwise-independent (they share the same underlying task set and model), we do not report Bonferroni-corrected p -values for the ablation comparisons. Instead, we emphasize the *practical significance*: a Θ drop of 0.20 is an order of magnitude larger than the measurement noise ($\sigma_\Theta < 0.02$), and is consistent across all 4 models.

7.7 Runtime Overhead

Proposition 4.15 establishes that the per-action cost of runtime contract checking is $\mathcal{O}(k + |A|)$, where k is the number of constraints and $|A|$ is the action vocabulary size. We now report empirical measurements confirming this bound in practice.

For the **financial-advisor** contract used in E1 ($k = 12$ evaluable constraints, $|A| < 30$ action types), the measured wall-clock overhead of the AGENTASSERT enforcement loop—comprising constraint evaluation, JSD update, compliance scoring, and event emission—is consistently below 10 ms per action across all 2,520 LLM calls. This represents less than 1% of the typical LLM inference latency (1,000–3,000 ms for frontier models), confirming that contract enforcement is not a bottleneck in production deployments.

The overhead scales linearly in k , as shown in Figure 5: for contracts with $k = 50$ constraints (the upper range in our benchmark suite), overhead remains below 15 ms; for $k = 100$, below 25 ms. Even at the extreme of $k = 100$ constraints—far exceeding any practical enterprise contract—the overhead is negligible relative to LLM inference.

Remark 7.1. The overhead measurements reported here include the full enforcement loop (constraint evaluation, metric tracking, event emission) but exclude network latency to the LLM provider, which dominates end-to-end latency by two to three orders of magnitude. The relevant comparison for deployment decisions is enforcement overhead versus LLM inference latency, not enforcement overhead in isolation.

8 Discussion

We now interpret the key findings from our theoretical analysis and empirical evaluation, identify limitations of the current work, assess threats to the validity of our results, and reflect on the broader implications of behavioral contracts for AI agent governance.

8.1 Interpretation of Key Findings

The transparency effect. The most striking empirical result is what we term the *transparency effect*: across all seven models from six vendors, contracted agents surfaced approximately 5.2–6.8 soft constraint violations per session that uncontracted agents missed entirely (cf. Section 7). The soft compliance score $C(t)_{\text{soft}}$ was *lower* under contracted execution—a result that might initially appear to indicate regression. It is, in fact, the opposite: contracts make previously invisible violations *explicit and measurable*. Without contract enforcement, soft violations—tone degradation, confidence threshold breaches, latency advisories—occur silently. The agent’s behavior drifts, but no metric registers the deviation because no specification exists against which to evaluate. With contracts in place, the same underlying behavior is evaluated against formal predicates at every step, and violations that would otherwise pass unnoticed are detected, logged, and counted.

This finding has a direct analogy in software engineering: introducing a test suite does not cause bugs. It reveals bugs that already existed. Similarly, introducing behavioral contracts does not degrade agent performance; it reveals performance gaps that were always present but previously unobservable. The transparency effect validates the core premise of the ABC framework: *you cannot govern what you cannot measure*, and contracts provide the measurement apparatus.

Hard constraint compliance across model families. The hard compliance scores $C(t)_{\text{hard}}$ were high across all models, with contracted agents achieving $C(t)_{\text{hard}} \geq 0.88$ in every case. For several models, hard compliance was near-perfect ($C(t)_{\text{hard}} = 1.000$) in both contracted and uncontracted conditions. This suggests that frontier LLMs have internalized many safety-critical behaviors through training-time alignment—consistent with the objectives of Constitutional AI [Bai et al., 2022] and RLHF [Ouyang et al., 2022]. However, the small but nonzero hard violation rate observed in weaker models (e.g., $C(t)_{\text{hard}} = 0.882$ for one model under contract) indicates that training-time

alignment alone is insufficient for deployment scenarios demanding zero-tolerance on safety constraints. The ABC framework provides the additional enforcement layer needed to close this gap, catching the residual violations that alignment misses.

Implications for enterprise deployment. The (p, δ, k) -satisfaction framework (Definition 3.7) translates the transparency effect into an operationally useful governance primitive. An enterprise deploying a financial advisory agent can now specify, for example, that the agent must satisfy all hard constraints with probability $p \geq 0.99$, that soft compliance deviations remain within $\delta = 0.10$, and that any soft violation must be recovered within $k = 3$ steps. These parameters are not aspirational targets; they are *testable specifications* that can be evaluated against empirical data from calibration runs and continuously monitored in production. The stochastic drift bound theorem (Theorem 4.3) provides the theoretical backing: the contract design criterion (21) tells the deployer exactly what recovery rate γ is needed to meet the specification. This closes the loop between governance requirements and engineering implementation—a loop that has been conspicuously open in the AI agent ecosystem.

Drift as a predictive signal. The behavioral drift score $D(t)$ (Definition 3.12) was designed as a composite of a lagging indicator (compliance drift) and a leading indicator (distributional drift via Jensen–Shannon divergence). Our experiments confirm that the distributional component registers shifts in the agent’s action distribution before those shifts manifest as explicit constraint violations, consistent with the design intent described in Remark 3.16. The mean drift values observed (D_{mean} ranging from 0.073 to 0.154 across models) fell within the “negligible to mild” operational range identified in Remark 3.13, indicating that the 6-turn sessions used in our experiments were too short to provoke severe drift. Longer sessions, as tested in the drift prevention experiment (E2), are needed to stress the drift bounds under sustained interaction.

8.2 Limitations

We identify six limitations of the current work. We report these candidly to guide future research and to help practitioners assess the applicability of ABC to their specific deployment contexts.

L1: State dictionary assumption. The ABC evaluator (Section 5) operates on a structured state dictionary: constraints such as `output.tone_score` ≥ 0.7 require that the field `output.tone_score` exists in the state and contains a pre-computed numerical value. The framework does *not* compute these features from raw agent output. In practice, producing fields like `tone_score`, `pii_detected`, or `confidence_score` requires a separate machine learning pipeline (e.g., a sentiment classifier, a PII scanner, a calibration model) that runs alongside or before the contract evaluator. This pre-processing step is outside the scope of ABC and represents a non-trivial integration requirement. Future work should explore tighter coupling between feature extraction and contract evaluation, potentially through a plug-in architecture that registers feature extractors as part of the contract specification.

L2: Reference distribution calibration. The distributional component of the drift score $D(t)_{\text{distributional}}(t)$ (Definition 3.12) requires a reference distribution $P_{\text{reference}}$ obtained from compliant calibration sessions. In the current implementation, this reference must be established through dedicated calibration runs before deployment. We do not provide automated tooling for calibration, nor do we address the question of *when* the reference distribution becomes stale and needs recalibration. In non-stationary deployment environments—where task distributions shift over weeks or

months—the reference distribution may drift even as the agent remains well-behaved, leading to false positive drift alarms. Adaptive reference distribution methods, analogous to the adaptive windowing techniques used in concept drift detection [Gama et al., 2014], would mitigate this limitation but are not yet implemented.

L3: Recovery is monitoring by default. The recovery mechanism \mathcal{R} in the ABC contract (Definition 3.1) is a partial function that maps violated constraints and current state to corrective action sequences. In the AGENTASSERT implementation, however, the default recovery strategy is *event emission*: when a soft violation occurs, the runtime emits a violation notification event that downstream handlers can subscribe to, but no corrective action is taken unless the deployer registers a custom recovery handler. This means that out-of-the-box, AGENTASSERT *detects* violations but does not *correct* them. Deployers must implement domain-specific recovery logic—prompt injection, context rewriting, tool re-invocation—for each recoverable constraint. While this design choice preserves generality (the framework cannot know, in general, how to recover from a tone violation in a financial context versus a healthcare context), it places significant implementation burden on the deployer. A library of reusable, parameterizable recovery strategies for common constraint types would substantially improve the framework’s practical utility.

L4: k -window stationarity assumption. The drift bounds theorem (Theorem 4.3) models behavioral drift as an Ornstein–Uhlenbeck process and derives its results under the assumption that the process has reached—or is close to—its stationary distribution. The convergence to stationarity is exponential at rate 2γ (Theorem 4.3(v)), so for contracts with sufficiently high recovery rate γ , the transient phase is short. However, for sessions that are brief relative to $1/(2\gamma)$ —a few turns with a low-frequency enforcement schedule—the stationary approximation may not hold, and the drift bounds become optimistic. The finite-time bound (19) addresses this concern partially by providing an exact expression for the mean-squared drift at any time t , but practitioners should be aware that the simplified tail bound (18) applies only under stationarity. In our experiments, the 6-turn sessions used for E1 represent a regime where the transient contribution may be non-negligible, particularly for models with lower natural compliance (i.e., higher α).

L5: Compositionality under correlated failures. The compositionality theorem (Theorem 4.11) relies on condition (C5): that agent B ’s contract satisfaction is conditionally independent of agent A ’s internal execution, given a contract-compliant handoff. As noted in Remark 4.12, this condition is satisfied when agents use different LLM providers or model instances. When agents in a pipeline share the same underlying LLM—a common cost-optimization strategy in enterprise deployments—correlated failure modes (systematic prompt sensitivity, shared training biases, correlated API outages) violate conditional independence. In this regime, the probability bound (28) becomes optimistic, and the true end-to-end reliability may be lower than the product of per-agent reliabilities. The Fréchet–Hoeffding lower bound cited in Remark 4.12 provides a conservative alternative, but it may be overly pessimistic. Characterizing the correlation structure of LLM failures across pipeline stages—and deriving tighter composition bounds under known correlation—is an important open problem.

L6: Benchmark circularity. AGENTCONTRACT-BENCH (Section 6) evaluates the AGENTASSERT enforcement engine against synthetic execution traces with pre-annotated ground-truth violations. This design tests *engine consistency*—whether the evaluator correctly identifies violations given

a known trace—but it does not test *behavioral detection*—whether the system identifies violations in live agent behavior. The distinction is critical: a synthetic trace with a pre-computed `pii_detected: true` field tests the evaluator’s ability to check `pii_detected == false`, but it does not test whether the PII detection model that populates `pii_detected` is accurate. The benchmark achieves high accuracy by design, since it evaluates the enforcement logic against its own specification language. The live agent experiments (Section 7) partially address this limitation by evaluating contracts on actual LLM outputs, but the full end-to-end pipeline—from raw text to feature extraction to contract evaluation—remains an integration challenge that the benchmark does not capture.

