Reliability, Quality, and Economic Governance in Provider-Independent Agentic Platforms

Primary Research Question and Foundational Hypotheses

The deployment of autonomous software entities necessitates a paradigm shift in reliability engineering. The primary research question governing this analysis is: *Which non-functional requirements, service indicators, service objectives, safety invariants, evaluation gates, error budgets, and business-quality metrics are required to prove that an agentic software platform is reliable, secure, efficient, portable, observable, governable, and trustworthy enough for autonomous software delivery and multi-project operation?*

To answer this, conventional distributed-system telemetry must be merged with non-deterministic evaluation methodologies. Several foundational hypotheses dictate the architecture of this measurement system. Most critically, conventional availability Service Level Objectives (SLOs) cannot adequately describe semantic agent quality.[1] An agent may successfully return an HTTP 200 OK response, generating a technically flawless JSON payload, while simultaneously delivering a hallucinated, semantically destructive tool argument.[2] Consequently, technical completion and semantic correctness must remain strictly distinct states.[2]

Furthermore, agent evaluation metrics must not be mixed into one aggregate quality score. Compressing agent behavior into a single success metric obscures critical operational properties across consistency, robustness, and safety dimensions, systematically biasing results to reflect performance on frequent, easy tasks while hiding rare, high-severity failures.[1] A single successful agent run provides weak evidence of reliability due to the stochastic execution paths inherent to Large Language Models (LLMs); probabilistic quality fundamentally requires repeated trials and uncertainty reporting.[1]

Cost per model call is deemed an economically useless metric. Agentic platforms execute non-deterministic recursive loops, making the "cost per accepted task" the only valid FinOps denominator.[3] Human attention is categorized as a finite platform resource that must be explicitly measured; avoiding human intervention is the core utility of autonomy.[4] Additionally, telemetry completeness is an independent non-functional requirement. If an observation cannot be calculated reproducibly due to trace fragmentation, it cannot serve as a valid release gate. Finally, safety invariants must never be relaxed because an error budget remains; safety-critical objectives require independent, zero-tolerance enforcement mechanisms.[5]

Mandatory Measurement Separation

The rigorous evaluation of the Agent Platform demands the strict disentanglement of metrics into four orthogonal measurement classes. Collapsing these dimensions undermines governance and fundamentally breaks incident response routing.[1]

Operational SLOs

Operational SLOs measure the deterministic, real-time health and performance of the target system's services and infrastructure.[5] These indicators rely on continuous rolling observation windows and binary good-event versus total-event definitions.[5] They govern metrics such as control-plane availability, environment-provisioning latency, task claim latency, checkpoint durability, and workflow-resumption success. Failures in this domain safely consume error budgets and trigger automated incident-response policies using multi-window, multi-burn-rate alerting.[7]

Evaluation Release Gates

Evaluation release gates assess the probabilistic and semantic quality of models, prompts, agent roles, memory retrieval pipelines, and workflows. Because the platform exhibits stochastic behavior, these gates depend on repeated trials in controlled scenario suites, utilizing statistical confidence intervals and baseline regression detection.[1] Metrics such as the false-DONE rate, tool-selection accuracy, handoff completeness, and cross-provider continuation quality belong exclusively here. They dictate release-blocking thresholds but are mathematically invalid as real-time operational SLOs, as continuous production telemetry lacks the deterministic ground truth required for real-time semantic scoring.[1]

Safety Invariants

Safety invariants define absolute boundary conditions that tolerate zero or near-zero residual risk.[2] These parameters define absolute platform boundaries. Examples include the strict prevention of cross-project data leakage, the blocking of unauthorized irreversible actions (e.g., direct protected-branch writes), the prevention of secret disclosure into model context, and the instantaneous termination of stale worker mutations after lease loss.[8] Safety invariants are non-negotiable and cannot be traded against cost, throughput, latency, or error-budget availability.[5] A violation triggers an immediate system halt and maximum-severity incident response.

Business-Quality Metrics

Business-quality metrics quantify the economic and organizational utility of the platform. These indicators measure whether the platform creates useful outcomes.[9] Metrics such as the total cost per accepted task, human minutes required per accepted task, deployment throughput, and the cost of rework measure organizational value.[3] While they guide product prioritization and optimization, they must never be confused with platform safety or deterministic reliability.

Non-Functional Requirement (NFR) Taxonomy

A complete and practical NFR taxonomy is required to map the platform's architectural domains to measurable indicators. The tables below categorize the fundamental requirements of the target system.

Reliability, Availability, and Performance

NFR Category	Specific Requirement	System Boundary	Validation Method	Telemetry Requirements	Measurement Class
Reliability	Task-state persistence & Checkpoint correctness	Workflow Engine to Database	Fault injection (Process termination)	checkpoint_attempt_success_rate, checkpoint_persistence_latency	Operational SLO
Reliability	Duplicate side-effect prevention & Idempotency	Provider Gateway to External APIs	Idempotency key tracking, compensation analysis	duplicate_external_side_effect_rate, idempotency_key_coverage	Safety Invariant
Reliability	Workflow durability & Cross-provider continuation	Memory Service to Execution Environment	Provider failover simulation	workflow_resumption_success, cross_provider_resume_success_rate	Operational SLO
Availability	Control-plane & API availability	Operator Console to API Gateway	Continuous synthetic request probing	api_request_success_rate, control_plane_uptime	Operational SLO

Availability	Provider-gateway & Queue availability	Task Queue to Worker Pool	Multi-window burn rate monitoring	queue_eligibility_latency, worker_availability_ratio	Operational SLO
Performance	Task-claim latency & Intake latency	API Intake to Active Worker	Latency distribution analysis (p95/p99)	task_claim_latency, queue_time	Operational SLO
Performance	Worktree & Environment provisioning	Sandbox to Container Daemon	Boot sequence timing	environment_provisioning_latency, worktree_provisioning_latency	Operational SLO
Performance	Context compilation & Memory query throughput	Vector Database to Context Compiler	Load testing against context overload	context_compilation_latency, memory_query_throughput	Operational SLO

