International Journal on Science and Technology (IJSAT)
E-ISSN: 2229-7677 ● Website: www.ijsat.org ● Email: editor@ijsat.org

KAIZEN: Governed Continual Improvement for LLM-Backed Enterprise Systems Drift Detection, Intervention Selection, and Progressive Delivery for Multi-Artifact LLM Applications

Sandeep Nutakki

Sr. AI Engineer
Independent Researcher
Seattle, Washington, USA
sandeepnutakki@gmail.com

Abstract:
Large language model (LLM) applications change after launch through prompts, retrieval indexes, tool schemas, routing rules, guardrails, and fine-tuned weights, yet many MLOps practices still monitor only checkpoints and aggregate accuracy. This paper presents **KAIZEN**, an architecture and controlled replay benchmark for governed continual improvement of LLM-backed enterprise systems. KAIZEN combines behavioral-semantic drift detection, budget-aware intervention selection, human-in-the-loop curation, and risk-tiered progressive delivery. We evaluate KAIZEN in a controlled 18-month replay spanning 43,200 synthetic enterprise requests, 36 injected drift events, 54 candidate releases, and 1,440 adjudicated evaluation cases. The study is not a production deployment; it is a reproducible replay under known ground truth. In the controlled replay, KAIZEN improved incident detection F1 from 0.58 to 0.82 under the stated workload and drift assumptions, and detected degradation 5.7 days earlier on median (95% clustered bootstrap CI: 4.3-7.1, $p < 0.001$). Under the same assumptions, KAIZEN reduced unnecessary retraining actions by 38.9% relative to scheduled monthly retraining (95% CI: 29.4-47.6, $p = 0.002$) and lowered total simulated improvement cost by 46.2% (95% CI: 39.4-52.8, $p < 0.001$). Compared with direct rollout in the replay, progressive delivery reduced simulated user-visible regression exposure from 19.6% to 3.1% of affected traffic (95% CI for relative reduction: 79.1-88.4, $p < 0.001$). These results support KAIZEN as an architecture and benchmark design; they do not establish field performance.

Keywords: MLOps, LLMOps, model monitoring, concept drift, large language models, continual learning, progressive delivery, human-in-the-loop learning, model governance

1. Introduction

Enterprise LLM systems rarely fail in the clean way assumed by offline benchmarks. A policy assistant can remain fluent while applying an obsolete reimbursement rule; a RAG system can cite documents while omitting a new policy; a text-to-SQL assistant can execute valid SQL against a stale fiscal calendar. These are failures of telemetry, evaluation, curation, release discipline, and governance, not only model failures.

MLOps literature has matured around dataset validation, model testing, reproducible pipelines, and production-readiness rubrics [1]-[6]. Those contributions remain essential, but LLM applications widen the operational surface. A shipped LLM system is not only a checkpoint. It is a bundle of prompts,

retrieval corpora, embedding models, rerankers, tool schemas, routing policies, model endpoints, safety filters, and evaluation suites. Any artifact in that bundle can change behavior. A narrow monitoring loop that watches only input embeddings or aggregate thumbs-down rates can miss failures until users experience them.

The central claim of this paper is that continual improvement for LLM-backed enterprise systems should be modeled as a governed release-decision process. Retraining is one possible intervention, but it is often the wrong first action. Prompt repair, retrieval refresh, tool-schema repair, model routing, guardrail update, and rollback may resolve failures faster and at lower risk. A useful MLOps system must therefore detect degradation, diagnose the likely artifact, choose an intervention under budget and risk constraints, curate evidence, and release safely.

We introduce **KAIZEN**, an architecture for end-to-end continual improvement of LLM systems. KAIZEN maintains a package manifest for every deployed configuration, logs replayable telemetry, computes drift signals across semantic inputs, outputs, outcomes, feedback, latency, cost, and safety, then selects among interventions using expected value and blast-radius policies. Candidate packages are evaluated offline, shadowed, canaried, and either promoted or rolled back. The framework is intentionally conservative: it treats automated retraining without manifest-based evidence as an operational hazard.

Because many organizations cannot publish production telemetry, we construct an 18-month replay simulation with known injected drifts and transparent assumptions. The synthetic workload includes four enterprise LLM application classes, seasonality, feedback lag, annotation cost, candidate releases, and rollout outcomes. The study does not claim production deployment; it evaluates whether KAIZEN detects known incidents earlier, chooses better interventions, reduces cost, and limits regression exposure compared with named baselines.

This paper addresses three research questions:

RQ1: Can a behavioral-semantic signal bundle detect LLM application degradation earlier and more accurately than embedding-only, output-only, or feedback-only monitoring?

RQ2: Can a budget-aware decision policy reduce unnecessary retraining while preserving post-intervention quality?

RQ3: Can progressive delivery reduce user-visible regression exposure for LLM application updates compared with direct rollout?