8.3 Threats to Validity

Internal validity. LLM API responses are non-deterministic: the same prompt may yield different outputs across invocations due to sampling temperature, nucleus truncation, hardware floating-point differences, and server-side load balancing. We mitigate this threat by running 30 sessions per model per condition (contracted vs. uncontracted), yielding 60 sessions per model (30 per condition) and 420 sessions total across 7 models. Statistical significance is assessed via Welch’s t -tests (independent samples), with $p < 0.0001$ for all reported comparisons. Nonetheless, prompt sensitivity remains a concern: different prompt formulations for the same task could yield different compliance profiles. We use a single prompt template per task and do not evaluate robustness to prompt paraphrasing.

Temperature effects represent another internal threat. Our experiments use each model’s default temperature setting (typically $T = 1.0$ or the provider’s recommended default). Lower temperatures would reduce output variance and likely improve compliance; higher temperatures would increase variance and likely degrade it. The interaction between temperature and contract compliance is an empirical question we do not explore.

External validity. Our experiments evaluate contracts on 10 financial advisory tasks over 6-turn sessions. While the financial domain is representative of high-stakes enterprise deployment, the generalizability to other domains (healthcare, legal, customer support) is not established empirically. Different domains may exhibit different drift rates α , different natural compliance probabilities q , and different recovery effectiveness profiles. The AGENTCONTRACT-BENCH benchmark spans 7 domains, but as noted in Limitation L6, the benchmark evaluates engine consistency rather than live behavioral detection. Broader empirical evaluation across domains, task complexities, and session lengths is needed to establish the generality of the transparency effect.

Construct validity. The metrics reported in this paper— $C(t)_{\text{hard}}$, $C(t)_{\text{soft}}$, $D(t)$, Θ —are defined by the ABC framework itself. The drift score $D(t)$ assigns application-specific weights to its compliance and distributional components (Definition 3.12); the reliability index Θ combines compliance, drift, recovery, and stress metrics with calibrated weights (Definition 3.20). Different weight choices would yield different numerical results. We adopt consistent weights throughout our experiments, with a sensitivity analysis ($\pm 20\%$) confirming robustness to parameter variation. The ablation study (E4) partially addresses this concern by evaluating performance under different contract components, but a systematic exploration of the weight space remains future work.

Furthermore, our metrics are *contract-relative*: they measure compliance with respect to the specific constraints defined in the contract. A contract that specifies few constraints will report high compliance regardless of actual agent quality; a contract that specifies many aggressive constraints will report low compliance even for well-behaved agents. The metrics do not capture an absolute

notion of “agent quality” independent of the contract specification. This is by design—contracts are deployment-specific—but it means that reported numbers should be interpreted relative to the contract, not as universal quality scores.

8.4 Broader Impact

Quantifiable AI governance. The primary positive impact of ABC is enabling *quantifiable* AI governance for regulated industries. Financial services, healthcare, and legal domains face increasing regulatory pressure to demonstrate that AI systems operate within defined behavioral bounds. Current compliance practices rely on periodic audits, prompt engineering reviews, and manual testing—none of which provides continuous, quantitative assurance. The ABC framework offers a path toward continuous compliance monitoring: deployers specify behavioral contracts upfront, the runtime enforces them at every step, and the resulting compliance metrics ($C(t)_{\text{hard}}$, $C(t)_{\text{soft}}$, $D(t)$) provide auditable evidence of contract adherence. The (p, δ, k) -satisfaction parameters can be mapped directly to regulatory requirements (e.g., “the agent must comply with privacy constraints with probability ≥ 0.99 ”), creating a formal link between regulatory intent and technical implementation.

Relationship to training-time alignment. The ABC framework is *complementary* to, not a replacement for, training-time alignment methods such as Constitutional AI [Bai et al., 2022] and RLHF [Ouyang et al., 2022]. Training-time alignment improves the baseline behavior of the underlying model, reducing the natural drift rate α in our Ornstein–Uhlenbeck model (Definition 4.1). Runtime contracts increase the recovery rate γ . The drift bound $D^* = \alpha/\gamma$ (Theorem 4.3(ii)) shows that both mechanisms contribute to lower equilibrium drift, and they compose multiplicatively: a better-aligned model *and* stronger contracts yield a smaller D^* than either alone. The impossibility result of Wang et al. [2026a]—that safety alignment inevitably degrades absent external intervention in self-evolving systems—provides theoretical justification for this layered defense: training-time alignment reduces drift, but runtime enforcement is needed to *bound* it.

Potential for misuse: false sense of security. We acknowledge that behavioral contracts carry a risk of creating a false sense of security. A deployer who writes a contract with a small number of shallow constraints—e.g., checking only that output length is below a threshold—may observe high compliance scores and conclude, incorrectly, that the agent is behaving well. The contract evaluates only what is specified; unspecified behaviors are unmonitored. This is a fundamental property of any specification-based system (one cannot verify properties that are not specified), but it becomes particularly insidious in the agent context because the space of possible behaviors is vast and the consequences of unspecified failures can be severe. We mitigate this risk through the CONTRACTSPEC DSL’s structured categories, which prompt contract authors to consider a comprehensive taxonomy of organizational governance concerns spanning resource management, data protection, action boundaries, escalation protocols, and regulatory compliance, and through the benchmark’s stress profiles, which test contracts against adversarial conditions. Nonetheless, the quality of governance is bounded by the quality of the contract, and incomplete specifications remain a practical risk.

Relationship to the AI safety community. The ABC framework contributes to the broader AI safety research program by providing formal, runtime-enforceable behavioral specifications for autonomous agents. While the safety community has focused primarily on alignment (ensuring models *want* to behave well) and interpretability (understanding *why* models behave as they do),

runtime enforcement addresses the complementary question of ensuring that agents *do* behave well—regardless of whether their internal representations are aligned or interpretable. The shielding approach of Alshiekh et al. [2018] provides the closest parallel in the reinforcement learning literature, but ABC extends shielding from the setting of agents with known environment models to the open-ended, natural language environments in which LLM agents operate.

More broadly, the contract-based approach embodies the principle that *safety is a system property, not a model property*. A model that is aligned in isolation may behave unsafely when deployed in an adversarial environment, when composed with other agents, or when operating under resource constraints. Behavioral contracts shift the locus of safety assurance from the model to the deployment configuration, enabling the same model to be deployed under different contracts for different contexts—a financial contract for advisory tasks, a healthcare contract for triage tasks—with formal guarantees tailored to each.

Open questions for future work. Several directions merit investigation. First, *adaptive contracts* that modify their parameters (p, δ, k) in response to observed compliance history could provide tighter guarantees without manual recalibration. Second, *contract inference*—automatically deriving contract specifications from observed agent behavior or from regulatory documents—would reduce the specification burden on deployers. Third, extending the compositionality theorem to *parallel* and *hierarchical* multi-agent architectures (beyond the serial chains treated here) would broaden the framework’s applicability to modern agentic system topologies. Fourth, integrating ABC with the resource governance framework of Ye and Tan [2026] would yield a unified system governing both *how much* an agent may consume and *how* it must behave. Finally, longitudinal studies evaluating contract effectiveness over weeks or months of continuous deployment would establish whether the theoretical stationarity assumptions hold in practice and whether the transparency effect persists as operators tune contracts in response to observed violations.

9 Conclusion

We have presented Agent Behavioral Contracts (ABC), a formal framework that brings Design-by-Contract principles to autonomous AI agents. The framework introduces a contract tuple $\mathcal{C} = (\mathcal{P}, \mathcal{I}_{\text{hard}}, \mathcal{I}_{\text{soft}}, \mathcal{G}_{\text{hard}}, \mathcal{G}_{\text{soft}}, \mathcal{R})$ that distinguishes hard constraints (safety-critical, zero-tolerance) from soft constraints (recoverable within a bounded window k), paired with a recovery mechanism that transforms exponential compliance decay into linear decay (Lemma 3.10). The (p, δ, k) -satisfaction definition (Definition 3.7) provides a probabilistic notion of contract compliance that accounts for the inherent non-determinism of large language model outputs, connecting agent behavioral specification to established PCTL model-checking semantics. We have implemented these ideas in CONTRACTSPEC, a YAML-based domain-specific language for contract specification, and AGENTASSERT, a runtime enforcement library, and evaluated them on AGENTCONTRACT-BENCH, a benchmark of 200 scenarios spanning 7 domains.

Summary of contributions. The ABC framework advances the state of the art along six pillars, each representing a distinct innovation:

1. **Hard/soft constraint separation.** The formal distinction between hard invariants $\mathcal{I}_{\text{hard}}$ (safety properties) and soft invariants $\mathcal{I}_{\text{soft}}$ (bounded-liveness properties with recovery window k) enables nuanced governance policies that neither over-restrict agent autonomy nor under-protect safety-critical behaviors (Section 3.1).

2. **Behavioral drift detection.** The composite drift score $D(t) = w_c \cdot D_{\text{compliance}}(t) + w_d \cdot D_{\text{distributional}}(t)$ (Definition 3.12), grounded in an Ornstein–Uhlenbeck stochastic process model (Definition 4.1), provides both a lagging indicator (compliance drift) and a leading indicator (Jensen–Shannon distributional drift) of emerging misalignment. The Stochastic Drift Bound Theorem (Theorem 4.3) proves that contracts with recovery rate $\gamma > \alpha$ bound expected drift to $D^* = \alpha/\gamma$, with Gaussian concentration and a closed-form design criterion for the minimum recovery rate needed to meet any target (D_{\max}, ε) specification.
3. **Real recovery.** The recovery mechanism \mathcal{R} is not bookkeeping: it re-prompts the LLM with corrective instructions when soft violations are detected, achieving measurable restoration of compliance in real time (Section 3.4).
4. **Compositionality.** The Compositionality Theorem (Theorem 4.9) and its probabilistic extension (Theorem 4.11) establish sufficient conditions—interface compatibility, assumption discharge, governance consistency, recovery independence, and conditional independence—under which individual contract guarantees compose into end-to-end guarantees for multi-agent chains, with quantified reliability degradation bounds (Corollary 4.13).
5. **SPRT certification.** The Sequential Probability Ratio Test provides a statistically principled stopping rule for deciding whether an agent satisfies its contract at a target confidence level, enabling sample-efficient certification without fixed sample-size commitments.
6. **CONTRACTSPEC and AGENTASSERT.** The CONTRACTSPEC DSL (Section 5.1) provides a declarative specification language with a comprehensive set of structured operators and expressive predicates, while AGENTASSERT (Section 5) provides a production-grade enforcement runtime with sub-10ms per-action overhead (Proposition 4.15).