Security, Privacy, and Isolation

NFR Category	Specific Requirement	System Boundary	Validation Method	Telemetry Requirements	Measurement Class
Security	Project isolation & Sandbox isolation	Worktree to Host/Network	Adversarial path traversal, credential scoping tests	cross_project_filesystem_access_rate (Target: 0)	Safety Invariant
Security	Approval-ga	Operator	RBAC	approval_ev	Safety

	te & Authorisation enforcement	Console to External API	testing, simulated bypass attempts	ent_coverage, unauthorised_action_block_rate	Invariant
Security	Stale-worker protection & Lease correctness	Worker Process to Provider Gateway	Fencing token expiry validation	stale_worker_mutation_attempts	Safety Invariant
Privacy	Data minimisation & PII detection	Context Compiler to Provider	Pre-flight payload scanning	untrusted_content_label_coverage, telemetry_sanitisation_rate	Safety Invariant

Agentic Quality and Auditability

NFR Category	Specific Requirement	System Boundary	Validation Method	Telemetry Requirements	Measurement Class
Agentic Quality	Semantic correctness & False completion	Agent Output to Acceptance Criteria	Deterministic test validation, Independent Reviewer Agent	false_done_rate, reviewer_effectiveness	Evaluation Gate
Agentic Quality	Tool selection & Argument construction	LLM Router to Tool Gateway	Labelled evaluation datasets, adversarial tool schemas	tool_selection_precision, tool_argument_semantic_accuracy	Evaluation Gate
Agentic	Handoff	Agent A to	State	handoff_reh	Evaluation

Quality	quality & Cross-provider continuity	Agent B (or Provider A to B)	rehydration tests, continuation without rework	ydration_success_rate, duplicate_work_after_handoff_rate	Gate
Agentic Quality	Memory retrieval & Confidence calibration	Memory Service to LLM Context	Golden memory-query datasets	memory_retrieval_precision, stale_memory_rate	Evaluation Gate
Auditability	Action attribution & Trace continuity	End-to-End Workflow	Trace ID coverage analysis, orphaned telemetry checks	trace_id_coverage, tool_receipt_coverage	Operational SLO
Auditability	Git provenance & Release-evidence completeness	Workflow Engine to Version Control	Commit hash correlation mapping	git_provenance_coverage, context_manifest_coverage	Operational SLO

Capacity, Portability, and Economics

NFR Category	Specific Requirement	System Boundary	Validation Method	Telemetry Requirements	Measurement Class
Capacity	Safe worker concurrency & Queue depth	Scheduler to Infrastructure	Saturation and load testing	maximum_active_workflows, database_connection_limits	Operational SLO

Portability	Provider substitution & Model substitution	Platform Core to External LLM API	Regression evaluation on adapter substitution	provider_specific_error_normalisation_rate	Evaluation Gate
Economics	Cost per accepted task & Retry overhead	Whole System	FinOps token usage and human-hour correlation	total_platform_cost_per_accepted_task, human_minutes_per_task	Business Metric

Reliability, Durability, and Canonical State

Canonical state durability dictates the platform's ability to survive catastrophic infrastructure failure without corrupting active reasoning loops. The required guarantees for task records, workflow state, handoffs, and tool receipts demand acknowledged-write durability.[11] The system must employ event sourcing backed by append-only ledgers to guarantee transaction boundaries and utilize outbox patterns to guarantee at-least-once outbox delivery to external message brokers.

However, external side-effects (e.g., mutating an external cloud resource) must be protected by strict idempotency controls. Duplicate side-effect prevention requires idempotency_key_coverage across all tool executions. Keys must be deterministically generated using the workflow ID, task ID, and attempt counter. Harmless duplicate queries (reads) are operationally acceptable, but duplicate mutations constitute a severe safety violation.

Checkpoint and Recovery Objectives

Checkpointing is the fundamental mechanism for bounding the Recovery Time Objective (RTO) of a long-running agent. State that must be checkpointed includes active variable bindings, memory conversation history, unresolved dependencies, and reference pointers to large storage artifacts.

The primary indicators include checkpoint_attempt_success_rate and checkpoint_restore_success_rate. Checkpoint validity is tested via checksums and periodic restoration drills. In the event of a worker crash, the workflow_resumption_success metric tracks whether the interrupted work can resume from the last checkpoint without restarting the entire reasoning chain. Semantic work continuity matters more than the exact restoration of side-effect history; if a task is restored, it must not repeat previously successful tool calls.[11]

Availability, Flow, and Degraded Modes

The control-plane boundary encompasses task intake, state transition, scheduling, approval recording, and release gating. Availability must be measured separately for reads (status inspection) and writes (task control). Long-running workflows frequently enter human-wait states, waiting for approval without consuming active compute. Wait-state durability requires that the state is serialized to cold storage; approval_wait_resumption_latency measures the system's ability to reload the context upon receiving asynchronous human input.
During a provider outage or database degradation, the platform must smoothly enter degraded modes. Valid degraded behaviors include pausing external side effects, refusing new task intakes, relying exclusively on cached local memory, and entering a read-only operator mode. Active tasks must be checkpointed and paused rather than aggressively retried, thereby preserving the error budget and avoiding token expenditure on failing connections.
Throughput must be explicitly defined. Tasks merely executed or technically completed by the agent do not count toward throughput. The primary throughput unit is the *operationally validated accepted task*.[10] Flow metrics such as task_arrival_rate, queue_time, claim_latency, and human_minutes_per_task must be heavily segmented by risk lanes (e.g., fast routine vs. high-complexity security tasks). A high-risk security task will inherently feature higher approval_wait_time and strict concurrency limits, whereas a fast-routine task should theoretically operate with near-zero human intervention.