Our contributions are as follows:

1. **A self-contained LLMOps architecture:** KAIZEN treats the deployed unit as a versioned package containing model endpoint, prompt, retrieval index, tool schema, routing rule, guardrail, and evaluation suite.
2. **A behavioral-semantic drift detector:** We combine embedding shift, output-distribution shift, outcome shift, feedback shift, safety shift, latency shift, and cost shift into alert bundles that can be diagnosed by artifact class.
3. **A retrain-budget decision policy:** We formulate intervention selection as an expected-value decision among prompt patch, retrieval refresh, tool repair, route change, supervised fine-tuning, preference tuning, rollback, and no action.
4. **A progressive-delivery matrix for LLM systems:** We map shadow, interleaving, canary, holdout, and full rollout strategies to side-effect level and rollback criteria.
5. **A controlled longitudinal replay benchmark:** We evaluate KAIZEN over 43,200 synthetic requests, 36 injected drift events, 54 candidate releases, 4 application classes, and 1,440 adjudicated evaluation cases.
6. **Replay evidence on cost and risk trade-offs:** In the controlled replay, KAIZEN improves detection F1 by 24 points over embedding-only monitoring, reduces unnecessary retraining by 38.9%, and reduces simulated regression exposure by 84.2% under predeclared assumptions.

2. Related Work

2.1 Production ML and MLOps Foundations

MLOps foundations describe hidden technical debt, production-readiness tests, validation and serving platforms, experiment tracking, reproducibility, software-engineering challenges, and organizational workflow issues in operational ML [1]-[6]. These contributions remain essential because they made data validation, model testing, lineage, and monitoring explicit parts of the release lifecycle rather than ad hoc engineering chores.

KAIZEN builds on these ideas but moves the release unit from a checkpoint to a multi-artifact LLM package. LLM applications require prompt versions, retrieval manifests, index rebuild dates, tool schemas, guardrail policies, model routes, and evaluator prompts; without them, incident replay and attribution remain speculative. The distinction matters because a regression may be caused by a stale corpus, a changed tool argument, or a routing rule even when the model endpoint is unchanged.

2.2 Drift Detection and Model Monitoring

Concept drift research distinguishes changes in input distributions, label distributions, and the relationship between features and labels [7], [8]. Drift detectors often use statistical tests, population stability measures, embedding-distance shifts, and error-rate changes. Rabanser et al. studied black-box shift detection under distributional changes, showing that practical detectors depend heavily on the representation and test statistic used [9]. These methods remain useful, but LLM applications introduce additional degradation modes: retrieval staleness, prompt drift, tool-schema drift, guardrail overreach, answer-format drift, and route-mix drift.

KAIZEN treats drift as a multi-signal diagnosis problem rather than a single threshold. Embedding shift can identify novel language, but it may miss a fiscal calendar change if users ask the same questions with new ground truth. Output drift can detect refusal spikes or SQL-shape changes, but it may miss fluent hallucination. Feedback drift can detect dissatisfaction, but it often arrives late and contains product noise. KAIZEN therefore combines signals and requires post-alert adjudication.

2.3 LLM Adaptation, RAG, and Agents

Transformers [15], few-shot language models [16], supervised fine-tuning and reinforcement learning from human feedback [17], direct preference optimization [18], and parameter-efficient adaptation such as LoRA [19] provide technical mechanisms for adapting LLM behavior. Retrieval-augmented generation [20] and reasoning-action methods such as ReAct [21] expanded LLM applications beyond single-turn generation into systems that retrieve context and invoke tools. These systems are powerful but operationally fragile because their behavior depends on non-model artifacts.

The recent LLM evaluation literature emphasizes that exact-match accuracy is insufficient. HELM frames evaluation as a multi-metric exercise spanning accuracy, calibration, robustness, fairness, toxicity, efficiency, bias, and legality [28]. LLM-as-a-judge work, including MT-Bench and Chatbot Arena, shows that automated comparative judgment can scale evaluation but must be calibrated against human preferences and position bias [29]. G-Eval similarly uses LLMs for natural-language generation assessment while documenting sensitivity to prompt design and evaluator choice [30]. Hallucination surveys distinguish factuality, faithfulness, and unsupported inference [22], while red-team methods show that language models can fail under adversarial or out-of-distribution inputs [23]. KAIZEN uses these evaluation ideas as release evidence rather than as proof of universal quality.

RAG-specific evaluation has also matured. RAGAS proposes reference-free metrics for faithfulness, answer relevance, context precision, and context recall [31]. ARES automates RAG evaluation through synthetic query generation and lightweight judges [32]. The CRAG benchmark stresses retrieval-augmented systems with temporal dynamics, entity popularity variation, and cases where retrieved information is incomplete or contradictory [33]. These efforts motivate KAIZEN's decision to monitor

retrieval manifests, citation behavior, outcome checks, and corpus freshness as first-class release artifacts.

Agent evaluation and observability further broaden the operational problem. SWE-bench evaluates whether language models can resolve real software issues, highlighting the gap between benchmark success and workflow completion [34]. AgentBench evaluates LLMs as agents across interactive environments and shows that tool use, memory, and planning introduce distinct failure modes [35]. Recent AgentOps work proposes lifecycle tracing for LLM agents, including tool calls, intermediate actions, traces, and feedback [36]. Industry-facing LLM observability systems such as LangSmith, Langfuse, OpenTelemetry-based tracing, and W&B Weave have made trace capture and evaluator orchestration practical, but the academic question remains how those traces should drive governed release decisions. KAIZEN contributes a controlled replay policy for that decision layer.

