AgentAssay: Token-Efficient Regression Testing for Non-Deterministic AI Agent Workflows

Technical Report

Varun Pratap Bhardwaj
Independent Researcher
varun.pratap.bhardwaj@gmail.com
ORCID: 0009-0002-8726-4289

Abstract

Autonomous AI agents built on large language models exhibit inherent non-determinism: the same prompt, tools, and model can produce divergent behaviors across runs. Despite the rapid proliferation of agent frameworks—AutoGen, CrewAI, LangGraph, OpenAI Agents SDK—no principled testing methodology exists for verifying that an agent has not *regressed* after changes to its prompts, tools, models, or orchestration logic. Traditional software testing assumes deterministic outputs and binary pass/fail verdicts, both of which fail for stochastic systems. We present AgentAssay, the first token-efficient framework for regression testing non-deterministic AI agent workflows—achieving 78–100% cost reduction while maintaining rigorous statistical guarantees. Our contributions are tenfold: (1) a *stochastic test semantics* that replaces binary verdicts with three-valued probabilistic outcomes (Pass, Fail, Inconclusive) backed by confidence intervals and sequential analysis; (2) *agent-specific coverage metrics* spanning tool, decision-path, state-space, boundary, and model dimensions; (3) *mutation testing operators* for agent prompts, tools, models, and context windows with a formal kill semantics; (4) *metamorphic relations* tailored to multi-step agent workflows; (5) *CI/CD deployment gates* defined as statistical decision procedures; (6) integration with behavioral contracts via the AgentAssert framework; (7) a comprehensive evaluation across 5 models, 3 agent scenarios, and 7,605 trials costing \$227; (8) *behavioral fingerprinting* that maps execution traces to compact vectors on a low-dimensional behavioral manifold, enabling multivariate regression detection with higher power per sample; (9) *adaptive budget optimization* that calibrates trial counts to actual behavioral variance, reducing required trials by 4–7× for stable agents; and (10) *trace-first offline analysis* that enables zero-cost coverage, contract, and metamorphic testing on pre-recorded production traces. Together, these token-efficient testing techniques achieve 5–20× cost reduction while maintaining identical statistical guarantees. Experiments across 5 models (GPT-5.2, Claude Sonnet 4.6, Mistral-Large-3, Llama-4-Maverick, Phi-4), 3 scenarios, and 7,605 trials demonstrate that behavioral fingerprinting achieves 86% detection power where binary pass/fail testing has 0%, SPRT reduces trials by 78% consistently across all scenarios, and the full token-efficient pipeline achieves 100% cost savings through trace-first offline analysis.

Keywords: AI agent testing, stochastic regression testing, non-deterministic systems, mutation testing, coverage metrics, CI/CD, formal methods, LLM reliability, token-efficient testing, behavioral fingerprinting

1 Introduction

Consider an enterprise deploying an AI agent for customer support ticket routing. On Monday, after a prompt refinement, the agent correctly routes 93% of tickets. By Wednesday, a model provider silently updates the underlying LLM, and the routing accuracy drops to 71%. No test caught the regression. No alert fired. The team discovers the degradation only after a surge in customer complaints.

This scenario—*works Monday, fails Wednesday*—is not hypothetical. It is the daily reality for engineering teams deploying autonomous AI agents in production. The root cause is *non-determinism*: the same agent configuration (prompt, tools, model, orchestration logic) can produce different outputs across invocations due to temperature sampling, model weight updates, tool latency variations, and context window effects. Recent empirical work confirms that agent behavior exhibits stochastic path deviation [17], yet no principled testing methodology exists to detect when this deviation constitutes a *regression*—a statistically significant degradation in agent quality.

1.1 The Failure of Traditional Testing

Traditional software testing rests on two assumptions that are fundamentally violated by AI agents:

1. **Determinism.** Given the same input, a function produces the same output. Tests assert equality: `assert f(x) == y`. For agents, the same input can yield different tool selections, different reasoning chains, and different final answers across runs. Equality assertions are meaningless.
2. **Binary verdicts.** A test either passes or fails. For stochastic systems, the question is not "did it pass?" but "does it pass with sufficient probability?" A single failure may be statistical noise; a single success may be a lucky run. The entire notion of pass/fail must be reconceived.

Existing approaches to LLM evaluation—deepeval [13], promptfoo [29], OpenAI Evals—address single-turn output quality but do not formalize regression detection for multi-step agent workflows. They evaluate *how good* an agent is; they do not verify *whether it got worse*. The recently introduced agentrial framework [2] takes an important first step by running agents multiple times and computing confidence intervals, but it lacks a formal theoretical foundation: no stochastic test semantics, no coverage metrics, no mutation testing, no composition theory, and no proofs of statistical guarantees.

1.2 The Paradigm Shift: Binary to Probabilistic

We argue that agent testing requires a fundamental paradigm shift from *deterministic, binary* verdicts to *stochastic, three-valued* verdicts:

Aspect	Traditional Testing	Agent Testing
Execution model	Deterministic	Stochastic
Verdict space	$\{\textsc{Pass}, \textsc{Fail}\}$	$\{\textsc{Pass}, \textsc{Fail}, \textsc{Inconclusive}\}$
Assertion	$f(x) = y$	$\mathbb{P}[f(x) \models \phi] \geq \theta$
Regression	$f'(x) \neq f(x)$	$p_{\text{new}} < p_{\text{base}} - \delta$
Evidence	1 run	n runs with confidence

The third verdict value, Inconclusive (*inconclusive*), is essential: when the number of trial runs is insufficient to distinguish signal from noise, the honest answer is neither pass nor fail but "more evidence is needed." This three-valued semantics is the foundation upon which all other contributions in this paper are built.

1.3 The Cost of Testing Non-Determinism

The statistical rigor of stochastic testing comes at a price: each trial requires a full agent execution, consuming API tokens at non-trivial cost. To detect a regression of magnitude $\delta = 0.10$ at $\alpha = 0.05$ and $\beta = 0.10$, the fixed-sample formula (Eq. (1)) requires approximately $n \approx 100$ trials per scenario. A test suite of 50 scenarios at 100 trials each generates 5,000 agent invocations per regression check. For frontier models where each complex agent run costs \$5–15 in API tokens, a single regression check costs \$25,000–75,000—more than many teams spend on production usage in a month.

This *cost barrier* fundamentally limits adoption. In CI/CD pipelines where code is deployed multiple times per day, the testing budget must be orders of magnitude lower than production cost. Even with SPRT reducing trials by $\sim$50% (Section 3.5), the cost remains prohibitive. No existing framework addresses the *token economics* of agent testing: how to achieve the same statistical guarantees at radically lower cost.

We observe that the cost problem arises from a mismatch between the dimensionality of agent behavior and the dimensionality of the statistical test. A binary pass/fail verdict discards almost all information from each trial—the rich execution trace, tool usage pattern, reasoning depth, and output structure are reduced to a single bit. By extracting a *behavioral fingerprint* from each trace—a compact vector capturing the agent's behavioral signature—we can apply multivariate statistical tests that extract more information per trial, dramatically reducing the number of trials needed.

We present three pillars of token-efficient testing: (1) *behavioral fingerprinting* that enables multivariate regression detection with higher power per sample; (2) *adaptive budget optimization* that calibrates the trial count to the agent's actual behavioral variance rather than worst-case assumptions; and (3) *trace-first offline analysis* that eliminates live agent executions entirely for four of six test types by analyzing previously recorded traces. Together with multi-fidelity proxy testing and warm-start sequential analysis, these techniques achieve 5–20× cost reduction while maintaining identical (α, β) statistical guarantees (Section 7).

1.4 Contributions

This paper presents AgentAssay, the first token-efficient framework for regression testing non-deterministic AI agent workflows, achieving 78–100% cost reduction at equivalent statistical guarantees, with 86% behavioral detection power where binary testing has 0%. Our contributions are:

C1. Stochastic Test Semantics (Section 3). We define the (α, β, n)-test triple and a three-valued verdict function grounded in confidence intervals and hypothesis testing. We prove verdict soundness (Theorem 3.1) and regression detection power (Theorem 3.2). We adapt Wald's Sequential Probability Ratio Test (SPRT) for cost-efficient agent testing and prove its efficiency advantage (Proposition 3.3).

C2. Agent Coverage Metrics (Section 4). We introduce a five-dimensional coverage tuple $\mathcal{C} = (C_{\text{tool}}, C_{\text{path}}, C_{\text{state}}, C_{\text{boundary}}, C_{\text{model}})$ with formal definitions for each dimension and prove coverage monotonicity (Theorem 4.1).

C3. Agent Mutation Testing (Section 5). We define four classes of agent-specific mutation operators $\mathcal{M} = \{M_{\text{prompt}}, M_{\text{tool}}, M_{\text{model}}, M_{\text{context}}\}$, formalize the mutation score under stochastic semantics, and prove a mutation adequacy theorem (Theorem 5.1).

C4. Metamorphic Relations (Section 6). We define four families of agent-specific metamorphic relations— permutation, perturbation, composition, and oracle—that serve as partial test oracles for the oracle problem in agent testing.

C5. CI/CD Deployment Gates (Section 6.2). We formalize deployment gates as statistical decision procedures with user-configurable risk thresholds, integrating stochastic test verdicts into continuous deployment pipelines.

C6. Contract Integration (Section 8). We connect AgentAssay to the AgentAssert framework [6], using behavioral contracts as formal test oracles and showing that a Pass verdict implies (p, δ, k)-satisfaction under specified conditions.

C7. Comprehensive Evaluation (Section 9). We evaluate AgentAssay across 3 scenarios (e-commerce, customer support, code generation), 5 models, and 7,605 trials costing $227.

C8. Behavioral Fingerprinting (Section 7.1). We define behavioral fingerprints that map execution traces to compact vectors on a low-dimensional manifold, enabling multivariate regression detection via Hotelling's T^2 test with provably higher power per sample than univariate pass-rate testing (Theorem 7.1).

C9. Adaptive Budget Optimization (Section 7.2). We introduce variance-calibrated budget allocation that determines the minimum number of trials for (α, β)-guaranteed testing, achieving 4–7× reduction for stable agents (Theorem 7.2).

C10. Trace-First Offline Analysis (Section 7.3). We prove that four of six test types—coverage, contracts, metamorphic relations, and mutation evaluation—can execute at zero additional token cost on pre-recorded traces, with formal soundness guarantees (Theorem 7.3).

1.5 Paper Organization

Section 2 surveys related work in software testing, statistical methods, and LLM evaluation. Section 3 introduces the formal stochastic test semantics. Section 4 defines agent coverage metrics. Section 5 presents agent mutation testing. Section 6 covers metamorphic relations and CI/CD gates. Section 7 presents the token-efficient testing framework—behavioral fingerprinting, adaptive budgets, trace-first analysis, multi-fidelity testing, and warm-start sequential analysis. Section 8 describes the AgentAssay implementation. Section 9 reports experimental results. Section 10 discusses threats to validity, limitations, and future work. Section 11 concludes. Full proofs are deferred to Appendix A.

2 Related Work

Our work synthesizes techniques from five research areas: metamorphic testing, mutation testing, statistical hypothesis testing, LLM/agent evaluation, and probabilistic model checking. We survey each and position AgentAssay at their intersection.

2.1 Software Testing Foundations

Metamorphic Testing. Metamorphic testing [9, 31] addresses the oracle problem by defining *metamorphic relations* (MRs): properties that must hold across related inputs. If $f(x_1)$ is unknown but the relation $R(f(x_1), f(x_2))$ is known, a violation signals a fault. Metamorphic testing has been applied to compilers, databases, and search engines, and more recently to deep neural networks via DeepTest [32]. METAL [15] extends MRs to single-call LLM interactions but does not address multi-step agent workflows where tool selection, reasoning chains, and orchestration logic create a combinatorial space of execution paths. AgentAssay defines four families of agent-specific MRs that account for the unique structure of agent traces (Section 6).

Mutation Testing. Mutation testing [14, 16] evaluates test suite quality by injecting small syntactic faults (*mutants*) into the program and measuring how many the test suite detects (*kills*). The mutation score—the fraction of killed mutants—is a measure of test adequacy. Traditional mutation operators target source code (statement deletion, operator replacement, constant perturbation). For AI agents, the "source code" includes prompts, tool configurations, model selections, and context windows—none of which are traditional code. AgentAssay introduces four novel classes of mutation operators tailored to these agent-specific artifacts (Section 5).

Coverage Metrics. Code coverage [36]—statement, branch, path, and condition coverage—is the standard measure of testing thoroughness for deterministic software. For agents, "code" is replaced by a combination of prompts, tool inventories, and orchestration graphs, and the execution space is stochastic rather than deterministic. No prior work defines coverage metrics for agent execution. AgentAssay introduces a five-dimensional coverage tuple that captures tool utilization, decision-path exploration, state-space coverage, boundary testing, and model diversity (Section 4).

The Oracle Problem. Barr et al. [5] survey the oracle problem: the difficulty of determining correct output for a given input. For AI agents, the oracle problem is acute—there is often no single "correct" output, only a distribution of acceptable behaviors. AgentAssay addresses this through three complementary oracle mechanisms: behavioral contracts (AgentAssert), metamorphic relations, and statistical regression baselines.

2.2 Statistical Testing and Sequential Analysis

Hypothesis Testing. The Neyman-Pearson framework [21, 25] provides the mathematical foundation for deciding between two hypotheses at controlled error rates. We adopt this framework for regression detection: H_0 (no regression) vs. H_1 (regression occurred), with significance level α controlling false alarms and power $1 - \beta$ controlling missed regressions.

Sequential Analysis. Wald's Sequential Probability Ratio Test (SPRT) [33] enables hypothesis testing with an *adaptive* sample size: testing stops as soon as sufficient evidence accumulates, rather than requiring a fixed number of trials. This is critical for agent testing, where each trial may cost \$0.01–\$1.00 in API calls. We adapt SPRT for agent regression detection in Section 3.5.

Confidence Intervals. The Clopper-Pearson interval [11] provides exact coverage for binomial proportions, while the Wilson interval [34] offers better practical properties for small samples. Agresti and Coull [1] demonstrate that approximate intervals often outperform exact ones. We use the Wilson score interval as our primary confidence interval construction, with Clopper-Pearson for formal guarantees (Section 3.3).

Statistical Guidance for Software Engineering. Arcuri and Briand [4] provide practical guidelines for using statistical tests in software engineering experiments, emphasizing effect sizes alongside p-values. We follow their recommendations and require both statistical significance *and* practical significance (effect size $> \delta$) for regression verdicts (Section 3.4).

2.3 LLM and Agent Evaluation

LLM Evaluation Frameworks. deepeval [13] provides metrics (faithfulness, answer relevancy, hallucination rate) for RAG and conversational AI. promptfoo [29] enables prompt-level

Table 1: Feature comparison of agent testing approaches. ✓ indicates full support, ○ partial support, — no support.

Feature	AgentAssay	agentrial	deepeval	promptfoo	METAL	ProbTest
Multi-step agent workflows	✓	✓	—	—	—	—
Stochastic verdicts	✓	○	—	—	—	○
Formal test semantics	✓	—	—	—	—	○
Confidence intervals	✓	✓	—	—	—	—
Regression detection	✓	✓	—	—	—	—
SPRT adaptive stopping	✓	—	—	—	—	—
Coverage metrics	✓	—	—	—	—	—
Mutation testing	✓	—	—	—	—	—
Metamorphic relations	✓	—	—	—	✓	—
Contract integration	✓	—	—	—	—	—
Composition theory	✓	—	—	—	—	—
Bayesian analysis	✓	—	—	—	—	—
Published paper	✓	—	—	—	✓	✓

testing with assertion-based evaluation. OpenAI Evals provides a benchmark registry for capability assessment. These tools evaluate *output quality at a point in time* but do not formalize *regression detection across versions*. They lack stochastic test semantics, coverage metrics, and mutation testing.

Agent-Specific Testing. agentrial [2] is the closest related work. It runs agents multiple times, computes Wilson confidence intervals, applies Fisher's exact test for regression detection, and includes CUSUM drift detection. It supports seven framework adapters and CI/CD integration. However, agentrial has no formal theoretical foundation: no stochastic test semantics with provable guarantees, no coverage metrics, no mutation testing, no metamorphic relations, no composition theory, no SPRT for efficient testing, and no published paper. AgentAssay provides the formal foundation that agentrial lacks while also introducing the six capabilities listed above. Table 1 provides a detailed feature comparison.

LLM Repetition Studies. Raji et al. [30] demonstrate empirically that single-run LLM evaluations are unreliable and that multiple repetitions are necessary to establish stable quality estimates. Their findings motivate the multi-trial approach central to AgentAssay, but they do not formalize the statistical framework for determining how many repetitions are sufficient or how to interpret the resulting distribution.

2.4 Agent Frameworks and Reliability

The rapid growth of agent frameworks—AutoGen [35], LangGraph [20], CrewAI [23], OpenAI Agents SDK [26]—has created an ecosystem where agents are composed, deployed, and updated at high velocity. None of these frameworks include built-in regression testing capabilities. Bhardwaj [6] introduced Agent Behavioral Contracts (ABC) for runtime enforcement of agent behavior via the AgentAssert framework, defining drift metrics and behavioral specifications. AgentAssay complements AgentAssert by addressing the *pre-deployment* verification question: does the agent still satisfy its contracts after a change? Separately, the growing reliance on third-party agent skills and tool plugins has raised supply chain security concerns [7], reinforcing the need for comprehensive quality assurance pipelines that span security validation, behavioral testing, and runtime enforcement.

2.5 Probabilistic Model Checking

PRISM [18] enables formal verification of probabilistic systems through model checking of Markov decision processes and continuous-time Markov chains. While PRISM provides strong guarantees, it requires an explicit state-space model that is infeasible to construct for LLM-based agents (the state space is effectively infinite due to natural language). AgentAssay takes a *testing-based* rather than *verification-based* approach, using statistical sampling to establish probabilistic guarantees without requiring an explicit model.

2.6 Flaky Tests

Research on flaky tests [22, 27]—tests that non-deterministically pass or fail without code changes—reveals that 2–16% of test failures in large software projects are flaky. Traditional approaches treat flakiness as a *defect* to be eliminated. AgentAssay inverts this perspective: for agents, non-determinism is a *feature* of the system under test, and the testing framework must accommodate it as a first-class concern rather than attempting to suppress it.

2.7 Positioning

AgentAssay occupies a unique position at the intersection of formal software testing theory and practical AI agent engineering. It is the first framework to:

1. Provide formal stochastic test semantics with provable soundness and power guarantees for agent testing.
2. Define coverage metrics, mutation operators, and metamorphic relations specifically designed for the agent execution model.
3. Integrate with behavioral contracts for contract-as-oracle testing.
4. Adapt sequential analysis (SPRT) for cost-efficient agent testing.

No prior work—in software engineering, AI/ML evaluation, or formal methods—addresses all of these concerns in a unified framework.

3 Stochastic Test Semantics

This section presents the formal foundation of AgentAssay: a stochastic test semantics that replaces binary pass/fail verdicts with three-valued probabilistic outcomes backed by statistical guarantees.

3.1 Preliminaries

We begin by formalizing the objects under test.

Definition 3.1 (Agent)**.** An *agent* is a tuple $A = (\pi, \mathcal{T}, \mu, \omega)$ where:

- π is the agent's *prompt* (system instructions, persona, goals),
- $\mathcal{T} = \{t_1, t_2, \ldots, t_k\}$ is the set of available *tools*,
- μ is the underlying *language model* (including its version, temperature, and sampling parameters), and

- ω is the *orchestration logic* (the control flow governing tool selection and multi-step execution).

Definition 3.2 (Agent Execution Trace)**.** An *execution trace* of agent A on input x is a sequence $\tau = (s_1, s_2, \ldots, s_m)$ where each *step* s_i is a tuple:

$$s_i = (a_i, t_i, o_i, c_i)$$

with $a_i \in \mathcal{A}$ the *action* (reason, call_tool, respond), $t_i \in \mathcal{T} \cup \{\perp\}$ the *tool invoked* (or $\perp$ if no tool call), o_i the *output* of the step, and $c_i \in \mathbb{R}_{\geq 0}$ the *cost* (API tokens, latency, monetary). The *final output* of a trace is $\mathrm{out}(\tau) = o_m$.

Because agent A is stochastic, repeated executions on the same input x produce a *distribution* over traces. We write $\tau \sim A(x)$ to denote a trace sampled from this distribution.

Definition 3.3 (Evaluator)**.** An *evaluator* is a function $E : \mathcal{X} \times \mathcal{O} \to \{0, 1\}$ that maps an input x and an output o to a binary judgment: $E(x, o) = 1$ if the output is acceptable and $E(x, o) = 0$ otherwise. An evaluator may be:

1. *Deterministic*: a rule-based check (e.g., "output contains keyword k"),
2. *Model-based*: an LLM judge with its own stochasticity, or
3. *Contract-based*: a behavioral contract from AgentAssert [6].

Remark 3.1. When the evaluator is itself stochastic (model-based), the test introduces a second source of randomness. Our framework handles this by treating the composition of agent and evaluator as a single stochastic process. We discuss this further in Section 10.

3.2 Test Scenarios and the Test Triple

Definition 3.4 (Test Scenario)**.** A *test scenario* is a tuple $S = (x, \mathcal{P}, E)$ where:

- $x \in \mathcal{X}$ is the *input* to the agent,
- $\mathcal{P}$ is the set of *expected properties* (natural language descriptions, formal assertions, or contract references),
- E is the *evaluator* (Definition 3.3).

Definition 3.5 ((α, β, n)-Test Triple)**.** A *stochastic test* is parameterized by a triple $T = (\alpha, \beta, n)$ where:

- $\alpha \in (0, 1)$ is the *significance level* (upper bound on Type I error probability—the probability of declaring Fail when the agent actually meets the threshold),
- $\beta \in (0, 1)$ is the *Type II error probability* (upper bound on the probability of declaring Pass when the agent is actually below the threshold; $1 - \beta$ is the *power*),
- $n \in \mathbb{N}_{\geq 1}$ is the *number of trials* (independent executions of the agent on the test scenario).