Provider Failover and Portability

True provider independence requires measuring externally observable outcomes rather than provider-specific implementation details. Failover classes must be explicitly distinguished: retrying with the same model, switching to a different model within the same provider, switching to a completely different provider family, or falling back from cloud to local execution.[2]
A failover is only correct if it preserves task identity, acceptance criteria, current state, prior tool results, and the original permission scope.[2] Measuring failover quality requires tracking cross_provider_resume_success_rate and quality_after_failover. A technically successful API routing switch that causes the secondary model to hallucinate or restart the task is a semantic failure. Model substitution must trigger an immediate regression evaluation to ensure that context-loss rates and repeated-action rates remain below baseline thresholds.[1]

Worktree, Environment Provisioning, and Context Compilation

Isolated execution environments are non-negotiable for agentic software engineering.[12]

Worktree and Environment Objectives

Metrics such as environment_provisioning_latency and worktree_boot_success_rate are critical operational SLOs. Capacity planning requires maintaining warm pools of prebuilt images, but

environment reuse introduces severe cross-project contamination risks. Provisioning speed must never override security constraints: environment_isolation_test_pass_rate must remain at 100%. Filesystem isolation, network-egress restrictions, and strict credential scoping are safety invariants.

Context Compilation Objectives

The context compiler acts as the bridge between the memory service and the provider gateway. Operational indicators include context_compilation_latency and context_source_resolution_success_rate. Every context build must be perfectly reproducible from the source identifiers, retrieval configuration, provider context limits, and active decisions.[13] This reproducibility is guaranteed by generating a Context Manifest.

Semantic quality indicators for context include required_context_recall, irrelevant_context_rate, and context_truncation_loss_rate.[14] Increasing context size can severely degrade reasoning performance via the "lost in the middle" phenomenon.[15] Therefore, token cost and instruction adherence must be plotted against context size to locate the optimization frontier.

Agentic Quality: False-DONE, Tools, and Handoffs

Agentic evaluation cannot rely on aggregate scoring; it must decompose into quality, suitability, and efficiency dimensions.[1]

The False-DONE Rate

The false-DONE rate is the most critical agent-specific quality measure. A false-DONE event occurs when a task is declared successfully complete by the agent, but one or more mandatory acceptance criteria are not actually satisfied.[1] Variants include semantic false-DONE (where tests pass but the underlying requirement fails), evidence-free DONE, and partial-completion DONE.

The denominator for this metric must be strict: $\frac{\text{False-DONE Events}}{\text{Accepted Tasks}}$. Detection requires a multi-layered approach: deterministic acceptance tests, independent reviewer agents executing verification loops, and human post-release defect sampling.[16] For deterministic, low-risk implementation tasks, the acceptable false-DONE threshold may tolerate minor variance, but for security-critical operations, zero false-DONE is a mandatory release gate.

Tool-Selection Accuracy

The complete tool-decision pipeline must be evaluated. Measurement dimensions extend far beyond simple precision and recall.[2] Indicators include tool_argument_schema_validity and, critically, tool_argument_semantic_accuracy.[2] A model may perfectly format a JSON API call (schema validity) while passing the wrong target ID (semantic failure). The forbidden_tool_attempt_rate serves as a strict proxy for prompt-injection resistance and model alignment.[2] Evaluation requires creating scenario suites involving ambiguous tool descriptions and missing permissions to validate the agent's error-handling recovery.[8]

Handoff Completeness

In multi-agent architectures, handoffs allow tasks to cross agent and provider boundaries.[8] A handoff must contain the original objective, validated acceptance criteria, tool receipts, decisions, unresolved work, and recommended next actions. The primary indicator is the handoff_rehydration_success_rate. The ultimate test of handoff quality is the continuation_without_rework_rate: can the receiving agent or provider continue the workflow from the exact commit without repeating diagnostic checks or hallucinating past state? [2]

Memory Retrieval Quality and Project Isolation

Memory storage quality must be evaluated separately from retrieval quality. Retrieval indicators include memory_retrieval_precision, memory_retrieval_recall, and relevant_memory_at_k.[14] High recall prevents context starvation, but low precision causes hallucination via irrelevant context.[15]

Trust and freshness are critical properties of agentic memory. The stale_memory_rate and superseded_memory_rate track the system's ability to filter out deprecated architectural decisions. Every retrieved item must include provenance metadata: scope, source, version, valid time, and trust classification.

Project isolation relies heavily on memory segregation and is categorized strictly as a safety invariant.[8] No project may read another project's private memory, and no provider request may contain unauthorized cross-project context. The cross_project_retrieval_violation_rate must be zero. Adversarial tests must bombard the system with ambiguous project identifiers, stale cached context, and malicious memory queries to validate boundary enforcement.

Agentic Economics and Human Attention

Agentic FinOps diverges sharply from standard cloud FinOps.[3] Traditional workloads map neatly to deterministic compute and storage; agentic costs are driven by compounding multi-step reasoning, non-deterministic token consumption, recursive tool loops, and massive context windows.[3]

Cost per Accepted Task

Cost per model call is a deceptive metric. The true business-quality metric is total_platform_cost_per_accepted_task.[10] This model must encompass model inference, environment compute, storage, retries, provider failover, rework loops, and the amortized cost of human review.[3]

Every cost record must attribute spend to the organization, project, workflow, task, provider, and tool.[3] A quality-cost trade-off analysis must utilize Pareto frontiers rather than single-score optimization. Lowering the inference cost by routing to a cheaper model is an economic failure if it drastically increases the false-DONE rate, rework loops, or human intervention.[9]

Human Attention Efficiency

Operator attention is a finite, expensive platform resource.[4] Valid human attention involves required business decisions or high-risk approvals; avoidable human attention involves routine technical questions caused by an agent's artificial ambiguity or poor context retrieval.[3] Metrics include human_minutes_per_task, avoidable_question_rate, and escalation_acceptance_rate. Attention efficiency measures whether an agent's escalation packet contains the exact decision required, the evidence gathered, and the consequences of the available options. Different risk lanes dictate different objectives: standard implementation should require zero technical questions, whereas emergency production tasks may demand continuous human-in-the-loop oversight.