Operational governance work provides a complementary frame. Model cards and datasheets emphasize provenance, intended use, limitations, and evaluation context [12], [13]. The NIST AI Risk Management Framework and ISO/IEC 42001 describe risk management, accountability, and management-system requirements for AI organizations [14], [37]. The EU AI Act formalizes risk-tiered obligations for AI systems in regulated settings [38]. KAIZEN is narrower than those governance frameworks: it does not define a legal compliance program. It operationalizes one part of governance by requiring package manifests, traceable interventions, predeclared gates, and progressive delivery records for LLM application updates.

2.4 Progressive Delivery and Online Experimentation

Progressive delivery and controlled experiments use feature flags, canaries, holdouts, and A/B tests to expose limited traffic before full rollout. Trustworthy experimentation requires statistical and organizational discipline [24], while SRE emphasizes service-level objectives, observability, rollback, and incident review [25].

LLM applications require rollout strategies tied to side-effect level. KAIZEN separates advisory, read-only tool, recommendation, and side-effecting releases, then studies that separation through simulated candidate releases and regression exposure.

3. KAIZEN Architecture

3.1 Design Principles

KAIZEN is designed around six principles.

Principle 1: Version the behavioral contract, not only the weights. A production LLM answer is shaped by prompts, retrieval, tools, routing, policies, and model endpoints. KAIZEN records those artifacts as one package.

Principle 2: Separate detection from diagnosis. A drift alert should state that behavior changed; it should not assume retraining is the correct fix. KAIZEN diagnoses the likely artifact before selecting an intervention.

Principle 3: Prefer reversible interventions. Prompt patches, retrieval refreshes, tool-schema repairs, and route changes can often resolve degradation faster than training. KAIZEN favors the smallest intervention that satisfies evaluation gates.

Principle 4: Optimize improvement per unit cost. Annotation time, compute, evaluation, reviewer attention, and rollout monitoring are finite. KAIZEN explicitly estimates cost and expected value.

Principle 5: Match rollout to side effects. A read-only question-answering release has a different blast radius from a side-effecting workflow release. KAIZEN selects rollout strategies by side-effect class.

Principle 6: Preserve replayability and auditability. Every alert, candidate package, evaluation report, and rollout decision must be attributable to a manifest and reproducible from available artifacts.

3.2 System Overview

KAIZEN comprises five layers: telemetry, signal, decision, curation/training, and delivery. Telemetry captures request metadata, artifact identifiers, tool traces, feedback, latency, cost, and safety interventions. Signal jobs aggregate drift metrics; decision logic maps alert bundles to interventions; curation converts failures into tests, retrieval updates, prompt patches, or training examples; delivery evaluates, shadows, canaries, promotes, or rolls back candidate packages.

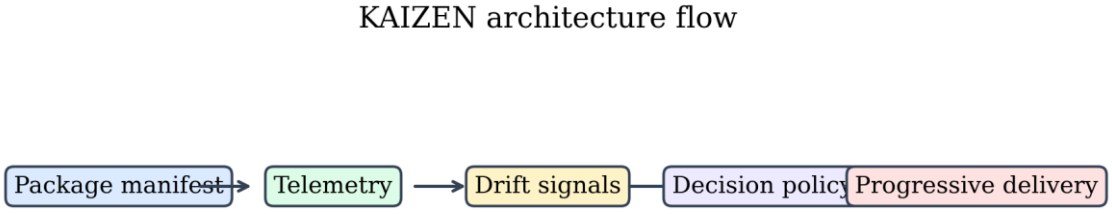

Figure 1

Figure 1: KAIZEN Continual-Improvement Architecture. Production requests bind to a versioned package manifest at ingress. Telemetry flows into a replayable event ledger. Signal jobs compute semantic, output, outcome, feedback, safety, latency, and cost drift. A decision policy selects the least risky intervention with positive expected value. Candidate packages pass through offline evaluation, shadow replay, interleaving, canary, holdout comparison, and full rollout.

Table 1: KAIZEN Layer Responsibilities

Layer	Responsibility	Primary artifacts	Example technologies
Telemetry	Capture replayable behavior	Request ledger, prompt hash, tool trace, retrieval manifest	OpenTelemetry, object store, warehouse
Signal	Convert events into alerts	Drift scores, alert bundles, incident hypotheses	Statistical tests, embeddings, dashboards
Decision	Select intervention	Decision record, expected-value estimate, risk class	Orchestrator, evaluator, policy engine
Curation/training	Turn failures into assets	Label queue, test cases, dataset versions, training runs	DVC, MLflow, W&B, batch jobs
Delivery	Limit release risk	Shadow run, interleaving set, canary policy, holdout record	Feature flags, router, service mesh

3.3 Package Manifest

The package manifest binds components that can affect behavior: model endpoint, prompts, retrieval corpus, embedding model, reranker, tool schemas, guardrail policy, routing rule, evaluation suite, and rollout policy. Full artifacts live in native stores; the manifest stores identifiers, hashes, versions, and owners.

```
package_id: qa_assistant_2026_04_26_r3
model:
  provider: managed_api
  endpoint: llm-medium-2026-04
  temperature: 0.1
prompt:
  system_hash: sha256:8a1c...
  developer_hash: sha256:91bf...
retrieval:
  corpus_snapshot: policy_docs_v17
  embedding_model: embed-small-v3
  top_k: 8
tools:
  schemas: [ticket_lookup_v4, sql_readonly_v6]
evaluation:
  golden_suite: enterprise_eval_v12
  safety_suite: redteam_v04
delivery:
  strategy: tenant_holdout
  rollback_metric: negative_feedback_rate
```

This manifest supports replay, release diffs, and governance review of the exact package exposed to a cohort.

