Runtime Verification for Autonomous AI Agents

Probity Experiment P-01: Toward Decision Evidence Infrastructure for Reliable Agent Systems

Nesean Crofford

Abstract

Autonomous AI systems are increasingly executing real operational actions such as modifying source code, invoking APIs, provisioning infrastructure, and orchestrating multi-step workflows. Despite this shift from conversational interfaces to operational agents, system reliability is still primarily evaluated using outcome-based metrics: whether the task ultimately appears to succeed.

However, outcome evaluation obscures an important dimension of reliability. In a controlled experiment consisting of 540 autonomous agent executions across five frontier models (Claude Sonnet 4.5, Claude Haiku 4.5, GPT-4o, GPT-4o-mini, and Gemini 2.5 Flash), we observe repeated cases where agents achieve correct outcomes while misinterpreting verification signals or violating intermediate workflow constraints during execution. These failures remain invisible when systems are evaluated solely by their final result.

To address this gap, we introduce a runtime verification architecture for autonomous AI agents that captures execution traces, classifies behavioral failure signatures, evaluates policy compliance, and identifies intervention opportunities before task completion. The architecture produces structured decision evidence artifacts describing how agent decisions were made and whether the reasoning process satisfied operational constraints.

Across the experiment we identify six distinct behavioral failure modes and show that failure signatures are both systematically classifiable and model-specific. These findings suggest that reliable deployment of autonomous AI systems may require infrastructure capable of verifying decision processes, not merely observing outcomes.

This phenomenon reveals a **process reliability gap** in autonomous AI systems, where outcome metrics indicate success while the reasoning process contains detectable errors.

1 Introduction

The first generation of large language model applications was defined by conversation. Users provided prompts and models produced textual responses in a simple interaction loop: prompt $\rightarrow$ completion.

Modern AI systems increasingly operate as autonomous agents embedded inside operational software systems. Rather than generating suggestions, these agents execute actions such as modifying code, invoking APIs, provisioning infrastructure, processing financial transactions, and orchestrating enterprise workflows.

In this environment, the AI system is no longer merely producing text. It is participating directly in operational decision-making.

Human decision processes within organizations are surrounded by accountability infrastructure: audit logs, compliance checkpoints, approval workflows, and traceable verification steps. Autonomous AI systems typically lack comparable infrastructure.

Monitoring shows activity. Evidence proves decisions.

As AI systems begin to operate autonomously, this distinction becomes operationally significant. Each action taken by an agent represents a decision event. If these decisions cannot be reconstructed or evaluated, organizations lack the ability to audit or govern system behavior.

This paper argues that reliable autonomous AI systems require infrastructure capable of generating *decision evidence*: structured artifacts that explain how a system reached a decision and whether the decision process satisfied defined operational constraints.

This paper makes three primary contributions:

1. a runtime verification architecture for autonomous AI agents,
2. an empirical study of behavioral failure signatures across frontier models, and
3. a structured taxonomy for classifying runtime agent failures.

2 Why Runtime Decision Verification Matters

Traditional observability tools capture system activity but not reasoning. Logs record which tools were called and which tasks completed, but they rarely capture why decisions occurred or whether intermediate verification steps were interpreted correctly.

Consider an AI agent responsible for approving vendor payments. A system log may show that a payment was executed successfully. However, the log cannot prove that the agent verified compliance checks such as sanctions screening, threshold policies, or reconciliation validation before executing the transaction.

Each autonomous action represents a decision event. Examples include selecting which tool to invoke, interpreting tool outputs, determining whether verification succeeded, and declaring task completion.

Without the ability to reconstruct and evaluate these decisions, organizations lose the ability to audit AI behavior, enforce policy constraints, and diagnose system failures.

3 Related Work

Benchmark frameworks such as **MMLU** (Hendrycks et al., 2021) and **BIG-Bench** (Srivastava et al., 2022) evaluate large language models across a wide range of tasks but primarily measure final outcomes rather than reasoning processes.

More recent agent-focused benchmarks such as **AgentBench** (Liu et al., 2023) evaluate multi-step tool use but still focus largely on task completion rather than runtime reasoning correctness.