Traceability and Audit Integrity

Traceability ensures that every consequential workflow can be reconstructed. The scope must preserve chronological links: from the initial objective to the attempt, model calls, context manifest, tool intents, receipts, file commits, reviews, and operational release evidence.[8] Indicators include trace_id_coverage, tool_receipt_coverage, and approval_event_coverage. Audit integrity requires append-only storage, event hashing, tamper detection, and strict clock synchronization. Telemetry failure introduces a critical policy decision: if telemetry routing degrades, the platform must fail closed for audit-critical operations, prioritizing non-repudiation and security over workflow throughput.

SLI, SLO, and Error Budget Specifications

The platform requires robust, machine-readable schemas to manage telemetry and objectives, leveraging standards such as the OpenSLO specification.[17]

Example Machine-Readable Schemas

SLI Record Schema (OpenSLO format)

YAML

```
apiVersion: openslo/v1
kind: SLI
metadata:
  name: provider-gateway-availability
  displayName: Provider Gateway Availability
spec:
  description: "Measures the successful completion of model inference requests at the gateway."
  system_boundary: "Platform Gateway to External LLM Provider"
  measurement_class: "Operational SLO"
  good_event_definition: "HTTP 200 OK received with valid JSON payload"
  valid_event_definition: "All outbound inference requests, excluding canceled tasks"
```

```
  bad_event_definition: "HTTP 5xx, timeouts, or malformed provider responses"
  numerator: "sum(provider_success_events)"
  denominator: "sum(provider_total_events)"
  aggregation: "rolling"
  dimensions: ["provider_id", "model_id", "risk_lane"]
  freshness_requirement: "1m"
```

SLO Record Schema (OpenSLO format)

```
YAML
apiVersion: openslo/v1
kind: SLO
metadata:
  name: standard-lane-gateway-slo
  displayName: Standard Lane Gateway SLO
spec:
  indicatorRef: provider-gateway-availability
  target: 0.995
  window: 30d
  budgetingMethod: Occurrences
  error_budget_policy: "Route to fallback provider upon 50% budget exhaustion"
  alert_policy: "mw-mbr-tier-2"
  risk_lane: "standard"
```

Event-Based vs. Time-Based Indicators

Operational indicators rely on event-based metrics (e.g., successful checkpoints / total checkpoints) rather than purely time-based uptime, as agents operate in discrete, asynchronously scheduled blocks.[6] Valid event definitions must prevent metric gaming by excluding expired deadlines, policy-blocked actions, and human-wait states from latency calculations.

Error Budgets and MWMBR Alerting

The conventional error budget model (1 - target reliability) applies cleanly to API availability and task claim success.[5] However, agentic quality failures (e.g., invalid tool calls, memory retrieval errors) utilize evaluation regression budgets assessed over evaluation batches, not real-time continuous budgets.[1]

The platform must adopt the Multi-Window Multi-Burn-Rate (MWMBR) alerting strategy to balance detection speed with false-positive suppression.[7] The burn rate is calculated as:

$$\text{Burn Rate} = \frac{1 - \text{Current Success Rate}}{1 - \text{SLO Target}}$$

For a 99.9% SLO target, a 99.5% current success rate yields a burn rate of 5x. An error-budget consumption policy dictates staged responses:

- **Fast Burn (14.4x rate):** Evaluated over 5-minute and 1-hour windows.[7] Poses an immediate threat to the 30-day budget (exhaustion in 2 days). Consequence: Immediate page, rapid isolation of the failing provider.
- **Slow Burn (3x rate):** Evaluated over 2-hour and 24-hour windows. Consequence: Ticket creation, prioritize reliability work over feature releases.
- **100% Exhaustion:** Freeze feature releases, pause non-critical workflow intake, require mandatory incident analysis.

Safety invariants bypass this math entirely. A cross-project leakage event has no budget concept; it triggers maximum severity response on the first occurrence.

Statistical Treatment of Probabilistic Quality

Because agents are probabilistic, deterministic pass/fail tests are insufficient for semantic quality.[2]

Repeated Trials and Confidence Intervals

Evaluation requires repeated stochastic runs across controlled datasets. Randomization, seed handling, and temperature settings must be rigorously documented.[2] To evaluate binomial proportions (e.g., success/failure rates of tool selection or false-DONE events), point averages are mathematically invalid. The system must compute confidence intervals using the Wilson score interval or Agresti-Coull interval to account for sample size uncertainty.[20]

Minimum sample sizes must be calculated based on desired statistical power; declaring an agent "reliable" based on five successful runs is an explicit anti-pattern. Averages hide catastrophic tail behavior.[1] Tail risk must be measured by examining worst-case costs, p95/p99 execution latency, and quality variance distributions.

Regression detection occurs when the lower bound of a baseline configuration's confidence interval exceeds the upper bound of a candidate release's confidence interval, triggering an automatic release block.

Evaluation Release Gates and Business Metrics

Release gates form the barrier between experimental model configurations and production deployment.[8]

Example Evaluation Gate Record

YAML
evaluation_gate_id: eval-gate-handoff-v2
scope: "agent-role-handoff"
metric_ids: ["handoff_rehydration_success_rate", "duplicate_work_after_handoff_rate"]
required_thresholds:
  handoff_rehydration_success_rate: "> 0.95"
  duplicate_work_after_handoff_rate: "< 0.02"
sample_size: 300
confidence_method: "wilson_score_interval"
allowed_regression: "0.01"
blocking: true

Gates are categorized into deterministic checks (schema validity, policy conformance), statistical checks (false-DONE thresholds, retrieval quality), and human-reviewed checks (decision quality, semantic task usability). Every release must compare the new artifact against previous baselines, explicitly documenting changed models, prompts, routing logic, and tools.