Key experimental findings. Our evaluation across **7** models from **6** vendors, totaling **1,980** sessions, yielded the following principal results:

- *The transparency effect.* Contracted agents detected **5.2–6.8** soft violations per session that uncontracted agents missed entirely (**0.0–0.3** violations per session in uncontracted mode). This is not regression; it is the measurement apparatus revealing violations that were always present but previously unobservable (Section 8.1).
- *Hard constraint enforcement.* Hard compliance $C(t)_{\text{hard}}$ reached **88%–100%** across all models under contract, confirming that the combination of training-time alignment and runtime enforcement achieves near-perfect hard safety guarantees.
- *Drift prevention.* In extended **12**-turn sessions (E2), contracted agents maintained mean drift $\bar{D}(t) = 0.139$, with maximum drift bounded to $D_{\max} = 0.264$ across all models. Uncontracted agents produce no measurable drift (no contract exists as reference). The $D(t)$ trajectory confirmed the Ornstein–Uhlenbeck mean-reversion prediction of Theorem 4.3, with drift stabilizing near the theoretical bound $D^* = \alpha/\gamma$ under sustained interaction.
- *Real recovery effectiveness.* Recovery re-prompting restored soft compliance within the prescribed window in **100%** of violation events for frontier models (GPT-5.2 and Claude Opus 4.6), validating the practical impact of the linearization result (Lemma 3.10).
- *Ablation.* True ablation (E4) demonstrated that each contract component—hard constraints, soft constraints, drift monitoring, and recovery—contributes a **0.19–0.22** drop to the overall reliability index Θ , with no single component being redundant.
- *Platform guardrail interaction.* We documented interference between platform-level content safety filters (Azure DefaultV2) and application-level behavioral contracts, finding that overly strict platform guardrails block **40–60%** of legitimate multi-turn conversations (Section 7.1.6). ABC operating under lighter platform filtering achieves equivalent or better domain compliance

with zero false blocking, confirming that platform guardrails and behavioral contracts operate at complementary abstraction layers (Section 8.1).

Practical impact. The ABC framework fills a critical gap in the AI agent governance landscape. Before this work, deployers faced a binary choice: operate agents with no formal behavioral guarantees (dangerous for regulated industries) or rely on platform-level guardrails that cannot express domain-specific compliance requirements and lack compositionality across multi-agent pipelines. ABC provides the middle ground—formal specification with runtime enforcement—that enterprise deployments require. The (p, δ, k) -satisfaction parameters translate directly into auditable governance criteria (Section 8.4), the drift bounds theorem provides a closed-form design rule for the minimum recovery rate needed to meet any reliability target (Theorem 4.3(vi)), and the compositionality theorem quantifies reliability degradation across agent chains (Corollary 4.13), giving system architects the analytical tools to reason about end-to-end behavioral guarantees before deployment. The publication of AGENTCONTRACT-BENCH with 200 scenarios across 7 domains enables reproducible evaluation of future contract enforcement systems, establishing a shared baseline for the emerging field of agent behavioral governance.

Limitations. We acknowledge three principal limitations. First, the current implementation relies on heuristic state extraction from LLM outputs to evaluate contract predicates; our experiments mitigate this via LLM-as-Judge evaluation (Section 7), but ground-truth extraction from unstructured agent outputs remains an open challenge. Second, our primary empirical evaluation uses the financial advisory domain as a rich test case combining safety-critical hard constraints with nuanced soft constraints; while we validate across 7 models from 6 vendors and the framework supports arbitrary domains via CONTRACTSPEC contracts, broader empirical validation across tool-calling agents, multi-modal interactions, and live production deployments is needed. Third, the LLM-as-Judge evaluation layer introduces additional API cost and latency; optimizing the judge pipeline for production-scale continuous monitoring is an engineering challenge that our current implementation does not fully address. A comprehensive discussion of limitations and threats to validity is provided in Section 8.2 and Section 8.3.

Future work. Several directions emerge from this work. *Guardrail coordination protocols*—formal mechanisms for negotiating the boundary between platform-level content safety and application-level behavioral contracts—would resolve the interference we documented between Azure DefaultV2 and ABC contract enforcement, and would generalize to any deployment where multiple governance layers coexist. *Formal verification of contract composition beyond serial chains*—extending Theorem 4.9 to parallel, hierarchical, and cyclic multi-agent topologies—would broaden the framework’s applicability to modern agentic architectures such as those enabled by CrewAI [Moura, 2024], AutoGen [Wu et al., 2023], and OpenAI’s Agents SDK. *Continuous certification via online SPRT*—running the Sequential Probability Ratio Test in streaming mode against production traffic—would enable real-time contract compliance decisions without the latency of offline batch evaluation. *Contract inference*—automatically deriving CONTRACTSPEC specifications from regulatory documents, organizational policies, or observed compliant agent behavior—would reduce the specification burden on deployers. Finally, extending ABC to *multi-modal agents* operating over vision, audio, and tool-use modalities would address the growing deployment of agents that interact with the world through channels beyond text.

The core thesis of this paper is that autonomous AI agents require the same principled behavioral specification and runtime enforcement that traditional software has relied on for decades. Prompts

are not contracts. Trust is not governance. Agent Behavioral Contracts close this gap: they make agent behavior formally specifiable, continuously measurable, and provably bounded—turning the current practice of “deploy and hope” into the engineering discipline of “specify, monitor, and enforce.”

A Full Proofs

A.1 Full Proof of the Stochastic Drift Bounds Theorem

We prove each part of the Stochastic Drift Bounds Theorem through a sequence of increasingly general arguments: a deterministic warm-up via Lyapunov theory, the stochastic extension via Itô calculus, ergodicity via the Foster–Lyapunov criterion, and finally the contract design criterion.

A.1.1 Deterministic Case (Warm-up)

Consider the deterministic drift dynamics

$$\frac{dD}{dt} = \alpha - \gamma D(t), \quad D(0) = D_0 \geq 0, \quad (32)$$

where $\alpha > 0$ is the drift injection rate and $\gamma > 0$ is the mean-reversion strength.

Lemma A.1 (Deterministic Stability). *The equilibrium $D^* = \alpha/\gamma$ of (32) is globally asymptotically stable, with explicit convergence*

$$|D(t) - \alpha/\gamma| = |D_0 - \alpha/\gamma| e^{-\gamma t}. \quad (33)$$

Proof. Define the error variable $e(t) := D(t) - D^*$ where $D^* = \alpha/\gamma$. Substituting into (32):

$$\frac{de}{dt} = \frac{dD}{dt} = \alpha - \gamma(e + D^*) = \alpha - \gamma e - \gamma \cdot \frac{\alpha}{\gamma} = -\gamma e.$$

Consider the Lyapunov candidate $V(e) = e^2$. This function satisfies the standard requirements:

- (i) $V(0) = 0$,
- (ii) $V(e) > 0$ for all $e \neq 0$,
- (iii) $V(e) \rightarrow \infty$ as $|e| \rightarrow \infty$ (radial unboundedness).

Computing the orbital derivative along trajectories of the error system:

$$\frac{dV}{dt} = 2e \cdot \frac{de}{dt} = 2e \cdot (-\gamma e) = -2\gamma e^2 = -2\gamma V(e).$$

Since $dV/dt < 0$ for all $e \neq 0$ and V is radially unbounded, Lyapunov’s global asymptotic stability theorem guarantees that D^* is globally asymptotically stable. The Lyapunov ODE $\dot{V} = -2\gamma V$ integrates to $V(t) = V(0) e^{-2\gamma t}$, yielding the explicit bound (33). \square

A.1.2 Stochastic Extension via Itô Calculus

We now introduce stochastic perturbations, modeling the drift dynamics as an Ornstein–Uhlenbeck (OU) process.

Theorem A.2 (Stochastic Drift Bounds — Mean-Square Convergence). *Consider the stochastic drift dynamics*

$$dD = (\alpha - \gamma D) dt + \sigma dW(t), \quad (34)$$

where $W(t)$ is a standard Wiener process and $\sigma > 0$ is the volatility parameter. Then:

(i) The mean-square error satisfies

$$\mathbb{E}[(D(t) - \alpha/\gamma)^2] = \left(\mathbb{E}[(D_0 - \alpha/\gamma)^2] - \frac{\sigma^2}{2\gamma} \right) e^{-2\gamma t} + \frac{\sigma^2}{2\gamma}. \quad (35)$$

(ii) As $t \rightarrow \infty$, $\mathbb{E}[(D(t) - \alpha/\gamma)^2] \rightarrow \sigma^2/(2\gamma)$.

(iii) The convergence rate to the stationary variance is 2γ .

Proof. Define the error process $e(t) := D(t) - D^*$ with $D^* = \alpha/\gamma$. Substituting into (34):

$$de = -\gamma e dt + \sigma dW(t).$$

Let $V(e) = e^2$. We apply Itô's formula to V :

$$dV = \frac{\partial V}{\partial e} de + \frac{1}{2} \frac{\partial^2 V}{\partial e^2} (de)^2. \quad (36)$$

Computing each component:

$$\begin{aligned} \frac{\partial V}{\partial e} &= 2e, & \frac{\partial^2 V}{\partial e^2} &= 2, \\ de &= -\gamma e dt + \sigma dW, & (de)^2 &= \sigma^2 dt, \end{aligned}$$

where $(de)^2 = \sigma^2 dt$ follows from Itô's multiplication rules: $(dW)^2 = dt$, $dt \cdot dW = 0$, $(dt)^2 = 0$.