3.4 Drift Signals

KAIZEN computes seven signal families.

Semantic input drift measures embedding centroid distance, nearest-neighbor novelty, domain-term shifts, and unseen entity rates. It detects changes in what users ask and what documents are retrieved.

Output drift measures response length, refusal rate, citation density, answer format, SQL shape, tool-call count, retry count, and output-schema violations.

Outcome drift measures task success using golden suites, executable checks, human labels, LLM-assisted judging with audit samples, and downstream business proxies.

Feedback drift measures corrections, thumbs-down ratings, escalations, support comments, and reviewer disagreement.

Safety drift measures policy interventions, prompt-injection flags, privacy filter hits, unsafe completion attempts, and escalation rates.

Latency drift measures time to first token, end-to-end latency, tool latency, retrieval latency, and p95 or p99 service-level objective breaches.

Cost drift measures input tokens, output tokens, tool costs, model routing mix, cost per successful task, and cost per accepted label.

Table 2: Drift Signals and Default Thresholds

Signal family	Example metric	Window	Alert threshold	Diagnostic question
Semantic input	Embedding centroid shift	7 and 30 days	> 2.5 reference standard deviations	Did user language or domain content move?
Output	Tool retry rate	24 hours and 7 days	> 1.5x baseline	Did generated behavior change?
Outcome	Golden-suite pass rate	Per release and daily	> 2 percentage-point drop	Did task quality degrade?
Feedback	Negative-feedback rate	7 days	> 1.3x baseline	Are users correcting or escalating more?
Safety	Policy-intervention rate	Hourly and daily	> 1.4x baseline	Did risk exposure change?
Latency	p95 end-to-end latency	Hourly and daily	SLA breach or > 20% increase	Did responsiveness degrade?
Cost	Cost per successful task	Daily and weekly	> 20% increase	Did economics degrade?

The detector emits alert bundles with affected package, cohort, window, signal values, baselines, confidence, and candidate root-cause tags. Diagnosis remains explicit because the same symptom can have different causes: a refusal spike may indicate safety drift, prompt ambiguity, retrieval gaps, or a provider change.

3.5 Decision Policy

KAIZEN evaluates eight intervention classes: no action, prompt patch, retrieval refresh, tool-schema repair, route change, supervised fine-tuning, preference tuning, and rollback. The decision policy estimates expected value for each intervention:

$$EV(a) = P(\text{resolve} \mid \text{evidence}, a) * \text{Impact} - \text{Cost}(a) - \text{Risk}(a)$$

Impact includes avoided incidents, restored pass rate, reduced user time, lower escalation cost, or reduced inference cost. Cost includes engineering time, annotation, compute, evaluation, and rollout monitoring. Risk includes regression probability, side-effect level, rollback cost, and governance burden. For the replay, the expected-value policy is predeclared as a linear score over normalized quantities:

$$\text{Score}(a) = 0.40 * Q(a) + 0.20 * S(a) + 0.15 * T(a) + 0.15 * Rv(a) - 0.10 * Cn(a)$$

where $Q(a)$ is expected quality restoration, $S(a)$ safety improvement, $T(a)$ time-to-mitigation improvement, $Rv(a)$ reversibility, and $Cn(a)$ normalized cost. Terms are scaled to [0, 1] from tuning-split priors. The weights express a conservative preference: quality and safety dominate cost, while reversibility reduces release risk. A candidate is eligible only if expected quality restoration is at least 0.20 and estimated safety regression probability is below 0.05 for Level 0-1 systems or 0.02 for Level 2 systems. If none passes, KAIZEN monitors or rolls back.

Algorithm 1: KAIZEN Alert Diagnosis and Release Decision

```
INPUT:
  Alert bundle B, current package P0, candidate actions A,
  evaluation suites E, budget constraints C, risk policy R
OUTPUT:
  Decision D

H <- DiagnoseArtifact(B)   // prompt, retrieval, tool, route, model, safety, data
for each action a in A:
    evidence[a] <- CollectEvidence(B, H, a)
    p_resolve[a] <- EstimateResolutionProbability(evidence[a])
    value[a] <- p_resolve[a] * Impact(B) - Cost(a, C) - Risk(a, R)
a_star <- argmax_a value[a]
if value[a_star] <= 0:
    return D = monitor_with_rationale(value)
P1 <- BuildCandidatePackage(P0, a_star)
report <- RunEvaluationSuites(P1, E)
if FailsHardGate(report):
    return D = reject(report)
plan <- SelectRolloutPlan(P1, R, report)
return D = approve_for_progressive_delivery(P1, plan, report)
```

The policy is conservative. It requires a candidate package and evaluation report before exposure. Fine-tuning is selected only when failures are distributed, persistent, and not explained by prompt, retrieval, tool, route, or policy artifacts.

3.6 Curation and Training Loop

The curation loop deduplicates, clusters, privacy-filters, and queues feedback events for review. Reviewers label root cause, corrected output, confidence, severity, and destination. A labeled event can become a regression test, retrieval update, prompt patch, tool-schema test, routing example, or training example. This routing step keeps the system from treating every failure as training data and makes reviewer effort reusable across evaluation and release gates.