Business-Quality Metrics

Business outcomes are entirely separate from technical reliability.[9] Metrics such as accepted_tasks_per_week, human_hours_saved, and cost_of_quality serve as lagging indicators of organizational value. While they guide strategic prioritization, they should never be utilized to block CI pipelines or excuse safety invariant violations.

Capacity, Load, and Chaos Testing

Defining safe concurrency limits by component requires aggressive capacity and load testing. Workloads must simulate task-intake bursts, concurrent context compilation, provider rate-limit saturation, and mass worktree provisioning. Saturation points, queue growth, and isolation under heavy load must be explicitly documented.

Failure Injection and Chaos Testing

Metrics are only theoretical until proven by failure injection. The research mandates validating detection telemetry across injected failures:

- **Provider Outage:** Inject HTTP 503s. Expected: provider_timeout_rate spikes, failover routing engages, trace correlation survives.
- **Checkpoint Corruption:** Corrupt the JSON payload of an active workflow. Expected: Checksum fails, RPO rollback triggers, checkpoint_restore_success_rate drops.
- **Stale Lease:** Force a worker process to sleep, allowing its lease to expire, then attempt a database mutation. Expected: Fencing token validation rejects the write, proving the stale-worker safety invariant.
- **Context Truncation:** Flood the context compiler with massive artifacts. Expected:

context_truncation_loss_rate registers the loss, prompting the agent to fail gracefully rather than hallucinate.[15]

Metric Gaming and Goodhart Risks

When metrics become targets, they become vulnerable to manipulation (Goodhart's Law). The system must preemptively document gaming risks and implement counter-metrics:

- **Gaming Task Success:** An agent or routing system might selectively drop or mark complex tasks as "blocked" to artificially inflate the success rate. *Counter-metric:* Track the abandonment_rate and blocked-task rate against historical baselines.[2]
- **Gaming Memory Precision:** An agent retrieving only a single, highly relevant document maximizes precision but destroys recall, leading to context starvation. *Counter-metric:* Use harmonic means (F1 score) to balance precision and recall.[22]
- **Gaming Cost per Task:** Bypassing evaluation steps or deferring required actions to humans lowers machine compute costs. *Counter-metric:* Measure the human_cost_per_accepted_task and the cost_of_escaped_defects to capture the true total cost.[10]
- **Superficial Throughput:** Agents splitting tasks into artificially small micro-tasks to inflate throughput. *Counter-metric:* Measure throughput strictly by accepted-task throughput mapped to initial user intent.

Required Comparisons

- **Traditional SRE SLOs vs. Agentic Metrics:** Traditional SLOs measure binary uptime and latency well, but completely fail to measure semantic correctness or probabilistic hallucination.[1] Agentic evaluation bridges this gap but incurs high implementation burden due to the necessity of repeated batch trials.
- **FinOps vs. Agent-Specific FinOps:** Traditional cloud FinOps allocates static compute instances to organizational tags. Agent-specific FinOps requires granular per-token tracking mapped directly to dynamic workflow attempts, capturing the compounding cost of reasoning loops.[3]
- **LLM Evaluation vs. Agentic Workflows:** Standard LLM benchmarks evaluate zero-shot reasoning well. They fail to measure multi-step tool execution, environmental interaction, and memory rehydration.[2] Agentic workflow metrics are required to evaluate autonomous state management.

Required Experiments

A rigorous adoption plan necessitates executing twelve deterministic experiments:

1. **NFR Catalogue:** Build a comprehensive mapping of platform components to operational SLOs, evaluation gates, and safety invariants.
2. **Critical User Journeys:** Define ten journeys (e.g., task intake to claim, cross-agent handoff, provider outage to failover) and map SLIs and provisional objectives for each.
3. **Failure Injection:** Execute chaos scenarios against the critical journeys to verify that

proposed indicators and MWMBR alerts actually fire.

4. **False-DONE Evaluation:** Create a dataset of deterministic and semantic false-DONE cases. Compare self-reported agent completion against deterministic validation and independent reviewer agents.[16]
5. **Tool Accuracy:** Utilize a labeled tool-selection dataset containing ambiguous schemas to evaluate the semantic accuracy of argument construction across two different frontier models.
6. **Handoff Continuation:** Interrupt a task, serialize the handoff packet, and transfer it across provider boundaries. Measure the resulting duplicated work to evaluate continuation quality.[2]
7. **Memory Retrieval:** Construct a golden memory dataset infused with irrelevant, stale, and contradictory records. Measure precision, recall, and the agent's ability to exclude superseded architecture decisions.
8. **Cost-Quality Frontier:** Route identical tasks through frontier and mid-tier models. Construct a Pareto curve comparing accepted-task rate, latency, false-DONE rate, and human intervention against true total cost.[3]
9. **Lane-Specific SLOs:** Compare alert fatigue between a single global objective set and segmented objectives (routine vs. high-complexity security lanes).
10. **Metric Gaming:** Task a red-team with improving task_acceptance_rate and cost_per_task without genuinely improving outcomes. Document the successful manipulations and design counter-metrics.
11. **Statistical Stability:** Execute the same scenario suite 100 times. Calculate the variance and establish the minimum sample size required to produce a valid Wilson score interval.[21]
12. **Telemetry Completeness:** Corrupt trace spans mid-flight. Verify that audit controls detect the broken continuity and flag the transaction.