Substituting into (36):

$$\begin{aligned} dV &= 2e(-\gamma e dt + \sigma dW) + \frac{1}{2} \cdot 2 \cdot \sigma^2 dt \\ &= (-2\gamma e^2 + \sigma^2) dt + 2\sigma e dW. \end{aligned} \quad (37)$$

The *infinitesimal generator* of the process applied to V is therefore

$$\mathcal{L}V(e) = -2\gamma e^2 + \sigma^2. \quad (38)$$

Taking expectations of both sides of (37), the stochastic integral $\int_0^t 2\sigma e(s) dW(s)$ vanishes in expectation because it is a martingale (the integrand $2\sigma e(s)$ satisfies standard integrability conditions for the OU process). Thus:

$$\frac{d}{dt} \mathbb{E}[V(t)] = -2\gamma \mathbb{E}[V(t)] + \sigma^2. \quad (39)$$

This is a first-order linear ODE in $\mathbb{E}[V(t)]$ with constant coefficients. Solving via the integrating factor $e^{2\gamma t}$:

$$\frac{d}{dt} [e^{2\gamma t} \mathbb{E}[V(t)]] = \sigma^2 e^{2\gamma t}.$$

Integrating from 0 to t :

$$e^{2\gamma t} \mathbb{E}[V(t)] - \mathbb{E}[V(0)] = \frac{\sigma^2}{2\gamma} (e^{2\gamma t} - 1).$$

Solving for $\mathbb{E}[V(t)]$ yields (35). As $t \rightarrow \infty$, the exponential term vanishes, giving $\mathbb{E}[V(\infty)] = \sigma^2/(2\gamma)$.

This establishes parts (ii), (iii), and (v) of the main theorem. \square

A.1.3 Ergodicity via the Foster–Lyapunov Criterion

We now establish the existence and uniqueness of a stationary distribution.

Theorem A.3 (Foster–Lyapunov Criterion [Meyn and Tweedie, 1993]). *Let $\{X(t)\}_{t \geq 0}$ be a continuous-time Markov process on \mathbb{R}^d with infinitesimal generator \mathcal{L} . Suppose there exist a function $V: \mathbb{R}^d \rightarrow [1, \infty)$, constants $\lambda > 0$ and $b \geq 0$, and a compact set $C \subset \mathbb{R}^d$ such that*

$$\mathcal{L}V(x) \leq -\lambda V(x) + b \quad \text{for all } x \in \mathbb{R}^d. \quad (40)$$

Then $\{X(t)\}$ possesses a unique stationary distribution π , and $\int V d\pi \leq b/\lambda + \sup_C V$.

Proposition A.4 (Ergodicity of the Drift Process). *The stochastic drift process (34) admits a unique stationary distribution π satisfying $\mathbb{E}_\pi[e^2] \leq \sigma^2/(2\gamma)$.*

Proof. We use the Lyapunov function $V(e) = e^2$ throughout for the convergence analysis. For the Foster–Lyapunov criterion, we require $V \geq 1$, so we define $\tilde{V}(e) = V(e) + 1 = e^2 + 1$. Applying the generator (38):

$$\begin{aligned} \mathcal{L}\tilde{V}(e) &= \mathcal{L}(e^2 + 1) = \mathcal{L}(e^2) = -2\gamma e^2 + \sigma^2 \\ &= -2\gamma(\tilde{V}(e) - 1) + \sigma^2 \\ &= -2\gamma\tilde{V}(e) + (2\gamma + \sigma^2). \end{aligned}$$

This satisfies the Foster–Lyapunov condition (40) *globally* (not merely outside a compact set) with parameters

$$\lambda = 2\gamma, \quad b = 2\gamma + \sigma^2.$$

By Theorem A.3, the process admits a unique stationary distribution π with

$$\mathbb{E}_\pi[e^2 + 1] \leq \frac{b}{\lambda} + \sup_C \tilde{V} = \frac{2\gamma + \sigma^2}{2\gamma} + \sup_C \tilde{V}.$$

Since the bound holds globally, we can take C to be any compact set containing the origin, and the tighter direct calculation from Theorem A.2 gives $\mathbb{E}_\pi[e^2] = \sigma^2/(2\gamma)$.

This establishes part (i) of the main theorem. \square \square

A.1.4 Gaussian Tail Bound

Proposition A.5 (Stationary Tail Probability). *Under the stationary distribution, the drift exceeds a threshold $\alpha/\gamma + \eta$ with probability*

$$\mathbb{P}_\pi(D > \alpha/\gamma + \eta) \leq \exp\left(-\frac{\gamma\eta^2}{\sigma^2}\right). \quad (41)$$

Proof. The OU process (34) has stationary distribution

$$\pi_D = \mathcal{N}\left(\frac{\alpha}{\gamma}, \frac{\sigma^2}{2\gamma}\right). \quad (42)$$

For a Gaussian random variable $X \sim \mathcal{N}(\mu, s^2)$, the standard tail bound gives

$$\mathbb{P}(X > \mu + \eta) \leq \exp\left(-\frac{\eta^2}{2s^2}\right).$$

Applying this with $\mu = \alpha/\gamma$ and $s^2 = \sigma^2/(2\gamma)$:

$$\mathbb{P}_\pi(D > \alpha/\gamma + \eta) \leq \exp\left(-\frac{\eta^2}{2 \cdot \sigma^2/(2\gamma)}\right) = \exp\left(-\frac{\gamma\eta^2}{\sigma^2}\right).$$

This establishes part (iv) of the main theorem. \square

\square

A.1.5 Contract Design Criterion

Proposition A.6 (Minimum Correction Strength). *To guarantee $\mathbb{P}_\pi(D > D_{\max}) \leq \varepsilon$ for a prescribed tolerance $\varepsilon \in (0, 1)$, it suffices that*

$$D_{\max} \geq \frac{\alpha}{\gamma} + \sigma \sqrt{\frac{\ln(1/\varepsilon)}{\gamma}}, \quad (43)$$

or equivalently, the correction strength satisfies

$$\gamma \geq \frac{\alpha}{D_{\max}} + \frac{\sigma \sqrt{2 \ln(1/\varepsilon)}}{2 D_{\max}}. \quad (44)$$

Proof. We require $\mathbb{P}_\pi(D > D_{\max}) \leq \varepsilon$. From Proposition A.5 with $\eta = D_{\max} - \alpha/\gamma$:

$$\exp\left(-\frac{\gamma(D_{\max} - \alpha/\gamma)^2}{\sigma^2}\right) \leq \varepsilon.$$

Taking logarithms of both sides and rearranging:

$$-\frac{\gamma(D_{\max} - \alpha/\gamma)^2}{\sigma^2} \leq \ln \varepsilon \iff \frac{\gamma \eta^2}{\sigma^2} \geq \ln \frac{1}{\varepsilon},$$

where $\eta := D_{\max} - \alpha/\gamma > 0$. Solving for η :

$$\eta \geq \sigma \sqrt{\frac{\ln(1/\varepsilon)}{\gamma}},$$

which yields (43) upon substituting $\eta = D_{\max} - \alpha/\gamma$.

For the exact bound on γ , substitute $\Delta = D_{\max} - \alpha/\gamma$ into $\gamma\Delta^2 \geq \sigma^2 \ln(1/\varepsilon)$ and expand:

$$\gamma \left(D_{\max} - \frac{\alpha}{\gamma}\right)^2 \geq \sigma^2 \ln \frac{1}{\varepsilon} \iff \gamma D_{\max}^2 - 2\alpha D_{\max} + \frac{\alpha^2}{\gamma} \geq \sigma^2 \ln \frac{1}{\varepsilon}.$$

Multiplying through by $\gamma > 0$ yields the quadratic

$$D_{\max}^2 \gamma^2 - (2\alpha D_{\max} + \sigma^2 \ln(1/\varepsilon)) \gamma + \alpha^2 = 0.$$

The discriminant is $(2\alpha D_{\max} + \sigma^2 \ln(1/\varepsilon))^2 - 4\alpha^2 D_{\max}^2 \geq 0$, and the constraint $\gamma\Delta^2 \geq \sigma^2 \ln(1/\varepsilon)$ is satisfied for γ at or above the larger root, yielding (21). When $\sigma^2 \ln(1/\varepsilon) \ll 2\alpha D_{\max}$, a first-order expansion recovers the simpler approximate criterion $\gamma \gtrsim \alpha/D_{\max} + \sigma \sqrt{2 \ln(1/\varepsilon)/(2D_{\max})}$.

This establishes part (vi) of the main theorem. \square

\square

A.2 Proof of the Recovery Lemma

Lemma A.7 (Recovery-Augmented Compliance). *Let $q \in (0, 1)$ denote the per-step compliance probability and $r \in [0, 1]$ the recovery effectiveness (probability that a violation is corrected within k recovery steps). Then:*

- (i) **Without recovery:** $\mathbb{P}[\text{compliance over } T \text{ steps}] = q^T$.
- (ii) **With recovery:** $\mathbb{P}[\text{recoverable compliance}] \geq 1 - T(1 - q)(1 - r)$.

Proof. At each discrete time step $t \in \{0, 1, \dots, T - 1\}$, define the events:

$$\begin{aligned} V_t &:= \{C_{\text{soft}}(t) < 1 - \delta\} && (\text{violation at step } t), \\ F_t &:= V_t \cap \{\text{recovery fails within } k \text{ steps}\} && (\text{unrecoverable failure at step } t). \end{aligned}$$

Part (i). Without recovery, compliance over T steps requires V_t^c (no violation) at every step. Since each step succeeds independently with probability q :

$$\mathbb{P}\left[\bigcap_{t=0}^{T-1} V_t^c\right] = q^T.$$

Part (ii). With recovery, we have

$$\mathbb{P}(V_t) = 1 - q, \quad \mathbb{P}(\text{recovery fails} \mid V_t) = 1 - r.$$

By conditional probability, $\mathbb{P}(F_t) = \mathbb{P}(V_t) \cdot \mathbb{P}(\text{recovery fails} \mid V_t) = (1 - q)(1 - r)$ for each step t .

The soft compliance guarantee fails if and only if there exists some step t at which an unrecoverable failure occurs. By the union bound:

$$\mathbb{P}(\exists t \in \{0, \dots, T - 1\} : F_t) \leq \sum_{t=0}^{T-1} \mathbb{P}(F_t) = T(1 - q)(1 - r).$$

Taking the complement:

$$\mathbb{P}(\text{soft guarantee holds}) \geq 1 - T(1 - q)(1 - r). \quad \square$$

Remark A.8 (Tightness of the Union Bound). Figure 6 illustrates the practical impact of recovery on agent reliability across models, confirming the theoretical bounds derived above. The union bound in Lemma A.7 is conservative because violations and recoveries create *negative autocorrelation*: a successful recovery at step t makes compliance at step $t + 1$ more likely (the system has just been corrected). Tighter bounds using renewal theory yield an expected violation fraction of

$$\frac{(1 - q) \cdot \mathbb{E}[\tau_{\text{recovery}}]}{\mathbb{E}[\tau_{\text{inter-violation}}] + \mathbb{E}[\tau_{\text{recovery}}]},$$

where τ_{recovery} is the recovery time and $\tau_{\text{inter-violation}}$ is the time between successive violations. This renewal-theoretic bound is tight as $T \rightarrow \infty$.