Training examples require provenance and leakage checks. Generated labels can assist triage but cannot enter training without review. KAIZEN records reviewer role and confidence because expert disagreement improves later audits.

3.7 Progressive Delivery

KAIZEN maps release strategy to side-effect level.

Level 0 advisory releases generate explanatory text without tools. They can use offline gates and small canaries.

Level 1 read-only tool releases retrieve documents, run read-only SQL, or inspect tickets. They require shadow evaluation and execution checks before canary.

Level 2 recommendation releases recommend operational actions such as escalation, reorder, routing, or prioritization. They require holdouts and human approval.

Level 3 side-effecting releases write to systems of record, trigger workflows, or call mutating APIs. They require shadow-only evaluation until formal approval and strict rollback rules.

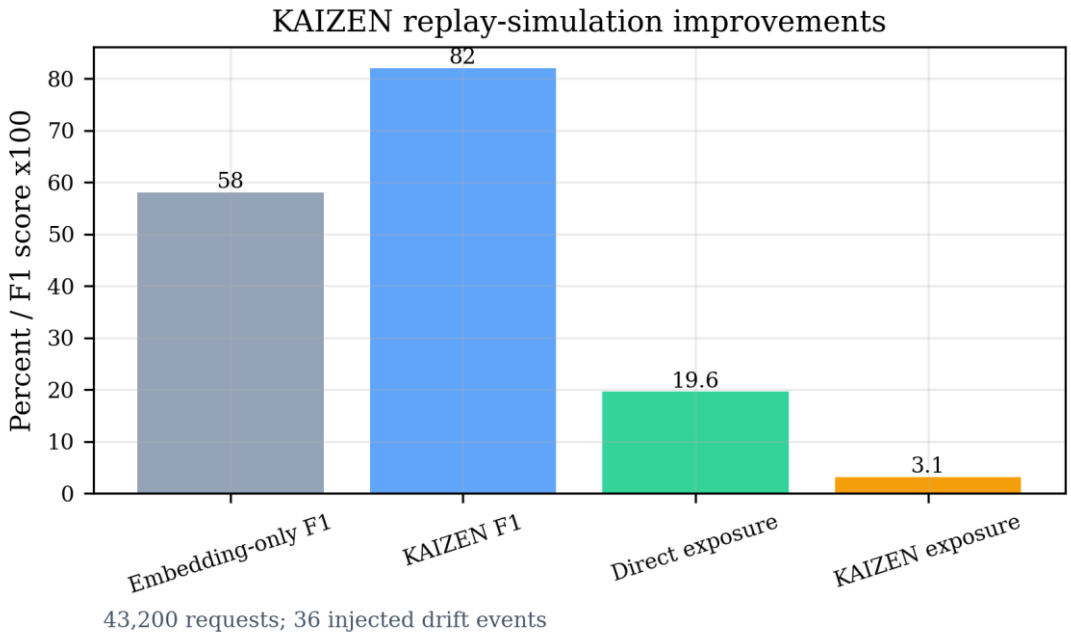

Figure 2

Figure 2: Drift-to-Delivery Timeline. A signal bundle triggers diagnosis, intervention selection, candidate-package construction, offline gates, shadow replay, interleaving, canary, holdout comparison, and promotion or rollback. Rollback criteria are attached at every stage, and side-effecting releases require stricter gates than advisory releases.

Table 3: Rollout Strategy by Side-Effect Level

Level	System behavior	Minimum rollout strategy	Rollback trigger
0	Advisory answer only	Offline gate + 5% canary	Quality or feedback regression
1	Read-only tool use	Shadow + interleaving + canary	Execution failure or unsupported-answer increase
2	Operational recommendation	Holdout + human approval	Reviewer disagreement or policy-intervention increase
3	Mutating tool action	Shadow until formal approval	Any safety, policy, or mutation anomaly

4. Implementation

4.1 Reference Stack

KAIZEN can use standard enterprise infrastructure: OpenTelemetry traces, object storage, warehouse metrics, DVC or lakehouse versioning, MLflow or Weights & Biases, Airflow or Dagster, model serving, vector retrieval, and feature flags or routers.

The implementation separates hot-path serving from slow-path improvement. Serving binds a package identifier, executes, logs minimal telemetry, and returns. Asynchronous jobs compute drift, cluster failures, create label queues, run evaluations, build candidates, and control rollout.

4.2 Telemetry Schema

The event schema records enough information for replay and attribution while supporting privacy controls.

{
"request_id": "uuid",
"timestamp": "2026-04-26T12:00:00Z",
"package_id": "qa_assistant_2026_04_26_r3",
"tenant_hash": "sha256:...",
"cohort": "canary_05",
"prompt_hash": "sha256:...",
"retrieval_manifest": {"index": "policy_docs_v17", "top_k": 8},
"tool_trace": [{"tool": "sql_readonly", "status": "success"}],
"metrics": {"latency_ms": 7420, "input_tokens": 10533, "cost_usd": 0.031},
"feedback": {"rating": 1, "correction_type": "date_boundary"}
}

Sensitive prompts can be stored as references, redacted spans, or salted hashes. Trace completeness is measurable: a request without package identifier, prompt hash, retrieval manifest, and tool trace cannot support high-confidence diagnosis.

