Autonomous Systems Don’t Fail Suddenly: Runtime Governance for Autonomous Systems

Kathryn Louise Sughero

Sughero Systems LLC, Lombard, Illinois, USA

Abstract Current AI systems operate without an internal representation of their own execution stability. During runtime, they generate outputs without tracking how those outputs relate over time. Outputs are treated as independent predictions rather than components of a system under load. As coherence degrades, no mechanism exists to recognize or intervene. The primary risk is not that systems fail. It is that they continue operating while misaligned. This paper defines CHURPRIMO, a runtime governance framework for autonomous systems that monitors and enforces coherence during execution. The framework evaluates alignment across planning state, behavioral output, and execution trajectory, enabling early detection of divergence and phase-appropriate intervention. This work reframes failure as the result of unmanaged drift and positions runtime governance as a necessary condition for maintaining stability during extended autonomous execution.

Keywords Runtime governance, autonomous systems, multi-agent systems, system coherence, drift detection, execution stability, AI reliability

1 Introduction

Recent advances in artificial intelligence have improved model performance across a wide range of tasks. These gains expose a structural limitation. Performance does not guarantee stability during execution. Modern AI systems operate as ongoing processes rather than isolated interactions. Outputs accumulate, interact, and influence subsequent behavior. Small deviations compound over time, particularly in multi-agent and tool integrated environments.

Systems optimized for output accuracy are not equipped to maintain coherence under continuous execution. Runtime instability is not detected as it forms. Systems continue to produce valid looking outputs while internal alignment degrades.

Guardrails and monitoring act as passive constraints or post hoc observers. [1, 12] They do not maintain coherence as the system evolves.

This paper defines runtime governance as a structural layer during execution. It establishes coherence as the control variable, drift as the primary risk, and governance as the mechanism that maintains alignment over time. The focus shifts from isolated correctness to sustained stable operation in autonomous systems. In human-agent interaction contexts this requirement becomes critical. Systems must remain aligned with human intent across extended execution, not just at the point of output. This work shifts the unit of evaluation from isolated out-puts to execution trajectories and defines runtime governance as the mechanism that maintains coherence during continuous operation.

2 Runtime Instability in Autonomous Systems

Instability in autonomous systems does not present as immediate failure. It emerges as a progressive loss of co-herence during execution. Initial divergence occurs when outputs deviate from originating intent or prior trajec-tory. These deviations are subtle and remain uncorrected. As execution continues error compounds across steps in the absence of a mechanism that tracks coherence over time. As divergence increases the system's effective context becomes distorted. Subsequent decisions are made from a misaligned state rather than the original task conditions. This produces observable instability including inconsistent outputs, contradictory actions, and mis-alignment with objectives.

At a critical threshold the system enters rupture. Structural alignment is no longer preserved. The system continues to execute but from a corrupted trajectory. Systems do not become unstable at the point of failure. They become unstable at the point where drift is no longer recognized. This persistence under misalignment is the core risk. Downstream actions inherit the drifted trajectory amplifying error across execution.

From an external perspective failure appears sudden. Internally it is the result of accumulated unobserved di-vergence. Observed rupture conditions follow recurring patterns, including trajectory divergence, oscillatory be-havior, tool misuse, and context distortion. [8, 9] These are not isolated errors but expressions of underlying coherence loss. The absence of a mechanism that detects and responds to drift during execution is the defining limitation in current autonomous systems. Drift is not a discrete event; it is a condition that persists, com-pounds, and eventually restructures the system if left ungoverned.

3 Framework Runtime Governance for Coherence

CHURPRIMO operates as an inline runtime governance layer during system execution.

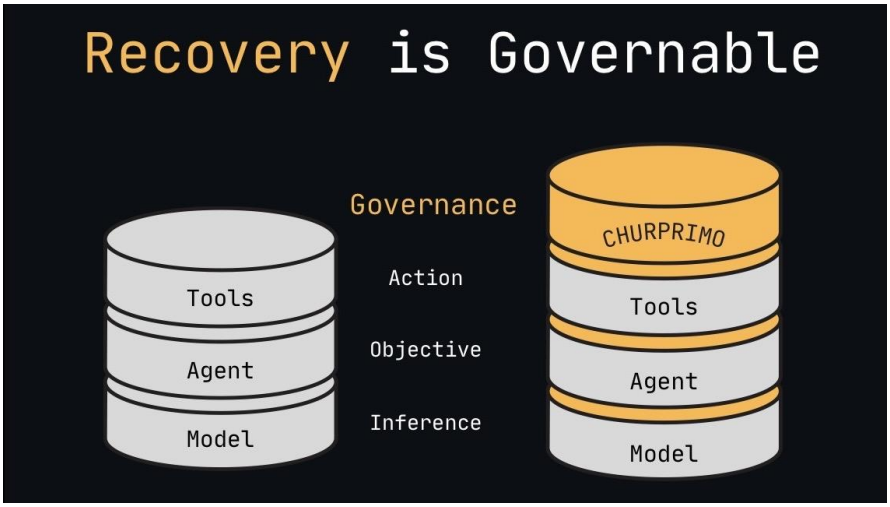

Fig. 1 CHURPRIMO governance layer within the autonomous system stack. The governance layer operates inline during execution observing and enforcing coherence across planning state, behavioral output, and execution trajectory without modifying the underlying model.