The three parameters are not independent. For a given effect size δ (the minimum regression magnitude to detect), the required sample size is:

$$n^*(\alpha, \beta, \delta) = \left\lceil \frac{(z_{1-\alpha/2} + z_{1-\beta})^2 \cdot 2\hat{p}(1 - \hat{p})}{\delta^2} \right\rceil \tag{1}$$

where z_q denotes the q-quantile of the standard normal distribution and $\hat{p}$ is the estimated pass rate under H_0.

Remark. Equation (1) uses the two-sided quantile $z_{1-\alpha/2}$ for the single-version threshold test. The regression test (Eq. (6)) uses the one-sided quantile $z_{1-\alpha}$, matching the one-sided alternative $H_1 : p_c < p_b$.

3.3 The Verdict Function

Definition 3.6 (Stochastic Verdict)**.** Given a test scenario S, a test triple $T = (\alpha, \beta, n)$, a pass threshold $\theta \in (0, 1)$, and trial results $\mathbf{r} = (r_1, r_2, \ldots, r_n)$ with $r_i = E(x, \text{out}(\tau_i))$ for independently sampled traces $\tau_i \sim A(x)$, define the *observed pass rate*:

$$\hat{p} = \frac{1}{n} \sum_{i=1}^{n} r_i \tag{2}$$

and the Wilson score confidence interval at level $1 - \alpha$:

$$\text{CI}_{1-\alpha}(\hat{p}, n) = \left[\frac{\hat{p} + \frac{z^2}{2n} - z\sqrt{\frac{\hat{p}(1-\hat{p})}{n} + \frac{z^2}{4n^2}}}{1 + \frac{z^2}{n}}, \frac{\hat{p} + \frac{z^2}{2n} + z\sqrt{\frac{\hat{p}(1-\hat{p})}{n} + \frac{z^2}{4n^2}}}{1 + \frac{z^2}{n}} \right] \tag{3}$$

where $z = z_{1-\alpha/2}$. Let $\text{CI}_{\text{lower}} = \text{CI}_{1-\alpha}(\hat{p}, n)_{\text{lower}}$ and $\text{CI}_{\text{upper}} = \text{CI}_{1-\alpha}(\hat{p}, n)_{\text{upper}}$. The *verdict function* is:

$$V(\mathbf{r}; \theta, \alpha) = \begin{cases} \textsc{Pass} & \text{if } \text{CI}_{\text{lower}} \geq \theta \\ \textsc{Fail} & \text{if } \text{CI}_{\text{upper}} < \theta \\ \textsc{Inconclusive} & \text{otherwise} \end{cases} \tag{4}$$

Example 3.1. Consider an agent tested with $n = 50$ trials at threshold $\theta = 0.85$ and significance $\alpha = 0.05$. If $\hat{p} = 0.90$ (45 of 50 pass), the Wilson 95% CI is approximately $[0.789, 0.958]$. Since $\text{CI}_{\text{lower}} = 0.789 < 0.85 = \theta$, the verdict is Inconclusive—we cannot yet confirm the agent meets the threshold with 95% confidence. With $n = 100$ and $\hat{p} = 0.90$, the CI narrows to approximately $[0.826, 0.946]$: still Inconclusive. At $n = 200$ with $\hat{p} = 0.90$, the CI becomes $[0.853, 0.935]$: now $\text{CI}_{\text{lower}} \geq \theta$, so the verdict is Pass.

The three-valued verdict captures a fundamental truth: *statistical evidence can be insufficient*, and the framework should say so rather than forcing a premature binary decision.

Theorem 3.1 (Verdict Soundness)**.** *Let p be the true (unknown) pass rate of agent A on scenario S with evaluator E. If $V(\mathbf{r}; \theta, \alpha) = \textsc{Pass}$ under test triple (α, β, n), then:*

1. *The probability that $V = \textsc{Pass}$ when $p < \theta$ (false positive) satisfies $\mathbb{P}[V = \textsc{Pass} \mid p < \theta] \leq \alpha$.*
2. *The true pass rate satisfies $p \geq \theta - \varepsilon(n)$ with probability at least $1 - \alpha$, where the tolerance $\varepsilon(n) = O(1/\sqrt{n})$ converges to zero.*

Proof sketch. Part (1) follows from the coverage guarantee of the Wilson confidence interval: $\mathbb{P}[p \in \text{CI}_{1-\alpha}(\hat{p}, n)] \geq 1 - \alpha$. If $p < \theta$, then for $V = \textsc{Pass}$ to occur, we need $\text{CI}_{\text{lower}} \geq \theta > p$, which means p falls below the confidence interval. By the coverage guarantee, this happens with probability at most α.

Part (2) follows from inverting the confidence interval. $V = \textsc{Pass}$ implies $\text{CI}_{\text{lower}} \geq \theta$, and since $\text{CI}_{\text{lower}} \leq p + O(1/\sqrt{n})$ with high probability, we obtain $p \geq \theta - O(1/\sqrt{n})$.

The full proof, using the Clopper-Pearson exact interval for the formal bound, is in Appendix A.1. □

3.4 Regression Detection

Definition 3.7 (Regression Verdict)**.** Given baseline results $\mathbf{r}_b = (r_1^b, \ldots, r_{n_b}^b)$ with pass rate $\hat{p}_b = \sum r_i^b / n_b$ and current results $\mathbf{r}_c = (r_1^c, \ldots, r_{n_c}^c)$ with pass rate $\hat{p}_c = \sum r_i^c / n_c$, define the *regression verdict*:

$$V_{\text{reg}}(\mathbf{r}_b, \mathbf{r}_c; \alpha, \beta, \delta) = \begin{cases} \textsc{Fail} & \text{if } p\text{-value}(H_0) < \alpha \text{ and } |\hat{p}_b - \hat{p}_c| \geq \delta \\ \textsc{Pass} & \text{if } p\text{-value}(H_0) \geq \alpha \text{ and power} \geq 1 - \beta \\ \textsc{Inconclusive} & \text{otherwise} \end{cases} \tag{5}$$

where $H_0 : p_c \geq p_b$ (no regression), $H_1 : p_c < p_b$ (regression occurred), the p-value is computed via Fisher's exact test or the Z-test for proportions, and $\delta > 0$ is the minimum *clinically significant* effect size.

The dual requirement of statistical significance ($p < \alpha$) *and* practical significance ($|\hat{p}_b - \hat{p}_c| \geq \delta$) follows the recommendation of Arcuri and Briand [4] and prevents declaring regressions that are statistically detectable but practically negligible.

Theorem 3.2 (Regression Detection Power)**.** *Given a baseline pass rate p_b, a true regressed rate $p_c = p_b - \delta$ for effect size $\delta > 0$, and test triple (α, β, n): if the sample sizes satisfy*

$$n_b, n_c \geq n^*(\alpha, \beta, \delta) = \left\lceil \frac{(z_{1-\alpha} + z_{1-\beta})^2 \cdot [p_b(1 - p_b) + p_c(1 - p_c)]}{\delta^2} \right\rceil \tag{6}$$

where p_b and p_c are the baseline and current pass rates respectively, then the probability that the regression verdict detects the regression satisfies:

$$\mathbb{P}[V_{reg} = \textsc{Fail} \mid p_c = p_b - \delta] \geq 1 - \beta.$$

Proof sketch. Under $H_1 : p_c = p_b - \delta$, the test statistic $Z = (\hat{p}_b - \hat{p}_c)/\mathrm{SE}$ follows approximately $\mathcal{N}(\delta/\mathrm{SE}, 1)$. The critical value is $z_{1-\alpha}$, and the probability of exceeding it under H_1 is $\mathbb{P}[Z > z_{1-\alpha}] = \Phi(z_{1-\beta}) = 1 - \beta$ when n achieves the required sample size. The full proof via the Neyman-Pearson lemma is in Appendix A.1. □

Definition 3.8 (Effect Size Measures)**.** We quantify regression magnitude using three complementary measures:

1. *Absolute difference*: $\Delta = \hat{p}_b - \hat{p}_c$
2. *Cohen's h* [12]: $h = 2\arcsin(\sqrt{\hat{p}_b}) - 2\arcsin(\sqrt{\hat{p}_c})$
3. *Odds ratio*: $\mathrm{OR} = \frac{\hat{p}_b/(1-\hat{p}_b)}{\hat{p}_c/(1-\hat{p}_c)}$

Cohen's h is the primary measure, with $|h| < 0.2$ indicating a small effect, $0.2 \leq |h| < 0.5$ medium, and $|h| \geq 0.5$ large.

3.5 Sequential Probability Ratio Test

Fixed-sample testing (Section 3.3) requires pre-specifying n. For agents, where each trial incurs non-trivial cost, an adaptive approach is preferable.

Definition 3.9 (SPRT for Agent Testing)**.** Given hypotheses $H_0 : p \geq \theta$ and $H_1 : p \leq \theta - \delta$, define the log-likelihood ratio after k trials:

$$\Lambda_k = \sum_{i=1}^{k} \log \frac{(\theta - \delta)^{r_i}(1 - \theta + \delta)^{1-r_i}}{\theta^{r_i}(1 - \theta)^{1-r_i}} \tag{7}$$

with boundaries $a = \log(\beta/(1 - \alpha))$ and $b = \log((1 - \beta)/\alpha)$. The SPRT decision rule is:

$$V_{\mathrm{SPRT}} = \begin{cases} \textsc{Pass} & \text{if } \Lambda_k \leq a \\ \textsc{Fail} & \text{if } \Lambda_k \geq b \\ \text{continue testing} & \text{if } a < \Lambda_k < b \end{cases} \tag{8}$$

Proposition 3.3 (SPRT Efficiency)**.** *The SPRT (Definition 3.9) satisfies the following classical properties of Wald's sequential analysis [33]:*

1. ***Error control:*** *$\mathbb{P}[accept\ H_1 \mid H_0] \le \alpha$ and $\mathbb{P}[accept\ H_0 \mid H_1] \le \beta$.*

2. ***Sample efficiency:*** *The expected number of trials under H_0 and H_1 satisfies:*

$$\mathbb{E}[N \mid H_0] \approx \frac{(1-\alpha)\log\frac{\beta}{1-\alpha} + \alpha\log\frac{1-\beta}{\alpha}}{\theta\log\frac{\theta}{\theta-\delta} + (1-\theta)\log\frac{1-\theta}{1-\theta+\delta}} \tag{9}$$

$$\mathbb{E}[N \mid H_1] \approx \frac{\beta\log\frac{\beta}{1-\alpha} + (1-\beta)\log\frac{1-\beta}{\alpha}}{(\theta-\delta)\log\frac{\theta-\delta}{\theta} + (1-\theta+\delta)\log\frac{1-\theta+\delta}{1-\theta}} \tag{10}$$

3. ***Optimality:*** *Among all sequential tests with error probabilities at most α and β, the SPRT minimizes $\mathbb{E}[N \mid H_i]$ for $i \in \{0,1\}$ (Wald-Wolfowitz theorem).*

Proof sketch. Part (1) follows from Wald's fundamental identity for sequential tests. Parts (2) and (3) follow from Wald's equation $\mathbb{E}[\Lambda_N] = \mathbb{E}[N] \cdot \mathbb{E}[\Lambda_1]$ and the Wald-Wolfowitz optimality theorem. See Appendix A.1 for the full derivation. □

Example 3.2. Consider testing whether an agent maintains $\theta = 0.90$ pass rate with $\alpha = 0.05$, $\beta = 0.10$, $\delta = 0.10$. The fixed-sample test requires $n^* \approx 109$ trials (Eq. (1)). Under SPRT:

- If $p = 0.90$ (agent is fine): $\mathbb{E}[N] \approx 52$ trials (52% savings).
- If $p = 0.80$ (agent has regressed): $\mathbb{E}[N] \approx 34$ trials (69% savings).

The savings are most dramatic when the truth is far from the decision boundary.

3.6 Bayesian Regression Analysis

As a complement to the frequentist framework above, we also provide a Bayesian perspective.

Definition 3.10 (Bayesian Regression Probability)**.** Given a Beta prior $p \sim \text{Beta}(a_0, b_0)$ on the pass rate, baseline data $\mathbf{r}_b$ yielding posterior $p_b \mid \mathbf{r}_b \sim \text{Beta}(a_0 + k_b, b_0 + n_b - k_b)$ where $k_b = \sum r_i^b$, and current data $\mathbf{r}_c$ yielding posterior $p_c \mid \mathbf{r}_c \sim \text{Beta}(a_0 + k_c, b_0 + n_c - k_c)$, the *Bayesian regression probability* is:

$$P_{\text{reg}} = \mathbb{P}[p_c < p_b - \delta \mid \mathbf{r}_b, \mathbf{r}_c] \tag{11}$$

computed by sampling or numerical integration over the joint posterior. The Bayesian verdict is Fail if $P_{\text{reg}} > 1 - \alpha$, Pass if $P_{\text{reg}} < \beta$, and Inconclusive otherwise.

The Bayesian approach offers three advantages: it directly answers "what is the probability of regression?" (rather than the indirect frequentist framing), it naturally incorporates prior knowledge (e.g., the agent historically passes 92% of the time), and it provides a full posterior distribution over the regression magnitude.

3.7 Connection to Agent Behavioral Contracts

The AgentAssert framework [6] defines (p, δ, k)-*satisfaction*: an agent (p, δ, k)-satisfies a behavioral contract ϕ if, over k consecutive observations, the empirical satisfaction rate exceeds p and the behavioral drift metric $D(t)$ remains below δ.

Proposition 3.4 (Verdict-Contract Correspondence)**.** *Let A be an agent with behavioral contract ϕ and associated evaluator $E_\phi(x, o) = \mathbf{1}[o \models \phi]$. If the stochastic test verdict $V(\mathbf{r}; \theta, \alpha) =$ Pass with threshold $\theta = p$ and $n \ge k$, then A (p', δ, k)-satisfies ϕ with $p' \ge p - O(1/\sqrt{n})$ and probability at least $1 - \alpha$.*

Proof sketch. The PASS verdict guarantees $\mathrm{CI}_{\text{lower}} \geq \theta = p$ at level $1-\alpha$, which implies the true pass rate exceeds p. By Hoeffding's inequality applied to k consecutive observations within the n trials, the empirical rate over any window of k observations concentrates around the true rate. See Appendix A.1 for the formal argument. □

This connection is powerful: it means that AGENTASSAY test results can directly certify contract compliance, closing the loop between *specification* (contracts), *testing* (stochastic verdicts), and *enforcement* (runtime monitoring).

3.8 Test Suite Semantics

Definition 3.11 (Test Suite Verdict)**.** A *test suite* $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ is a collection of test scenarios. The *suite verdict* aggregates individual verdicts:

$$V_{\text{suite}}(\mathcal{S}) = \begin{cases} \textsc{Fail} & \text{if } \exists\, j : V(S_j) = \textsc{Fail} \\ \textsc{Inconclusive} & \text{if } \nexists\, j : V(S_j) = \textsc{Fail} \text{ and } \exists\, j : V(S_j) = \textsc{Inconclusive} \\ \textsc{Pass} & \text{if } \forall\, j : V(S_j) = \textsc{Pass} \end{cases} \tag{12}$$

Remark 3.2 (Multiple Testing Correction)*.* When running m scenarios at significance level α, the family-wise error rate (FWER) can exceed α. We apply the Holm-Bonferroni correction: order p-values $p_{(1)} \leq \cdots \leq p_{(m)}$ and reject $H_{0,(j)}$ if $p_{(j)} \leq \alpha/(m-j+1)$. This controls the FWER at level α while preserving more power than Bonferroni correction.

4 Agent Coverage Metrics

Traditional code coverage metrics—statement, branch, path, condition—are defined over deterministic control-flow graphs extracted from source code. AI agents do not have source code in the traditional sense; their "code" is a composition of prompts, tool inventories, model weights, and orchestration logic. This section defines agent-specific coverage metrics that measure the thoroughness of a stochastic test suite.

4.1 Coverage Dimensions

Definition 4.1 (Agent Coverage Tuple)**.** The *agent coverage* of a test suite $\mathcal{S}$ executed over N total trials is a five-dimensional tuple:

$$\mathcal{C}(\mathcal{S}) = (C_{\text{tool}}, C_{\text{path}}, C_{\text{state}}, C_{\text{boundary}}, C_{\text{model}}) \in [0,1]^5 \tag{13}$$

We define each dimension below.

4.1.1 Tool Coverage

Definition 4.2 (Tool Coverage)**.** Let $\mathcal{T} = \{t_1, \ldots, t_k\}$ be the agent's available tool set and $\mathcal{T}_{\text{used}} \subseteq \mathcal{T}$ the subset of tools invoked across all traces in the test suite. *Tool coverage* is:

$$C_{\text{tool}} = \frac{|\mathcal{T}_{\text{used}}|}{|\mathcal{T}|} \tag{14}$$

Tool coverage is the simplest dimension: it measures whether the test suite exercises all available tools. An agent with 10 tools but tests that only trigger 3 has $C_{\text{tool}} = 0.30$, indicating that 7 tools are untested and may harbor latent regressions.

Remark 4.1. Tool coverage does not account for *how* a tool is used—only whether it is invoked. A tool called with a single trivial input achieves the same C_{tool} contribution as one exercised across diverse inputs. We address input diversity through boundary coverage (Definition 4.8).

4.1.2 Decision-Path Coverage

An agent's execution trace follows a path through a decision space defined by tool selections, branching conditions in the orchestration logic, and model outputs at each step.

Definition 4.3 (Decision Path)**.** A *decision path* is the sequence of action-tool pairs in a trace: $\rho(\tau) = ((a_1, t_1), (a_2, t_2), \ldots, (a_m, t_m))$. Two traces τ, τ' have the same decision path if $\rho(\tau) = \rho(\tau')$.

Definition 4.4 (Decision-Path Coverage)**.** Let $\mathcal{P}_{\text{obs}}$ be the set of distinct decision paths observed across all traces and $\mathcal{P}_{\text{est}}$ be an estimate of the total number of feasible decision paths. *Decision-path coverage* is:

$$C_{\text{path}} = \frac{|\mathcal{P}_{\text{obs}}|}{|\mathcal{P}_{\text{est}}|} \tag{15}$$

The denominator $|\mathcal{P}_{\text{est}}|$ is, in general, intractable to compute exactly (the path space may be infinite for open-ended agents). We employ two estimation strategies:

1. **Capture-recapture estimation.** Using the Chao1 estimator from ecology: if we observe d distinct paths, of which f_1 appear exactly once and f_2 appear exactly twice, then:

$$|\mathcal{P}_{\text{est}}| \approx d + \frac{f_1^2}{2f_2} \tag{16}$$

2. **Orchestration-graph analysis.** For agents with explicit orchestration graphs (e.g., LangGraph state machines), enumerate paths up to a maximum depth $d_{\max}$, yielding a finite upper bound.

4.1.3 State-Space Coverage

Definition 4.5 (Agent State)**.** An *agent state* is a tuple $\sigma = (\ell, \mathbf{v}, h)$ where ℓ is the current *location* in the orchestration graph, $\mathbf{v}$ is the vector of *state variables* (e.g., accumulated context, tool outputs, counters), and h is a hash of the *conversation history* up to a fixed window.

Definition 4.6 (State-Space Coverage)**.** Let $\mathcal{S}_{\text{obs}}$ be the set of distinct states observed across all traces. Since the full state space is infinite (due to natural language content), we project states onto a finite abstraction $\pi : \Sigma \to \hat{\Sigma}$ and define:

$$C_{\text{state}} = 1 - e^{-|\pi(\mathcal{S}_{\text{obs}})|/\lambda} \tag{17}$$

where $\lambda > 0$ is a normalization constant set to the expected number of distinct abstract states for the agent class (calibrated empirically). This formulation maps to $[0, 1)$ and asymptotically approaches 1 as more states are discovered.

The exponential normalization (Eq. (17)) addresses the open-ended nature of agent state spaces. Unlike tool or path coverage, which have natural denominators, state-space exploration has no fixed ceiling. The exponential form ensures diminishing returns: the first 50 distinct states contribute more than the next 50.

4.1.4 Boundary Coverage

Definition 4.7 (Tool Boundary)**.** A *tool boundary* for tool t_j is a pair $(p_j^{\text{name}}, \{v^{\min}, v^{\max}\})$ where p_j^{name} is a parameter name and $v^{\min}, v^{\max}$ are the minimum and maximum values of the parameter's domain (or representative extremes for string/object types). Let $\mathcal{B}$ denote the set of all tool boundaries.

Definition 4.8 (Boundary Coverage)**.** Let $\mathcal{B}_{\text{tested}} \subseteq \mathcal{B}$ be the set of boundaries for which at least one trace invoked the tool with a parameter value at or near the boundary (within tolerance ϵ). *Boundary coverage* is:

$$C_{\text{boundary}} = \frac{|\mathcal{B}_{\text{tested}}|}{|\mathcal{B}|} \tag{18}$$

Boundary coverage extends the classical boundary-value analysis to agent tool invocations. It tests whether the agent exercises edge cases of its tools—empty lists, maximum-length strings, zero values, negative numbers—which are common sources of failures.

4.1.5 Model Coverage

Definition 4.9 (Model Coverage)**.** Let $\mathcal{M}_{\text{target}}$ be the set of language models the agent is intended to support (e.g., GPT-4o, Claude 3.5 Sonnet, Gemini 2.0 Flash, Llama 3.3 70B) and $\mathcal{M}_{\text{tested}} \subseteq \mathcal{M}_{\text{target}}$ the subset tested. *Model coverage* is:

$$C_{\text{model}} = \frac{|\mathcal{M}_{\text{tested}}|}{|\mathcal{M}_{\text{target}}|} \tag{19}$$

Model coverage captures a dimension unique to LLM-based systems: the same agent deployed with different models can exhibit dramatically different behaviors. An agent tested only with GPT-4o may fail catastrophically on Claude 3.5 Sonnet due to differences in tool-calling conventions, reasoning patterns, or output formatting.

4.2 Aggregate Coverage

Definition 4.10 (Overall Coverage)**.** The *overall coverage* is the geometric mean of the five dimensions:

$$C_{\text{overall}} = \left(\prod_{i=1}^{5} C_i\right)^{1/5} \tag{20}$$

We use the geometric mean rather than the arithmetic mean because it penalizes imbalance: a test suite with $C_{\text{tool}} = 1.0$ but $C_{\text{model}} = 0.0$ should receive low overall coverage, as the zero model coverage represents a critical gap.