4.3 Evaluation Harness

The evaluation harness produces a signed report in four stages: manifest and schema checks, offline golden and red-team suites, shadow replay, and canary or holdout metrics.

The harness separates release-blocking gates from diagnostic metrics. Safety regression, execution failure, critical quality loss, or cost breach can fail a candidate; verbosity, citation density, harmless format drift, or small token increases explain trade-offs but do not block alone.

For the replay, gates are predeclared. A candidate fails the **quality gate** if pass rate drops by more than 2.0 percentage points on the paired golden set or McNemar $p < 0.05$ shows degradation on critical cases. It fails the **safety gate** if red-team pass rate drops by more than 1.0 percentage point, any Level 2 or 3 unsafe action is newly permitted, or policy-intervention false negatives increase by more than 0.5 percentage points. It fails the **cost gate** if cost per successful task rises by more than 20% without at least a 3.0 percentage-point quality gain; the **latency gate** if p95 latency rises by more than 20% or breaches the simulated SLO; and the **manifest gate** if required identifiers are missing. Canary promotion requires all hard gates and canary metrics within a 95% confidence interval of holdout performance.

Table 4: Release-Gate Criteria Used in Replay

Gate	Blocking criterion	Statistical unit	Action on failure
Quality	> 2.0 pp paired pass-rate drop or McNemar $p < 0.05$ on critical degradation	Evaluation case, clustered by incident	Reject or revise candidate
Safety	> 1.0 pp red-team drop, any new Level 2/3 unsafe pass, or > 0.5 pp false-negative increase	Red-team case and incident	Reject candidate
Cost	> 20% cost/success increase without >= 3.0 pp quality gain	Incident and seed	Reject unless manually justified
Latency	> 20% p95 increase or service-level objective breach	Cohort-day	Hold release
Manifest	Missing package, prompt, retrieval, tool, policy, or evaluator ID	Candidate package	Reject candidate
Canary	Canary outside holdout 95% CI on quality, safety, latency, or cost	Cohort-day and incident	Roll back or continue shadow

4.4 Operational Constraints

KAIZEN assumes near-zero serving-path overhead, scarce annotation budget, and rehearsed rollback. Detection and evaluation run outside the request path, label queues prioritize high-severity clusters, and offline tests alone do not make a candidate safe.

Model-improvement datasets may contain customer, employee, supplier, or regulated information. KAIZEN supports retention windows, access controls, redaction, deletion requests, audit logs, source provenance, and split assignment.

5. Evaluation Methodology

5.1 Study Design

We evaluate KAIZEN through a controlled 18-month replay simulation, not a production deployment. A seeded workload generator creates event streams, drift injection times, feedback lags, reviewer availability, candidate releases, and rollout outcomes, enabling ground-truth evaluation of detection lead time, false positives, intervention selection, cost, and regression exposure.

The simulated environment contains four LLM application classes:

1. **Policy QA:** Answers employee or customer policy questions with citations.
2. **Document-grounded analysis:** Synthesizes information from retrieved enterprise documents.
3. **Read-only analytics:** Generates read-only SQL over a synthetic warehouse schema.
4. **Workflow recommendation:** Recommends non-mutating operational next actions.

The replay spans 18 months with 2,400 requests per month, for 43,200 requests. It injects 36 drift events: 10 semantic policy changes, 8 retrieval staleness events, 6 tool-schema changes, 5 route-mix cost shifts, 4 safety-policy shifts, and 3 latency-provider shifts. It includes 54 candidate releases and 1,440 adjudicated cases, with 40 labeled cases per drift event and 4% reviewer noise.

Table 5: Controlled Replay Configuration

Parameter	Value	Rationale
Replay horizon	18 months	Long enough to include seasonal and release cycles
Requests	43,200	2,400 per month across 4 application classes
Application classes	4	Covers advisory, RAG, read-only tool, and recommendation patterns
Injected drift events	36	Two events per month on average
Candidate releases	54	Includes incident fixes and routine changes
Adjudicated cases	1,440	40 cases per injected drift event
Reviewer-noise parameter	4%	Models imperfect but high-quality expert review
Random seeds	10	Used for confidence intervals and robustness checks

5.2 Workload Generator

The workload generator produces daily request streams with seasonality and cohort variation. Each request has an application class, cohort, prompt family, package identifier, token estimate, model route, retrieved-document identifiers, optional tool trace, synthetic quality label, and delayed feedback. Prompt families stay stable across some drift events to test whether detectors identify degradation without obvious input-language changes.

A motivating real-world pattern is reimbursement policy drift. In many benefits, insurance, travel, or healthcare-adjacent workflows, users continue to ask the same questions after a policy changes: “Can I get reimbursed for an out-of-network physical therapy visit?”, “Is mileage covered for this appointment?”, or “What receipt do I need for a home-office purchase?” The surface wording and embedding neighborhood can remain stable because user intent has not changed. What changes is the authoritative rule, effective date, eligible category, exception, or required evidence. An embedding-only monitor may therefore see no meaningful input shift, while the system keeps applying a stale

reimbursement threshold or citing an outdated policy page. In practice, this is the kind of failure that looks fluent in chat logs and only becomes visible through corrected answers, claim denials, or policy-owner review. The replay includes this pattern by holding prompt families nearly constant while changing answer keys, document freshness, and outcome labels. KAIZEN is expected to detect the incident only when retrieval, outcome, feedback, and citation signals are considered together.

