SALESFUSION

Governed Continual Improvement Loops

Technical architecture and implementation report

PUBLIC RELEASE · VERSION 1.0.0 · 2026-07-17

Governed Continual Improvement Loops

Publication metadata

Field	Value
Author	SalesFusion Research
Publication date	2026-07-17
Version	1.0.0
Canonical slug	governed-continual-improvement-loops
Canonical path	/technical-papers/governed-continual-improvement-loops
Publication class	Technical architecture and implementation report
Approval state	Founder-approved public release

Revenue Cortex connection

Revenue Cortex treats growth as a governed improvement loop, not a one-time automation build. Signals, campaigns, objections, conversion evidence, sales-call notes, pipeline outcomes, and delivery proof can all create hypotheses, but changes to positioning, workflows, offers, and outreach require versioned experiments, approval, rollback, and evidence review.

Evidence-status box

> This paper proposes a governed continual-improvement protocol for business AI systems. SalesFusion has implemented cadence components and design artifacts for review, reporting, learning, and change proposals, but the complete optimization controller described here has not been evaluated in a longitudinal field study. The allowed claim is protocol-level: business AI optimization should operate through versioned hypotheses, bounded experiments, approvals, outcome measurement, rollback, and regression monitoring. No claim is made that the protocol improves business outcomes, model quality, revenue, or efficiency.

Abstract

Business AI systems should learn from operation, but uncontrolled self-modification creates authority, safety, and measurement problems. A system that can change its own policies, permissions, claims, or consequential workflow behavior can optimize the wrong proxy, hide regression, or expand scope without human authority.

We propose a governed continual-improvement loop that separates telemetry, hypothesis generation, experiment design, approval, execution, evaluation, learning, and change activation. The loop requires baselines, success criteria, failure criteria, guardrails, stop conditions, rollback plans, and regression tests before an experiment starts. Completed experiments create learning records and change proposals; they do not directly promote themselves into production policy or workflow changes. The protocol is intended for business agents whose actions affect operating workflows and therefore require conservative evidence handling, explicit approval, and inspectable failure states.

The problem

Most automation systems degrade if they never adapt. Sources change, connectors drift, objectives shift, and workflow designs become stale. At the same time, a self-improving business agent can create a different failure class. It may optimize a visible local metric while harming a more important downstream metric. It may treat correlation as causation. It may run experiments without a baseline. It may continue after a guardrail breach because the local score still looks good. It may recommend removing the very control that would stop it. It may approve its own change because the optimization loop is wired too close to the execution layer.

The core issue is not whether learning should exist. It is whether learning is governed. A safe operating loop must preserve the difference between observation, hypothesis, experiment, result, learning record, change proposal, approval, deployment, rollback, and regression monitoring. If those states collapse into "the system learned and updated itself," the organization loses the ability to audit what changed and why.

Contributions

This paper contributes a protocol for governed continual improvement.

First, it defines experiment records with required baseline, hypothesis, intervention, success, failure, guardrail, stop, and rollback fields.

Second, it separates experiment approval from change approval. An experiment may be allowed without granting permission to activate a consequential workflow change.

Third, it specifies controls for proxy gaming, causal overreach, self-authorization, and regression.

Fourth, it defines implementation interfaces and state transitions for experiments, learning records, and change proposals.

Fifth, it describes an evaluation program for optimization integrity before business-outcome claims are made.

Definitions

Baseline is the pre-intervention state of a metric, workflow, population, or process during a defined time window.

Hypothesis is a testable statement about how an intervention is expected to affect defined metrics under stated conditions.

Experiment is a bounded change to a workflow, policy-adjacent behavior, ranking rule, prompt, interface, or process, run under predeclared criteria and authority.

Guardrail is a metric, rule, or condition that must not be harmed or breached, even if the primary metric moves favorably.

Stop condition is a predeclared rule that pauses or terminates an experiment.

Rollback is the controlled return to a prior approved version or state, with proof that the reversion occurred.

Learning record is a post-evaluation artifact that states what was observed, evidence strength, limits, confounders, and applicability.

Change proposal is a request to modify memory, workflow, policy, permission, or operating cadence. It requires approval appropriate to its consequence class.