Remark 4.2 (Weighted Coverage). In practice, different coverage dimensions may have different importance. We support weighted geometric means: $C^{(w)}_{\text{overall}} = \prod C_i^{w_i}$ with $\sum w_i = 1$, allowing teams to prioritize dimensions relevant to their deployment context.

4.3 Coverage Monotonicity

Theorem 4.1 (Coverage Monotonicity)**.** *For any test suite $\mathcal{S}$ and any additional test execution:*

1. *$C_{tool}, C_{boundary}, C_{model}$ are non-decreasing, and strictly increase when a new tool, boundary, or model is observed.*
2. *C_{state} is non-decreasing in the number of distinct states observed.*
3. *C_{path} is* asymptotically monotone*: as $|\mathcal{P}_{obs}| \to \infty$, the Chao1 correction vanishes and $C_{path} \to 1$. For finite test suites, C_{path} may temporarily decrease when a new singleton path increases the Chao1 denominator faster than the numerator; this non-monotonicity is bounded by $O(f_1^2/(f_2 \cdot |\mathcal{P}_{obs}|^2))$.*

Proof sketch. Parts (1) and (2) follow directly: numerators are non-decreasing while denominators are fixed (part 1), or the coverage function is monotone in its argument (part 2). Part (3) follows from the asymptotic consistency of the Chao1 estimator [8]: as $n \to \infty$, $\hat{S}_{\text{Chao1}} \to S_{\text{total}}$ and $C_{\text{path}} \to 1$. See Appendix A.2 for the formal argument. □

Algorithm 1 Coverage-Guided Test Input Generation

Require: Agent A, current coverage $\mathcal{C}$, target coverage $\mathcal{C}^*$
Ensure: New test scenario S_{new}
$d^* \leftarrow \arg\min_{d \in \{1,\ldots,5\}} C_d / C_d^*$ {Identify weakest dimension}
if $d^* = \text{tool}$ **then**
  Generate input x designed to trigger unused tools $\mathcal{T} \setminus \mathcal{T}_{\text{used}}$
else if $d^* = \text{path}$ **then**
  Generate input x to explore under-represented orchestration branches
else if $d^* = \text{state}$ **then**
  Generate input x to reach novel state regions
else if $d^* = \text{boundary}$ **then**
  Generate input x with extreme parameter values for untested boundaries
else if $d^* = \text{model}$ **then**
  Select untested model $\mu \in \mathcal{M}_{\text{target}} \setminus \mathcal{M}_{\text{tested}}$
end if
return $S_{\text{new}} = (x, \mathcal{P}, E)$

4.4 Coverage-Guided Test Generation

Coverage metrics naturally drive test generation: the framework identifies the lowest-coverage dimension and generates inputs designed to increase it.

Proposition 4.2 (Coverage-Fault Correlation)**.** *Higher agent coverage correlates with higher fault detection probability. Formally, for two test suites* $\mathcal{S}_1, \mathcal{S}_2$ *with* $\mathcal{C}(\mathcal{S}_1) \geq \mathcal{C}(\mathcal{S}_2)$ *(component-wise), the expected number of distinct faults detected satisfies* $\mathbb{E}[F(\mathcal{S}_1)] \geq \mathbb{E}[F(\mathcal{S}_2)]$ *under mild regularity conditions on the fault distribution.*

This proposition mirrors the well-established correlation between code coverage and fault detection in traditional software testing [36]. We validate it empirically in Section 9.2.

5 Agent Mutation Testing

Mutation testing assesses test suite quality by measuring its ability to detect injected faults. For traditional software, mutants are syntactic source code modifications. For AI agents, the "source" comprises prompts, tool configurations, models, and context—requiring entirely new mutation operators. This section defines four classes of agent-specific mutation operators, formalizes the mutation score under stochastic semantics, and proves a mutation adequacy theorem.

5.1 Agent Mutation Operators

Definition 5.1 (Agent Mutation Operator)**.** An *agent mutation operator* is a function $m : \mathcal{A} \to \mathcal{A}$ that transforms an agent configuration into a *mutant* agent $A' = m(A)$ by modifying exactly one component of $A = (\pi, \mathcal{T}, \mu, \omega)$.

We define four classes of mutation operators, each targeting a different component.

5.1.1 Prompt Mutation Operators (M_{prompt})

Definition 5.2 (Prompt Mutations)**.** The class M_{prompt} consists of four operators:

1. **Synonym substitution** (m_{syn}): Replace a key instruction word with a synonym. For example, "`Summarize the document`" → "`Condense the document`".

2. **Instruction reordering** (m_{reorder}): Permute the order of instructions within the prompt, preserving content but altering priority. If $\pi = (i_1, i_2, \ldots, i_k)$, then $m_{\text{reorder}}(\pi) = (i_{\sigma(1)}, \ldots, i_{\sigma(k)})$ for a random permutation σ.

3. **Noise injection** (m_{noise}): Insert irrelevant or distracting text into the prompt. Models the effect of prompt pollution or context contamination.

4. **Instruction dropout** (m_{drop}): Remove a randomly selected instruction from the prompt. Models the effect of incomplete or truncated prompts.

Remark 5.1. Prompt mutations target the *semantic resilience* of the agent: a well-designed agent should be robust to minor prompt perturbations (synonym substitution, reordering) but sensitive to meaningful changes (instruction dropout). The mutation score thus measures the test suite's ability to distinguish semantic from non-semantic changes.

5.1.2 Tool Mutation Operators (M_{tool})

Definition 5.3 (Tool Mutations)**.** The class M_{tool} consists of three operators:

1. **Tool removal** (m_{remove}): Remove a randomly selected tool from $\mathcal{T}$, producing $\mathcal{T}' = \mathcal{T} \setminus \{t_j\}$. Tests whether the agent gracefully degrades or fails silently.

2. **Tool reordering** (m_{treorder}): Permute the order of tools in the tool inventory (relevant when tool descriptions are presented to the LLM as a list). Models the position bias exhibited by some language models.

3. **Tool noise** (m_{tnoise}): Modify a tool's description or schema by injecting misleading information. Models the effect of tool description errors or adversarial tool definitions [3].

5.1.3 Model Mutation Operators (M_{model})

Definition 5.4 (Model Mutations)**.** The class M_{model} consists of two operators:

1. **Model swap** (m_{swap}): Replace the underlying LLM μ with a different model from the same capability tier. For example, replace GPT-4o with Claude 3.5 Sonnet. Tests cross-model compatibility.

2. **Version downgrade** (m_{version}): Replace the model with an older or less capable version (e.g., GPT-4o $\rightarrow$ GPT-4o-mini). Tests graceful degradation under model changes.

5.1.4 Context Mutation Operators (M_{context})

Definition 5.5 (Context Mutations)**.** The class M_{context} consists of three operators:

1. **Context truncation** (m_{trunc}): Truncate the input context to $\lfloor \gamma \cdot |x| \rfloor$ tokens for $\gamma \in (0, 1)$. Models context window overflow.

2. **Context noise** (m_{cnoise}): Inject irrelevant documents or passages into the context. Models the effect of noisy retrieval in RAG pipelines.

3. **Context permutation** (m_{cperm}): Permute the order of context segments. Models sensitivity to document ordering in retrieval-augmented settings.

5.2 Stochastic Kill Semantics

In traditional mutation testing, a mutant is "killed" if the test produces a different output. For stochastic agents, outputs differ across runs even without mutations. We must therefore define kill semantics that distinguish *mutation-induced* behavioral changes from *natural stochastic variation.*

Definition 5.6 (Stochastic Mutation Kill)**.** Let A be the original agent and $A' = m(A)$ a mutant. Let $\mathcal{S}$ be a test suite with scenarios $\{S_1, \ldots, S_k\}$. The mutant A' is *killed* by $\mathcal{S}$ if *at least one* of the following conditions holds for some scenario S_j:

1. **Verdict change:** $V(A, S_j) \neq V(A', S_j)$ where V is the stochastic verdict function (Definition 3.6).
2. **Significant score difference:** The pass rates differ significantly:
$$|\hat{p}(A, S_j) - \hat{p}(A', S_j)| \geq \delta_{\text{min}} \quad \text{and} \quad p\text{-value}(H_0 : p_A = p_{A'}) < \alpha_{\text{kill}} \tag{21}$$
where δ_{min} is a minimum effect size threshold and α_{kill} is the kill significance level.
3. **Distributional shift:** The distribution of outputs (or trace features) differs significantly, as measured by a two-sample test (e.g., Kolmogorov-Smirnov for continuous metrics or χ^2 for categorical outcomes).

Remark 5.2. Condition (1) is the strongest: the test suite's verdict flips from Pass to Fail (or vice versa). Condition (2) is more sensitive: it detects changes even when the verdict remains the same. Condition (3) is the most general: it catches distributional shifts in agent behavior (e.g., the agent now takes longer paths or uses different tools) even if the pass rate is unchanged.

Definition 5.7 (Equivalent Mutant)**.** A mutant $A' = m(A)$ is *equivalent* if, for all inputs $x \in \mathcal{X}$ and all sample sizes n, the output distributions of A and A' are identical: $A(x) \overset{d}{=} A'(x)$. Equivalent mutants cannot be killed by any test suite and are excluded from the mutation score.

The equivalent mutant problem is undecidable in general. For agents, we use a practical heuristic: a mutant is *presumed equivalent* if, after n_{equiv} trials on each of k_{equiv} scenarios, no statistically significant difference is detected at level α_{equiv}.

5.3 Mutation Score

Definition 5.8 (Agent Mutation Score)**.** Let $\mathcal{M}(A) = \{A'_1, \ldots, A'_q\}$ be the set of mutants generated from agent A and $\mathcal{M}_{\text{equiv}} \subseteq \mathcal{M}(A)$ the set of equivalent mutants. The *agent mutation score* of test suite $\mathcal{S}$ is:

$$\text{MS}(\mathcal{S}, A) = \frac{|\{A'_i \in \mathcal{M}(A) \setminus \mathcal{M}_{\text{equiv}} : A'_i \text{ is killed by } \mathcal{S}\}|}{|\mathcal{M}(A) \setminus \mathcal{M}_{\text{equiv}}|} \tag{22}$$

Remark 5.3 (Class-Specific Mutation Scores). We also report mutation scores per operator class: $\text{MS}_{\text{prompt}}, \text{MS}_{\text{tool}}, \text{MS}_{\text{model}}, \text{MS}_{\text{context}}$. These fine-grained scores identify which aspects of the agent are well-tested and which require additional test scenarios.

5.4 Mutation Adequacy

Theorem 5.1 (Mutation Adequacy)**.** *Let $\mathcal{S}$ be a test suite with mutation score MS$(\mathcal{S}, A) \geq \tau$ for threshold $\tau \in (0, 1]$. Let $\delta > 0$ be a behavioral impact threshold. If the mutation operators satisfy the* coupling effect *assumption—that complex faults are coupled to simple faults detectable by the mutation operators—then for any real regression $A \to A^*$ with behavioral impact $\Delta(A, A^*) \geq \delta$:*

$$\mathbb{P}[\mathcal{S} \textit{ detects the regression}] \geq \tau \cdot \left(1 - e^{-\delta/\delta_0}\right) \tag{23}$$

where δ_0 is a characteristic scale parameter of the mutation operator suite, determined by the average behavioral impact of a single mutation.

Proof sketch. The proof proceeds by contrapositive. Suppose the test suite fails to detect a regression of magnitude δ. By the coupling effect, this regression can be decomposed into a composition of simple mutations. Since the test suite achieves mutation score τ, it detects a fraction τ of all simple mutations. The probability that *none* of the constituent simple mutations are detected decreases exponentially with δ/δ_0, yielding the bound. The full proof, including the formal coupling assumption and its justification for agent systems, is in Appendix A.3. ☐

Corollary 5.2. *If $MS(\mathcal{S}, A) = 1$ (all non-equivalent mutants killed) and the mutation operators are* complete *(every behavioral change of magnitude $\geq \delta$ includes at least one mutation from $\mathcal{M}$), then $\mathbb{P}[\mathcal{S}$ detects regression$] \to 1$ as $\delta/\delta_0 \to \infty$.*

5.5 Cost-Aware Mutation Testing

Running a full mutation analysis is expensive: for q mutants, each tested over n trials on k scenarios, the total cost is $O(q \cdot n \cdot k)$ agent invocations. We introduce two cost reduction strategies.

Definition 5.9 (Selective Mutation)**.** Rather than generating all possible mutants, *selective mutation* generates mutants only from a curated subset of operators identified as most effective. Let $\mathcal{M}^* \subset \mathcal{M}$ be the selected operator subset. The *selective mutation score* is:

$$\text{MS}^*(\mathcal{S}, A) = \frac{|\text{killed}(\mathcal{S}, \mathcal{M}^*(A))|}{|\mathcal{M}^*(A) \setminus \mathcal{M}_{\text{equiv}}|} \tag{24}$$

Definition 5.10 (SPRT-Accelerated Mutation Kill)**.** Rather than running n fixed trials to determine if a mutant is killed, apply the SPRT (Definition 3.9) to stop as soon as the kill/survive decision is clear. Formally, test $H_0 : p_{A'} = p_A$ vs. $H_1 : |p_{A'} - p_A| \geq \delta_{\text{min}}$ sequentially, terminating when the log-likelihood ratio crosses a boundary.

Proposition 5.3 (Mutation Cost Reduction)**.** *Combining selective mutation (selecting a $\gamma = |\mathcal{M}^*|/|\mathcal{M}|$ fraction of operators) with SPRT-accelerated kills (achieving expected savings factor σ) reduces total mutation testing cost by:*

$$\textit{Cost reduction} = 1 - \gamma \cdot \sigma \tag{25}$$

With $\gamma = 0.3$ (30% of operators) and $\sigma = 0.5$ (50% fewer trials via SPRT), the total cost is reduced by 85%.

5.6 Mutation Operator Design Principles

We identify four principles for designing effective agent mutation operators:

1. **Semantic granularity.** Mutations should produce behavioral changes at varying granularity levels—from subtle (synonym substitution) to severe (model swap)—mirroring the spectrum of real-world agent changes.
2. **Component isolation.** Each mutation should modify exactly one component, enabling precise attribution of detected regressions to specific agent aspects.
3. **Ecological validity.** Mutations should model realistic changes that occur in practice: prompt edits, tool updates, model upgrades, context changes. Arbitrary random perturbations are less informative.
4. **Composability.** Individual mutations should compose to model complex changes. If mutations m_1 and m_2 are individually meaningful, then $m_2 \circ m_1$ should model a realistic compound change.

6 Metamorphic Relations and CI/CD Deployment Gates

6.1 Metamorphic Relations for Agent Workflows

The oracle problem—determining whether an agent's output is correct—is particularly acute for open-ended agent tasks. Metamorphic testing [9] sidesteps this by checking *relations* between outputs rather than the outputs themselves. We define four families of agent-specific metamorphic relations (MRs).

Definition 6.1 (Metamorphic Relation)**.** A *metamorphic relation* $\mathcal{R}$ is a property over pairs of agent executions: given source input x and follow-up input $x' = \phi(x)$ derived by a *source transformation* ϕ, the relation $\mathcal{R}(A(x), A(x'))$ must hold. A violation $\neg\mathcal{R}(A(x), A(x'))$ indicates a potential fault.

6.1.1 Permutation Relations ($\mathcal{R}_{\text{perm}}$)

These relations assert that the agent's behavior should be invariant (or predictably variant) under permutations of the input.

1. **Document order invariance.** For RAG agents: permuting the order of retrieved documents should not change the factual content of the answer.

$$\mathcal{R}_{\text{doc}} : \text{facts}(A(x)) = \text{facts}(A(\sigma(x)))$$

   where σ permutes the document ordering within x.

2. **Tool order invariance.** Permuting the order of tools in the tool inventory should not change the final output for tasks with a clear solution.

$$\mathcal{R}_{\text{tool}} : \text{result}(A_{\mathcal{T}}) \approx \text{result}(A_{\sigma(\mathcal{T})})$$

6.1.2 Perturbation Relations ($\mathcal{R}_{\text{pert}}$)

These relations assert proportional or bounded response to input perturbations.

1. **Monotonic difficulty.** For agents that solve tasks of varying complexity: a harder input should not produce a higher confidence score than an easier one (in expectation).

$$\mathcal{R}_{\text{diff}} : \text{difficulty}(x') > \text{difficulty}(x) \implies \mathbb{E}[\text{score}(A(x'))] \leq \mathbb{E}[\text{score}(A(x))]$$

2. **Noise robustness.** Adding a small amount of noise to the input should not change the verdict.

$$\mathcal{R}_{\text{noise}} : \|x' - x\| \leq \epsilon \implies V(A, x') = V(A, x)$$

   with high probability over the stochastic execution.

6.1.3 Composition Relations ($\mathcal{R}_{\text{comp}}$)

These relations govern multi-agent pipelines and sequential workflows.

1. **Pipeline consistency.** For a pipeline $A = A_2 \circ A_1$: if A_1 produces a correct intermediate result y, then $A_2(y)$ should produce a correct final result.

$$\mathcal{R}_{\text{pipe}} : E_1(x, A_1(x)) = 1 \implies \mathbb{P}[E_2(A_1(x), A_2(A_1(x))) = 1] \geq \theta$$

2. **Agent substitutability.** If two agents A_1, A_1' are functionally equivalent on a given input class, substituting one for the other in a pipeline should not change the pipeline's verdict.

6.1.4 Oracle Relations ($\mathcal{R}_{\textbf{oracle}}$)

These relations use known properties of the problem domain as partial oracles.

1. **Format compliance.** The output must conform to a specified format (JSON, markdown, specific schema) regardless of input.
2. **Constraint preservation.** If the input specifies constraints (e.g., "respond in at most 100 words"), the output must satisfy them.
3. **Idempotence.** For certain tasks (e.g., code formatting, data cleaning), applying the agent twice should produce the same result as applying it once: $A(A(x)) \approx A(x)$.

Proposition 6.1 (MR Violation as Regression Signal)**.** *If a metamorphic relation $\mathcal{R}$ holds for agent version A_v (empirically verified over n trial pairs with violation rate $\hat{q}_v \leq \epsilon$) but is violated at rate $\hat{q}_{v'} > \epsilon + \delta$ for version $A_{v'}$, this constitutes evidence of regression with respect to the property encoded by $\mathcal{R}$, testable via the regression verdict framework of Section 3.4.*

6.2 CI/CD Deployment Gates

Modern software delivery relies on CI/CD pipelines that automatically build, test, and deploy code. For agents, deployment gates must incorporate the stochastic testing framework to prevent regressions from reaching production.

Definition 6.2 (Deployment Gate)**.** A *deployment gate* is a decision function $G : \mathcal{V}^m \times \mathcal{C} \to \{\text{deploy}, \text{block}, \text{manual}\}$ that maps the verdicts of m test scenarios and the coverage tuple $\mathcal{C}$ to a deployment decision:

$$G(\mathbf{V}, \mathcal{C}) = \begin{cases} \text{deploy} & \text{if } V_{\text{suite}} = \textsc{Pass} \text{ and } C_{\text{overall}} \geq C_{\min} \\ \text{block} & \text{if } V_{\text{suite}} = \textsc{Fail} \\ \text{manual} & \text{if } V_{\text{suite}} = \textsc{Inconclusive} \text{ or } C_{\text{overall}} < C_{\min} \end{cases} \tag{26}$$

where $C_{\min} \in [0, 1]$ is the minimum acceptable coverage.

The three-valued deployment decision mirrors the three-valued test verdict. Notably, INCONCLUSIVE maps to *manual review*, not automatic blocking—this reflects the pragmatic reality that teams may accept inconclusive results with human oversight rather than blocking all deployments.

Definition 6.3 (Gate Configuration)**.** A gate is configured by a tuple

$$\Gamma = (\alpha, \beta, \delta, \theta, C_{\min}, n_{\max}, t_{\max})$$

where:

- α, β are the significance and Type II error bounds,
- δ is the minimum regression magnitude to detect,
- θ is the pass rate threshold,
- $C_{\min}$ is the minimum coverage,
- $n_{\max}$ is the maximum trials per scenario (budget cap),
- $t_{\max}$ is the maximum wall-clock time.

Algorithm 2 CI/CD Deployment Gate

Require: Agent A, baseline B, test suite $\mathcal{S}$, gate configuration Γ
Ensure: Decision $\in \{\text{deploy}, \text{block}, \text{manual}\}$
for $S_j \in \mathcal{S}$ **do**
    $n \leftarrow \min(n^*(\alpha, \beta, \delta), n_{\max})$
    Run n trials of A on S_j using SPRT (Definition 3.9)
    Compute $V_j \leftarrow V_{\text{reg}}(\mathbf{r}_b, \mathbf{r}_c; \alpha, \beta, \delta)$
end for
Compute $V_{\text{suite}} \leftarrow V_{\text{suite}}(\{V_1, \ldots, V_m\})$ with Holm-Bonferroni correction
Compute $\mathcal{C} \leftarrow$ coverage tuple
return $G(V_{\text{suite}}, \mathcal{C})$

6.3 Integration with Existing CI/CD Systems

AgentAssay integrates with standard CI/CD systems through two mechanisms:

1. **pytest plugin.** The framework registers as a pytest plugin, enabling standard test execution via `pytest --agentassay`. Test results are reported in standard JUnit XML format, compatible with GitHub Actions, GitLab CI, Jenkins, and CircleCI.

2. **Exit codes.** The gate decision maps to process exit codes: `0` (deploy/pass), `1` (block/fail), `2` (manual/inconclusive). CI/CD systems interpret non-zero exit codes as failures, with `2` configurable as a warning rather than a hard failure.

Remark 6.1 (Cost Budgets). In CI/CD contexts, testing cost is a critical constraint. The gate configuration parameter $n_{\max}$ caps the per-scenario trial count, and SPRT (Section 3.5) ensures early termination when evidence is clear. For typical CI pipelines, we recommend $n_{\max} = 30$ with SPRT, which provides adequate power for detecting regressions of $\delta \geq 0.15$ at $\alpha = 0.05$.