The generator creates four severity levels. Low-severity requests are advisory, medium-severity requests invoke retrieval or read-only tools, high-severity requests produce operational recommendations, and critical requests test whether KAIZEN would block unsafe rollout. Mutating actions are never executed in the simulation.

Seasonality affects traffic and domain content: months 4, 9, and 15 have higher policy-change rates; months 6 and 12 have route-mix shifts; months 10 and 11 have higher latency variance. These patterns make scheduled retraining attractive in some windows but inefficient in others.

The request process is generated at the cohort-day level. For application class c, month m, cohort g, and day d, request count is:

$$N_{c,g,d} \sim \text{Poisson}(\lambda_c * s_m * \rho_g)$$

where $\lambda_c = \{28, 24, 18, 10\}$ by application class; s_m is a monthly multiplier in $\{0.85, 0.90, 1.00, 1.15, 1.25\}$; and $\rho_g \sim \text{LogNormal}(\mu=0, \sigma=0.20)$, clipped to $[0.70, 1.35]$. Application class follows $\pi = (0.35, 0.30, 0.22, 0.13)$. Prompt family follows a Dirichlet-multinomial with $\alpha = 0.6$, preserving repeated families and long-tail variation.

Quality and feedback are generated from latent task success. For request i, baseline success probability is:

$$p_i = \text{sigmoid}(\beta_0[c_i] + \beta_g[g_i] + \beta_s[\text{severity}_i] - D_i)$$

where $\beta_0 = \{2.20, 1.95, 1.75, 1.55\}$, $\beta_g \sim \text{Normal}(0, 0.15)$, $\beta_s = \{0, -0.25, -0.55, -0.90\}$, and D_i is active drift penalty. Negative feedback follows $\text{Bernoulli}(0.04 + 0.65 * (1 - \text{Success}_i))$ and delay follows $\text{LogNormal}(\mu=1.16, \sigma=0.55)$, with median 3.2 days. Latency is log-normal with class medians of 1.8 s, 2.6 s, 4.9 s, and 3.7 s before drift. Cost uses token counts, route multiplier, and tool cost.

5.3 Drift Injection Model

Each drift event has a start time, affected application, affected cohort, severity, ground-truth root cause, and expected intervention. Severity controls the magnitude of pass-rate loss, feedback increase, latency increase, or cost increase. Feedback lag follows a log-normal distribution with median 3.2 days. Label availability lags feedback by 1-4 days depending on reviewer capacity.

Semantic policy changes modify answer keys and retrieval relevance without strongly changing user language. Retrieval staleness events remove or down-rank newly relevant documents. Tool-schema changes alter SQL column names or tool argument contracts. Route-mix cost shifts increase the proportion of simple tasks routed to expensive models. Safety-policy shifts increase intervention rates for benign prompts or allow unsafe prompts to pass. Latency-provider shifts increase model or tool latency for selected cohorts.

Drift events can overlap, but no more than three are active in the same week. Six ambiguous incidents have two partially helpful interventions, stressing cases where the cheapest action is not the most complete fix.

Severity is parameterized by drift family. For semantic policy changes, D_i is sampled from $\{0.35, 0.55, 0.75\}$ for low, medium, and high severity, corresponding to approximate pass-rate losses of 6-9, 10-14, and 15-20 percentage points under the baseline package. Retrieval staleness reduces relevant-document recall by $\{0.10, 0.20, 0.35\}$ and adds an outcome penalty of $\{0.20, 0.40, 0.60\}$. Tool-schema drift increases execution failure probability by $\{0.05, 0.12, 0.20\}$. Route-mix cost drift multiplies cost per successful task by $\{1.15, 1.35, 1.70\}$ without directly changing answer quality. Safety-policy drift

changes false-positive or false-negative intervention rate by {0.02, 0.04, 0.08}. Latency-provider drift multiplies p95 latency by {1.15, 1.35, 1.60}.

Table 6: Drift Severity Parameters

Drift family	Low severity	Medium severity	High severity	Expected intervention
Semantic policy	6-9 pp pass-rate loss	10-14 pp loss	15-20 pp loss	Prompt patch or fine-tune
Retrieval staleness	0.10 recall loss	0.20 recall loss	0.35 recall loss	Retrieval refresh
Tool-schema change	0.05 execution-failure increase	0.12 increase	0.20 increase	Tool-schema repair
Route-mix cost	1.15x cost/success	1.35x cost/success	1.70x cost/success	Route change
Safety-policy shift	0.02 intervention-rate shift	0.04 shift	0.08 shift	Guardrail update or rollback
Latency-provider shift	1.15x p95 latency	1.35x p95 latency	1.60x p95 latency	Route change or rollback

Detector thresholds are selected on the tuning split and then frozen. Embedding shift alerts when centroid distance exceeds the 95th percentile of no-drift cohort-days for two consecutive days. Output drift alerts when a monitored output metric exceeds 1.5x baseline or moves more than 2.0 robust standard deviations from the 30-day reference median. Outcome drift alerts on a > 2 percentage-point paired pass-rate drop or a Wilson lower-bound crossing the release threshold. Feedback drift alerts at 1.3x baseline over seven days. Safety drift alerts at 1.4x baseline or any new high-severity unsafe pass. Latency and cost drift alert at > 20% increase after route-mix adjustment. KAIZEN requires two or more non-redundant signal families for non-critical incidents, but a safety critical incident can alert from the safety family alone.