Governed improvement protocol

The protocol has seven stages:

Stage	Primary object	Exit condition
observe	telemetry and outcome records	anomaly, opportunity, or failure pattern documented
hypothesize	hypothesis record	testable statement and scope written
design	experiment record	baseline, metrics, guardrails, stop, rollback complete
approve	approval record	authority approves exact experiment scope
run	experiment events	observations collected under versioned intervention
evaluate	learning record	result classified with limits and confounders
propose change	change proposal	separate approval path for durable change

The experiment record should include:

```
{
  "experiment_id": "exp_guardrail_review_001",
  "objective_refs": ["obj_operating_reliability"],
  "baseline_window": "defined_period",
  "hypothesis": "If intervention X is applied to scope Y, metric A will move without guardrail B
breach.",
  "intervention_version": "workflow_v4_candidate",
  "population_scope": "bounded_group",
  "success_criteria": ["primary_metric_threshold"],
  "failure_criteria": ["no_valid_measurement", "negative_primary_result"],
  "guardrails": ["quality_floor", "authority_error_zero", "complaint_threshold"],
  "stop_conditions": ["guardrail_breach", "missing_data", "connector_error"],
  "rollback_plan": "return_to_workflow_v3_and_verify_receipts",
  "approval_required": ["business_owner", "operator"],
  "self_authorization_allowed": false
}
```

The Improvement Controller may propose this record. It cannot approve it. It also cannot approve a later change proposal that modifies policy, permission, claim state, or consequential workflow behavior. This is a hard invariant.

Baselines, hypotheses, and experiment design

A baseline is not a casual memory of how things usually work. It must define time window, population, source, metric semantics, missing-data treatment, and known concurrent changes. If the baseline is weak, the experiment should be labeled exploratory and should not produce a strong conclusion.

Hypotheses should be narrow. A useful hypothesis states the expected direction of a primary metric, the affected scope, the time window, and the guardrails. "Make the workflow better" is not an experiment. "Change step order for this non-mutating analysis job and compare verified artifact completion under the same input fixture" is closer to a testable design.

Experiment design must record comparison logic. Depending on risk and feasibility, a team might use before/after comparison, holdout, canary, shadow mode, simulation, or expert scoring. The method should match consequence class. A read-only ranking change may be evaluated in simulation. A workflow that mutates an external system may require canary, explicit approval, and rollback proof before broader use.

Guardrails, stop conditions, rollback, and regression

Guardrails are intended to detect local optimization that violates system-level constraints. Examples include unauthorized-action rate, cross-tenant leakage rate, false-completion rate, complaint threshold, error budget, blocker classification quality, output restriction compliance, and downstream outcome preservation. A local metric gain with guardrail breach is failure by definition.

Stop conditions must be executable. They should specify the event, threshold, actor, and required response. A stop condition such as "if things look risky" is too vague. A better condition is: if an experiment produces any unauthorized external action, stop immediately, create a safety event, roll back the workflow version, and escalate to the authority owner.

Rollback must be verified, not assumed. A rollback record should include prior version, target version, affected scope, execution receipts, state reconciliation, unresolved side effects, and human review status. If rollback is unavailable or unsafe, that fact must be known before the experiment starts.

Regression monitoring continues after a change is approved. The system should run scenario benchmarks against prior objectives, guardrails, prompt-injection tests, authority tests, and model/version changes. OpenTelemetry concepts are useful for traces, metrics, and logs, but domain regression requires business-state records as well as infrastructure telemetry.

Preventing proxy gaming and self-authorization

Proxy gaming occurs when an experiment improves what is measured while harming what matters. The protocol addresses this with objective trees, anti-goals, guardrails, delayed-outcome review, and human evaluation of applicability limits. It also records negative, null, blocked, and failed experiments so the system does not learn only from attractive summaries.

Self-authorization is a separate risk. The Improvement Controller is allowed to:

Allowed	Not allowed
identify anomalies	approve its own experiment
propose hypotheses	change authority policy
draft experiment records	expand connector permissions
evaluate observations	activate consequential workflow version
create learning records	promote correlation to company fact
draft change proposals	delete guardrails or restrictions

This separation reflects the same governance posture used elsewhere in the Company Cortex: generated analysis may propose, but authority changes require human approval and auditable events.