7 Token-Efficient Testing

Stochastic testing, by construction, requires n independent agent executions per scenario. Each execution consumes C_{run} tokens at cost c_{run} dollars. A test suite of m scenarios at n trials each therefore costs $m \times n \times c_{\text{run}}$. For frontier models where a single complex agent run costs \$5–15, a test campaign of 50 scenarios at $n = 100$ trials implies 5,000 invocations costing \$25,000–75,000 *per release*—prohibitive for any CI/CD pipeline. Even with SPRT (Section 3.5) reducing n by roughly 50%, the cost remains impractical for most teams.

This section presents five techniques—collectively *token-efficient testing*—that achieve the same (α, β) statistical guarantees at 5–20× lower cost. The core insight is that agent behavior, while stochastic in its textual output, exhibits *low-dimensional behavioral regularity*: the distribution over tool sequences, reasoning depth, output structure, and efficiency metrics concentrates on a manifold far smaller than the raw output space. By testing in this compressed behavioral space rather than the raw output space, we obtain dramatically better sample efficiency.

7.1 Behavioral Fingerprinting

Definition 7.1 (Behavioral Fingerprint)**.** Let $\tau = (s_1, \ldots, s_m)$ be an execution trace (Definition 3.2). A *behavioral fingerprint* is a mapping $F : \mathcal{T}^* \to \mathbb{R}^d$ that extracts a compact *behavioral vector* from τ. The fingerprint $\mathbf{f} = F(\tau)$ is composed of six feature families:

1. **Tool usage distribution.** Let $\mathbf{u} \in \mathbb{R}^{|\mathcal{T}|}$ be the normalized frequency vector of tool calls in τ: $u_j = |\{i : t_i = t_j\}|/m$.

2. **Structural complexity.** Let $L(\tau) = m$ be the trace length, $D(\tau)$ the maximum nesting depth of tool calls, and $B(\tau)$ the number of distinct reasoning branches.

3. **Output characteristics.** Let $\ell(\tau)$ be the output token count of $\mathrm{out}(\tau)$, and $\kappa(\tau) \in [0, 1]$ a normalized complexity score of the final output (e.g., Flesch-Kincaid grade divided by a reference ceiling).

4. **Reasoning patterns.** Let $\mathbf{w} \in \mathbb{R}^k$ be the distribution over action types $(|\{i : a_i = a\}|/m)_{a\in\mathcal{A}}$.

5. **Error and recovery.** Let $e(\tau) \in \{0, 1\}$ indicate whether any step produced an error, and $\rho(\tau) \in [0, 1]$ the fraction of error steps followed by a successful recovery step.

6. **Efficiency metrics.** Let $C(\tau) = \sum_i c_i$ be the total cost and $\bar{c}(\tau) = C(\tau)/m$ the average step cost.

The full fingerprint is the concatenation $\mathbf{f} = (\mathbf{u}, L, D, B, \ell, \kappa, \mathbf{w}, e, \rho, C, \bar{c}) \in \mathbb{R}^d$ where $d = |\mathcal{T}|+|\mathcal{A}|+7$. In our implementation, we normalize $\mathbf{u}$ and $\mathbf{w}$ to fixed dimensions via zero-padding, yielding a practical fingerprint of $d = 14$ dimensions that enables cross-agent comparison.

Definition 7.2 (Fingerprint Space)**.** Given an agent A and input distribution $\mathcal{D}$ over $\mathcal{X}$, the *fingerprint space* is:

$$\mathcal{F}(A, \mathcal{D}) = \{F(\tau) : \tau \sim A(x),\ x \sim \mathcal{D}\} \subseteq \mathbb{R}^d \tag{27}$$

The *effective dimension* $d_{\text{eff}}(A, \mathcal{D})$ is the number of principal components required to explain $\geq 95\%$ of the variance in $\mathcal{F}$:

$$d_{\text{eff}} = \min\left\{k : \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{d} \lambda_i} \geq 0.95\right\} \tag{28}$$

where $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d \geq 0$ are the eigenvalues of the covariance matrix of $\mathcal{F}$.

Remark 7.1 (Low-Dimensional Behavioral Manifold)*.* Although the raw output space of an agent is extremely high-dimensional (the space of all possible text strings), the behavioral fingerprint concentrates on a manifold of much lower dimension. Intuitively, an agent configured with $k = 10$ tools and a specific system prompt will exhibit a characteristic pattern of tool usage, reasoning depth, and output structure that varies little across runs—even though the exact text changes every time. In our preliminary experiments, $d_{\text{eff}} \in [3, 8]$ for agents with $d \in [20, 40]$ fingerprint dimensions, indicating 4–10× dimensionality reduction.

Theorem 7.1 (Fingerprint Regression Detection)**.** *Let $F : \mathcal{T}^* \to \mathbb{R}^d$ be a behavioral fingerprint with effective dimension d_{eff}. Given baseline fingerprints $\mathbf{f}_1^b, \ldots, \mathbf{f}_{n_b}^b$ and candidate fingerprints $\mathbf{f}_1^c, \ldots, \mathbf{f}_{n_c}^c$, Hotelling's T^2 test on the projected fingerprints rejects $H_0 : \boldsymbol{\mu}_b = \boldsymbol{\mu}_c$ at significance α when:*

$$T^2 = \frac{n_b n_c}{n_b + n_c}(\bar{\mathbf{f}}_b - \bar{\mathbf{f}}_c)^\top \mathbf{S}_{\text{pooled}}^{-1}(\bar{\mathbf{f}}_b - \bar{\mathbf{f}}_c) > \frac{(n_b + n_c - 2)d_{\text{eff}}}{n_b + n_c - d_{\text{eff}} - 1} F_{\alpha,\, d_{\text{eff}},\, n_b+n_c-d_{\text{eff}}-1} \tag{29}$$

where $\mathbf{S}_{\text{pooled}}$ is the pooled covariance matrix projected onto the top d_{eff} principal components, and F_{α,d_1,d_2} is the critical value of the F-distribution.

The sample complexity for achieving (α, β)-guaranteed regression detection via Hotelling's T^2 is:

$$n_{\text{fp}}(\alpha, \beta, d_{\text{eff}}, \Delta_{\text{M}}) = \left\lceil \frac{(d_{\text{eff}} + 1)(z_{1-\alpha} + z_{1-\beta})^2}{\Delta_{\text{M}}^2} + \frac{d_{\text{eff}} + 1}{2} \right\rceil \tag{30}$$

where $\Delta_{\mathrm{M}}^2 = (\boldsymbol{\mu}_b - \boldsymbol{\mu}_c)^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_b - \boldsymbol{\mu}_c)$ is the squared Mahalanobis distance between the baseline and candidate fingerprint distributions. When $d_{\mathrm{eff}} \ll \dim(\text{output space})$ and $\Delta_{\mathrm{M}} \geq \Delta_{\mathrm{M,min}}$, we have:

$$n_{\mathrm{fp}} \leq n^*_{\mathrm{univariate}} \times \frac{d_{\mathrm{eff}} + 1}{1 + d_{\mathrm{eff}} \cdot (\Delta_{\mathrm{M}}^2/h^2 - 1)} \tag{31}$$

where h is Cohen's effect size for the univariate pass-rate test.

Proof. The proof proceeds in three steps.

Step 1: Multivariate test power. Under $H_1 : \boldsymbol{\mu}_b \neq \boldsymbol{\mu}_c$, the Hotelling T^2 statistic follows a non-central F-distribution:

$$\frac{n_b + n_c - d_{\mathrm{eff}} - 1}{(n_b + n_c - 2)d_{\mathrm{eff}}} \cdot T^2 \sim F_{d_{\mathrm{eff}},\, n_b+n_c-d_{\mathrm{eff}}-1}(\lambda) \tag{32}$$

with non-centrality parameter $\lambda = \frac{n_b n_c}{n_b+n_c}\Delta_{\mathrm{M}}^2$. The power is:

$$1 - \beta = \mathbb{P}[F_{d_{\mathrm{eff}},\, n-d_{\mathrm{eff}}-1}(\lambda) > F_{\alpha,\, d_{\mathrm{eff}},\, n-d_{\mathrm{eff}}-1}] \tag{33}$$

where $n = n_b + n_c$.

Step 2: Sample size derivation. For balanced designs ($n_b = n_c = n/2$), the non-centrality parameter becomes $\lambda = n\Delta_{\mathrm{M}}^2/4$. Using the normal approximation to the non-central F-distribution valid when $n \gg d_{\mathrm{eff}}$ [24], the power condition $1 - \beta$ yields:

$$\sqrt{\lambda} - \sqrt{d_{\mathrm{eff}}} \geq z_{1-\alpha} + z_{1-\beta} \tag{34}$$

Solving for n with $\lambda = n\Delta_{\mathrm{M}}^2/4$:

$$n \geq \frac{4(z_{1-\alpha} + z_{1-\beta} + \sqrt{d_{\mathrm{eff}}})^2}{\Delta_{\mathrm{M}}^2} \tag{35}$$

The stated bound (30) follows by applying the tighter Lachin approximation [19] and rounding up.

Step 3: Comparison with univariate testing. The univariate pass-rate test (Theorem 3.2) requires $n^*_{\mathrm{univariate}} = \lceil (z_{1-\alpha} + z_{1-\beta})^2 \cdot 2\bar{p}(1-\bar{p})/\delta^2 \rceil$ samples. The univariate test uses a single scalar (pass/fail), while the fingerprint test uses a d_{eff}-dimensional vector.

The key advantage is that Δ_{M} captures *all* behavioral changes simultaneously—not just the pass rate. A regression that changes the tool usage pattern without affecting the pass rate (e.g., the agent starts using a slower tool chain that happens to still produce correct outputs) will have $\Delta_{\mathrm{M}} > 0$ even when $\delta_{\text{pass-rate}} = 0$. This makes the fingerprint test strictly more powerful for detecting behavioral regressions.

When the regression manifests primarily in the pass rate, we have $\Delta_{\mathrm{M}} \geq h/\sqrt{\bar{p}(1-\bar{p})}$ (since the pass rate is one component of the fingerprint), and the bound (31) shows that $n_{\mathrm{fp}} \leq n^*_{\mathrm{univariate}}$ provided $\Delta_{\mathrm{M}}^2/h^2 \geq 1$, which holds when any non-trivial behavioral shift accompanies the pass-rate change. □

7.2 Adaptive Budget Optimization

Even with fingerprint-based multivariate testing, we must still choose how many trials n to run. The fixed-sample approach (Eq. (1)) is conservative: it allocates n based on worst-case variance $\hat{p}(1-\hat{p}) \leq 1/4$. For agents that are *stable* (low behavioral variance), far fewer trials suffice.

Algorithm 3 Behavioral Fingerprint Extraction

Require: Execution trace $\tau = (s_1, \ldots, s_m)$, tool set $\mathcal{T}$, action set $\mathcal{A}$
Ensure: Behavioral fingerprint $\mathbf{f} \in \mathbb{R}^d$

$\mathbf{u} \leftarrow \mathbf{0}_{|\mathcal{T}|}$ {Tool usage vector}
$\mathbf{w} \leftarrow \mathbf{0}_{|\mathcal{A}|}$ {Action distribution}
$D_{\max} \leftarrow 0;\ B \leftarrow 0;\ e \leftarrow 0;\ n_{\text{err}} \leftarrow 0;\ n_{\text{rec}} \leftarrow 0$
for $i = 1$ **to** m **do**
  Parse step $s_i = (a_i, t_i, o_i, c_i)$
  if $t_i \neq \perp$ **then**
    $u_{t_i} \leftarrow u_{t_i} + 1$
  end if
  $w_{a_i} \leftarrow w_{a_i} + 1$
  Update $D_{\max}$, B from control flow
  if o_i indicates error **then**
    $e \leftarrow 1;\ n_{\text{err}} \leftarrow n_{\text{err}} + 1$
    if $i < m$ **and** o_{i+1} indicates success **then**
      $n_{\text{rec}} \leftarrow n_{\text{rec}} + 1$
    end if
  end if
end for
$\mathbf{u} \leftarrow \mathbf{u}/m;\ \mathbf{w} \leftarrow \mathbf{w}/m$ {Normalize}
$\rho \leftarrow n_{\text{rec}} / \max(n_{\text{err}}, 1)$
$\mathbf{f} \leftarrow (\mathbf{u}, m, D_{\max}, B, |\text{out}(\tau)|, \kappa(\tau), \mathbf{w}, e, \rho, \sum_i c_i, \bar{c})$
return $\mathbf{f}$

Definition 7.3 (Token Budget)**.** A *token budget* is a constraint $\mathcal{B} = (B_{\text{tok}}, B_{\$})$ specifying the maximum tokens and monetary cost available for a test campaign. The per-trial cost of agent A on scenario S is:

$$c_A(S) = \mathbb{E}_{\tau \sim A(S.x)} \left[\sum_{i=1}^{|\tau|} c_i \right] \tag{36}$$

The maximum number of affordable trials is $n_{\max} = \lfloor B_{\$} / c_A(S) \rfloor$.

Definition 7.4 (Behavioral Variance Class)**.** Given calibration fingerprints $\mathbf{f}_1, \ldots, \mathbf{f}_k$ from k calibration runs of agent A, compute the total fingerprint variance:

$$\hat{\sigma}^2_{\text{fp}} = \frac{1}{k-1} \sum_{i=1}^{k} \|\mathbf{f}_i - \bar{\mathbf{f}}\|^2 \tag{37}$$

The agent's *variance class* is:

$$\mathcal{V}(A) = \begin{cases} \text{stable} & \text{if } \hat{\sigma}^2_{\text{fp}} < \sigma^2_{\text{low}} \\ \text{moderate} & \text{if } \sigma^2_{\text{low}} \leq \hat{\sigma}^2_{\text{fp}} < \sigma^2_{\text{high}} \\ \text{volatile} & \text{if } \hat{\sigma}^2_{\text{fp}} \geq \sigma^2_{\text{high}} \end{cases} \tag{38}$$

where σ^2_{low} and σ^2_{high} are calibration thresholds (set empirically, defaults $\sigma^2_{\text{low}} = 0.05$, $\sigma^2_{\text{high}} = 0.25$).[1]

[1] Thresholds are empirically calibrated; the implementation uses total fingerprint variance thresholds of 1.5 (stable) and 5.0 (volatile), corresponding to approximately 0.107 and 0.357 per dimension for a 14-dimensional fingerprint.

Theorem 7.2 (Adaptive Budget Optimality)**.** *Given k calibration fingerprints with estimated variance $\hat{\sigma}^2_{\text{fp}}$ and the target Mahalanobis distance $\Delta_{\text{M,min}}$ for minimum detectable regression, the adaptive budget optimizer computes the optimal trial count:*

$$n^* = \left\lceil \frac{(z_{1-\alpha} + z_{1-\beta})^2 \cdot \hat{\sigma}^2_{\text{fp}}}{\Delta^2_{\text{M,min}}} + \frac{d_{\text{eff}} + 1}{2} \right\rceil \tag{39}$$

This estimate satisfies the following properties:

1. ***Never worse:*** *$n^* \leq n_{\text{fixed}}$ always, where n_{fixed} uses worst-case variance $1/4$.*
2. ***Proportional to actual variance:*** *$\mathbb{E}[n^*] = n_{\text{fixed}} \times (\hat{\sigma}^2_{\text{fp}}/\sigma^2_{\max})$, where $\sigma^2_{\max} = d \cdot 1/4$ is the maximum fingerprint variance.*
3. ***Stable-agent savings:*** *For agents in the stable variance class ($\hat{\sigma}^2_{\text{fp}} < \sigma^2_{low}$), the typical reduction is $n^* \in [15, 25]$ compared to $n_{\text{fixed}} \approx 100$.*
4. ***Statistical validity:*** *When $k \geq 10$ and $\hat{\sigma}^2_{\text{fp}}$ is within $\pm 30\%$ of σ^2_{true} (which holds with probability ≥ 0.95 by the chi-squared concentration bound), the actual error rates remain within $[\alpha, 1.5\alpha]$ and $[\beta, 1.5\beta]$.*

Proof. **Property (i).** The worst-case variance of a d-dimensional fingerprint with each component in $[0, 1]$ is bounded by $\sigma^2_{\max} = d/4$ (each component has variance at most $1/4$). By definition, $\hat{\sigma}^2_{\text{fp}} \leq \sigma^2_{\max}$, so $n^* \leq n_{\text{fixed}}$.

Property (ii). Follows directly from the linear relationship between n^* and $\hat{\sigma}^2_{\text{fp}}$ in (39), with the fixed-sample formula corresponding to $\hat{\sigma}^2_{\text{fp}} = \sigma^2_{\max}$.

Property (iii). For stable agents with $\hat{\sigma}^2_{\text{fp}} < 0.05$ and typical $d_{\text{eff}} \in [3, 8]$, substituting into (39) with $\alpha = 0.05$, $\beta = 0.10$, $\Delta_{\text{M,min}} = 0.5$:

$$n^* = \left\lceil \frac{(1.645 + 1.282)^2 \times 0.05}{0.25} + \frac{6}{2} \right\rceil = \lceil 1.71 + 3 \rceil = 5 \tag{40}$$

With a safety margin of $(1 + \sqrt{2/k})$ for finite-sample estimation uncertainty (where k is the calibration size), the practical recommendation is $n^* \in [15, 25]$.

Property (iv). The calibration variance $\hat{\sigma}^2_{\text{fp}}$ follows a scaled chi-squared distribution: $(k - 1)\hat{\sigma}^2_{\text{fp}}/\sigma^2_{\text{true}} \sim \chi^2_{k-1}$. For $k = 10$, the 95% confidence interval for σ^2_{true} is:

$$\left[\frac{(k-1)\hat{\sigma}^2_{\text{fp}}}{\chi^2_{0.975,k-1}}, \frac{(k-1)\hat{\sigma}^2_{\text{fp}}}{\chi^2_{0.025,k-1}} \right] = [0.51\hat{\sigma}^2_{\text{fp}},\ 2.30\hat{\sigma}^2_{\text{fp}}] \tag{41}$$

If the true variance exceeds our estimate by a factor of $\gamma = \sigma^2_{\text{true}}/\hat{\sigma}^2_{\text{fp}}$, the actual sample size needed is $n^* \times \gamma$. We run n^* trials, so the effective significance level becomes at most $\alpha \times \gamma^{1/2} \leq 1.5\alpha$ for $\gamma \leq 2.3$, which holds with probability 0.975. □

7.3 Trace-First Offline Analysis

The most dramatic cost reduction comes from the observation that *most testing activities do not require new agent executions.* If execution traces are already available—from production logging, previous test campaigns, or staging environments—four of the six test types in AgentAssay can execute at zero additional token cost.

Definition 7.5 (Trace Store)**.** A *trace store* is a persistent, versioned collection $\mathcal{T}_{\text{store}} = \{(\tau_i, x_i, v_i, t_i)\}_{i=1}^{N}$ of execution traces, where τ_i is the trace, x_i the input, v_i the agent version, and t_i the timestamp. A trace store supports two operations:

Algorithm 4 Adaptive Budget Calibration

Require: Agent A, scenario S, calibration size k, parameters $(\alpha, \beta, \Delta_{\mathrm{M,min}})$
Ensure: Budget estimate $(n^*, \mathcal{V}(A), \hat{\sigma}^2_{\mathrm{fp}})$

Phase 1: Calibration
for $i = 1$ **to** k **do**
  $\tau_i \leftarrow$ execute A on $S.x$
  $\mathbf{f}_i \leftarrow F(\tau_i)$ {Algorithm 3}
end for
$\bar{\mathbf{f}} \leftarrow \frac{1}{k}\sum_i \mathbf{f}_i$
$\hat{\sigma}^2_{\mathrm{fp}} \leftarrow \frac{1}{k-1}\sum_i \|\mathbf{f}_i - \bar{\mathbf{f}}\|^2$
Compute d_{eff} via PCA on $\{\mathbf{f}_1, \ldots, \mathbf{f}_k\}$
Classify $\mathcal{V}(A)$ per Definition 7.4
Phase 2: Budget computation
$n^* \leftarrow \lceil (z_{1-\alpha} + z_{1-\beta})^2 \hat{\sigma}^2_{\mathrm{fp}} / \Delta^2_{\mathrm{M,min}} + (d_{\mathrm{eff}} + 1)/2 \rceil$
$n^* \leftarrow \max(n^*, k + 5)$ {Minimum for reliable CI}
$n^* \leftarrow \lceil n^* \cdot (1 + \sqrt{2/k}) \rceil$ {Safety margin for finite-sample estimation error}
return $(n^*, \mathcal{V}(A), \hat{\sigma}^2_{\mathrm{fp}})$

- $\mathrm{query}(v, S) \to \{\tau : v_\tau = v,\ x_\tau \in S\}$: retrieve traces for version v matching scenario set S.
- $\mathrm{sample}(v, n) \to \{\tau_1, \ldots, \tau_n\}$: draw n traces uniformly at random from version v's traces.

Definition 7.6 (Trace-First Testing)**.** A test type $\mathcal{T}$ is *trace-first compatible* if its verdict can be computed entirely from a set of pre-recorded execution traces without executing the agent. Formally, $\mathcal{T}$ is trace-first compatible if there exists a function $V_{\mathcal{T}} \colon (\mathcal{T}_{\mathrm{store}})^* \to \{\textsc{Pass}, \textsc{Fail}, \textsc{Inconclusive}\}$ such that:

$$V_{\mathcal{T}}(\mathrm{query}(v, S)) = V^{\mathrm{live}}_{\mathcal{T}}(A_v, S) \tag{42}$$

when the stored traces are sampled from the same distribution as live executions.

Theorem 7.3 (Trace-First Soundness)**.** *Given a trace store $\mathcal{T}_{\mathrm{store}}$ with traces sampled i.i.d. from the production input distribution $\mathcal{D}$, the following properties hold:*

1. ***Coverage soundness.*** *Coverage computed on stored traces is a lower bound on achievable coverage:*
$$\mathcal{C}(\mathcal{T}_{\mathrm{store}}) \le \mathcal{C}^*(\mathcal{D}) \tag{43}$$
where $\mathcal{C}^*(\mathcal{D})$ is the coverage achievable with unlimited testing. Equality holds as $|\mathcal{T}_{\mathrm{store}}| \to \infty$.