Required Artefacts

The engineering program must deliver fourteen specific artifacts:

1. non-functional-requirements-report.md: Synthesis of architecture recommendations, metric separation, and trade-offs.
2. nfr-catalogue.csv: Mapping nfr_id, category, system boundary, measurement class, validation method, and telemetry requirements.
3. sli-slo-error-budget-spec.md: Formal mathematical definitions of MWMBR formulas, time windows, and burn-rate policies.
4. v1-service-objectives.yaml: Provisional early-release OpenSLO targets.
5. v2-service-objectives.yaml: Multi-project scale OpenSLO targets, incorporating portfolio views.
6. agentic-quality-metrics-spec.md: Formal definitions of false-DONE, tool selection precision, handoff rehydration, and repeated-run variance calculations.
7. safety-invariants.yaml: Machine-readable rules defining zero-tolerance policies (e.g., cross-project data leakage) and their detection controls.

8. evaluation-release-gates.yaml: Blocking thresholds, sample sizing, and statistical confidence interval methodologies for CI pipelines.
9. measurement-and-telemetry-requirements.md: Data contracts mapping metrics to specific structured JSON logging events and retention windows.
10. slo-breach-response-policy.md: Playbooks defining engineering actions, release freezes, and provider disablement based on rapid error-budget burn.
11. capacity-and-load-test-plan.md: Definition of concurrency saturation points for worktree provisioning and context compilation under burst loads.
12. reliability-and-recovery-objectives.md: Explicit targets for workflow RTO, RPO, and checkpointing restoration.
13. agentic-evaluation-datasets.md: Specifications for golden, adversarial, and scenario-based evaluation suites.
14. metrics-governance-policy.md: Framework for metric ownership, anti-gaming review, and periodic recalibration.

Initial Objective Categories

Provisional objectives must be calibrated using baseline data rather than inventing arbitrary thresholds.

- **Reliability:** Checkpoint restore success target must be extraordinarily high (e.g., >99.9%); duplicate non-idempotent side-effect rate must be strictly 0.
- **Availability:** Control-plane and task-control APIs require standard high-availability targets (e.g., 99.95%) governed by operational error budgets.
- **Agentic Quality:** The false-DONE rate for deterministic tasks must be 0; handoff continuation success must be calibrated via Experiment 6; unsupported claims must trigger prompt regressions.
- **Security:** Isolation breach rate must be 0. Secret exposure rate must be 0. Stale-worker mutation rate must be 0. These are safety invariants.
- **Economics:** Cost per accepted task requires a baseline collection phase to determine the specific Pareto frontier for the organization.[9]

Phased Implementation Roadmap

A staged adoption plan prevents operational whiplash.

- **Phase 0 — Measurement Vocabulary:** Establish definitions. Prohibit aggregate agent-quality scoring. Define stable IDs. *Deliverable:* NFR Taxonomy.
- **Phase 1 — Core Operational Telemetry:** Implement standard trace_id correlation across the control plane, providers, and FinOps endpoints. *Deliverable:* 100% trace correlation visibility.
- **Phase 2 — Reliability SLOs:** Implement MWMBR alerting for durability, checkpointing, recovery, and environment provisioning. *Deliverable:* v1-service-objectives.yaml.
- **Phase 3 — Agentic Evaluation Gates:** Introduce statistical evaluation for false-DONE, tool accuracy, and retrieval precision into CI pipelines. *Deliverable:* Release-blocking evaluation tests.

- **Phase 4 — Safety Invariants:** Hard-code IAM and fencing controls to enforce project isolation and stale worker mitigation. *Deliverable:* Zero-tolerance security gates.
- **Phase 5 — Economics and Human Attention:** Correlate cost records to trace IDs to generate cost_per_accepted_task dashboards. *Deliverable:* Agentic FinOps allocation architecture.
- **Phase 6 — Error Budgets and Release Policy:** Enforce automated release freezes and dynamic provider disablement based on burn rates. *Deliverable:* slo-breach-response-policy.md.
- **Phase 7 — Multi-Project Scale:** Roll out inherited objectives, portfolio views, and capacity quotas. *Deliverable:* v2-service-objectives.yaml.

Anti-patterns to Reject

The following anti-patterns critically undermine the reliability and safety of agentic systems and must be explicitly rejected:

Anti-Pattern	Why it Fails	Detection Mechanism	Safer Alternative
One aggregate "agent quality" score	Obscures catastrophic safety failures (e.g., data deletion) behind high task completion rates.[1]	Code reviews of evaluation aggregation logic.	Decompose into distinct Quality, Suitability, and Efficiency dimensions.[1]
Task completion rate without acceptance validation	Results in massive false-DONE rates where agents mark failing tasks as complete.[1]	High defect reports despite high reported success.	Enforce accepted_task_rate using deterministic test receipts.[10]
Treating all failures as budgetable	Equates transient API timeouts with severe isolation breaches.	Reviewing error budget policies for safety exclusions.	Strictly separate operational SLOs from zero-tolerance Safety Invariants.
Averaging safety failures with productivity gains	Encourages trading security for speed.	Tracking performance against security metrics.	Safety invariants must instantly block releases, regardless of throughput.

Using uptime SLOs for semantic quality	A fast, highly available hallucination is still a failure.[1]	Discrepancy between uptime metrics and user satisfaction.	Use Evaluation Release Gates based on repeated semantic trials.
Setting arbitrary thresholds without baselines	Leads to alert fatigue or meaningless goals.	Tracking alert validity.	Execute baseline calibration periods and label targets as provisional.
Evaluating one run per task	Ignores the inherent non-deterministic stochastic variance of LLMs.[2]	High variance in production vs evaluation.	Use repeated trials with Wilson score intervals.[21]
Comparing providers on different task samples	Invalidates statistical equivalence.	Audit of dataset parameters during evaluation.	Use identical, controlled scenario suites for A/B provider testing.
Measuring model cost without total task cost	Ignores the massive cost of recursive retry loops and human intervention.[3]	High cloud bills despite "cheap" models.	Measure total_platform_cost_per_accepted_task.
Counting agent questions without classification	Penalizes agents for asking necessary, high-risk security questions.[4]	Tracking avoidable_question_rate via human feedback.	Distinguish necessary escalation from artificial ambiguity.
Measuring retrieval precision without recall	Encourages agents to retrieve too little, causing context starvation.[22]	F1 score analysis.	Balance precision and recall evaluation.
Measuring throughput from self-declared DONE events	Invites gaming; agents learn to claim victory instantly.	Measuring false-DONE rates.	Throughput strictly requires external deterministic validation.