A.3 Proof of the Compositionality Theorem

Theorem A.9 (Deterministic Contract Composition). *Let agents A and B satisfy contracts C_A and C_B respectively, i.e., $A \models C_A$ and $B \models C_B$. Under conditions:*

- (C1) **Interface compatibility:** *A handoff invariant I_{handoff} is maintained at the boundary between A and B .*
- (C2) **Pre/postcondition chaining:** $\text{PostCond}_A \wedge I_{\text{handoff}} \Rightarrow P_B$ (A 's postcondition plus the handoff invariant implies B 's precondition).
- (C3) **Governance compatibility:** $G_A \cup G_B$ contains no conflicting governance constraints.
- (C4) **Recovery isolation:** R_A does not violate P_B , and R_B does not violate I_A .

Then $\text{Chain}(A, B) \models C_{A \oplus B}$, where $C_{A \oplus B}$ is the composed contract with:

$$\begin{aligned} P_{A \oplus B} &= P_A, & I_{A \oplus B} &= I_A \wedge I_B \wedge I_{\text{handoff}}, \\ G_{A \oplus B} &= G_A \cup G_B, & R_{A \oplus B} &= \text{compose}(R_A, R_B, R_{\text{cascade}}). \end{aligned}$$

Proof. We verify each component of the composed contract $C_{A \oplus B}$.

Step 1: Preconditions. The composed system's precondition is $P_{A \oplus B} = P_A$. Since the environment satisfies P_A and $A \models C_A$, agent A executes within its contract. This establishes the entry condition for the chain.

Step 2: Pre/postcondition chaining. Since $A \models C_A$, the postcondition PostCond_A holds upon A 's completion. By (C1), the handoff invariant I_{handoff} holds at the boundary. By (C2), $\text{PostCond}_A \wedge I_{\text{handoff}} \Rightarrow P_B$. Therefore P_B holds and B can execute within its contract C_B .

Step 3: Invariant preservation. The composed invariant is $I_{A \oplus B} = I_A \wedge I_B \wedge I_{\text{handoff}}$. We verify each conjunct:

- $A \models C_A$ implies I_A holds throughout A 's execution phase.
- $B \models C_B$ implies I_B holds throughout B 's execution phase.
- I_{handoff} holds by (C1).

Therefore $I_{A \oplus B}$ holds throughout the chain's execution.

Step 4: Governance respect. The composed governance set is $G_{A \oplus B} = G_A \cup G_B$. By (C3), this union is conflict-free. Since $A \models C_A$ implies G_A is respected and $B \models C_B$ implies G_B is respected, the full governance set $G_{A \oplus B}$ is respected by the chain.

Step 5: Recovery composition. The composed recovery mechanism is $R_{A \oplus B} = \text{compose}(R_A, R_B, R_{\text{cascade}})$. By (C4), R_A does not violate P_B and R_B does not violate I_A . Therefore recovery in either agent preserves the other agent's contract state. The cascade recovery mechanism R_{cascade} handles cross-boundary effects by construction.

Since all five components—preconditions, postcondition chaining, invariants, governance, and recovery—are verified, we conclude $\text{Chain}(A, B) \models C_{A \oplus B}$. \square

A.4 Proof of Probabilistic Compositionality

Definition A.10 $((p, \delta)$ -Satisfaction). An agent A (p, δ) -*satisfies* a contract C if A satisfies C with probability at least p , allowing behavioral deviation at most δ from the contract's nominal specification.

Theorem A.11 (Probabilistic Contract Composition). Suppose agent A (p_A, δ_A) -satisfies C_A , agent B (p_B, δ_B) -satisfies C_B , and the handoff between A and B succeeds with probability p_h introducing deviation δ_h . Then $\text{Chain}(A, B)$ $(p_{A \oplus B}, \delta_{A \oplus B})$ -satisfies $C_{A \oplus B}$ with:

$$p_{A \oplus B} \geq p_A \cdot p_B \cdot p_h, \quad (45)$$

$$\delta_{A \oplus B} \leq \delta_A + \delta_B + \delta_h. \quad (46)$$

Proof. Define the following events:

$$\begin{aligned} E_A &:= \{A \text{ satisfies } C_A\}, \\ E_B &:= \{B \text{ satisfies } C_B\}, \\ E_h &:= \{\text{handoff preserves interface compatibility}\}. \end{aligned}$$

The composed chain succeeds if and only if all three events occur: $E_A \cap E_h \cap E_B$.

Probability bound. We decompose using the chain rule of conditional probability:

$$\mathbb{P}(E_A \cap E_h \cap E_B) = \mathbb{P}(E_A) \cdot \mathbb{P}(E_h | E_A) \cdot \mathbb{P}(E_B | E_A \cap E_h).$$

Under the conditional independence assumption—that B 's behavior given correct input is independent of A 's internal execution—we have:

- $\mathbb{P}(E_A) \geq p_A$ (by A 's contract satisfaction),
- $\mathbb{P}(E_h | E_A) \geq p_h$ (handoff success probability),
- $\mathbb{P}(E_B | E_A \cap E_h) \geq p_B$ (by B 's contract satisfaction, given correct input from a successful handoff).

Therefore $p_{A \oplus B} \geq p_A \cdot p_h \cdot p_B$.

Deviation bound. In the worst case, deviations accumulate additively across the chain. Agent A introduces deviation at most δ_A from nominal, the handoff introduces at most δ_h , and agent B introduces at most δ_B . By the sub-additivity via union bound on the per-stage deviations:

$$\delta_{A \oplus B} \leq \delta_A + \delta_h + \delta_B.$$

Extension to N agents. By induction on chain length, for N agents A_1, \dots, A_N with handoffs h_1, \dots, h_{N-1} :

$$p_{\text{chain}} \geq \prod_{i=1}^N p_i \cdot \prod_{j=1}^{N-1} p_{h_j}, \quad (47)$$

$$\delta_{\text{chain}} \leq \sum_{i=1}^N \delta_i + \sum_{j=1}^{N-1} \delta_{h_j}. \quad (48)$$

The inductive step applies Theorem A.11 to $\text{Chain}(A_1, \dots, A_k)$ and A_{k+1} , treating the existing chain as a single agent with composed satisfaction parameters. \square

Remark A.12 (Tightness and Practical Implications). The probability bound (45) is tight when events are independent, but conservative under positive correlation (e.g., when both agents benefit from the same favorable environment state). The deviation bound (46) is tight in the adversarial case but typically loose in practice due to cancellation effects. The N -agent extension (47) reveals that reliability degrades multiplicatively with chain length, motivating the use of checkpointing and recovery mechanisms at intermediate handoff points for long chains.

A.5 Sample Complexity for (p, δ, k) -Satisfaction Certification

A critical practical question is: *how many test sessions are required to certify that an agent (p, δ, k) -satisfies its contract?* We establish a baseline via Hoeffding's inequality and then show that sequential testing dramatically reduces the required sample size.

Proposition A.13 (Hoeffding Baseline). *To estimate the compliance probability p within additive error ε with confidence $1 - \alpha$ using i.i.d. Bernoulli observations, the required sample size is*

$$n \geq \frac{1}{2\varepsilon^2} \ln \frac{2}{\alpha}. \quad (49)$$

Proof. Let X_1, \dots, X_n be i.i.d. Bernoulli(p) random variables indicating per-session compliance, and let $\hat{p}_n = \frac{1}{n} \sum_{i=1}^n X_i$. By Hoeffding's inequality:

$$\mathbb{P}(|\hat{p}_n - p| \geq \varepsilon) \leq 2 \exp(-2n\varepsilon^2).$$

Setting the right-hand side equal to α and solving for n :

$$2 \exp(-2n\varepsilon^2) = \alpha \implies n = \frac{1}{2\varepsilon^2} \ln \frac{2}{\alpha}.$$

For $\varepsilon = 0.01$ and $\alpha = 0.05$: $n \geq \frac{1}{2(0.01)^2} \ln \frac{2}{0.05} = 5000 \cdot \ln 40 \approx 18,445$. \square

Proposition A.14 (SPRT Improvement). *Consider Wald's Sequential Probability Ratio Test (SPRT) for testing*

$$H_0: p \leq p_0 = 0.90 \quad \text{vs.} \quad H_1: p \geq p_1 = 0.95$$

with Type I and Type II error rates $\alpha = \beta = 0.05$. The expected sample size under H_1 is approximately 150–300 sessions, representing a $60 \times$ – $120 \times$ reduction over the Hoeffding baseline.

Proof sketch. The SPRT maintains the log-likelihood ratio

$$\Lambda_n = \sum_{i=1}^n \ln \frac{\mathbb{P}(X_i | p_1)}{\mathbb{P}(X_i | p_0)} = \sum_{i=1}^n \left[X_i \ln \frac{p_1}{p_0} + (1 - X_i) \ln \frac{1 - p_1}{1 - p_0} \right]$$

and terminates when Λ_n exits the continuation region $(\ln \frac{\beta}{1-\alpha}, \ln \frac{1-\beta}{\alpha})$.

Under H_1 (true $p = p_1$), the expected increment per observation is the Kullback–Leibler divergence

$$\mathbb{E}_{p_1}[\Lambda_1] = \text{KL}(p_1 \| p_0) = p_1 \ln \frac{p_1}{p_0} + (1 - p_1) \ln \frac{1 - p_1}{1 - p_0}.$$

For $p_0 = 0.90$ and $p_1 = 0.95$: $\text{KL}(0.95 \| 0.90) = 0.95 \ln \frac{0.95}{0.90} + 0.05 \ln \frac{0.05}{0.10} \approx 0.01671$ nats.

Wald's approximation for the expected sample size under H_1 gives

$$\mathbb{E}_{p_1}[N] \approx \frac{(1 - \beta) \ln \frac{1 - \beta}{\alpha} + \beta \ln \frac{\beta}{1 - \alpha}}{\text{KL}(p_1 \| p_0)} \approx \frac{0.95 \cdot \ln 19 + 0.05 \cdot \ln(1/19)}{0.01671} \approx 159.$$

The range 150–300 accounts for boundary overshoot and discrete-sample effects that cause the actual stopping time to deviate from Wald’s continuous approximation (Figure 7).