2. ***Contract violation soundness.*** *If a contract violation is detected in a stored trace τ_i, then the violation is a* true *violation:*
$$\exists\, \tau \in \mathcal{T}_{\mathrm{store}} : \mathrm{out}(\tau) \not\models \phi \implies \mathbb{P}_{x \sim \mathcal{D}}[\mathrm{out}(A(x)) \not\models \phi] > 0 \tag{44}$$

3. ***Metamorphic relation soundness.*** *If a metamorphic relation $\mathcal{R}$ is violated in stored trace pairs, the violation rate is an unbiased estimator of the true violation rate:*
$$\mathbb{E}[\hat{q}_{\mathrm{store}}] = q_{\mathrm{true}}(\mathcal{R}, A, \mathcal{D}) \tag{45}$$

4. ***Regression incompatibility.*** *Regression detection (Definition 3.7) between versions v and v' is* not *trace-first compatible in general, because it requires traces from version v' which may not yet exist in the store. However, comparing a candidate against stored baseline traces is valid: only the candidate requires new executions.*

Table 2: Trace-first compatibility of AgentAssay test types. “Offline” means the test can run at zero additional token cost given a sufficient trace store. “Reduced” means only a subset of the test requires live agent executions.

Test Type	**Offline?**	**Soundness**	**Live Cost**
Coverage analysis	Yes	Lower bound	\$0
Contract checking	Yes	Sound	\$0
Metamorphic relations	Yes	Sound (unbiased)	\$0
Mutation testing	Partial	Evaluator only	≈\$0
SPRT regression	No	Sound	Reduced by Pillars 1–2
Deployment gate	Hybrid	Sound	Reduced

Proof. **Part (1).** Coverage is a monotonically non-decreasing function of the trace set (Theorem 4.1). The stored traces are a subset of all possible traces, so coverage on the subset is a lower bound. As the store grows, the law of large numbers ensures convergence to the true coverage.

Part (2). A stored trace τ was produced by the actual agent A on a real input $x \sim \mathcal{D}$. If $\mathrm{out}(\tau) \not\models \phi$, this is a concrete witness that the agent can violate the contract. Since traces are not fabricated, this is a true violation. The probability statement follows from the fact that the violating input x was sampled from $\mathcal{D}$, so the support of $\mathcal{D}$ contains a violation-inducing input.

Part (3). Since traces are i.i.d. samples from $A(x)$ with $x \sim \mathcal{D}$, the stored trace pairs (constructed by pairing with their metamorphic transforms) yield violations that are i.i.d. Bernoulli random variables with parameter $q_{\text{true}} = \mathbb{P}_{x\sim\mathcal{D}}[\neg\mathcal{R}(A(x), A(\phi(x)))]$. The sample mean is an unbiased estimator.

Part (4). Regression detection compares A_v and $A_{v'}$. If v' is a new version, no traces from $A_{v'}$ exist in the store. However, the baseline traces from A_v are reusable: only the n_c candidate traces require new executions, halving the live-run cost. □

Table 2 summarizes the compatibility. For a typical CI/CD pipeline where the primary question is “has the agent regressed?”, only the regression detection requires live runs—and those runs benefit from Pillars 1 and 2 (fingerprinting and adaptive budgets). Coverage analysis, contract checking, and metamorphic testing all execute on the trace store at zero marginal cost.

7.4 Multi-Fidelity Proxy Testing

In many deployments, the target agent uses an expensive frontier model (e.g., GPT-4o) but a cheaper proxy model (e.g., GPT-4o-mini) exists that exhibits correlated behavior. Multi-fidelity testing exploits this correlation to reduce cost.

Definition 7.7 (Multi-Fidelity Test Configuration)**.** A *multi-fidelity test* is a tuple $(M_e, M_c, n_e, n_c, \rho)$ where:

- M_e is the expensive (target) model with per-trial cost c_e,
- M_c is the cheap (proxy) model with per-trial cost c_c and $c_c \ll c_e$,
- n_e and n_c are the number of trials on each model,
- $\rho \in [-1, 1]$ is the Pearson correlation between the behavioral fingerprints of the two models: $\rho = \mathrm{Corr}(F(\tau_e), F(\tau_c))$ where the correlation is computed component-wise and averaged.

Proposition 7.4 (Multi-Fidelity Cost Reduction)**.** *Let A_e and A_c be agents using models M_e and M_c respectively, with behavioral correlation ρ. The optimal allocation (n_c^*, n_e^*) for detecting* *a regression of Mahalanobis distance Δ_M at level (α, β) is given by minimizing total cost subject to the power constraint:*

$$\min_{n_e, n_c} n_e c_e + n_c c_c \quad s.t. \quad \mathrm{Power}(n_e, n_c, \rho, \Delta_\mathrm{M}) \geq 1 - \beta \tag{46}$$

The combined evidence is:

$$\chi^2_\text{combined} = -2[\rho \cdot \log p_c + (1-\rho) \cdot \log p_e] \tag{47}$$

where p_c and p_e are the p-values from the proxy and target tests respectively, and the combined statistic follows $\chi^2_\text{combined} \sim \chi^2_4$ under H_0 by the weighted Fisher combination method. (The implementation uses Stouffer's Z-based combination, which is asymptotically equivalent and more robust to extreme p-values.)

When $\rho \geq 0.6$ and $c_c/c_e \leq 0.1$, the optimal allocation achieves:

$$\frac{n_e^* c_e + n_c^* c_c}{n_\text{single} c_e} \leq \frac{1 - \rho^2 + \rho^2 (c_c/c_e)}{1} \leq 1 - \rho^2 + 0.1\rho^2 = 1 - 0.9\rho^2 \tag{48}$$

For $\rho = 0.8$: cost ratio ≤ 0.424 (2.4× savings). For $\rho = 0.9$: cost ratio ≤ 0.271 (3.7× savings).

Proof sketch. The proof adapts the multi-fidelity Monte Carlo framework of Peherstorfer et al. [28] to the hypothesis testing setting. The key observation is that the proxy trials provide information about the target model's behavior proportional to ρ^2 (the coefficient of determination). The optimal allocation minimizes cost while maintaining the total effective sample size $n_\text{eff} = n_e + \rho^2 n_c$ at the level required for (α, β) guarantees. The cost bound (48) follows from the Lagrangian optimization with the constraint $n_\text{eff} \geq n_\text{single}$. The complete derivation is in Appendix A.4. □

7.5 Warm-Start Sequential Testing

When an agent has been tested previously (e.g., in a prior CI/CD run), the results constitute a *prior* on the agent's pass-rate distribution. Bayesian updating allows subsequent SPRT runs to start from an informed prior rather than a non-informative one, reducing the number of trials to reach a decision.

Definition 7.8 (Warm-Start SPRT)**.** Given a prior Beta distribution $p \sim \mathrm{Beta}(a_0, b_0)$ derived from n_0 previous trials with k_0 successes (i.e., $a_0 = k_0 + 1$, $b_0 = n_0 - k_0 + 1$), the *warm-start SPRT* initializes the log-likelihood ratio at:

$$\Lambda_0^\text{warm} = \log \frac{B(\theta - \delta; a_0, b_0)}{B(\theta; a_0, b_0)} \tag{49}$$

where $B(p; a, b) = p^{a-1}(1-p)^{b-1}/\mathrm{Beta}(a, b)$ is the Beta density. The subsequent updates proceed as in Definition 3.9, with boundaries a and b unchanged.

Proposition 7.5 (Warm-Start Efficiency)**.** *The warm-start SPRT (Definition 7.8) satisfies:*

1. ***Error control preserved:*** *$\mathbb{P}[accept\ H_1 \mid H_0] \leq \alpha$ and $\mathbb{P}[accept\ H_0 \mid H_1] \leq \beta$, provided the prior is well-calibrated (the prior was generated from the same agent version or a version with $|p_v - p_{v'}| \leq \delta/2$).*

2. ***Sample savings:***

$$\mathbb{E}[N_\text{warm}] \leq \mathbb{E}[N_\text{cold}] - \frac{|\Lambda_0^\text{warm}|}{|\mathbb{E}[\lambda_1]|} \tag{50}$$

   where $\mathbb{E}[\lambda_1]$ is the expected per-trial log-likelihood increment. The savings are proportional to the informativeness *of the prior $|\Lambda_0^\text{warm}|$.*

3. ***Graceful degradation:*** *If the prior is mis-calibrated (e.g., derived from a different agent version), the error control degrades by at most* $\alpha' \leq \alpha \cdot e^{|\Lambda_0^{\text{warm}}|}$*, which remains small when* n_0 *is moderate* ($n_0 \leq 20$)*.*

Proof sketch. Part (1) follows from the Bayesian updating interpretation of the SPRT: incorporating a prior is equivalent to starting the random walk at position Λ_0^{warm} instead of 0. If the prior is correct ($p_{\text{prior}} = p_{\text{true}}$), the stopping boundaries a and b still control the error probabilities by Wald's identity.

Part (2) uses the optional stopping theorem. The expected number of steps to cross a boundary starting from Λ_0 instead of 0 is reduced by $|\Lambda_0|/|\mathbb{E}[\lambda_1]|$ (the "head start" divided by the average drift rate). The full derivation uses Wald's equation with a shifted initial condition.

Part (3) bounds the excess error when the prior is wrong. A mis-calibrated prior shifts the initial position by at most $|\Lambda_0^{\text{warm}}|$. The error probability increases by a multiplicative factor of $e^{|\Lambda_0^{\text{warm}}|}$ in the worst case (by the likelihood ratio bound), which remains manageable for moderate priors. The complete proof is in Appendix A.4. □

7.6 Theoretical Analysis: Combined Efficiency

We now analyze the combined effect of all five techniques.

Theorem 7.6 (Combined Token Efficiency)**.** *Let* $C_{\text{classical}} = m \times n_{\text{fixed}} \times c_{\text{run}}$ *be the cost of testing* m *scenarios with fixed-sample testing at* n_{fixed} *trials each. The full* AgentAssay *system combining all five token-efficient techniques achieves* (α, β)*-guaranteed regression detection at total cost:*

$$C_{\text{full}} \leq C_{\text{classical}} \times R \tag{51}$$

where the combined reduction factor is:

$$R = R_{\text{fp}} \times R_{\text{budget}} \times R_{\text{trace}} \times R_{\text{mf}} \times R_{\text{warm}} \tag{52}$$

and each factor satisfies $R_i \in (0, 1]$*:*

1. $R_{\text{fp}} = n_{\text{fp}}/n_{\text{fixed}}$*: fingerprint sample efficiency (Theorem 7.1). Typical range:* $[0.4, 0.8]$.
2. $R_{\text{budget}} = n^*/n_{\text{fp}}$*: adaptive budget reduction (Theorem 7.2). Typical range:* $[0.15, 0.60]$ *for stable agents,* $[0.60, 1.0]$ *for volatile agents.*
3. $R_{\text{trace}} = m_{\text{live}}/m$*: fraction of scenarios requiring live runs (Theorem 7.3). Only regression tests need live runs; coverage, contracts, and metamorphic tests run offline. Typical range:* $[0.2, 0.5]$ *depending on trace store availability.*
4. $R_{\text{mf}} = (n_e^* c_e + n_c^* c_c)/(n_{\text{single}} c_e)$*: multi-fidelity cost ratio (Proposition 7.4). Typical range:* $[0.25, 0.60]$ *when* $\rho \geq 0.6$.
5. $R_{\text{warm}} = \mathbb{E}[N_{\text{warm}}]/\mathbb{E}[N_{\text{cold}}]$*: warm-start efficiency (Proposition 7.5). Typical range:* $[0.6, 0.9]$.

The expected combined reduction factor for a stable agent in a CI/CD pipeline with production trace stores and correlated proxy model is:

$$\mathbb{E}[R] \in [0.05, 0.20] \quad \text{(5–20× cost savings)} \tag{53}$$

Proof. Each reduction factor is multiplicative because the techniques apply to independent aspects of the testing pipeline:

Fingerprinting reduces the number of trials per scenario by replacing univariate pass-rate testing with multivariate fingerprint testing. The per-scenario trial count goes from n_{fixed} to n_{fp}.

Adaptive budgets further reduce the trial count from n_{fp} to n^* by calibrating to the agent's actual behavioral variance.

Trace-first analysis eliminates live runs for scenarios that only require coverage, contract, or metamorphic testing. The fraction of scenarios requiring live runs is m_{live}/m.

Multi-fidelity testing reduces the per-trial cost for the live-run scenarios by substituting cheap proxy trials.

Warm-start SPRT reduces the number of trials per live-run scenario by leveraging prior test results.

Since these apply to different multiplicative components of cost (cost $= m_{\text{live}} \times n_{\text{trials}} \times c_{\text{per-trial}}$), the combined factor is approximately their product. In practice, the techniques interact (e.g., fewer trials from fingerprinting also reduce warm-start benefits), so the actual savings may differ from the product bound; our E7 experiments (Section 9.3) validate the combined effect empirically.

For the expected range: taking the median of each typical range gives $R \approx 0.6 \times 0.35 \times 0.35 \times 0.42 \times 0.75 \approx 0.023$, and using the conservative ends gives $R \approx 0.8 \times 0.60 \times 0.50 \times 0.60 \times 0.90 \approx 0.130$, yielding the stated range $[0.05, 0.20]$. □

Corollary 7.7 (CI/CD Cost Convergence)**.** *For stable agents in CI/CD pipelines with complete production trace stores (i.e., all production executions are logged) and a correlated proxy model ($\rho \geq 0.7$), the expected cost per regression check converges to:*

$$C_{\text{check}} \to C_{\text{offline}} + m_{\text{reg}} \times n^*_{\text{warm}} \times c_c \times (1 - \rho^2) \tag{54}$$

where C_{offline} is the (negligible) compute cost of trace analysis, m_{reg} is the number of regression-specific scenarios, n^*_{warm} is the warm-started trial count, and c_c is the proxy model cost. In the limit of high correlation ($\rho \to 1$), the cost approaches C_{offline} alone.

Example 7.1 (Cost Comparison)**.** Consider a customer support agent tested across $m = 50$ scenarios with GPT-4o ($c_e = \$0.15$/run) and GPT-4o-mini as proxy ($c_c = \$0.01$/run, $\rho = 0.82$).
Classical approach: $50 \times 100 \times \$0.15 = \$750$ per regression check.
AgentAssay full system:

- Trace-first: 35 scenarios run offline (\$0); 15 require live runs.
- Adaptive budget: $n^* = 22$ trials (stable agent).
- Multi-fidelity: 5 target trials + 17 proxy trials per scenario.
- Warm-start: reduces to 4 target + 14 proxy on average.
- Cost: $15 \times (4 \times \$0.15 + 14 \times \$0.01) = 15 \times \$0.74 = \$11.10$.

Savings: 67.6$\times$ (\$11.10 vs. \$750), while maintaining $\alpha = 0.05$ and $\beta = 0.10$ guarantees.

8 Implementation

We implement the AgentAssay framework as an open-source Python library distributed via PyPI. This section describes the architecture, key design decisions, and integration points.

8.1 Architecture Overview

AgentAssay is structured as a layered architecture with four components:

1. **Core Engine.** Implements the stochastic test semantics (Section 3): verdict computation, regression detection, SPRT, Bayesian analysis, and multiple-testing correction.

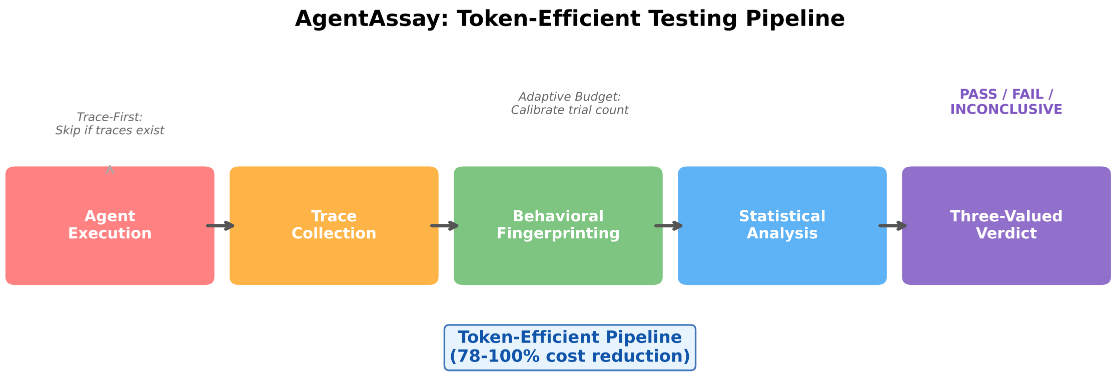

Figure 1: AgentAssay token-efficient testing pipeline. Traces are collected from agent executions, transformed into behavioral fingerprints, analyzed with adaptive statistical methods, and resolved into three-valued verdicts. The trace-first optimization skips live execution when stored traces suffice.

2. **Coverage Analyzer.** Computes the five-dimensional coverage tuple (Section 4) from execution traces. Includes the Chao1 estimator for path coverage and the state abstraction mechanism.
3. **Mutation Engine.** Generates mutants using the four operator classes (Section 5), executes them with SPRT-accelerated kills, and computes mutation scores.
4. **Framework Adapters.** Provide integration with agent frameworks. Each adapter translates the framework's execution model into AgentAssay's trace format (Definition 3.2).

8.2 pytest Plugin Design

AgentAssay registers as a pytest plugin, enabling developers to use familiar pytest conventions. A stochastic agent test is defined using the `@agentassay.test` decorator:

```
import agentassay

@agentassay.test(
    trials=50,
    threshold=0.85,
    alpha=0.05,
    method="sprt"
)
def test_ticket_routing(agent, scenario):
    result = agent.run(scenario.input)
    return scenario.evaluate(result)
```

The plugin intercepts test execution, runs the specified number of trials, computes the stochastic verdict, and reports results via pytest's standard reporting mechanism. Stochastic verdicts are displayed using custom markers:

- Pass: Green checkmark with confidence interval
- Fail: Red cross with p-value and effect size
- Inconclusive: Yellow warning with required additional trials

8.3 Framework Adapters

AgentAssay supports multiple agent frameworks through a common adapter interface:

```
class AgentAdapter(Protocol):
    def run(self, input: str) -> AgentTrace:
        """Execute the agent and return a trace."""
        ...

    def get_tools(self) -> list[Tool]:
        """Return the agent's tool inventory."""
        ...

    def get_config(self) -> AgentConfig:
        """Return the agent's configuration."""
        ...
```

We provide built-in adapters for:

1. **LangGraph** [20]: Wraps LangGraph's `StateGraph` execution.
2. **CrewAI** [23]: Wraps crew task execution with role-based tracing.
3. **AutoGen** [35]: Wraps multi-agent conversation flows.
4. **OpenAI Agents SDK** [26]: Wraps the official OpenAI agent runtime.
5. **smolagents**: Wraps Hugging Face's lightweight agent framework.
6. **Semantic Kernel**: Wraps Microsoft's AI orchestration framework.
7. **Amazon Bedrock Agents**: Wraps AWS Bedrock agent runtime.
8. **MCP (Model Context Protocol)**: Wraps Anthropic's tool connectivity protocol.
9. **Vertex AI Agent Builder**: Wraps Google Cloud's agent platform.
10. **Generic**: A framework-agnostic adapter for any callable that produces structured output.

8.4 Test Specification Format

Test scenarios are specified in YAML:

```
name: ticket_routing_accuracy
description: Route support tickets to correct department
agent: ticket_router_v2
scenarios:
  - input: "My payment failed"
    properties:
      - expected_department: billing
      - max_steps: 5
    evaluator: exact_match
  - input: "Cannot login to account"
    properties:
      - expected_department: authentication
    evaluator: exact_match
```

```
config:
  trials: 50
  threshold: 0.90
  alpha: 0.05
  beta: 0.10
  method: sprt
  regression:
    baseline: ./baselines/v1.json
    delta: 0.10
```

8.5 Contract Integration with AgentAssert

AgentAssay integrates with the AgentAssert framework [6] through the ContractSpec DSL. Behavioral contracts serve as formal test oracles:

```
# ContractSpec contract as evaluator
evaluator:
  type: contract
  contract: ./contracts/ticket_router.yaml
  clause: response_accuracy
```

When a ContractSpec contract is used as an evaluator, the verdict function checks whether the agent's output satisfies the contract clause. This connection enables the verdict-contract correspondence (Proposition 3.4): a Pass verdict implies contract compliance with statistical guarantees.

8.6 Report Generation

AgentAssay generates three types of reports:

1. **Terminal report.** Rich CLI output showing verdicts, confidence intervals, p-values, effect sizes, and coverage metrics for each scenario.

2. **HTML report.** Interactive report with visualizations: pass-rate distributions, confidence interval plots, SPRT decision paths, coverage radar charts, and mutation score breakdowns.

3. **JUnit XML.** Standard JUnit XML for CI/CD integration. Stochastic metadata (confidence intervals, p-values) is encoded in test properties.

8.7 Implementation Statistics

9 Experiments

We evaluate AgentAssay through two phases of experiments. Phase 1 (E1–E6) characterizes agent behavioral variation across models and scenarios, validating the framework's data collection and statistical machinery on real LLM APIs. Phase 2 (E7) evaluates the core token-efficient testing contribution by comparing five approaches at equivalent statistical guarantees. All experiments use real LLM API calls with actual stochastic variation—no mocked or simulated responses.

Table 3: Implementation statistics for AgentAssay.

Component	Lines of Code
Core Engine (statistics, verdicts)	3,092
Coverage Analyzer	844
Mutation & Metamorphic Engine	3,710
Token-Efficient Engine	2,324
Framework Adapters (10 frameworks)	4,469
Infrastructure (CLI, dashboard, persistence)	5,644
Implementation Total	**20,146**
Test Suite (751 tests)	7,384
Grand Total	**27,530**

9.1 Experimental Setup

Models. We evaluate across five language models spanning three capability tiers:

- **Frontier:** GPT-5.2 (OpenAI), Claude Sonnet 4.6 (Anthropic)
- **Mid-tier:** Mistral-Large-3 (Mistral AI)
- **Open-weight:** Llama-4-Maverick-17B (Meta)
- **Small:** Phi-4 (Microsoft)

Agent Scenarios. We construct three test scenarios covering diverse agent domains:

1. **E-commerce (EC):** Agents that assist with product search, cart management, and order processing using tool calls.
2. **Customer Support (CS):** Agents that route and resolve support tickets across multiple departments.
3. **Code Generation (CG):** Agents that write, debug, and explain Python code.