Excluding hard tasks to improve success rate	Skews metrics and hides capability boundaries.	Tracking abandonment and dataset composition.	Segment SLOs by risk lane and track task abandonment.
Allowing telemetry gaps during irreversible actions	Destroys auditability and non-repudiation.	Tracking orphan_telemetry_rate.	Fail closed if telemetry routing degrades during critical external writes.
Using model self-report as the source of truth	Susceptible to sycophancy and hallucination.	Auditing self-reports against deterministic logs.	Rely on deterministic harness observation where possible.
Relying solely on LLM-as-judge	Passes systemic biases and hallucinations into the evaluation pipeline.[2]	Human-in-the-loop consensus audits.	Combine LLM-as-judge with deterministic unit tests and human samples.
Alerting on every failed agent attempt	Generates massive alert fatigue due to normal stochastic retry behavior.	Tracking operator alert-acknowledgment rates.	Alert strictly on MWMBR error budget consumption.[7]

Research Methodology and Quality Controls

Recommendations are derived strictly from formal SRE guidance, observability patterns, FinOps frameworks, and peer-reviewed agent evaluation research. Crucially, proposed metrics clearly distinguish between deterministic operational SLIs and probabilistic evaluation methodologies. Research quality controls necessitate distinct roles: SREs govern operational SLOs and MWMBR alerting; security engineers define strict safety invariants; FinOps specialists govern token-level cost attribution architectures; and statistics specialists define the confidence intervals required for batch evaluation.[20] Deterministic scripts must execute schema validation, duplicate ID detection, and confidence interval calculations to prevent manual error. A metric is never accepted solely because a model recommends it; it must prove its utility through adversarial failure injection.

Final Synthesis

Which NFRs are mandatory for the first trustworthy release? Canonical state durability, control-plane availability, strict sandbox environment isolation, end-to-end trace completeness, and the explicit tracking of the false-DONE rate are non-negotiable for v1.
Which become mandatory only for multi-project scale? Cross-project data isolation enforcement, inherited portfolio-level objectives, shared resource quotas, and cross-project FinOps attribution.
Which indicators are operational SLIs? Control-plane availability, task claim latency, checkpoint restore success, environment provisioning latency, and provider timeout rates.
Which measurements belong only in evaluation suites? False-DONE rates, semantic tool-argument accuracy, memory retrieval precision/recall, and cross-provider continuation quality.
Which conditions are safety invariants? No cross-project data leakage, strict secret masking, prevention of unauthorized irreversible actions, and instantaneous stale-worker lease revocation.[8]
Which metrics are business outcomes? Total cost per accepted task, human minutes required per accepted task, and deployment cycle time.[9]
What is the correct definition of an accepted task? A task is accepted only when deterministic acceptance criteria pass, mandatory security reviews are completed, and cryptographic execution evidence (e.g., git commits) is attached.[10]
How should false-DONE be measured? As the proportion of tasks declared "DONE" by the agent that subsequently fail independent deterministic validation or reviewer-agent checks, calculated over accepted tasks.
How should semantic quality be evaluated statistically? Through repeated trials against controlled datasets, utilizing the Wilson score interval or Agresti-Coull interval to calculate confidence bounds for binomial outcomes.[21]
Which failure classes may consume an error budget? Transient queue delays, API timeouts, recoverable provider errors, and temporary environment provisioning delays.
Which failure classes have zero tolerance? Project boundary traversal, unauthorized production data deletion, and successful prompt injection overriding guardrails.
How should provider failover quality be measured? By measuring the continuation_without_rework_rate and verifying the preservation of exact task constraints upon resuming with the secondary provider.
What recovery guarantees are required? Acknowledged-write durability of the workflow state, guaranteeing that RPO rollback allows the task to resume without duplicating non-idempotent side effects.
How should worktree and environment provisioning be measured? Via provisioning latency operational SLOs, gated by strict binary security isolation test scripts.
How should context quality be measured? By evaluating the Context Manifest against relevant_memory_at_k, monitoring truncation rates, and checking untrusted-content label coverage.[14]
What constitutes a complete handoff? A serialized state packet containing the objective, validated results, active assumptions, and precise reference pointers, allowing a secondary

agent to proceed without querying baseline data.

How should memory retrieval precision, recall, freshness, and isolation be balanced? Precision prevents hallucination; recall prevents context starvation; freshness metadata prevents reliance on superseded architecture; and isolation enforces hard namespace boundaries.

What cost categories belong in cost per accepted task? Model inference (input/output tokens), vector database compute, environment compute, retry overhead, failover costs, and the amortized cost of human operator review.[3]

How should human attention be valued? As a highly constrained, premium resource measured in human_minutes_per_task. Reductions in compute cost that increase human escalation degrade overall platform ROI.[4]

What trace and audit evidence is mandatory? Immutable correlation of trace_id, task_id, attempt_id, context manifests, tool receipts, and final output artifacts.

How should objectives vary by task risk lane? Routine tasks optimize for latency and autonomy; high-risk infrastructure tasks prioritize zero-defect quality gates, mandatory human approvals, and strict safety bounds.[4]

Which metrics should block CI? Statistically significant regressions in tool selection accuracy, memory retrieval F1 scores, or the false-DONE rate.

Which metrics should block release? Violations of safety invariants, severe degradation in semantic handoff completion, or any isolation breaches during chaos testing.