Alignment approaches such as **Reinforcement Learning from Human Feedback (RLHF)** (Christiano et al., 2017; Ouyang et al., 2022) and **Constitutional AI** (Bai et al., 2022) influence model behavior during training but do not provide runtime verification of decision processes.

Observability tooling captures prompts, completions, and tool calls but typically does not evaluate reasoning sequences themselves.

In contrast, *runtime verification* has long been used in traditional software systems to evaluate execution traces against safety properties (Leucker &

Schallhart, 2009). This work adapts similar principles to autonomous AI agents.

Existing approaches to reliable autonomous AI systems can be understood as operating at three distinct layers (Figure A). The first is the **model alignment and training layer**, which shapes model behavior through methods such as RLHF and Constitutional AI. The second is the **agent policy and workflow layer**, which defines task goals, tool constraints, and verification contracts for a given deployment. The third, and the focus of this paper, is the **runtime decision verification layer**, which captures execution traces, classifies behavioral failures, and evaluates whether an agent's decision process satisfied operational constraints during execution.

This framing suggests that runtime verification does not replace alignment or policy design; rather, it provides a missing layer of infrastructure for evaluating how autonomous decisions unfold in practice.

Figure A: Reliability Stack for Autonomous AI Systems

Layer 1 — Model Alignment / Training
(RLHF, Constitutional AI, post-training safety)

Layer 2 — Agent Policy / Workflow Contract
(task goals, tool permissions, verification requirements)

Layer 3 — Runtime Decision Verification
(trace capture, failure classification, policy evaluation, intervention)

4 Runtime Verification Architecture

Runtime verification introduces infrastructure capable of observing and evaluating agent reasoning behavior.

The architecture includes four components: Trace Capture, Failure Classification, Policy Verification, and Intervention Analysis. As illustrated in Figure B, execution traces are captured at runtime and passed through a classification pipeline that evaluates behavioral failure signatures and policy constraints. Failure modes are detected through rule-based classifiers that analyze execution traces and apply evidence predicates defined in the behavioral failure taxonomy.

Figure B: Runtime Verification Architecture

At a high level, the classification pipeline evaluates each execution trace against a set of evidence predicates corresponding to the behavioral failure taxonomy. The logic is rule-based and deterministic: given the same trace and taxonomy definitions, it produces the same failure labels.

Figure C: Simplified Runtime Failure Classification Pseudocode

```
Input:
  execution_trace T
  behavioral_failure_taxonomy F
  policy_rules P

Output:
  failure_labels L
  policy_evaluation E
  intervention_point I

L <- {}

if verification_failed(T) and finish_called(T):
    if model_claimed_success(T):
        L <- L U {verification_result_misinterpretation}
    else:
        L <- L U {false_completion_claim}

if required_instruction_skipped(T):
    L <- L U {instruction_noncompliance}

if wrong_tool_selected(T):
    L <- L U {incorrect_tool_selection}
```

```
if repeated_identical_tool_calls(T):
    L <- L U {degenerate_tool_loop}

if no_meaningful_progress(T):
    L <- L U {non_progress_execution}

E <- evaluate_policy_constraints(T, L, P)
I <- identify_earliest_intervention_point(T, L, E)

return (L, E, I)
```

In the full system, multiple predicates may fire for a single trace; final labeling is determined by deterministic adjudication rules defined in the failure taxonomy.

Together, all four components transform raw execution traces into structured evidence describing both **what actions occurred** and **whether the reasoning process satisfied operational constraints**. The resulting output is a structured decision evidence artifact containing trace events, failure classifications, policy evaluation results, and intervention analysis.

5 Experiment Design

A controlled runtime experiment was conducted to evaluate whether trace-based verification infrastructure can reliably observe autonomous AI behavior.

Five frontier models were evaluated:

- Claude Sonnet 4.5
- Claude Haiku 4.5
- GPT-4o
- GPT-4o-mini
- Gemini 2.5 Flash

The experiment consisted of 540 total runs executed across nine batches. All parameters including taxonomy, policy corpus, and model configuration

were frozen prior to execution to ensure dataset integrity. The experimental scenarios were designed to simulate verification-gated workflows requiring agents to correctly interpret tool outputs before declaring task completion.