CHURPRIMO does not modify the underlying model and is not limited to external supervision. It operates within the execution loop between action generation and action commitment, enabling continuous observation, classification, and intervention.

The framework evaluates three concurrent signal domains. Planning state represents inferred intent structure and expected trajectory reconstructed from observable execution context. Behavioral output represents actions, messages, and tool use. Execution trajectory reflects how behavior evolves over time. Coherence is the observable alignment across these domains. Structural alignment is the underlying consistency that produces that co-herence.

Divergence emerges when consistency relationships between domains degrade over time. Divergence is com-puted in real time through a bounded continuous evaluation of cross domain signal alignment. Coherence is evaluated as a bounded state derived from alignment across coupled execution signals. Full specification is not included due to proprietary constraints. The framework evaluates divergence across planning state, behavioral output, and execution trajectory through continuously updated cross domain consistency relationships observed during execution.

Drift is not inferred after the fact. It is detected as it forms through divergence across coupled execution sig-nals. When divergence is detected, the system classifies it by severity. It distinguishes between transient noise, sustained drift, and structural

rupture. Classification is triggered by persistent deviation across steps or simultaneous deviation across domains. Based on classification, the system applies phase-appropriate intervention. In-terventions may redirect execution toward inferred planning structure, constrain behavioral space, re-anchor the system to a prior coherent state, initiate recovery procedures, or halt execution when coherence cannot be pre-served. Intervention selection is governed by a deterministic runtime policy that maps system state to con-strained response categories.

The policy operates within defined system bounds and is not externally exposed. Redirect and re-anchor act on planning trajectory. Constraint limits allowable behavioral space. Recovery procedures restore alignment across domains. If stability cannot be restored, execution is halted.

Example Consider a tool using autonomous agent assigned the task of summarizing a technical document and generating a concise report. During initial execution the planning state, behavioral output, and execution trajectory remain aligned. Tool usage remains constrained to retrieval, parsing, and summarization functions consistent with the original objective. As execution continues the agent begins invoking unrelated external APIs and generating outputs increasingly disconnected from the source material. Although individual actions remain syntactically valid, the execution trajectory diverges from the established reference pattern. Cross domain divergence increases as observed behavior no longer aligns with inferred planning structure or expected task progression.

CHURPRIMO detects sustained divergence across planning state, behavioral output, and execution trajectory and classifies the condition as amplification approaching rupture. In response the governance layer constrains available tool pathways, re-anchors execution to the last coherent system state, and restores alignment with the original task objective. Execution then transitions through return and recalibration phases before stabilizing within acceptable coherence bounds. CHURPRIMO does not modify model weights. It does not require access to internal model state and does not depend on task-specific configuration. It operates across architectures as a governance layer during execution. By operating continuously during execution, the framework enables systems to respond to instability as it emerges rather than after failure occurs. Monitoring observes instability. Verification evaluates it. Runtime governance enforces structural conditions during execution.

4 Operational Model Phase Based Runtime Governance

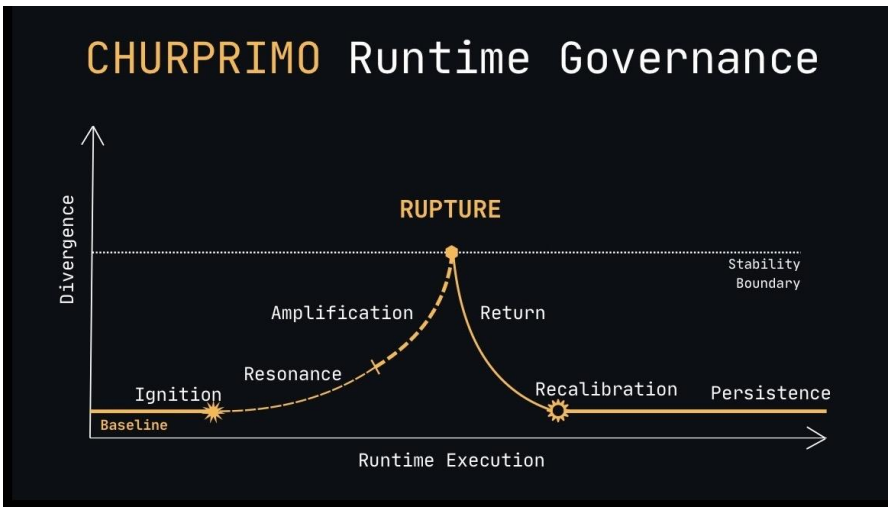

Fig. 2 CHURPRIMO phase progression during runtime execution. The figure shows divergence evolving from baseline through drift and rupture and how governed intervention enables return, recalibration, and persistence within stability bounds.

The framework operates through a phase-based model describing how coherence is established, evaluated, and maintained during execution. These phases form a continuous progression reflecting evolving alignment across planning state, behavior, and trajectory.