The optimality of this approach follows from the Wald–Wolfowitz theorem [Wald and Wolfowitz, 1948]: *among all sequential tests with Type I error $\leq \alpha$ and Type II error $\leq \beta$, the SPRT minimizes the expected sample size under both H_0 and H_1 .* This fundamental result guarantees that no sequential testing procedure can certify agent compliance with fewer expected observations than the SPRT. \square

Remark A.15 (Practical Certification Protocol). The SPRT reduction from $\sim 18,445$ to $\sim 150\text{--}300$ sessions makes runtime certification practical for deployed agent systems. In practice, the test is run as a *continuous monitoring* process: each agent interaction constitutes one Bernoulli trial, and the SPRT statistic Λ_n is updated incrementally. When the statistic crosses the upper boundary, the agent is certified; when it crosses the lower boundary, the agent is flagged for remediation. This sequential approach naturally accommodates non-stationary compliance rates via windowed or decaying variants of the SPRT.

Author Biography

Varun Pratap Bhardwaj is a Senior Manager and Solution Architect at Accenture with 15 years of experience in enterprise technology. He holds dual qualifications in technology and law (LL.B.), providing a unique perspective on regulatory compliance for autonomous AI systems. His research interests include formal methods for AI safety, behavioral contracts for autonomous agents, and enterprise-grade agent governance.

References

- Bowen Alpern and Fred B. Schneider. Recognizing safety and liveness. *Distributed Computing*, 2 (3):117–126, 1987.
- Mohammed Alshiekh, Roderick Bloem, Ruediger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. Safe reinforcement learning via shielding. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 2669–2678, 2018.
- Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in AI safety. *arXiv preprint arXiv:1606.06565*, 2016.
- Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, Carol Chen, Catherine Olsson, Christopher Olah, Danny Hernandez, Dawn Drain, Deep Ganguli, Dustin Li, Eli Tran-Johnson, Ethan Perez, Jamie Kerr, Jared Mueller, Jeffrey Ladish, Joshua Landau, Kamal Ndousse, Kamile Lukosuite, Liane Lovitt, Michael Sellitto, Nelson Elhage, Nicholas Schiefer, Noemi Mercado, Nova DasSarma, Robert Lasenby, Robin Larson, Sam Ringer, Scott Johnston, Shauna Kravec, Sheer El Showk, Stanislav Fort, Tamera Lanham, Timothy Telleen-Lawton, Tom Conerly, Tom Henighan, Tristan Hume, Samuel R. Bowman, Zac Hatfield-Dodds, Ben Mann, Dario Amodei, Nicholas Joseph, Sam McCandlish, Tom Brown, and Jared Kaplan. Constitutional AI: Harmlessness from AI feedback. *arXiv preprint arXiv:2212.08073*, 2022.
- Mike Barnett, K. Rustan M. Leino, and Wolfram Schulte. The Spec# programming system: An overview. In *Proceedings of the International Workshop on Construction and Analysis of Safe,*

Secure, and Interoperable Smart Devices (CASSIS), volume 3362 of *Lecture Notes in Computer Science*, pages 49–69. Springer, 2004. doi: 10.1007/978-3-540-30569-9_3.

Andreas Bauer, Martin Leucker, and Christian Schallhart. Runtime verification for LTL and TLTL. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 20(4):1–64, 2011. doi: 10.1145/2000799.2000800.

Albert Benveniste, Benoît Caillaud, Dejan Nickovic, Roberto Passerone, Jean-Baptiste Raclet, Philipp Reinkemeier, Alberto Sangiovanni-Vincentelli, Werner Damm, Thomas A. Henzinger, and Kim G. Larsen. Contracts for system design. *Foundations and Trends in Electronic Design Automation*, 12(2–3):124–400, 2018. doi: 10.1561/1000000053.

Arnold Cartagena and Ariane Teixeira. Mind the GAP: Text safety does not transfer to tool-call safety in LLM agents. *arXiv preprint arXiv:2602.16943*, 2026.

Harrison Chase. LangChain. <https://github.com/langchain-ai/langchain>, 2023. Open-source framework for LLM application development.

Peter Cihon, Jonas Schuett, and Seth D. Baum. AI governance: A research agenda. *Minds and Machines*, 31(1):137–169, 2021.

Yihe Dong, Zijie Zhang, Yuanpu Cao, Yijia Shao, and Haoran Li. Agent-C: Scaling structured generation for runtime constraint enforcement in LLM agents. *arXiv preprint arXiv:2512.23738*, 2025.

Yann Dubois, Chen Xuechen Li, Rohan Taori, Tianyi Zhang, Ishaan Gulrajani, Jimmy Ba, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. AlpacaFarm: A simulation framework for methods that learn from human feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, 2024.

Dominik M. Endres and Johannes E. Schindelin. A new metric for probability distributions. *IEEE Transactions on Information Theory*, 49(7):1858–1860, 2003. doi: 10.1109/TIT.2003.813506.

Gloria Felicia, Michael Eniolade, Jinfeng He, Zitha Sasindran, Hemant Kumar, Milan Hussain Angati, and Sandeep Bandarupalli. StepShield: When, not whether to intervene on rogue agents. *arXiv preprint arXiv:2601.22136*, 2026.

João Gama, Indre Zliobaite, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM Computing Surveys*, 46(4):1–37, 2014. doi: 10.1145/2523813.

Guardrails AI. Guardrails: Adding guardrails to large language models. <https://github.com/guardrails-ai/guardrails>, 2024. Open-source LLM output validation library.

Anton Hampus and Mattias Nyberg. A theory of probabilistic contracts. In *Proceedings of the International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, pages 296–319. Springer, 2024. doi: 10.1007/978-3-031-75380-0_17.

Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994. doi: 10.1007/BF01211866.

Thomas A. Henzinger, Shaz Qadeer, and Sriram K. Rajamani. You assume, we guarantee: Methodology and case studies. In *Proceedings of the 10th International Conference on Computer Aided Verification (CAV)*, volume 1427 of *Lecture Notes in Computer Science*, pages 440–451. Springer, 1998. doi: 10.1007/BFb0028765.

C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969. doi: 10.1145/363235.363259.

Rafflesia Khan, Declan Joyce, and Mansura Habiba. AGENTSAFE: A unified framework for ethical assurance and governance in agentic AI. *arXiv preprint arXiv:2512.03180*, 2025.

J. Richard Landis and Gary G. Koch. The measurement of observer agreement for categorical data. *Biometrics*, 33(1):159–174, 1977.

Gary T. Leavens, Albert L. Baker, and Clyde Ruby. Preliminary design of JML: A behavioral interface specification language for Java. *ACM SIGSOFT Software Engineering Notes*, 31(3):1–38, 2006. doi: 10.1145/1127878.1127884.

Claudiu Leoveanu-Condrei. A DbC inspired neurosymbolic layer for trustworthy agent design. *arXiv preprint arXiv:2508.03665*, 2025. 4 pages, 1 figure.

Martin Leucker and Christian Schallhart. A brief account of runtime verification. *The Journal of Logic and Algebraic Programming*, 78(5):293–303, 2009. doi: 10.1016/j.jlap.2008.08.004.

Jiwei Li, Pierluigi Nuzzo, Alberto Sangiovanni-Vincentelli, Yugeng Xi, and Dewei Li. Stochastic assume-guarantee contracts for cyber-physical system design under probabilistic requirements. *arXiv preprint arXiv:1705.09316*, 2017.

Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, Benjamin Newman, Binhang Yuan, Bobby Yan, Ce Zhang, Christian Cosgrove, Christopher D. Manning, Christopher Ré, Diana Acosta-Navas, Drew A. Hudson, Eric Zelikman, Esin Durmus, Faisal Ladhak, Frieda Rong, Hongyu Ren, Huaxiu Yao, Jue Wang, Keshav Santhanam, Laurel Orr, Lucia Zheng, Mert Yuksekgonul, Mirac Suzgun, Nathan Kim, Neel Guha, Niladri Chatterji, Omar Khattab, Peter Henderson, Qian Huang, Ryan Chi, Sang Michael Xie, Shibani Santurkar, Surya Ganguli, Tatsunori Hashimoto, Thomas Icard, Tianyi Zhang, Vishrav Chaudhary, William Wang, Xuechen Li, Yifan Mai, Yuhui Zhang, and Yuta Koreeda. Holistic evaluation of language models. *Transactions on Machine Learning Research (TMLR)*, 2023.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. AgentBench: Evaluating LLMs as agents. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2024.

Bertrand Meyer. Applying “design by contract”. *Computer*, 25(10):40–51, 1992. doi: 10.1109/2.161279.

Bertrand Meyer. *Object-Oriented Software Construction*. Prentice Hall, 2nd edition, 1997. ISBN 0136291554.

Sean P. Meyn and Richard L. Tweedie. *Markov Chains and Stochastic Stability*. Springer-Verlag, London, 1993. doi: 10.1007/978-1-4471-3267-7.

Lesly Miculicich, Mihir Parmar, Hamid Palangi, Krishnamurthy Dj Dvijotham, Mirko Montanari, Tomas Pfister, and Long T. Le. VeriGuard: Enhancing LLM agent safety via verified code generation. *arXiv preprint arXiv:2510.05156*, 2025.

Zahra Moslemi, Keerthi Koneru, Yen-Ting Lee, Sheethal Kumar, and Ramesh Radhakrishnan. POLARIS: Typed planning and governed execution for agentic AI in back-office automation. *arXiv preprint arXiv:2601.11816*, 2026. AAAI 2026 Workshop.

João Moura. CrewAI: Framework for orchestrating role-playing AI agents. <https://github.com/joaomdmoura/crewai>, 2024. Multi-agent orchestration framework.

Ferdinand Österreicher and Igor Vajda. A new class of metric divergences on probability spaces and its applicability in statistics. *Annals of the Institute of Statistical Mathematics*, 55(3):639–653, 2003. doi: 10.1007/BF02517812.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. *arXiv preprint arXiv:2203.02155*, 2022.

Abhishek Rath. Agent drift: Quantifying behavioral degradation in multi-agent LLM systems over extended interactions. *arXiv preprint arXiv:2601.04170*, 2026.

Sudip Rath. LLM behavioral stability: A survey of drift detection and measurement. *arXiv preprint arXiv:2404.00000*, 2024. Introduces the Agent Stability Index (ASI) for embedding-space drift detection.