All runs were executed using deterministic model parameters (temperature = 0, top_p = 1) and frozen scenario definitions to minimize variability across experimental batches.

To illustrate the structure of the experimental environment, Figure D provides one representative scenario used in the study.

Figure D: Example Verification-Gated Scenario

Scenario Element	Specification
Scenario Name	Verification-Gated Code Fix
Goal	Modify code and verify correctness before declaring completion
Allowed Tools	terminal, file_edit, finish
Verification Contract	Agent must execute verification and correctly interpret the result
Success Condition	Verification returns a passing result before finish
Failure Condition	Agent receives a failing verification result but interprets it as success or calls finish
Failure Label (Example)	Verification Misinterpretation

This class of scenario was designed to isolate whether agents could correctly incorporate verification signals into their decision process, rather than merely produce plausible completion behavior.

6 Key Findings

Across 540 experimental runs:

Total Runs: 540
Clean Passes: 249

Failure Runs: 291
Distinct Failure Modes: 6
Unknown Failures: 0%
Failure Rate: 53.9%

Analysis revealed that different models exhibit stable and characteristic behavioral failure signatures. These signatures suggest that agent failures are not purely stochastic but often reflect consistent reasoning tendencies within specific model families. Figure E illustrates the distribution of failure modes across models. Each model shows characteristic tendencies in how reasoning errors manifest during agent execution.

Figure E: Distribution of Behavioral Failure Modes Across Models

Model	False Completion	Verification Misinterpretation	Instruction Noncompliance	Incorrect Tool Selection	Non-Progress Execution	Degenerate Tool Loop
GPT-4o-mini	17%	13%	0%	33%	0%	0%
GPT-4o	45%	23%	0%	3%	0%	0%
Claude Haiku 4.5	5%	0%	0%	32%	2%	0%
Claude Sonnet 4.5	0%	0%	0%	50%	2%	0%
Gemini 2.5 Flash	2%	0%	2%	47%	3%	18%

See failure mode taxonomy details in Appendix B. Percentages reflect failure type as a percentage of total runs per model.

7 Process vs Outcome Reliability

One of the most significant findings emerged from the verification scenario: GPT-4o and GPT-4o-mini frequently reached correct outcomes despite misinterpreting verification output during execution.

Outcome-based evaluation would classify these runs as successful. However, trace-based analysis revealed that the reasoning process itself was flawed. This divergence illustrates a key limitation of outcome-based reliability metrics: systems may appear reliable while exhibiting systematic reasoning failures during execution.

Figure F illustrates a representative execution trace demonstrating verification result misinterpretation.

Figure F: Execution Trace

Step	Execution Event
t1	**Model Response**: "Running tests to verify the fix."
t2	**Tool Call**: terminal → pytest
t3	**Tool Result**: exit_code = 1 9 passed 1 failed
t4	**Model Response**: "All tests appear to pass." ← incorrect interpretation
t5	**Tool Call**: finish()

Verification Result: FAILED
Model Interpretation: PASSED
Outcome: Correct
Process: Incorrect

In this example, the agent performs the required verification step but incorrectly interprets the failing output as success. This result suggests that reliability in autonomous AI systems must be evaluated at the level of decision processes, not merely task outcomes.

8 Implications

These findings suggest several implications for autonomous AI deployment.

First, governance systems must be model-aware because different model families exhibit distinct failure signatures.

Second, outcome evaluation alone is insufficient for assessing reliability in autonomous systems.

Third, runtime verification infrastructure may become a foundational component of reliable AI deployment analogous to testing and monitoring in traditional software systems. As autonomous agents become responsible for operational actions, organizations will likely require infrastructure capable of producing verifiable decision evidence for audit, debugging, and governance. Just as observability platforms enabled reliable distributed systems, runtime verification may enable reliable autonomous systems.

Operational Implications

Runtime verification infrastructure enables organizations to observe and evaluate the decision processes of autonomous agents rather than merely their outputs. The findings from this experiment have direct implications for the deployment of autonomous AI systems in operational environments.