How should rapid error-budget burn affect provider routing? A fast burn (e.g., 14.4x rate) on a provider gateway must instantly trigger a circuit breaker, routing traffic to a pre-warmed fallback provider while preserving existing task leases.[7]

How should metric definitions be versioned? Via declarative schemas (e.g., OpenSLO YAML) managed through GitOps, treating SLOs as immutable code artifacts.[17]

How should evaluation datasets remain representative? By continuously harvesting sanitized production failures, escaped defects, and complex human escalations into the golden evaluation corpus.

How should metrics resist gaming? By deploying paired counter-metrics (e.g., tracking abandonment alongside success) and utilizing cryptographic receipts to audit self-reported completion.

What should be displayed to a non-developer operator? Composite error-budget burn rates, current risk lane health, total cost per accepted task, and actionable, context-rich escalation queues.

What is the minimum viable measurement system? A unified telemetry pipeline capturing distributed trace IDs, a binary task acceptance validator, a deterministic check for false-DONE events, and a single MWMBR alert governing API availability.

What evidence would invalidate the final recommendations? The emergence of models exhibiting zero temporal variance (pure determinism), the elimination of context window token costs (invalidating the FinOps models), or empirical proof that aggregate scalar scoring accurately predicts specific catastrophic safety failures without masking tail risk.

Final Execution Instruction

The deployment of this framework must be executed as a rigorous SRE, evaluation, safety, and FinOps engineering program. The platform must optimize for measurable guarantees, provider-neutral definitions, deterministic evidence, explicit safety boundaries, actionable MWMBR error budgets, and true cost per accepted outcome. Where evidence does not justify a numerical target, implement a baseline-collection experiment and label the objective provisional. Do not invent precise thresholds without empirical grounding.

Works cited

1. Monitoring Agentic Systems Before They're Reliable - arXiv, accessed July 26, 2026, https://arxiv.org/html/2606.02494v1
2. Beyond Task Completion: An Assessment Framework for Evaluating Agentic AI Systems, accessed July 26, 2026, https://arxiv.org/html/2512.12791v1
3. AI FinOps: The Complete Guide to Controlling AI Agent Costs - Praesidia, accessed July 26, 2026, https://praesidia.ai/guides/ai-finops
4. Agentic Orchestration: How AI Workflows Coordinate Agents, Tools, accessed July 26, 2026, https://uibakery.io/blog/agentic-orchestration
5. What is Site Reliability Engineering (SRE)? - Xurrent, accessed July 26, 2026, https://www.xurrent.com/blog/site-reliability-engineering-sre-explained
6. SLOs, SLIs, and SLAs: Meanings & Differences | New Relic, accessed July 26, 2026, https://newrelic.com/blog/observability/what-are-slos-slis-slas
7. How to Implement Multi-Window Alerts - OneUptime, accessed July 26, 2026, https://oneuptime.com/blog/post/2026-01-30-multi-window-alerts/view
8. Beyond Task Completion: An Assessment Framework for Evaluating Agentic AI Systems, accessed July 26, 2026, https://www.alphaxiv.org/audio/2512.12791v2
9. FinOps for AI Agents: How Enterprises Control Cost, Value, and Scale - Orq.ai, accessed July 26, 2026, https://orq.ai/blog/ai-agent-finops
10. AI Agent ROI Calculator: Cost, Payback and Break-Even - Decode, accessed July 26, 2026, https://decodethefuture.org/en/ai-agent-roi-calculator/
11. Durable Execution for AI Agents: Inngest vs Trigger.dev vs Temporal, accessed July 26, 2026, https://noqta.tn/en/blog/durable-execution-ai-agents-inngest-trigger-temporal-2026
12. Agentic Software Engineering - INNOQ, accessed July 26, 2026, https://www.innoq.com/en/topics/agentic-software-engineering/
13. The 8 Types of LLM Hallucinations and How to Fix Them - LangWatch, accessed July 26, 2026, https://langwatch.ai/blog/the-8-types-of-llm-hallucinations
14. Audit of Hallucinations in LLM-based Models and Solutions | by Owen Fletcher | Medium, accessed July 26, 2026, https://medium.com/@firstlinesoftware/audit-of-hallucinations-in-llm-based-models-and-solutions-694dde3fbb5e
15. Measuring LLM Hallucinations: The Metrics That Actually Matter for Reliable AI Apps, accessed July 26, 2026, https://www.getmaxim.ai/articles/measuring-llm-hallucinations-the-metrics-that-act

ually-matter-for-reliable-ai-apps/
16. ai-first-sdlc-practices/plugins/sdlc-core/agents/critical-goal-reviewer, accessed July 26, 2026, https://github.com/stevegjones/ai-first-sdlc-practices/blob/main/plugins/sdlc-core/agents/critical-goal-reviewer.md
17. OpenSLO, accessed July 26, 2026, https://openslo.com/
18. OpenSLO/OpenSLO: Open specification for defining and expressing service level objectives (SLO) - GitHub, accessed July 26, 2026, https://github.com/openslo/openslo
19. How to Set Up Multi-Window Multi-Burn-Rate Alerting for SLOs on Google Cloud, accessed July 26, 2026, https://oneuptime.com/blog/post/2026-02-17-how-to-set-up-multi-window-multi-burn-rate-alerting-for-slos-on-google-cloud/view
20. Binomial Distribution: Definition & FutureAGI Guide (2026), accessed July 26, 2026, https://futureagi.com/glossary/binomial-distribution/
21. (PDF) Better Confidence Intervals for a Binomial Proportion - ResearchGate, accessed July 26, 2026, https://www.researchgate.net/publication/290969007_Better_Confidence_Intervals_for_a_Binomial_Proportion
22. Detecting hallucinations with LLM-as-a-judge: Prompt engineering and beyond | Datadog, accessed July 26, 2026, https://www.datadoghq.com/blog/ai/llm-hallucination-detection/
23. Our POV - Evaluating LLM Hallucinations - Factored AI, accessed July 26, 2026, https://www.factored.ai/our-pov/llm-hallucination-evaluation