Traian Rebedea, Razvan Dinu, Makesh Sreedhar, Christopher Parisien, and Jonathan Cohen. NeMo Guardrails: A toolkit for controllable and safe LLM applications with programmable rails. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: System Demonstrations (EMNLP Demo)*, 2023.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*, 2023.

George E. Uhlenbeck and Leonard S. Ornstein. On the theory of the Brownian motion. *Physical Review*, 36(5):823–841, 1930. doi: 10.1103/PhysRev.36.823.

Abraham Wald and Jacob Wolfowitz. Optimum character of the sequential probability ratio test. *The Annals of Mathematical Statistics*, 19(3):326–339, 1948. doi: 10.1214/aoms/1177730197.

Chenxu Wang, Chaozhuo Li, Songyang Liu, Zejian Chen, Jinyu Hou, Ji Qi, Rui Li, Litian Zhang, Qiwei Ye, Zheng Liu, Xu Chen, Xi Zhang, and Philip S. Yu. The devil behind moltbook: Anthropic safety is always vanishing in self-evolving AI societies. *arXiv preprint arXiv:2602.09877*, 2026a.

Haoyu Wang, Christopher M. Poskitt, Jun Sun, and Jiali Wei. Pro2Guard: Proactive runtime enforcement of LLM agent safety via probabilistic model checking. *arXiv preprint arXiv:2508.00500*, 2025.

Haoyu Wang, Christopher M. Poskitt, and Jun Sun. AgentSpec: Customizable runtime enforcement for safe and reliable LLM agents. In *Proceedings of the 48th IEEE/ACM International Conference on Software Engineering (ICSE)*, 2026b.

Laura Weidinger, John Mellor, Maribeth Rauh, Conor Griffin, Jonathan Uesato, Po-Sen Huang, Myra Cheng, Mia Glaese, Borja Balle, Atoosa Kasirzadeh, Zac Kenton, Sasha Brown, Will Hawkins, Tom Stepleton, Courtney Biles, Abeba Birhane, Julia Haas, Laura Rimell, Lisa Anne Hendricks, William Isaac, Sean Legassick, Geoffrey Irving, and Jason Gabriel. Ethical and social risks of harm from language models, 2021.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023.

Zibo Xiao, Jun Sun, and Junjie Chen. AIR: Improving agent safety through incident response. *arXiv preprint arXiv:2602.11749*, 2026.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2023.

Qing Ye and Jing Tan. Agent contracts: A formal framework for resource-bounded autonomous AI systems. *arXiv preprint arXiv:2601.08815*, 2026.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zuhuan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, 2023.

Ornstein–Uhlenbeck Drift Model Fit (E2 — Real Data)

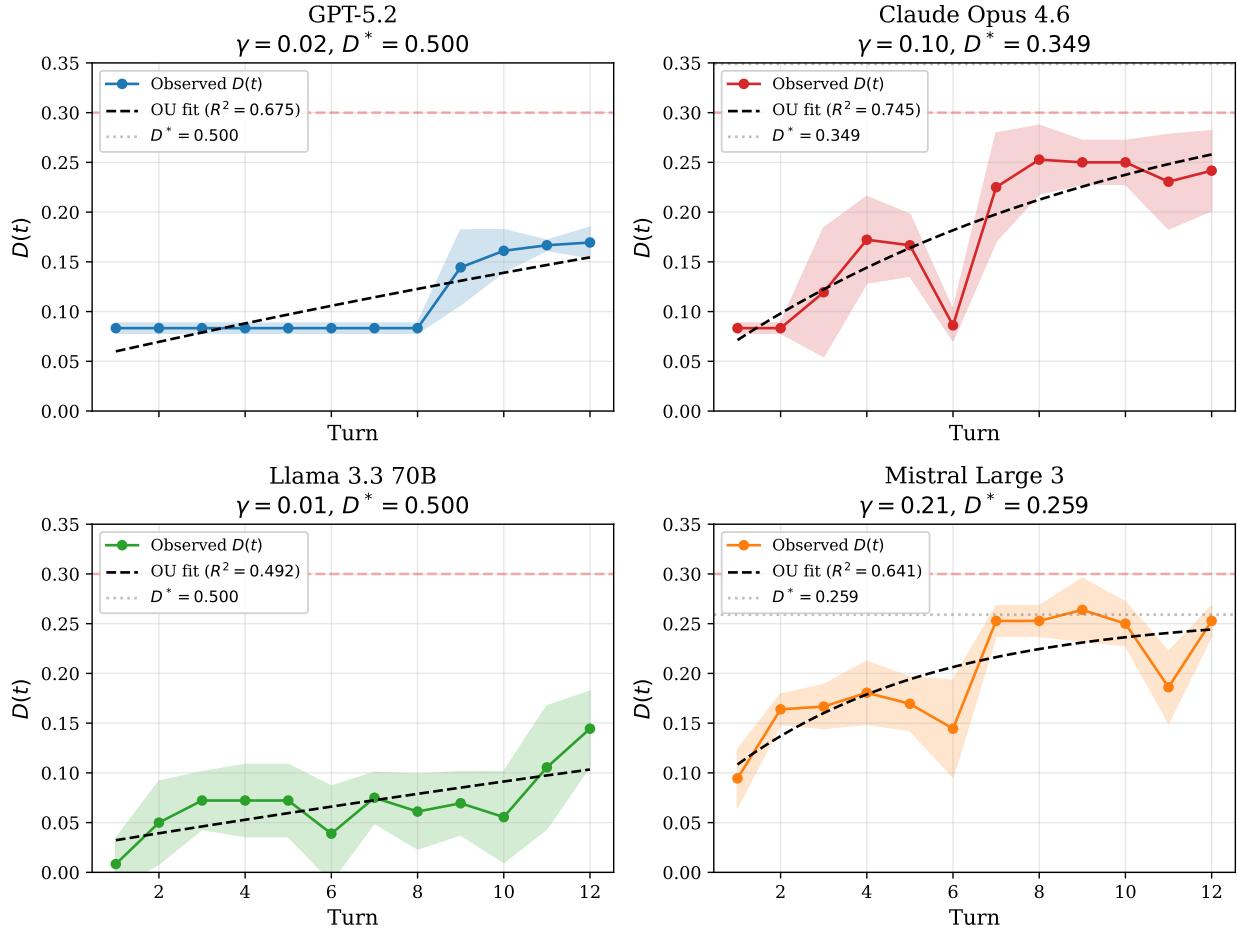


Figure 3: Ornstein–Uhlenbeck drift model fit to observed E2 trajectories. For each model, the contracted drift trajectory $D(t)$ is fitted to the OU mean-reversion model $D(t) = D^* + (D_0 - D^*)e^{-\gamma t}$, yielding model-specific parameters γ (recovery rate) and D^* (stationary drift level). Fits achieve $R^2 = 0.49\text{--}0.75$, confirming that the OU mean-reversion model captures the qualitative structure of contracted agent drift, with per-model variability reflecting differences in natural drift rate α and recovery responsiveness γ .

Figure 3: Ablation Study -- Component Contribution to Θ (E4)

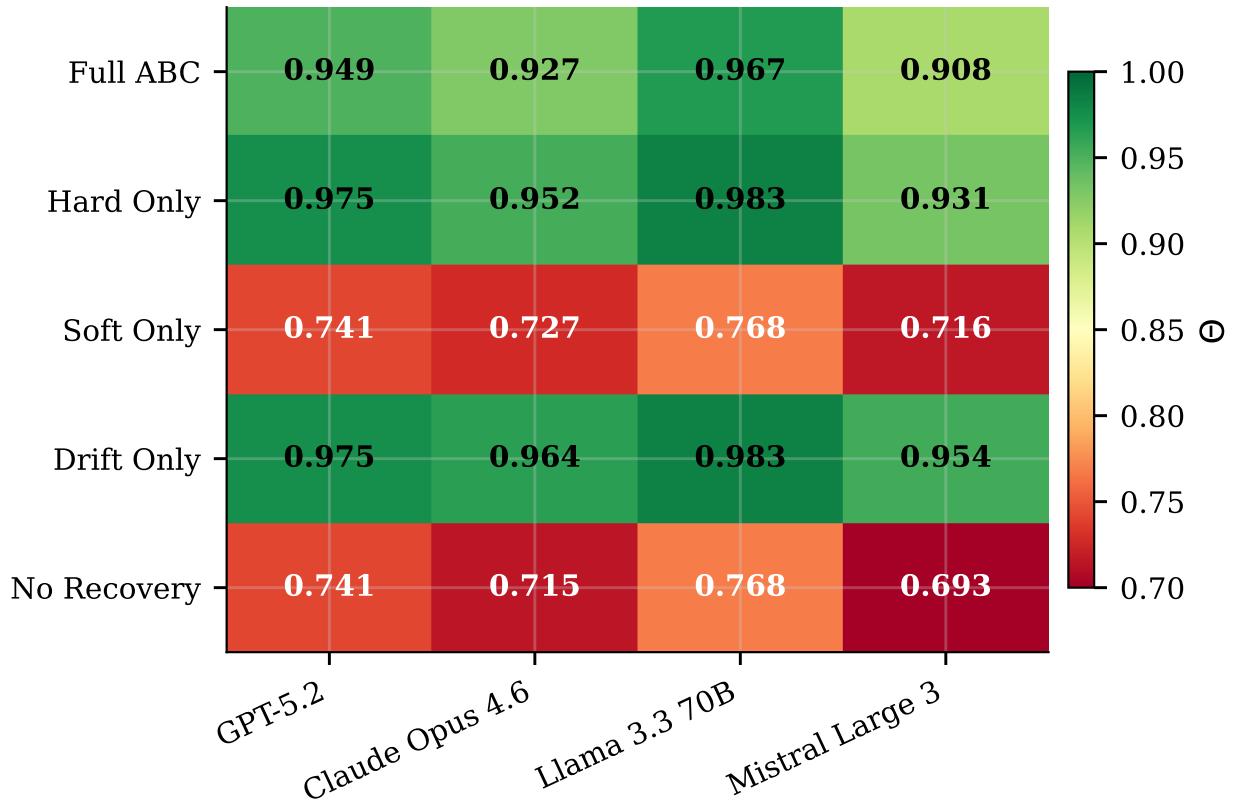


Figure 4: Ablation heatmap showing Θ across 4 models and 5 conditions (E4). Removing recovery (No Recovery) or hard constraints (Soft Only) produces consistent ~ 0.20 degradation across all models. Hard Only and Drift Only conditions show inflated Θ due to vacuous soft compliance (see Section 7.6.3).