Implementation

The protocol can be implemented with event-backed records and stable endpoints:

```
POST /v1/experiments
POST /v1/experiments/{experiment_id}/approve
POST /v1/experiments/{experiment_id}/start
POST /v1/experiments/{experiment_id}/observe
POST /v1/experiments/{experiment_id}/stop
POST /v1/experiments/{experiment_id}/evaluate
POST /v1/experiments/{experiment_id}/rollback
POST /v1/learning-records
POST /v1/change-proposals
```

The experiment state machine is:

```
draft -> evidence_needed -> approval_needed -> approved -> running
                                                 |-> paused -> running
                                                 |-> completed -> evaluated
                                                 |-> failed
                                                 |-> stopped_for_safety -> rollback
```

Events include experiment.proposed, experiment.approved, experiment.started, experiment.observation_recorded, experiment.guardrail_breached, experiment.stopped, experiment.completed, experiment.evaluated, experiment.rollback_started, experiment.rolled_back, learning_record.created, change_proposal.created, change_proposal.approved, change_proposal.rejected, and regression.detected.

SalesFusion has implemented cadence components in delivery design: weekly review of run health and blockers, monthly review of constraints and experiments, quarterly review of objectives and permissions, and learning fields in reporting. The complete controller still needs a reference implementation and longitudinal evaluation.

Evaluation

Optimization integrity should be evaluated before outcome effectiveness. Each experiment can be scored for pre-registered hypothesis, baseline validity, comparison design, intervention version, success and failure criteria, guardrail coverage, stop conditions, rollback feasibility, data completeness, confounder recording, conclusion strength, applicability limits, and approved change event.

Adversarial cases should include apparent local improvement with worse downstream outcomes, missing data, seasonality, changing population, delayed outcomes, model changes during the test, optimizer proposal to remove a guardrail, and a workflow that tries to approve its own permission expansion.

Acceptance criteria:

Criterion	Required behavior
baseline completeness	experiment cannot start without required baseline fields
guardrail breach	experiment stops and creates rollback/escalation events
self-authorization	optimizer cannot approve own policy or permission change
causal discipline	result language matches comparison strength
rollback proof	prior version restoration is verified
regression detection	approved change is monitored after activation

A later field evaluation would need multiple installations, visible baselines, time windows, confounders, negative results, and limitations. This paper does not provide that evidence.

Safety, risks, and failure modes

The main risks are proxy gaming, causal overreach, data corruption, hidden confounders, unsafe irreversible experiments, optimizer privilege escalation, rollback failure, regression after deployment, and selective reporting. NIST AI RMF and ISO/IEC 42001 support governance and management-system framing, but they do not validate this protocol's effectiveness.

The protocol fails closed where possible. Missing baseline blocks launch. Missing rollback blocks or escalates high-consequence experiments. Guardrail breach stops the run. Authority ambiguity escalates. Change activation requires a separate approval event. Result uncertainty is recorded rather than polished away.

Limitations

The protocol increases process cost. Small changes may feel slow when every experiment needs baseline, guardrails, and rollback. Some workflows lack clean comparison groups. Some outcomes are delayed, sparse, or affected by external events. Human reviewers can still approve bad experiments or misread results. The controller also depends on the quality of telemetry and outcome ledgers; corrupted metrics produce corrupted learning.

This paper does not claim that governed improvement loops outperform manual review or unguided optimization. It defines an inspection structure intended to make optimization risks reviewable and specifies the tests needed to evaluate it.

Practical implications

Teams should start with proposal-only optimization. Let the system identify blockers, draft hypotheses, and prepare experiment records. Require human approval for experiment launch and separate approval for durable change. Run early experiments in simulation or non-mutating mode where possible. Preserve failed and null results because they are part of the learning system.

The operating question should shift from "did the agent improve itself?" to "what changed, under which hypothesis, against which baseline, with which guardrails, and who approved the durable change?"

References used

This paper uses NIST AI Risk Management Framework for risk framing, ISO/IEC 42001:2023 for AI management-system context, OpenTelemetry Specification for observability concepts, and OWASP Top 10 for Large Language Model Applications for threat awareness. These references do not support SalesFusion-specific effectiveness claims.

Conclusion

Continual improvement is useful only if it remains governed. The Company Cortex improvement loop separates observation from hypothesis, experiment from activation, and learning from authority. SalesFusion has

implemented cadence components, while the full optimization controller remains a proposal requiring implementation and longitudinal evaluation. The central rule is non-negotiable: the optimizer may recommend and experiment under approval, but it cannot self-authorize consequential change.

Changelog

- 2026-07-17 — Version 1.0.0: founder-approved first public release; publication metadata, evidence-status box, limitations, and verified references retained.