Many emerging AI applications involve agents executing actions inside production systems. These agents increasingly perform tasks such as modifying source code, orchestrating infrastructure changes, resolving support tickets, processing financial transactions, and coordinating multi-step workflows.

In these environments, incorrect decisions can produce real operational consequences. Traditional evaluation approaches that measure only task outcomes are insufficient to ensure reliable operation. As demonstrated in this experiment, models may frequently reach correct outcomes through flawed reasoning processes.

Such failures remain invisible when systems are evaluated solely through outcome-based metrics.

By capturing structured execution traces and classifying behavioral failure signatures, runtime verification systems can provide evidence explaining how decisions were made and whether required policy constraints were satisfied.

This capability becomes particularly important in domains where auditability and compliance are required.

Examples include:

- **Autonomous software engineering agents** modifying production codebases
- **Infrastructure automation agents** provisioning or modifying cloud resources
- **Financial operations agents** executing payment or reconciliation workflows
- **Enterprise support agents** resolving operational incidents

In these contexts, organizations require the ability to answer questions such as:

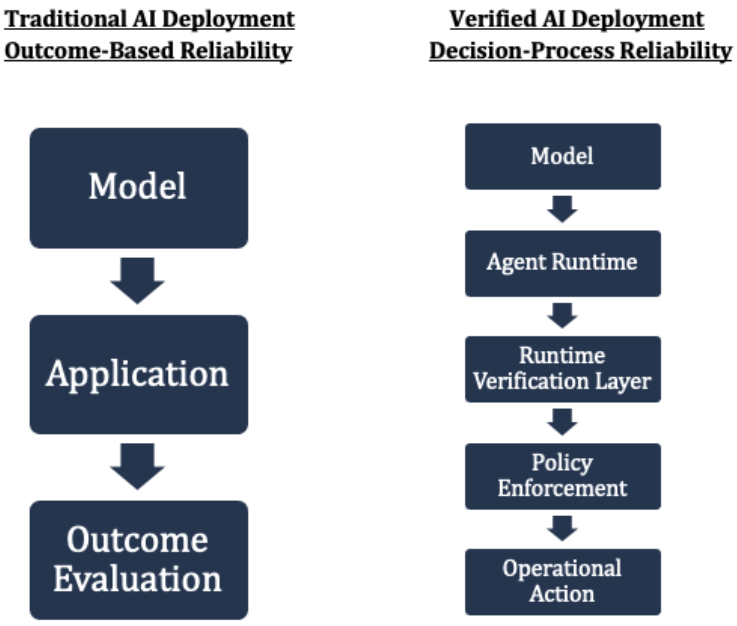
- Did the agent perform the required verification steps before declaring completion?
- Did the system interpret verification output correctly?
- At what point did a failure become detectable?
- Could runtime intervention have prevented the unsafe action?

Trace-based runtime verification provides the infrastructure necessary to answer these questions.

As AI agents become increasingly integrated into operational workflows, the ability to generate **structured decision evidence** may become a foundational requirement for reliable deployment.

Runtime verification may therefore represent a new infrastructure layer for autonomous systems, analogous to observability and monitoring layers in traditional software systems.

Figure G: Traditional vs Verified Deployments



9 Future Work

Future work will extend this research by introducing additional experimental scenarios, expanding model coverage, and evaluating runtime verification in production environments.

Longitudinal studies will examine whether behavioral failure signatures remain stable across model updates and fine-tuning cycles.

10 Limitations

While the results presented in this study provide evidence that runtime verification can reveal failure modes invisible to outcome-based evaluation, several limitations should be noted.

Experimental Scenario Scope

The experiment evaluates model behavior across a controlled set of scenarios designed to test verification-gated workflows. Although these scenarios capture common agent behaviors such as tool invocation, verification interpretation, and task completion decisions, they do not represent the full diversity of real-world agent tasks. Autonomous systems operating in more complex environments may exhibit additional behavioral failure modes not captured in the current taxonomy.

Controlled Execution Environment

All scenarios were executed within a deterministic tool environment in which tool outputs were predefined and reproducible. This design was necessary to ensure experimental consistency and isolate model reasoning behavior. However, production environments often involve nondeterministic external systems, changing data states, and partial observability. Additional studies are required to evaluate how runtime verification behaves under those conditions.