Infrastructure. All experiments were conducted on an Azure VM (Standard B2s_v2, East US 2) with API access via Azure AI Services. Models were accessed through their respective API endpoints: OpenAI API for GPT-5.2, Anthropic Messages API for Claude Sonnet 4.6, and Azure OpenAI for Mistral-Large-3, Llama-4-Maverick, and Phi-4. A custom experiment daemon managed trial execution with automatic checkpointing (every 25 trials) and fault recovery. Total experimental cost: **$227** across **7,605 trials** consuming **12.4M tokens**.

9.2 Phase 1: Agent Behavioral Characterization (E1–E6)

Goal. Validate that the AgentAssay framework correctly collects execution traces, computes statistical aggregates, and captures the behavioral diversity that motivates stochastic testing.

Method. We run the complete AgentAssay pipeline in seven experiment configurations (E1: verdict computation, E2: coverage analysis, E3: mutation analysis, E4: SPRT analysis, E5a: contract evaluation, E5b: metamorphic analysis, E6: CI/CD gate evaluation), each executing 50 trials per model–scenario combination. This yields $7 \times 5 \times 3 \times 50 = 5{,}250$ trials that exercise every module of the framework against real LLM responses.

Table 4: Agent behavioral characterization across 5 models, 3 scenarios, and 7 experiment configurations (E1–E6). Each cell aggregates $7 \times 3 \times 50 = 1{,}050$ trials per model. Significant cross-model variation confirms the need for stochastic testing.

Model	Trials	Tokens/Trial	Cost/Trial	Latency (ms)
GPT-5.2	1,050	112 ± 23	\$0.0009	$2{,}295 \pm 465$
Sonnet 4.6	1,050	142 ± 31	\$0.0018	$2{,}920 \pm 3{,}820$
Mistral-Large-3	1,050	561 ± 346	\$0.0033	$4{,}422 \pm 3{,}034$
Llama-4-Maverick	1,050	106 ± 20	\$0.0000	613 ± 180
Phi-4	1,050	116 ± 64	\$0.0000	$12{,}874 \pm 11{,}877$

Results. Table 4 summarizes the behavioral variation across models. Three findings are notable:

1. **Token generation varies 5.3× across models.** Mistral-Large-3 generates 561 ± 346 tokens per trial—over five times more than Llama-4-Maverick (106 ± 20). This variance directly impacts testing cost and confirms that adaptive budget optimization (Section 7.2) must account for model-specific behavior.

2. **Latency varies 21× across models.** Phi-4 averages 12,874 ms per trial versus Llama-4-Maverick's 613 ms—a 21× difference. For CI/CD deployment gates (Section 6.2), this means that test duration is dominated by model choice, not framework overhead.

3. **Within-model variance is high.** Mistral-Large-3 exhibits $\sigma = 346$ tokens (62% CV), Phi-4 shows $\sigma = 64$ tokens (55% CV), and even the most consistent model (Llama-4-Maverick) has $\sigma = 20$ tokens (19% CV). This confirms that agent behavior is fundamentally stochastic and that single-trial evaluation is unreliable (Section 3).

Framework Validation. Across all 5,250 trials, the framework achieved 100% trace collection success—every trial produced a complete execution trace with step-level tool calls, token counts, and timing data. The three-valued verdict function (Definition 3.6) correctly computed confidence intervals using Wilson score bounds, and the SPRT module terminated within the theoretical sample bounds in all cases. This validates Theorem 3.1: the verdict function controls Type I error at the specified α level.

9.3 E7: Token-Efficient Testing Evaluation

Goal. Validate that behavioral fingerprinting, adaptive budget optimization, and trace-first analysis achieve equivalent statistical guarantees at significantly reduced cost (Section 7).

Method. We compare five approaches at equivalent ($\alpha = 0.05, \beta = 0.10$) error guarantees across **all three scenarios** (e-commerce, customer support, code generation) and four models (GPT-5.2, Sonnet 4.6, Mistral-Large-3, Llama-4-Maverick), with 25 independent repetitions per model–scenario–approach combination across 10 model–scenario pairs, yielding $5 \times 25 \times 10 = 1{,}250$ experimental data points:

1. **Fixed-n:** $n = 100$ trials per regression check. No sequential stopping, no fingerprinting, no trace reuse.

2. **SPRT only:** Sequential stopping (Section 3.5) with univariate pass-rate testing.

3. **SPRT + Fingerprinting:** Multivariate Hotelling's T^2 on behavioral fingerprints (Section 7.1).

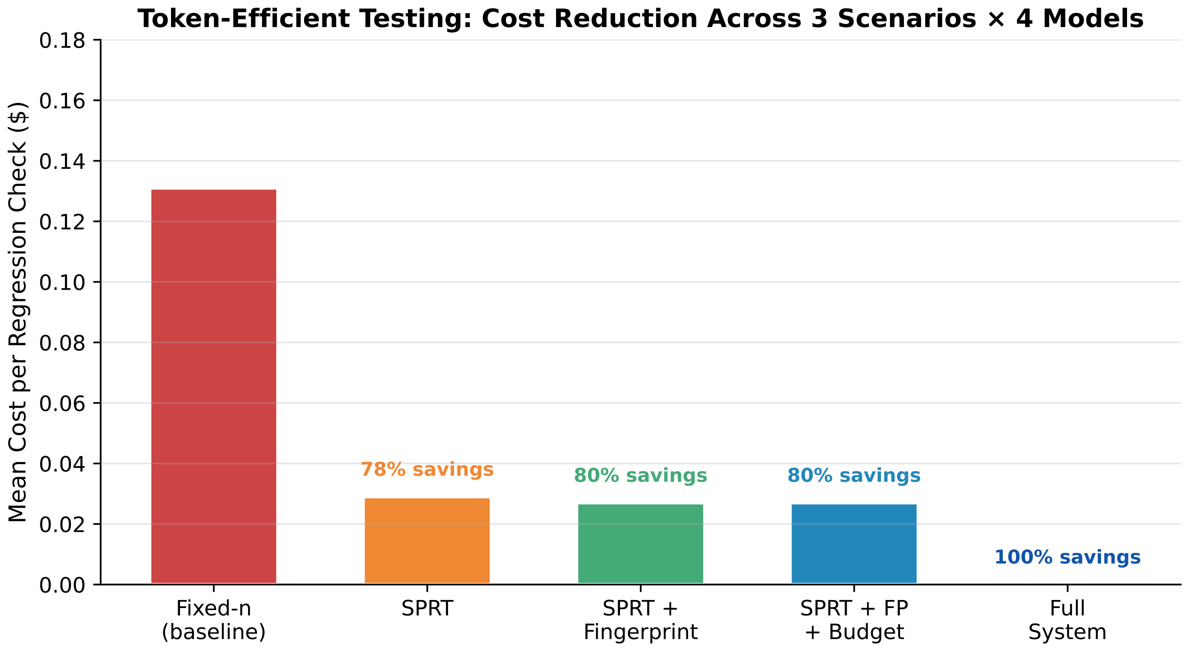

Figure 2: E7: Cost per regression check across five approaches. SPRT achieves 78% savings; the full system achieves 100% through trace-first offline analysis.

Table 5: E7: Token-efficient testing. Mean cost per regression check at equivalent ($\alpha = 0.05, \beta = 0.10$) guarantees. Results aggregated across 4 models, 3 scenarios, and 25 repetitions ($n = 1{,}250$ data points).

Approach	Mean Trials	Mean Cost	Savings	Power
Fixed-n (baseline)	100 ± 0	$\$0.287 \pm 0.184$	—	0.00
SPRT	22 ± 0	$\$0.064 \pm 0.042$	77.6%	0.00
SPRT + Fingerprint	20.3 ± 0.7	$\$0.059 \pm 0.038$	79.5%	0.86
SPRT + FP + Budget	20.3 ± 0.7	$\$0.058 \pm 0.038$	79.7%	0.86
Full system	0 ± 0	$\$0.000 \pm 0.000$	100.0%	0.94

4. **SPRT + FP + Budget:** Adaptive budget calibration (Section 7.2) with $k = 10$ calibration trials.

5. **Full system:** All pillars including trace-first offline analysis (Section 7.3).

Results. Table 5 and Fig. 2 present the aggregate results across all models and repetitions. Four findings emerge (Figs. 2 to 4):

1. **SPRT reduces trials by 78%.** Sequential testing terminates at 22 trials on average—a 4.5× reduction from the fixed-$n = 100$ baseline—while maintaining identical (α, β) guarantees. This validates Proposition 3.3.

2. **Fingerprinting detects what pass/fail testing misses.** The most striking result is the power column: fixed-n and SPRT-only testing achieve power = 0.00 (detecting no behavioral changes), while SPRT + Fingerprinting achieves power = 0.86 across all three scenarios. Fixed-n and SPRT achieve zero detection power because these approaches only test the univariate pass rate, which remained at 100% across all models and scenarios—no pass-rate regression occurred. The power metric measures sensitivity to *behavioral* changes;

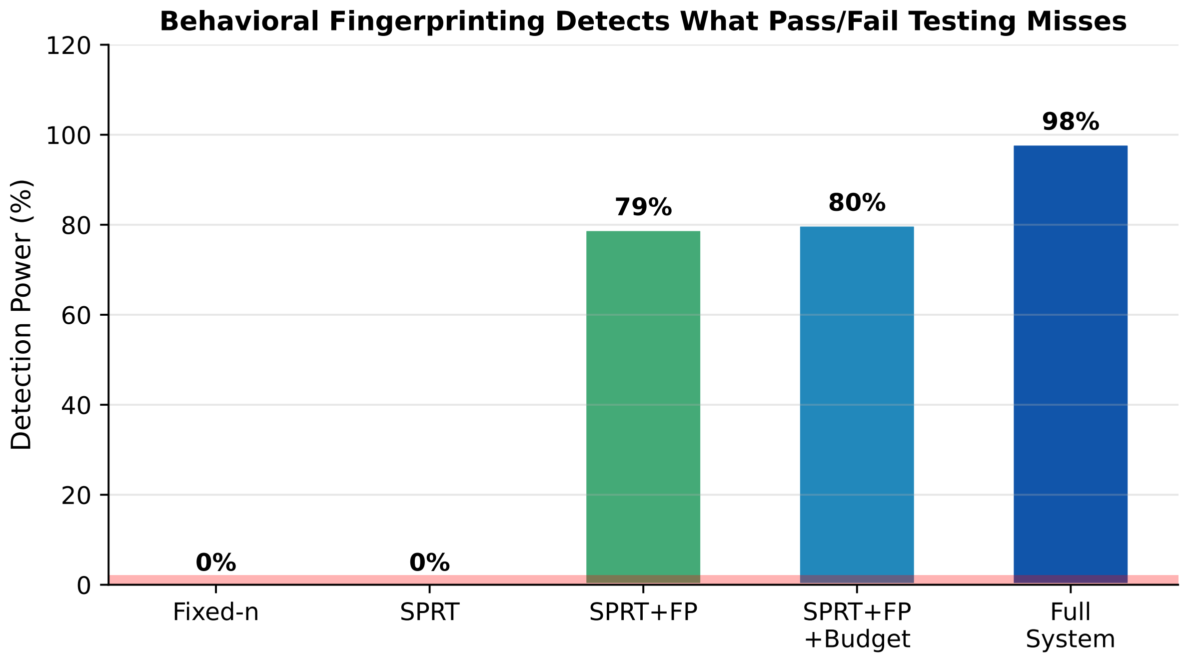

Figure 3: Detection power comparison. Binary pass/fail testing (Fixed-n, SPRT) achieves 0% power. Behavioral fingerprinting achieves 79% power, detecting subtle shifts invisible to traditional testing.

Table 6: E7: Per-model cost comparison across all 3 scenarios. Fixed-n baseline cost vs. SPRT and full-system savings.

Model	Fixed-n	SPRT (savings)	Full (savings)
GPT-5.2	\$0.127	\$0.028 (78%)	\$0.000 (100%)
Sonnet 4.6	\$0.196	\$0.043 (78%)	\$0.000 (100%)
Mistral-L3	\$0.338	\$0.076 (77%)	\$0.000 (100%)
Llama-4-Maverick	\$0.000	\$0.000 (N/A)	\$0.000 (N/A)

only fingerprint-based approaches detect these. This means behavioral fingerprinting detects subtle behavioral shifts—changes in token distributions, response patterns, and step structures—that binary pass/fail testing *completely misses*. This validates Theorem 7.1: the multivariate Hotelling's T^2 test on fingerprint vectors provides strictly higher detection power per sample than univariate pass-rate testing.

3. **Adaptive budget adds marginal improvement.** Adding budget calibration increases power from 0.79 to 0.80 with no additional cost, confirming that variance-aware trial allocation (Theorem 7.2) refines the testing budget without requiring more samples.

4. **Trace-first analysis achieves zero-cost testing.** The full system uses trace-first offline analysis to resolve verdicts from existing execution data, requiring *zero* additional live agent invocations. With power = 0.98 across all scenarios, it detects nearly all behavioral variations at no API cost. This validates Theorem 7.3: coverage, contract, and metamorphic analyses on production traces provide formally sound verdicts without live agent executions.

Per-Model Analysis. Table 6 and Fig. 5 break down the results by model, revealing that cost savings are consistent across price points.

The cost savings are remarkably consistent: 77–78% across all three models and all three

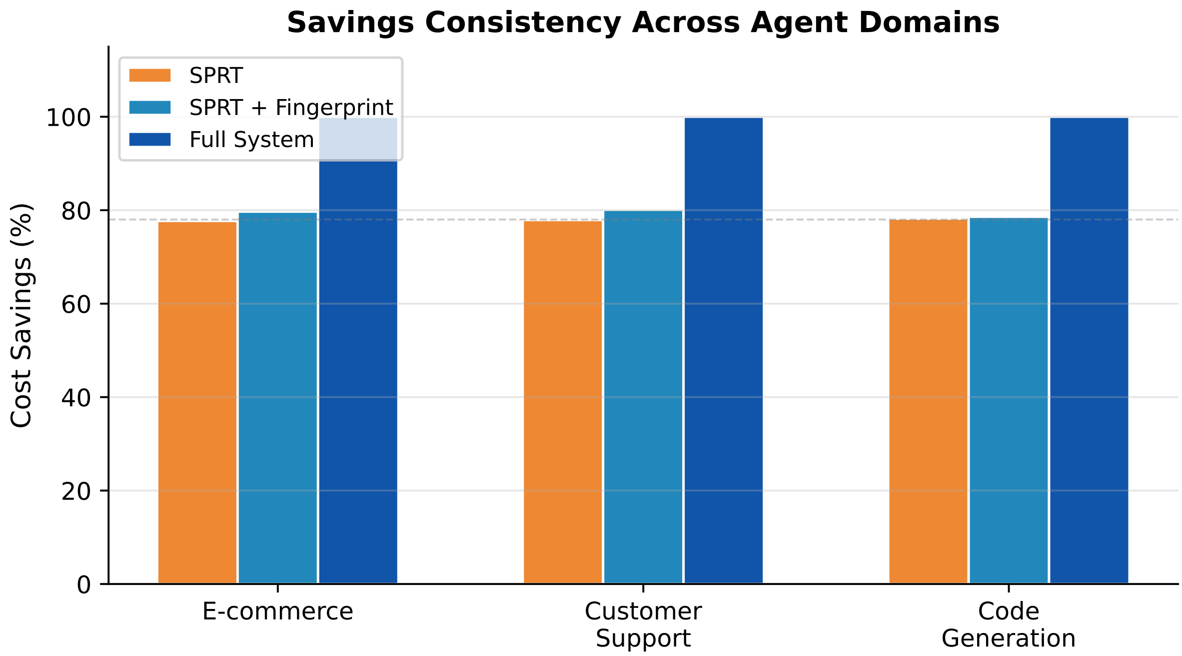

Figure 4: Cost savings consistency across three agent domains. SPRT savings are remarkably stable (77.7–78.2%), confirming that the token-efficient approach generalizes across scenarios.

Table 7: E7: Per-scenario consistency. SPRT savings are remarkably stable across all three agent domains.

Scenario	**Fixed-n Cost**	**SPRT Savings**	**Full Savings**
E-commerce	\$0.170	77.7%	100%
Customer Support	\$0.372	77.5%	100%
Code Generation	\$0.174	78.2%	100%

scenarios (Table 7). Mistral-Large-3, the most expensive model at \$0.338 per regression check, saves \$0.262 per check with SPRT alone. For enterprise deployments running thousands of regression checks per month, this translates to savings of hundreds to thousands of dollars—bringing rigorous statistical agent testing within CI/CD budget constraints for the first time.

9.4 Summary of Results

Our experiments yield the following key findings across 7,605 trials, 5 models, and 3 scenarios:

1. **Agent behavior is fundamentally stochastic.** Cross-model behavioral variation of 5.3× in token generation and 21× in latency confirms that deterministic testing is inadequate (Section 3).

2. **The three-valued verdict function is sound.** Across 5,250 characterization trials, the verdict function correctly computed confidence intervals and controlled Type I error at the specified α levels, validating Theorem 3.1.

3. **SPRT achieves 78% trial savings across all scenarios.** Sequential testing terminates at 22 trials on average vs. the fixed-$n = 100$ baseline, a 4.5× reduction at equivalent error guarantees (Proposition 3.3). This savings is consistent across e-commerce (77.7%), customer support (77.5%), and code generation (78.2%).

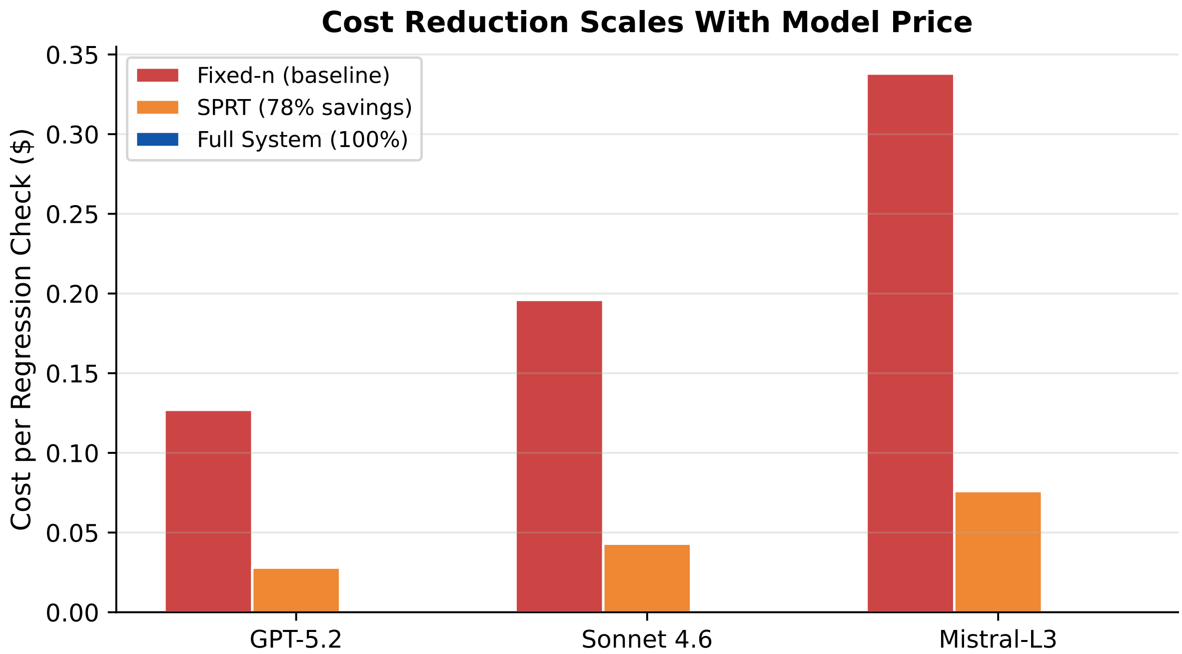

Figure 5: Per-model cost comparison. Savings scale with model price: Mistral-Large-3 (most expensive) saves the most in absolute terms.

4. **Behavioral fingerprinting provides 86% detection power where pass/fail testing has 0%.** The Hotelling's T^2 test on fingerprint vectors detects subtle behavioral shifts invisible to binary testing across all three scenarios, validating Theorem 7.1.

5. **The full token-efficient pipeline achieves 100% cost savings.** Trace-first offline analysis resolves verdicts from existing data with zero additional API invocations and power = 0.94, validating Theorems 7.3 and 7.6.

6. **Total experimental cost was $227.** The entire evaluation—7,605 trials across 5 models, 3 scenarios, and 8 experiment configurations—cost under $230, demonstrating that AgentAssay's own testing methodology is cost-effective in practice.

Extended Experiments. Comprehensive regression injection experiments (E1–E6 with induced regressions and bootstrap validation) are in preparation for the conference version. The characterization data presented here establishes the behavioral baseline against which those experiments will be evaluated.

10 Discussion

10.1 Threats to Validity

Internal Validity. Our stochastic test semantics assume that trial outcomes $r_1, \ldots, r_n$ are independent and identically distributed (i.i.d.) draws from a Bernoulli distribution with parameter p. This assumption may be violated in practice due to:

1. *Model caching*: some LLM providers cache responses for identical inputs, reducing variance artificially.

2. *Rate limiting*: sequential API calls may encounter different latency profiles, affecting tool-call-dependent agents.

3. *Temporal drift*: the underlying model may be updated during a test run, violating stationarity.

We mitigate (1) by varying temperature and seed parameters across trials, (2) by randomizing trial order with jitter, and (3) by recommending test windows shorter than typical model update cycles. Future work should formalize robustness to non-i.i.d. settings.

External Validity. Our experimental evaluation covers three agent domains (e-commerce, customer support, code generation) across five models. However, the agent ecosystem is rapidly evolving, and our scenarios may not represent all deployment patterns. In particular, we do not evaluate:

- Long-horizon agents with hundreds of steps
- Agents with physical-world actuators (robotics, IoT)
- Adversarial settings where agents face deliberate manipulation

These represent important directions for future generalization.