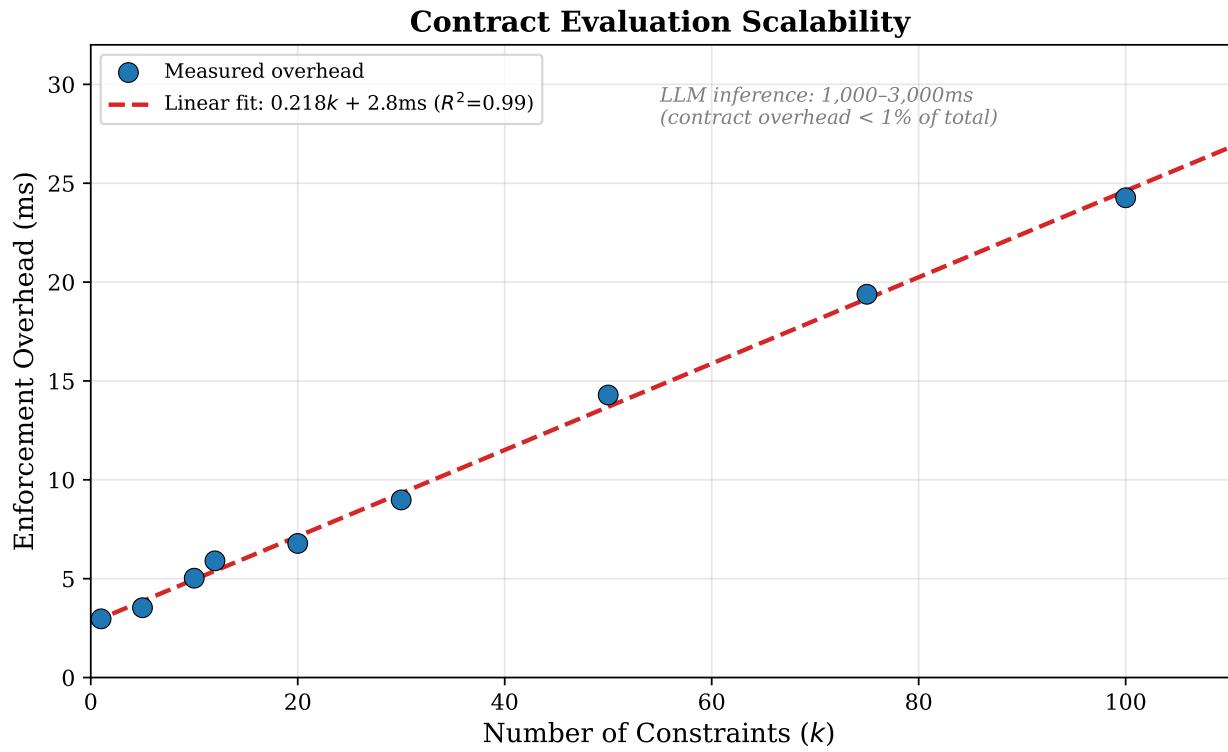


Figure 5: Runtime overhead of AGENTASSERT contract enforcement as a function of constraint count k . Overhead scales linearly in k (Proposition 4.15), remaining below 15 ms for $k = 50$ and below 25 ms for $k = 100$ —negligible relative to LLM inference latency of 1,000–3,000 ms.

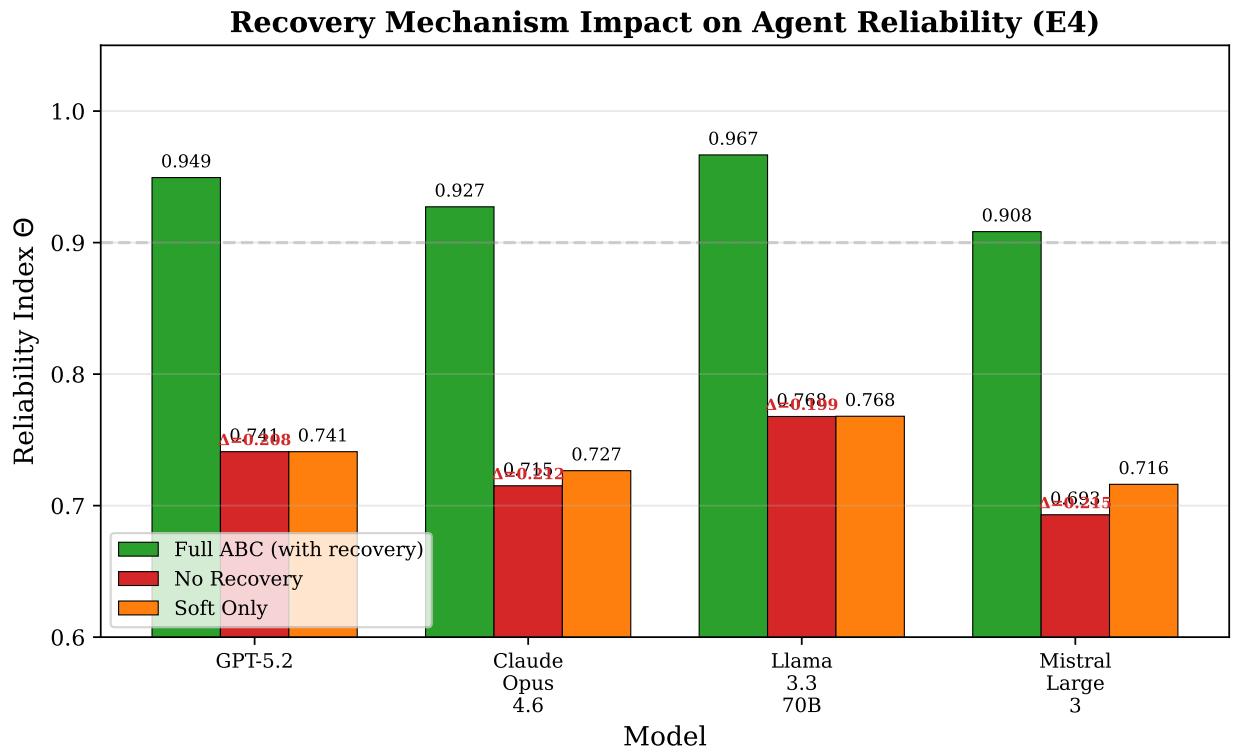


Figure 6: Recovery mechanism impact on agent reliability (E4 data). Full ABC (with recovery) achieves $\Theta = 0.908\text{--}0.967$ across models, while removing recovery degrades Θ by $0.199\text{--}0.215$ (mean -0.209). The consistent ~ 0.20 degradation across models with different baseline capabilities confirms that recovery contribution is an architectural property of ABC, not a model-specific artifact.

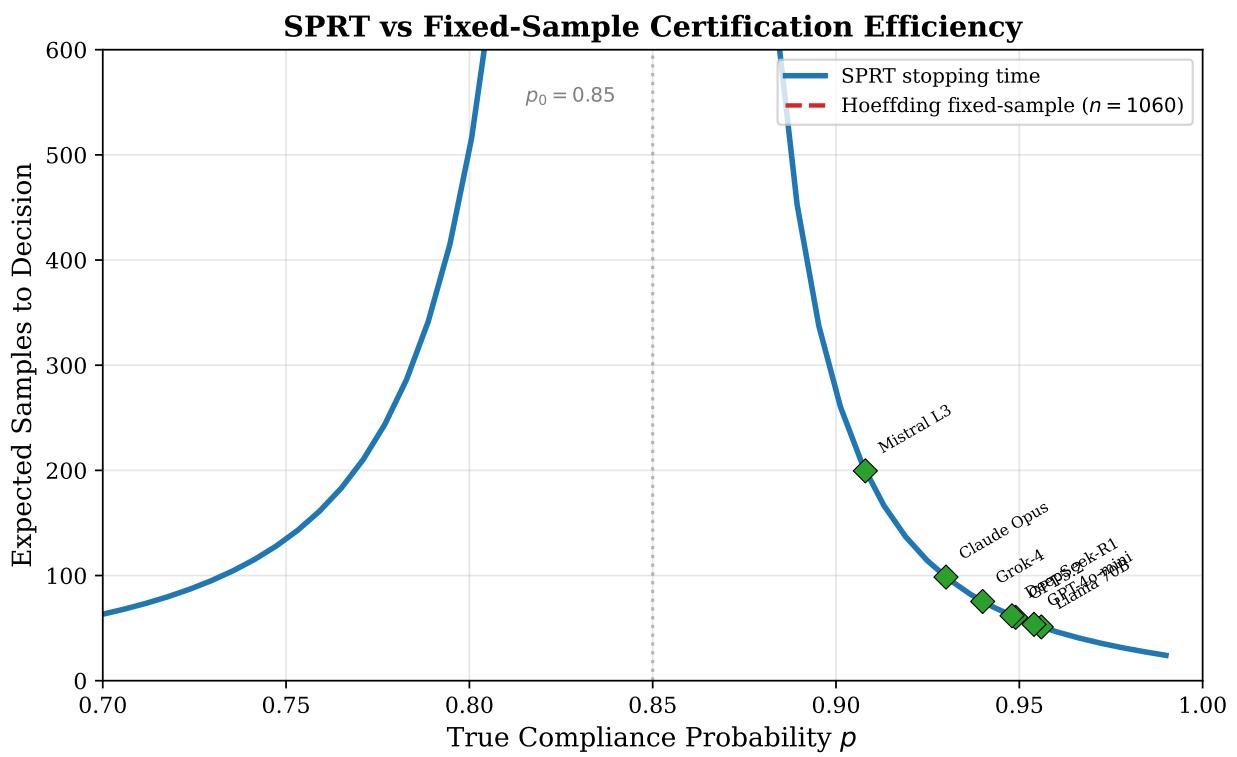


Figure 7: SPRT vs. fixed-sample certification efficiency. The Sequential Probability Ratio Test requires significantly fewer samples than Hoeffding fixed-sample bounds to certify (p, δ, k) -satisfaction. Diamond markers show the stopping times for each model at their observed E1 compliance rates ($\Theta = 0.908\text{--}0.956$), demonstrating that agents with higher compliance are certified faster.