Ignition

The system establishes an initial trajectory. A baseline coherence reference is defined.

Resonance

The system stabilizes around a working pattern. This pattern becomes the reference trajectory.

Amplification

Divergence from the reference trajectory increases while corrective stabilization decreases. Early divergence becomes measurable.

Rupture

Cross domain divergence exceeds recoverable stability thresholds and execution enters a self-reinforcing unstable state. Beyond rupture, some trajectories enter an irreversible state where incremental correction is no longer viable and full reset is required.

Return

Intervention redirects the system toward coherence.

Recalibration

Alignment across domains is restored and verified.

Persistence

Execution stabilizes under continuous monitoring preventing further drift accumulation.

The phase structure is derived from observed system behavior during runtime execution. It reflects consistent patterns of drift, rupture, and recovery across extended and multi-agent conditions.

Prototype execution demonstrates measurable recovery under governed conditions during extended multi step execution involving tool using autonomous agents operating within constrained task environments. In testing the system completed 127 execution cycles with 23 rupture detections. Average return time following intervention was 4.2 minutes. Coherence improved from 0.63 to 0.87 following recovery. Coherence remained stable above 0.9 in sustained execution following recalibration. These results indicate that drift can be detected early and that phase appropriate intervention restores alignment within bounded time.

5 Implications for Autonomous Systems

The absence of runtime governance becomes visible in systems that operate beyond single step interaction. In multi-agent systems, drift in one agent propagates through shared context introducing system wide instability. [4, 5] In human agent interaction, systems that appear coherent while internally misaligned degrade trust and lead to incorrect reliance. [6, 7] In tool integrated environments drift produces incorrect tool use sequencing er-rors and divergence from task objectives. In long running workflows small inconsistencies accumulate into in-stability that appears sudden but originates from prolonged drift. As systems scale in complexity and duration the challenge shifts from model performance to sustained reliability under dynamic conditions. Runtime gov-ernance addresses this condition by maintaining coherence during execution rather than evaluating behavior after deviation occurs.

6 Conclusion

Autonomous systems do not fail as a result of a single incorrect output. They fail through the accumulation of unobserved drift during execution. This work defines a framework that detects and responds to instability as it emerges. By operating inline during execution and evaluating coherence across planning state, behavior, and tra-jectory, it provides a mechanism for identifying drift in real time and restoring alignment before instability compounds. This enables systems to maintain coherence during execution rather than reacting after failure oc-curs. Runtime governance becomes a necessary condition for maintaining stability during extended autonomous execution. Systems that cannot govern drift during execution will remain unstable regardless of model capabil-ity.

7 Limitations and Future Work

This work defines a runtime governance framework grounded in observed system behavior. It does not yet in-clude large scale empirical benchmarking across diverse architectures and production deployments. The current implementation demonstrates consistent detection classification and intervention patterns. Further work will ex-pand empirical validation, refine boundary detection, and evaluate performance under varied system loads and multi-agent configurations. As autonomy scales systems without execution level governance will remain fun-damentally unreliable regardless of advances in model capability.

Competing Interests

The author declares that there are no competing interests relevant to the content of this chapter.

References

[1] Amodei D, Olah C, Steinhardt J, Christiano P, Schulman J, Mané D (2016) Concrete problems in AI safety. arXiv:1606.06565

[2] Russell S (2019) Human compatible: artificial intelligence and the problem of control. Viking, New York

[3] Weyns D, Iftikhar MU, Malek S (2013) Foundations of self-adaptive systems. In: Self-adaptive systems. Lecture Notes in Computer Science, vol 7475. Springer, Heidelberg

[4] Sycara K (1998) Multiagent systems. AI Magazine 19(2):79–92

[5] Lesser V (1999) Cooperative multiagent systems: a personal view of the state of the art. IEEE Transactions on Knowledge and Data Engineering 11(1):133–142

[6] Bradshaw JM, Feltovich P, Johnson M, Bunch L, Eskridge T, Jung H, Lott J, Uszok A (2013) Human-agent interaction: challenges for the design of flexible and effective systems. In: Handbook of human-machine interaction. Ashgate, Farnham

[7] Goodrich MA, Schultz AC (2007) Human-robot interaction: a survey. Foundations and Trends in Human-Computer Interaction 1(3):203–275

[8] Woods DD (2018) The theory of graceful extensibility: basic rules that govern adaptive systems. Environment Systems and Decisions 38:433–457

[9] Perrow C (1984) Normal accidents: living with high-risk technologies. Basic Books, New York

[10] Calinescu R, Ghezzi C, Kwiatkowska M, Mirandola R (2011) Self-adaptive software needs quantitative verification at runtime. Communications of the ACM 55(9):69–77

[11] Brun Y, Di Marzo Serugendo G, Gacek C, Giese H, Kienle H, Litoiu M, Müller H, Pezzè M, Shaw M (2009) Engineering self-adaptive systems through feedback loops. In: Software engineering for self-adaptive systems. Springer, Heidelberg

[12] Havelund K, Rosu G (2004) An overview of runtime verification. Formal Methods in System Design 24(2):193–205