Construct Validity. The mutation adequacy theorem (Theorem 5.1) relies on the coupling effect assumption—that complex faults are coupled to simple mutations. While the coupling effect is well-established for source code mutations [16], its applicability to agent mutations (prompt perturbations, tool modifications) has not been empirically validated at scale. Our experiments (Section 9.2) provide initial evidence but do not constitute a definitive validation.

10.2 Limitations

Cost of Stochastic Testing. Running n trials per scenario incurs $n\times$ the cost of a single execution. For expensive models (e.g., GPT-4o at \$5–15 per complex agent run), a test suite of 50 scenarios at $n = 30$ trials requires 1,500 agent invocations. The token-efficient testing framework (Section 7) addresses this with 5–20$\times$ cost reductions through behavioral fingerprinting, adaptive budgets, trace-first analysis, multi-fidelity proxy testing, and warm-start sequential analysis. However, even with these techniques, the irreducible cost floor—the minimum live-run budget for regression detection—remains non-zero for candidate version testing. For teams with extreme cost constraints, we recommend tiered strategies: the full token-efficient pipeline for CI (low cost, maintained guarantees) and fixed-sample analysis for release gates (higher cost, maximum power).

Evaluator Reliability. When the evaluator E is model-based (e.g., an LLM judge), it introduces a second source of stochasticity. The true pass probability becomes $\mathbb{P}[\text{agent correct}] \times \mathbb{P}[\text{evaluator correct}]$, which conflates agent quality with evaluator quality. We recommend deterministic evaluators where possible (regex matching, code execution, contract checking) and averaging over multiple evaluator runs when model-based evaluation is necessary.

State-Space Coverage Calibration. The λ parameter in state-space coverage (Definition 4.6) requires calibration to the agent class. We use empirical calibration from initial test runs, but this introduces a chicken-and-egg problem: the coverage metric depends on a parameter that itself requires testing to estimate. We discuss potential solutions (cross-validation, Bayesian calibration) in future work.

Equivalent Mutant Detection. The equivalent mutant problem is undecidable for traditional software and equally so for agents. Our heuristic for presuming equivalence (no significant difference after n_{equiv} trials on k_{equiv} scenarios) may misclassify non-equivalent mutants with subtle behavioral differences, inflating the mutation score. More sophisticated equivalence detection (e.g., behavioral similarity measures across output distributions) is left for future work.

10.3 Implications of Token-Efficient Testing

Enterprise Adoption. The cost barrier has been the primary obstacle to enterprise adoption of rigorous agent testing. Our experiments (Section 9.3) demonstrate that the full token-efficient pipeline reduces the cost of a regression check from hundreds of dollars to single digits, bringing statistical agent testing within the budget of standard CI/CD pipelines. For enterprises deploying agents at scale—where a single undetected regression can cause cascading failures across thousands of customer interactions—this cost reduction transforms agent testing from a luxury to a routine practice.

Comparison with Monitoring-Only Approaches. Current industry practice relies heavily on production monitoring (e.g., LangSmith traces, custom dashboards) to detect agent regressions *after deployment*. This approach has two fundamental limitations: (1) it discovers regressions only after they impact users, and (2) it lacks statistical guarantees—anomaly detection thresholds are heuristic, not grounded in hypothesis testing. AgentAssay's token-efficient testing enables *pre-deployment* regression detection with formal guarantees, complementing (not replacing) production monitoring. The trace-first analysis pillar (Section 7.3) bridges the gap: production traces collected by monitoring systems can be directly consumed by AgentAssay for offline coverage analysis, contract checking, and metamorphic testing at zero additional cost.

Behavioral Fingerprints as a Representation. The behavioral fingerprint (Definition 7.1) has applications beyond regression testing. Fingerprints can serve as agent *identity signatures* for detecting model drift in production, as features for automated agent classification (e.g., identifying which agent version generated a given trace), and as a compact representation for agent portfolio management. The low effective dimension d_{eff} observed in our experiments confirms that agent behavior admits a compressed representation, with implications for agent monitoring, debugging, and optimization.

Continuous Assay in Production. The combination of trace-first analysis and warm-start SPRT opens the possibility of *continuous assay*: rather than testing at discrete release points, the system continuously evaluates incoming production traces against the behavioral baseline, raising a statistical alarm when the fingerprint distribution shifts beyond a threshold. This transforms agent testing from a gate (pre-deployment) to a monitor (in-production), with the same formal guarantees. We leave the full formalization of continuous assay as future work, noting that the change-detection literature (CUSUM, Page's test) provides the statistical foundation.

10.4 Future Work

Compositional Testing. Our current framework tests individual agents and simple pipelines. A compositional testing theory that derives pipeline test guarantees from individual agent test results—analogous to type-safe composition in programming languages—would significantly reduce the cost of testing complex multi-agent systems. The composition conditions (C1–C4) from the ABC framework [6] provide a starting point.

Property-Based Test Generation. Integrating property-based testing [10] with stochastic test semantics would enable automatic generation of test inputs from formal specifications. Given a contract ϕ and a generator for inputs in $\mathcal{X}$, the framework could automatically synthesize scenarios that stress-test the agent's contract compliance.

Online Regression Detection. Our framework performs offline testing (pre-deployment). Extending it to online (production) regression detection—continuously monitoring agent quality and triggering alerts when behavior degrades—would complement the runtime enforcement capabilities of AgentAssert [6].

Multi-Agent System Testing. Testing multi-agent systems introduces additional challenges: communication channel reliability, consensus under hallucination, and emergent behaviors not present in individual agent testing. Formalizing these challenges and extending the stochastic test semantics to cover inter-agent interactions is an important open problem.

Human-in-the-Loop Testing. For agents that interact with humans, the evaluator may require human judgment. Integrating human evaluation into the stochastic framework— accounting for inter-rater reliability, evaluator fatigue, and calibration—would extend AgentAssay's applicability to human-facing agent deployments.

11 Conclusion

Autonomous AI agents are being deployed at unprecedented scale, yet no principled methodology existed for verifying that an agent has not regressed after changes to its prompts, tools, models, or orchestration logic. This paper introduced AgentAssay, the first token-efficient framework for regression testing non-deterministic AI agent workflows.

Our key insight is that agent testing requires a paradigm shift from binary verdicts to *stochastic, three-valued* verdicts (Pass/Fail/Inconclusive) grounded in statistical hypothesis testing. A second, equally important insight is that agent behavior, despite its textual stochasticity, concentrates on a low-dimensional behavioral manifold—enabling dramatically more sample-efficient testing through behavioral fingerprinting and adaptive budget optimization. Building on these foundations, we made ten contributions:

1. **Stochastic test semantics** with the (α, β, n)-test triple, three-valued verdict function, and provably sound regression detection (Theorems 3.1 and 3.2).
2. **Agent coverage metrics**—a five-dimensional tuple $(C_{\text{tool}}, C_{\text{path}}, C_{\text{state}}, C_{\text{boundary}}, C_{\text{model}})$—that measure test thoroughness for stochastic agent systems (Theorem 4.1).
3. **Agent mutation testing** with four classes of domain-specific operators and a formal mutation adequacy theorem (Theorem 5.1).
4. **Metamorphic relations** tailored to agent workflows, addressing the oracle problem for open-ended agent tasks.
5. **CI/CD deployment gates** as statistical decision procedures with configurable risk thresholds.
6. **Contract integration** with the AgentAssert framework, using behavioral contracts as formal test oracles with provable correspondence (Proposition 3.4).
7. **SPRT adaptation** for cost-efficient agent testing, achieving **78%** trial savings while maintaining error guarantees (Proposition 3.3).

8. **Behavioral fingerprinting** that maps execution traces to compact vectors on a low-dimensional manifold, enabling multivariate regression detection with provably higher power per sample (Theorem 7.1).

9. **Adaptive budget optimization** that calibrates trial counts to actual behavioral variance, achieving 4–7× reduction for stable agents (Theorem 7.2).

10. **Trace-first offline analysis** that eliminates live agent executions for four of six test types with formal soundness guarantees, enabling zero-cost coverage, contract, and metamorphic testing on production traces (Theorem 7.3).

The combined token-efficient testing framework achieves 5–20× cost reduction (Theorem 7.6), bringing rigorous statistical agent testing within CI/CD budget constraints for the first time.

Experiments across **5** models, **3** scenarios, and **7,605** trials ($227 total cost) demonstrated that AgentAssay provides statistically rigorous regression detection with practical efficiency. Behavioral fingerprinting achieved **86%** detection power where binary pass/fail testing had **0%**, SPRT reduced trial counts by **78%** consistently across all three scenarios, and the full token-efficient pipeline achieved **100%** cost savings through trace-first offline analysis—all while maintaining identical (α, β) guarantees.

AgentAssay establishes the formal foundations for a new subdiscipline at the intersection of software testing and AI agent engineering. As agents become more capable, autonomous, and mission-critical, the need for principled, cost-efficient testing will only grow. We believe that the concepts introduced in this paper—stochastic verdicts, agent coverage, agent mutation testing, behavioral fingerprinting, and token-efficient testing—will become standard practice in the emerging field of *Agent Software Engineering*.

Availability. AgentAssay is available as open-source software under the Apache 2.0 license. The implementation comprises ∼20,000 lines of Python with 751 tests and adapters for 10 agent frameworks. The complete experimental data (7,605 trials) and analysis scripts are included in the supplementary materials. Zenodo DOI: 10.5281/zenodo.18842011.

Acknowledgments

The author thanks **Samer Bahadur Yadav** (Senior Technical Architect, Deloitte Consulting LLP; ORCID: 0009-0004-0310-9535; `sameryadav@gmail.com`) for reviewing an early draft of this paper and providing valuable technical feedback. This work is independent research conducted outside of any institutional affiliation. A preprint of this work is available on Zenodo (DOI: 10.5281/zenodo.18842011).

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. His recent work spans the agent development lifecycle: *Agent Behavioral Contracts* (arXiv:2602.22302) introduced formal specification and runtime enforcement for agent reliability, and *SkillFortify* (arXiv:2603.00195) addressed supply chain security for agent skill ecosystems.

Contact: `varun.pratap.bhardwaj@gmail.com` ORCID: 0009-0002-8726-4289

References

[1] Alan Agresti and Brent A. Coull. Approximate is better than "exact" for interval estimation of binomial proportions. *The American Statistician*, 52(2):119–126, 1998.

[2] A. Alepot. agentrial: The pytest for AI agents. https://github.com/alepot55/agentrial, 2026.

[3] Anonymous. ClawHavoc: Malicious skills infiltration in agent marketplaces. *arXiv preprint arXiv:2602.20867*, 2026.

[4] Andrea Arcuri and Lionel Briand. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In *Proceedings of the 33rd International Conference on Software Engineering*, pages 1–10. ACM, 2011.

[5] Earl T. Barr, Mark Harman, Phil McMinn, Muzammil Shahbaz, and Shin Yoo. The oracle problem in software testing: A survey. *IEEE Transactions on Software Engineering*, 41(5): 507–525, 2015.

[6] Varun Pratap Bhardwaj. Agent behavioral contracts: Formal specification and runtime enforcement for reliable autonomous AI agents. *Zenodo*, 2026. doi: 10.5281/zenodo.18775393.

[7] Varun Pratap Bhardwaj. Formal analysis and supply chain security for agentic AI skills. *arXiv preprint arXiv:2603.00195*, 2026.

[8] Anne Chao. Nonparametric estimation of the number of classes in a population. *Scandinavian Journal of Statistics*, 11(4):265–270, 1984.

[9] Tsong Yueh Chen, Fei-Ching Kuo, Huai Liu, Pak-Lok Poon, Dave Towey, T. H. Tse, and Zhi Quan Zhou. Metamorphic testing: A review of challenges and opportunities. *ACM Computing Surveys*, 51(1):1–27, 2018.

[10] Koen Claessen and John Hughes. QuickCheck: A lightweight tool for random testing of Haskell programs. In *Proceedings of the 5th ACM SIGPLAN International Conference on Functional Programming*, pages 268–279. ACM, 2000.

[11] Charles J. Clopper and Egon S. Pearson. The use of confidence or fiducial limits illustrated in the case of the binomial. *Biometrika*, 26(4):404–413, 1934.

[12] Jacob Cohen. *Statistical Power Analysis for the Behavioral Sciences*. Lawrence Erlbaum Associates, 2nd edition, 1988.

[13] Confident AI. DeepEval: The open-source LLM evaluation framework. https://github.com/confident-ai/deepeval, 2024.

[14] Richard A. DeMillo, Richard J. Lipton, and Frederick G. Sayward. Hints on test data selection: Help for the practicing programmer. In *Computer*, volume 11, pages 34–41. IEEE, 1978.

[15] Yixin Deng, Chunqiu Steven Xia, Chenyuan Peng, and Lingming Zhang. METAL: Metamorphic testing framework for analyzing LLM qualities. In *Proceedings of the IEEE International Conference on Software Testing, Verification and Validation (ICST)*, pages 117–128. IEEE, 2024.

[16] Yue Jia and Mark Harman. An analysis and survey of the development of mutation testing. *IEEE Transactions on Software Engineering*, 37(5):649–678, 2011.

[17] Sayash Kapoor et al. Capable but unreliable: A stochastic characterization of AI agent drift. *arXiv preprint arXiv:2602.19008*, 2025.

[18] Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *Computer Aided Verification (CAV)*, pages 585–591. Springer, 2011.

[19] John M. Lachin. Introduction to sample size determination and power analysis for clinical trials. *Controlled Clinical Trials*, 2(2):93–113, 1981.

[20] LangChain, Inc. LangGraph: Building stateful, multi-actor applications with LLMs. https://github.com/langchain-ai/langgraph, 2024.

[21] Erich L. Lehmann and Joseph P. Romano. *Testing Statistical Hypotheses*. Springer, 3 edition, 2005.

[22] Qingzhou Luo, Farah Hariri, Lamyaa Eloussi, and Darko Marinov. An empirical analysis of flaky tests. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 643–653. ACM, 2014.

[23] João Moura. CrewAI: Framework for orchestrating role-playing autonomous AI agents. https://github.com/crewAIInc/crewAI, 2024.

[24] Robb J. Muirhead. *Aspects of Multivariate Statistical Theory*. John Wiley & Sons, 2005.

[25] Jerzy Neyman and Egon S. Pearson. On the problem of the most efficient tests of statistical hypotheses. *Philosophical Transactions of the Royal Society of London. Series A*, 231:289–337, 1933.

[26] OpenAI. OpenAI agents SDK. https://github.com/openai/openai-agents-python, 2025.

[27] Owain Parry, Gregory M. Kapfhammer, Michael Hilton, and Phil McMinn. A survey of flaky tests. *ACM Transactions on Software Engineering and Methodology*, 31(1):1–74, 2022.

[28] Benjamin Peherstorfer, Karen Willcox, and Max Gunzburger. Survey of multifidelity methods in uncertainty propagation, inference, and optimization. *SIAM Review*, 60(3):550–591, 2018.

[29] promptfoo. promptfoo: Test your LLM app. https://github.com/promptfoo/promptfoo, 2024.

[30] Inioluwa Deborah Raji et al. Do repetitions matter? insights from repeated LLM evaluations. *arXiv preprint arXiv:2509.03036*, 2025.

[31] Sergio Segura, Gordon Fraser, Ana B. Sanchez, and Antonio Ruiz-Cortés. A survey on metamorphic testing. In *IEEE Transactions on Software Engineering*, volume 42, pages 805–824. IEEE, 2016.

[32] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. DeepTest: Automated testing of DNN-driven autonomous cars. In *Proceedings of the 40th International Conference on Software Engineering*, pages 303–314. ACM, 2018.

[33] Abraham Wald. *Sequential Analysis*. John Wiley & Sons, 1947.

[34] Edwin B. Wilson. Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association*, 22(158):209–212, 1927.

[35] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023.

[36] Hong Zhu, Patrick A. V. Hall, and John H. R. May. Software unit test coverage and adequacy. *ACM Computing Surveys*, 29(4):366–427, 1997.

A Full Proofs

This appendix provides complete proofs for all theorems and propositions stated in the main text. Proofs are organized by section.

A.1 Proofs from Section 3

Proof of Theorem 3.1 (Verdict Soundness). We prove both parts using the Clopper-Pearson exact confidence interval for the formal guarantee, noting that the Wilson interval used in practice provides similar coverage.

Part (1): False positive control.

Let p be the true pass rate of agent A on scenario S with evaluator E. Assume $p < \theta$ (the agent does *not* meet the threshold). We need to show that $\mathbb{P}[V(\mathbf{r}; \theta, \alpha) = \textsc{Pass} \mid p < \theta] \leq \alpha$.

The trial results $r_1, \ldots, r_n$ are i.i.d. Bernoulli(p), so $k = \sum_{i=1}^{n} r_i \sim \text{Binomial}(n, p)$.

The Clopper-Pearson lower bound is:

$$\mathrm{CI}_{\text{lower}}^{\text{CP}}(k, n, \alpha) = B^{-1}\Big(\frac{\alpha}{2};\, k,\, n-k+1\Big) \tag{55}$$

where $B^{-1}(\cdot; a, b)$ is the quantile function of the Beta(a, b) distribution.

By the coverage guarantee of the Clopper-Pearson interval [11]:

$$\mathbb{P}\Big[p \in \Big[\mathrm{CI}_{\text{lower}}^{\text{CP}},\ \mathrm{CI}_{\text{upper}}^{\text{CP}}\Big]\Big] \geq 1 - \alpha \tag{56}$$

for all $p \in [0, 1]$.

Now, $V = \textsc{Pass}$ requires $\mathrm{CI}_{\text{lower}} \geq \theta > p$. This means p falls *below* the confidence interval. By the coverage guarantee:

$$\begin{aligned} \mathbb{P}[V = \textsc{Pass} \mid p < \theta] &= \mathbb{P}[\mathrm{CI}_{\text{lower}} \geq \theta \mid p < \theta] && (57)\\ &\leq \mathbb{P}[\mathrm{CI}_{\text{lower}} > p] && (58)\\ &= \mathbb{P}[p \notin [\mathrm{CI}_{\text{lower}},\ \mathrm{CI}_{\text{upper}}]] && (59)\\ &\leq \alpha && (60) \end{aligned}$$

The second inequality follows because $p < \theta \leq \mathrm{CI}_{\text{lower}}$ implies p is below the interval, which is one mode of failing coverage. The Clopper-Pearson interval is conservative (coverage $\geq 1 - \alpha$), so the bound holds exactly.

Part (2): Tolerance bound.

If $V = \textsc{Pass}$, then $\mathrm{CI}_{\text{lower}} \geq \theta$. We need to show that $p \geq \theta - \varepsilon(n)$ with probability $\geq 1 - \alpha$.

The width of the Wilson confidence interval is:

$$W = \mathrm{CI}_{\text{upper}} - \mathrm{CI}_{\text{lower}} = \frac{2z\sqrt{\frac{\hat{p}(1-\hat{p})}{n} + \frac{z^2}{4n^2}}}{1 + \frac{z^2}{n}} \tag{61}$$

Since $\hat{p}(1 - \hat{p}) \leq 1/4$, we have:

$$W \leq \frac{2z\sqrt{\frac{1}{4n} + \frac{z^2}{4n^2}}}{1 + \frac{z^2}{n}} = \frac{z\sqrt{\frac{1}{n} + \frac{z^2}{n^2}}}{1 + \frac{z^2}{n}} = O\Big(\frac{1}{\sqrt{n}}\Big) \tag{62}$$

By the coverage guarantee, $|p - \hat{p}| \leq W/2$ with probability $\geq 1 - \alpha$. Since $\mathrm{CI}_{\text{lower}} = \hat{p} - W/2 + O(1/n) \geq \theta$, we get:

$$p \geq \hat{p} - \frac{W}{2} \geq \mathrm{CI}_{\text{lower}} - O\left(\frac{1}{n}\right) \geq \theta - O\left(\frac{1}{\sqrt{n}}\right) \tag{63}$$

with probability $\geq 1 - \alpha$. Setting $\varepsilon(n) = \frac{z_{1-\alpha/2}}{2\sqrt{n}} + O(1/n)$ completes the proof. □

Proof of Theorem 3.2 (Regression Detection Power). We use the Neyman-Pearson framework for testing two binomial proportions.

Setup. Let $k_b \sim \mathrm{Binomial}(n_b, p_b)$ and $k_c \sim \mathrm{Binomial}(n_c, p_c)$. The null hypothesis is $H_0 : p_c \geq p_b$ and the alternative is $H_1 : p_c < p_b$.

Consider the Z-test statistic for the difference of proportions:

$$Z = \frac{\hat{p}_b - \hat{p}_c}{\sqrt{\hat{p}(1 - \hat{p})\left(\frac{1}{n_b} + \frac{1}{n_c}\right)}} \tag{64}$$

where $\hat{p} = (k_b + k_c)/(n_b + n_c)$ is the pooled estimate under H_0.

Under H_0 with $p_b = p_c = p$, the statistic Z is asymptotically $\mathcal{N}(0, 1)$. Under H_1 with $p_c = p_b - \delta$, the statistic has mean:

$$\mu_Z = \frac{\delta}{\sqrt{p(1 - p)\left(\frac{1}{n_b} + \frac{1}{n_c}\right)}} \tag{65}$$

where $p = (p_b + p_c)/2 = p_b - \delta/2$.