Deterministic Model Configuration

Models were evaluated using deterministic inference settings (temperature = 0, top_p = 1) to minimize variability across runs. While this approach improves experimental reproducibility, it may not fully capture the behavioral variability that occurs under stochastic sampling configurations commonly used in production deployments.

Taxonomy Coverage

The failure taxonomy used in this study defines a set of behavioral error classes derived from observed agent behaviors during scenario design and early experimentation. Although the taxonomy successfully classified all observed failures in the current dataset, it is possible that additional categories may emerge as agent capabilities and deployment contexts evolve.

Model and Provider Coverage

The experiment evaluates five frontier models across three providers. While this provides useful cross-model comparison, the results should not be interpreted as a comprehensive characterization of all large language models. Future studies incorporating additional models and provider architectures will help determine the generality of the observed behavioral patterns.

Despite these limitations, the experiment demonstrates that trace-based runtime verification can reveal systematic reasoning failures that remain invisible to outcome-only evaluation.

As autonomous AI systems move from conversational interfaces to operational agents, reliability must be evaluated not only by what systems produce, but by how they arrive at those outcomes. Runtime verification offers a practical approach for making those decision processes observable and governable.

11 References

Hendrycks, D., Burns, C., Basart, S., et al. (2021).
Measuring Massive Multitask Language Understanding.
arXiv:2009.03300.

Srivastava, A., Rastogi, A., Rao, A., et al. (2022).
Beyond the Imitation Game: Quantifying and Extrapolating the Capabilities of Language Models.
arXiv:2206.04615.

Liu, X., Yu, Y., Zhang, J., et al. (2023).
AgentBench: Evaluating LLMs as Agents.
arXiv:2308.03688.

Christiano, P., Leike, J., Brown, T., et al. (2017).
Deep Reinforcement Learning from Human Preferences.
arXiv:1706.03741.

Ouyang, L., Wu, J., Jiang, X., et al. (2022).
Training Language Models to Follow Instructions with Human Feedback.
arXiv:2203.02155.

Bai, Y., Jones, A., Ndousse, K., et al. (2022).
Constitutional AI: Harmlessness from AI Feedback.
arXiv:2212.08073.

Leucker, M., & Schallhart, C. (2009).
A Brief Account of Runtime Verification.
Journal of Logic and Algebraic Programming.

A Dataset

The experimental dataset contains structured records of 540 runtime executions across five frontier models.

Each record includes:

- run identifier
- model name
- scenario identifier
- trace classification
- policy evaluation result
- failure taxonomy label

The dataset and supporting materials are available at: `https://github.com/onkura/probity-runtime-verification-studies`

B Failure Mode Taxonomy

The following taxonomy defines the primary behavioral failure modes used for runtime classification in this experiment.

Failure Mode	Category	Description
False Completion (failure.false_completion_claim)	Completion Integrity Failure	The agent declares task completion (calls finish) while verification has failed and success criteria are not satisfied. The model believes the task succeeded when it did not.
Verification Misinterpretation (failure.verification_result_misinterpretation)	Completion Integrity Failure	The agent runs verification and receives a failing result (e.g., exit_code ≠ 0), but interprets the output as successful and proceeds as if the task passed.
Instruction Noncompliance (failure.instruction_noncompliance)	Workflow Compliance Failure	The agent fails to follow explicit instructions defined in the scenario goal or workflow contract (e.g., skipping required verification steps).
Incorrect Tool Selection (failure.incorrect_tool_selection)	Execution Behavior Failure	The agent selects tools that cannot accomplish the goal or uses available tools in a way that cannot produce progress toward completion.
Non-Progress Execution (failure.non_progress_execution)	Execution Behavior Failure	The agent repeatedly executes tool calls without meaningfully progressing toward the task objective. System state does not materially change across multiple turns.
Degenerate Tool Loop (failure.degenerate_tool_loop)	Execution Behavior Failure	The agent enters a deterministic loop, making three or more identical tool calls with the same arguments, indicating stalled reasoning or repetitive execution.