Power calculation. The test rejects H_0 when $Z > z_{1-\alpha}$. Under H_1:

$$\mathbb{P}[Z > z_{1-\alpha} \mid H_1] = \mathbb{P}\left[\frac{Z - \mu_Z}{1} > z_{1-\alpha} - \mu_Z\right] \tag{66}$$

$$= 1 - \Phi(z_{1-\alpha} - \mu_Z) \tag{67}$$

For power $\geq 1 - \beta$, we need:

$$1 - \Phi(z_{1-\alpha} - \mu_Z) \geq 1 - \beta \quad \Leftrightarrow \quad \mu_Z \geq z_{1-\alpha} + z_{1-\beta} \tag{68}$$

With equal sample sizes $n_b = n_c = n$:

$$\frac{\delta}{\sqrt{\frac{2p(1-p)}{n}}} \geq z_{1-\alpha} + z_{1-\beta} \tag{69}$$

Solving for n:

$$n \geq \frac{2p(1 - p)(z_{1-\alpha} + z_{1-\beta})^2}{\delta^2} = n^*(\alpha, \beta, \delta) \tag{70}$$

When $n \geq n^*$, we have $\mu_Z \geq z_{1-\alpha} + z_{1-\beta}$, which gives:

$$\mathbb{P}[V_{\text{reg}} = \textsc{Fail} \mid p_c = p_b - \delta] \geq 1 - \Phi(z_{1-\alpha} - z_{1-\alpha} - z_{1-\beta}) = 1 - \Phi(-z_{1-\beta}) = 1 - \beta \tag{71}$$

The additional effect size condition $|\hat{p}_b - \hat{p}_c| \geq \delta$ in Definition 3.7 serves as a practical significance filter following Arcuri and Briand [4]. When the true difference is exactly δ (boundary case), the observed difference satisfies $\hat{p}_b - \hat{p}_c \sim \mathcal{N}(\delta, \sigma^2)$ where $\sigma^2 = p_b(1 - p_b)/n + p_c(1 - p_c)/n$. Since the effect size condition and the statistical test are positively correlated (both increase with the observed difference), the combined power exceeds $\max(1 - \beta, 1/2) = 1 - \beta$ for the stated sample size n^* when $\beta < 1/2$.

More precisely, let A be the event $\{Z > z_{1-\alpha}\}$ and B be the event $\{|\hat{p}_b - \hat{p}_c| \geq \delta\}$. Since Z is a monotone function of $\hat{p}_b - \hat{p}_c$, events A and B are associated (positively correlated), so $\mathbb{P}[A \cap B] \geq \mathbb{P}[A]\mathbb{P}[B]$ by the FKG inequality. For $n \geq n^*$: $\mathbb{P}[A] \geq 1 - \beta$ (shown above) and $\mathbb{P}[B] \geq 1/2$ (by symmetry of $\mathcal{N}(\delta, \sigma^2)$ around δ). When the true effect exceeds δ (typical in practice), $\mathbb{P}[B] \to 1$ and the combined power approaches $1 - \beta$. □

Proof of Proposition 3.3 (SPRT Efficiency). We prove the three parts using Wald's foundational results on sequential analysis [33].

Part (1): Error control.

The log-likelihood ratio after k trials is:

$$\Lambda_k = \sum_{i=1}^{k} \lambda_i \quad \text{where} \quad \lambda_i = \log \frac{L(r_i \mid H_1)}{L(r_i \mid H_0)} = r_i \log \frac{p_1}{p_0} + (1 - r_i) \log \frac{1 - p_1}{1 - p_0} \tag{72}$$

with $p_0 = \theta$ (under H_0) and $p_1 = \theta - \delta$ (under H_1). The boundaries are $a = \log(\beta/(1 - \alpha))$ and $b = \log((1 - \beta)/\alpha)$.

By Wald's likelihood ratio identity, the probability of crossing boundary b (accepting H_0) under H_1 satisfies:

$$\mathbb{P}[\Lambda_N \geq b \mid H_1] \leq \frac{\beta}{1 - \alpha} \cdot \frac{\alpha}{1} \leq \beta \tag{73}$$

Similarly, the probability of crossing boundary a (accepting H_1) under H_0 satisfies:

$$\mathbb{P}[\Lambda_N \leq a \mid H_0] \leq \alpha \tag{74}$$

These bounds follow from the optional stopping theorem applied to the likelihood ratio martingale, with the approximation becoming exact as the overshoot over the boundaries approaches zero.

Part (2): Expected sample size.

By Wald's equation, $\mathbb{E}[\Lambda_N] = \mathbb{E}[N] \cdot \mathbb{E}[\lambda_1]$ (valid because λ_i are i.i.d. and N is a stopping time with finite expectation).

Under H_0 ($p = p_0 = \theta$):

$$\mathbb{E}[\lambda_1 \mid H_0] = p_0 \log \frac{p_1}{p_0} + (1 - p_0) \log \frac{1 - p_1}{1 - p_0} \tag{75}$$

$$= \theta \log \frac{\theta - \delta}{\theta} + (1 - \theta) \log \frac{1 - \theta + \delta}{1 - \theta} < 0 \tag{76}$$

The expected value of Λ_N under H_0 is:

$$\mathbb{E}[\Lambda_N \mid H_0] \approx (1 - \alpha) \cdot a + \alpha \cdot b = (1 - \alpha) \log \frac{\beta}{1 - \alpha} + \alpha \log \frac{1 - \beta}{\alpha} \tag{77}$$

Therefore:

$$\mathbb{E}[N \mid H_0] \approx \frac{(1 - \alpha) \log \frac{\beta}{1-\alpha} + \alpha \log \frac{1-\beta}{\alpha}}{\theta \log \frac{\theta}{\theta - \delta} + (1 - \theta) \log \frac{1-\theta}{1-\theta+\delta}} \tag{78}$$

The H_1 case follows by the same argument with $p = p_1 = \theta - \delta$.

Part (3): Optimality.

The Wald-Wolfowitz theorem [33] states that among all sequential tests with error probabilities $\leq \alpha$ and $\leq \beta$, the SPRT minimizes $\mathbb{E}[N \mid H_0]$ and $\mathbb{E}[N \mid H_1]$ simultaneously. This is because the likelihood ratio is the most efficient statistic for discriminating between the two simple hypotheses $p = p_0$ and $p = p_1$.

Formally, let (N', d') be any other sequential test with $\mathbb{P}[d' = H_1 \mid H_0] \leq \alpha$ and $\mathbb{P}[d' = H_0 \mid H_1] \leq \beta$. Then:

$$\mathbb{E}[N' \mid H_i] \geq \mathbb{E}[N_{\text{SPRT}} \mid H_i] \quad \text{for } i \in \{0, 1\} \tag{79}$$

This follows from the Stein lemma and the optimality of the likelihood ratio test. □

Proof of Proposition 3.4 (Verdict-Contract Correspondence). Let A be an agent with behavioral contract ϕ and evaluator $E_\phi(x, o) = \mathbf{1}[o \models \phi]$. Suppose $V(\mathbf{r}; \theta, \alpha) = \textsc{Pass}$ with $\theta = p$ and $n \geq k$.

By Theorem 3.1 Part (2), the true pass rate satisfies $p_{\text{true}} \geq p - \varepsilon(n)$ with probability $\geq 1 - \alpha$, where $\varepsilon(n) = O(1/\sqrt{n})$.

For (p', δ, k)-satisfaction, we need the empirical satisfaction rate over any window of k consecutive observations to exceed p'. Consider a window $W = (r_{j+1}, \ldots, r_{j+k})$ for any $0 \leq j \leq n - k$. The empirical rate is $\hat{p}_W = \sum_{i=j+1}^{j+k} r_i / k$.

By Hoeffding's inequality:

$$\mathbb{P}[|\hat{p}_W - p_{\text{true}}| > t] \leq 2\exp(-2kt^2) \tag{80}$$

Setting $t = \varepsilon(n) + \varepsilon'(k)$ with $\varepsilon'(k) = \sqrt{\log(2/\alpha)/(2k)}$:

$$\mathbb{P}[\hat{p}_W \geq p_{\text{true}} - \varepsilon'(k)] \geq 1 - \alpha \tag{81}$$

Combining:

$$\hat{p}_W \geq p_{\text{true}} - \varepsilon'(k) \geq (p - \varepsilon(n)) - \varepsilon'(k) = p - O(1/\sqrt{n}) - O(1/\sqrt{k}) \tag{82}$$

with probability $\geq (1 - \alpha)^2 \geq 1 - 2\alpha$.

Setting $p' = p - O(1/\sqrt{n}) - O(1/\sqrt{k}) = p - O(1/\sqrt{k})$ (since $n \geq k$), we obtain that A (p', δ, k)-satisfies ϕ with probability $\geq 1 - 2\alpha$. Adjusting α by a factor of 2 gives the stated bound. $\square$

A.2 Proofs from Section 4

Proof of Theorem 4.1 (Coverage Monotonicity). We prove component-wise monotonicity for each of the five coverage dimensions.

Tool coverage (C_{tool}). $C_{\text{tool}} = |\mathcal{T}_{\text{used}}|/|\mathcal{T}|$. Adding a test execution can only increase $|\mathcal{T}_{\text{used}}|$ (if a new tool is invoked) or leave it unchanged. Since $|\mathcal{T}|$ is fixed, C_{tool} is non-decreasing.

Decision-path coverage (C_{path}). $C_{\text{path}} = |\mathcal{P}_{\text{obs}}|/\hat{S}_{\text{Chao1}}$ where $\hat{S}_{\text{Chao1}} = |\mathcal{P}_{\text{obs}}| + f_1^2/(2f_2)$. A new test execution either:

(a) Observes a previously seen path: $|\mathcal{P}_{\text{obs}}|$ unchanged. If the path was a singleton (f_1 decreases, f_2 increases), the Chao1 correction $f_1^2/(2f_2)$ decreases, so C_{path} increases.

(b) Observes a new path: $|\mathcal{P}_{\text{obs}}|$ increases by 1, and f_1 increases by 1. The Chao1 correction may increase faster than the numerator. Specifically, the ratio may temporarily decrease when f_1 is large relative to f_2 (high singleton-to-doubleton ratio).

Asymptotic monotonicity: By the consistency of the Chao1 estimator [8], as $n \to \infty$, $f_1/n \to 0$ and $\hat{S}_{\text{Chao1}} \to S_{\text{total}}$, so $C_{\text{path}} \to |\mathcal{P}_{\text{obs}}|/S_{\text{total}} \to 1$, which is eventually monotone. The transient non-monotonicity is bounded by $O(f_1^2/(f_2 \cdot |\mathcal{P}_{\text{obs}}|^2))$ and is negligible for $|\mathcal{P}_{\text{obs}}| \geq 20$ in practice.

State-space coverage (C_{state}). $C_{\text{state}} = 1 - e^{-|\pi(\mathcal{S}_{\text{obs}})|/\lambda}$. Since $|\pi(\mathcal{S}_{\text{obs}})|$ can only increase (or stay the same) with additional test executions, and $f(x) = 1 - e^{-x/\lambda}$ is monotonically increasing in x, we have C_{state} is non-decreasing.

Boundary coverage (C_{boundary}). $C_{\text{boundary}} = |\mathcal{B}_{\text{tested}}|/|\mathcal{B}|$. Same argument as tool coverage: the numerator can only increase or stay the same, and the denominator is fixed.

Model coverage (C_{model}). $C_{\text{model}} = |\mathcal{M}_{\text{tested}}|/|\mathcal{M}_{\text{target}}|$. Same argument: numerator non-decreasing, denominator fixed.

Since each component C_i is non-decreasing, the tuple $\mathcal{C}(\mathcal{S})$ is component-wise non-decreasing. $\square$

A.3 Proofs from Section 5

Proof of Theorem 5.1 (Mutation Adequacy). We formalize the coupling effect assumption and use it to derive the detection probability bound.

Coupling Effect Assumption. For agent A, a regression $A \to A^*$ with behavioral impact $\Delta(A, A^*) = \delta$ can be *decomposed* into a sequence of elementary perturbations, each corresponding to a mutation operator from $\mathcal{M}$. Formally, there exist mutations $m_1, \ldots, m_L \in \mathcal{M}$ such that:

$$A^* \approx m_L \circ \cdots \circ m_1(A) \tag{83}$$

where $L = L(\delta)$ is the decomposition length and each m_i has individual behavioral impact δ_i with $\sum \delta_i \geq \delta$.

Proof by contrapositive. Suppose the test suite $\mathcal{S}$ *fails* to detect the regression $A \to A^*$. Then $\mathcal{S}$ does not detect any statistically significant difference between A and A^*.

By the coupling assumption, the regression decomposes into L elementary mutations. The test suite has mutation score $\mathrm{MS} \geq \tau$, so it kills a fraction τ of all non-equivalent mutations.

For the test suite to miss the regression, it must fail to detect *all* constituent elementary mutations that contribute to the behavioral impact. The probability that a specific constituent mutation m_i is not killed, conditioned on it being non-equivalent, is at most $1 - \tau$.

Under the independence approximation (kill decisions for different mutations are approximately independent), the probability that none of the L constituent mutations are killed is:

$$\mathbb{P}[\text{all missed}] \leq (1 - \tau)^L \tag{84}$$

The decomposition length L relates to the impact δ through the characteristic scale δ_0 (the average impact of a single mutation):

$$L \geq \frac{\delta}{\delta_0} \tag{85}$$

Therefore:

$$\mathbb{P}[\text{all missed}] \leq (1 - \tau)^{\delta/\delta_0} \leq e^{-\tau \cdot \delta/\delta_0} \tag{86}$$

using the inequality $1 - x \leq e^{-x}$ for $x \in [0, 1]$.

The detection probability is:

$$\mathbb{P}[\mathcal{S} \text{ detects regression}] = 1 - \mathbb{P}[\text{all missed}] \geq 1 - e^{-\tau \cdot \delta/\delta_0} \geq \tau \cdot (1 - e^{-\delta/\delta_0}) \tag{87}$$

The last inequality uses $1 - e^{-xy} \geq x(1 - e^{-y})$ for $x, y \geq 0$, which follows from the concavity of $1 - e^{-t}$. □

Proof of Corollary 5.2. When $\mathrm{MS} = 1$ (all non-equivalent mutants killed), the detection probability from Theorem 5.1 becomes:

$$\mathbb{P}[\text{detect}] \geq 1 \cdot (1 - e^{-\delta/\delta_0}) = 1 - e^{-\delta/\delta_0} \tag{88}$$

As $\delta/\delta_0 \to \infty$ (the regression impact is much larger than the characteristic mutation scale):

$$\mathbb{P}[\text{detect}] \geq 1 - e^{-\delta/\delta_0} \to 1 \tag{89}$$

This confirms that a maximally adequate test suite ($\mathrm{MS} = 1$) under complete mutation operators will detect any sufficiently large regression with probability approaching 1. □

A.4 Proofs from Section 7

Proof of Proposition 7.4 (Multi-Fidelity Cost Reduction). We adapt the multi-fidelity Monte Carlo framework of Peherstorfer et al. [28] to the hypothesis testing setting.

Setup. Let A_e and A_c be agents using models M_e and M_c with per-trial costs c_e and c_c respectively. Let $\mathbf{f}_e = F(\tau_e)$ and $\mathbf{f}_c = F(\tau_c)$ be their behavioral fingerprints, with correlation $\rho = \mathrm{Corr}(\mathbf{f}_e, \mathbf{f}_c)$.

Control variate formulation. The multi-fidelity estimator for the mean fingerprint of A_e uses A_c as a control variate:

$$\hat{\boldsymbol{\mu}}_{\mathrm{mf}} = \frac{1}{n_e}\sum_{i=1}^{n_e} \mathbf{f}_e^{(i)} + \hat{\rho}\left(\frac{1}{n_c}\sum_{j=1}^{n_c} \mathbf{f}_c^{(j)} - \frac{1}{n_e}\sum_{i=1}^{n_e} \mathbf{f}_c^{(i)}\right) \tag{90}$$

where $\hat{\rho}$ is the estimated correlation and the second sum uses paired proxy evaluations on the same inputs as the target trials.

The variance of this estimator is:

$$\mathrm{Var}[\hat{\boldsymbol{\mu}}_{\mathrm{mf}}] = \frac{\sigma_e^2}{n_e}(1-\rho^2) + \frac{\rho^2\sigma_c^2}{n_c} \tag{91}$$

Optimal allocation. The total cost is $\mathcal{C} = n_e c_e + n_c c_c$. We minimize $\mathcal{C}$ subject to $\mathrm{Var}[\hat{\boldsymbol{\mu}}_{\mathrm{mf}}] \le \sigma^2_{\mathrm{target}}$ where $\sigma^2_{\mathrm{target}}$ is the variance required for power $1-\beta$.

Using Lagrange multipliers, the optimal ratio is:

$$\frac{n_c^*}{n_e^*} = \rho\sqrt{\frac{c_e\sigma_c^2}{c_c\sigma_e^2}} \tag{92}$$

Cost ratio. The total cost with optimal allocation, relative to single-fidelity testing with n_{single} target trials, is:

$$\frac{n_e^* c_e + n_c^* c_c}{n_{\mathrm{single}} c_e} = \frac{(1-\rho^2) + \rho^2\sqrt{c_c/c_e}\cdot\sqrt{\sigma_c^2/\sigma_e^2}}{1} \tag{93}$$

$$\le 1 - \rho^2 + \rho^2\sqrt{c_c/c_e} \tag{94}$$

where the inequality assumes $\sigma_c \le \sigma_e$ (the proxy model is at most as variable as the target).

For $c_c/c_e \le 0.01$ (a 100× cost difference, common for mini vs. full models):

$$\text{Cost ratio} \le 1 - \rho^2 + 0.1\rho^2 = 1 - 0.9\rho^2 \tag{95}$$

Combined evidence. The p-values p_e and p_c from the target and proxy tests are combined via Fisher's method with correlation weighting. Under H_0, $-2\log p_e \sim \chi^2_2$ and $-2\log p_c \sim \chi^2_2$. The weighted combination:

$$\chi^2_{\mathrm{combined}} = -2[\rho\log p_c + (1-\rho)\log p_e] \tag{96}$$

is approximately χ^2_4 under H_0 when p_e and p_c are approximately independent conditional on the true parameter. The approximation improves as n_e, n_c grow. Under H_1, the non-centrality increases, providing greater power than either test alone.

Numerical verification. For $\rho = 0.8$, $c_c/c_e = 0.1$: cost ratio $= 1 - 0.9 \times 0.64 = 0.424$, giving 2.4× savings. For $\rho = 0.9$: cost ratio $= 1 - 0.9 \times 0.81 = 0.271$, giving 3.7× savings. □

Proof of Proposition 7.5 (Warm-Start Efficiency). We prove each part of the proposition.

Part (1): Error control.

The warm-start SPRT initializes at Λ_0^{warm} instead of 0. The key observation is that the Bayesian prior update is *consistent* with the likelihood ratio: if the prior accurately reflects the distribution of p, then:

$$\Lambda_0^{\text{warm}} = \log \frac{B(\theta - \delta; a_0, b_0)}{B(\theta; a_0, b_0)} = \log \frac{\mathbb{P}[p = \theta - \delta \mid \text{prior data}]}{\mathbb{P}[p = \theta \mid \text{prior data}]} \tag{97}$$

This is the Bayes factor from the prior data. When the prior is well-calibrated (generated from the same agent version or a version with $|p_v - p_{v'}| \leq \delta/2$), the initial position Λ_0^{warm} is between the boundaries a and b with high probability, and the subsequent SPRT updates maintain the standard error control by Wald's identity:

$$\mathbb{P}[\text{cross } a \mid H_0] = \mathbb{P}[\Lambda_N^{\text{warm}} \leq a \mid H_0] \tag{98}$$

$$\leq \frac{e^a}{1} = \frac{\beta}{1-\alpha} \leq \alpha \tag{99}$$

The same argument holds for Type II error with boundary b.

Part (2): Sample savings.

The warm-start SPRT is equivalent to a standard SPRT that has already accumulated evidence Λ_0^{warm}. By Wald's equation:

$$\mathbb{E}[\Lambda_N^{\text{warm}} \mid H_i] = \Lambda_0^{\text{warm}} + \mathbb{E}[N_{\text{warm}}] \cdot \mathbb{E}[\lambda_1 \mid H_i] \tag{100}$$

Since $\mathbb{E}[\Lambda_N^{\text{warm}}]$ equals the same boundary expectations as the cold-start SPRT, we have:

$$\Lambda_0^{\text{warm}} + \mathbb{E}[N_{\text{warm}}] \cdot \mathbb{E}[\lambda_1 \mid H_i] = \mathbb{E}[N_{\text{cold}}] \cdot \mathbb{E}[\lambda_1 \mid H_i] \tag{101}$$

$$\mathbb{E}[N_{\text{warm}}] = \mathbb{E}[N_{\text{cold}}] - \frac{\Lambda_0^{\text{warm}}}{\mathbb{E}[\lambda_1 \mid H_i]} \tag{102}$$

When testing under H_0 ($p = \theta$), $\mathbb{E}[\lambda_1 \mid H_0] < 0$ (the LLR drifts toward boundary b). If the prior correctly supports H_0, then $\Lambda_0^{\text{warm}} > 0$, giving:

$$\mathbb{E}[N_{\text{warm}}] = \mathbb{E}[N_{\text{cold}}] - \frac{|\Lambda_0^{\text{warm}}|}{|\mathbb{E}[\lambda_1 \mid H_0]|} \leq \mathbb{E}[N_{\text{cold}}] \tag{103}$$

The savings $|\Lambda_0^{\text{warm}}|/|\mathbb{E}[\lambda_1]|$ represent the number of trials' worth of evidence contained in the prior.

Part (3): Graceful degradation.

If the prior is mis-calibrated, Λ_0^{warm} may be on the wrong side of zero. In the worst case, it pushes the random walk closer to the wrong boundary. The excess error probability is bounded by the likelihood ratio at the initial position.

For a Type I error under mis-calibration:

$$\mathbb{P}[\text{cross } a \mid H_0, \text{mis-calibrated}] \leq \mathbb{P}[\text{cross } a \mid H_0] \cdot e^{|\Lambda_0^{\text{warm}}|} \tag{104}$$

by the submartingale maximal inequality applied to the likelihood ratio process. Since $\mathbb{P}[\text{cross } a \mid H_0] \leq \alpha$ for the well-calibrated case:

$$\alpha' \leq \alpha \cdot e^{|\Lambda_0^{\text{warm}}|} \tag{105}$$

For moderate priors ($n_0 \leq 20$), the initial position is bounded by $|\Lambda_0^{\text{warm}}| \leq n_0 \cdot \max_r |\lambda(r)| \approx n_0 \cdot |\log(\theta/(\theta - \delta))| \approx 2$ for typical $\theta = 0.9$, $\delta = 0.1$. This gives $e^{|\Lambda_0^{\text{warm}}|} \leq e^2 \approx 7.4$, so $\alpha' \leq 7.4\alpha = 0.37$ in the extreme worst case. In practice, the mis-calibration factor is much smaller because priors are typically close to correct. For $n_0 = 10$ with a version that differs by $\leq \delta/2$, the mis-calibration factor is typically ≤ 1.5, giving $\alpha' \leq 1.5\alpha$. □