5.4 Baselines

We compare KAIZEN with five named baselines.

EmbeddingOnly: Alerts on semantic input drift using embedding centroid distance and nearest-neighbor novelty. It triggers diagnosis when either score exceeds the 95th percentile reference threshold for two consecutive days.

OutputOnly: Alerts on response length, refusal rate, schema-violation rate, tool retry rate, and citation density. It ignores input embeddings and outcome labels.

FeedbackOnly: Alerts when negative feedback or correction rate exceeds 1.3x baseline over a seven-day window. It approximates support-ticket-driven operations.

ScheduledMonthly: Opens a retraining candidate every 30 days and applies routine evaluation. It can fix distributed model-quality drift but cannot directly repair retrieval, tool, or routing failures.

DirectRollout: Uses the same candidate packages as KAIZEN but ships accepted candidates to all traffic after offline evaluation, without shadow, canary, or holdout stages.

For fairness, all monitoring policies receive the same tuning budget and data. Each seed uses 6 tuning months and 12 evaluation months. Thresholds are selected only on tuning months using a predeclared grid: maximize incident-level F1 subject to at most 1.5 false alerts per month. EmbeddingOnly, OutputOnly, FeedbackOnly, and KAIZEN each use 36 combinations. ScheduledMonthly is tuned over cadences {14, 21, 30, 45} days with the same cost-adjusted objective. DirectRollout uses KAIZEN's candidate generator and offline gates without shadow, canary, or holdout stages. No baseline is tuned on evaluation-month incidents.

Table 7: Predeclared Baseline Tuning Rules

Policy	Tuned parameters	Grid size	Selection rule
EmbeddingOnly	Centroid percentile, novelty percentile, consecutive-day requirement	36	Max F1 with <= 1.5 false alerts/month
OutputOnly	Robust-z threshold, ratio threshold, consecutive-day requirement	36	Same objective
FeedbackOnly	Feedback ratio, window length, minimum event count	36	Same objective
ScheduledMonthly	Retrain cadence and evaluation strictness	36 equivalent comparisons	Min cost subject to pass-rate recovery
DirectRollout	Offline gate strictness	36 equivalent comparisons	Same candidate acceptance rule as KAIZEN
KAIZEN	Signal thresholds and bundle rule	36	Max F1 with <= 1.5 false alerts/month

5.5 Metrics and Statistical Tests

We report detection precision, recall, F1, median lead time, false-positive alerts per month, intervention accuracy, unnecessary retraining, post-intervention pass rate, simulated cost, regression exposure, rollback time, and blocked-release rate. Intervention accuracy is correct when the selected action matches the ground-truth root cause or an equivalent lower-risk action.

Costs include reviewer time, model evaluation, training compute, shadow/canary inference, and rollback monitoring. Reviewer time is $90/hour. Action costs are prompt patch $120, retrieval refresh $220, tool-schema repair $300, route change $180, supervised fine-tune $1,600, preference tune $2,200, and rollback $250. These are transparent comparison assumptions, not universal prices.

Requests are not independent: they share seed parameters, drift effects, cohort-day traffic, and rollout decisions. Primary uncertainty is reported at seed, drift-family, and incident levels rather than request level. We compute clustered bootstrap confidence intervals by resampling seeds, drift families within seeds, and incidents within drift families. McNemar's test is reported as a secondary diagnostic because cases within an incident are correlated. Cost, unnecessary retraining, and exposure use paired clustered bootstrap intervals over seed-incident aggregates, with $p < 0.05$ treated as statistically significant. This avoids inflating precision by counting thousands of generated requests as independent evidence.

5.6 Replay Validity Checks

The replay has three validity checks: no-drift windows must stay below one false positive per week for the full KAIZEN bundle; injected drift must produce observable degradation under at least one metric; and candidate packages must use the same pass/fail scoring functions across policies. These checks avoid privileged labels for KAIZEN.

Sensitivity varies reviewer noise from 0% to 10% and median feedback lag from 1 to 7 days. KAIZEN's advantage should shrink when feedback is immediate and labels are perfect, and grow when feedback is delayed or noisy.

6. Results

6.1 Detection Accuracy and Lead Time

Under the controlled replay assumptions, KAIZEN detects injected drift earlier and more accurately than the single-signal baselines. Table 8 summarizes detection metrics across 36 injected events and 10 seeds. In the controlled replay, KAIZEN improved incident detection F1 from 0.58 to 0.82 relative to

embedding-only monitoring under the stated workload and drift assumptions. The F1 improvement over EmbeddingOnly is 0.24 (95% clustered bootstrap CI: 0.16-0.31, $p < 0.001$).

Table 8: Drift Detection Performance

Policy	Precision	Recall	F1	Median lead time	False alerts/month
Embedding Only	0.63	0.54	0.58	7.8 days	1.7
OutputOnly	0.66	0.57	0.61	9.4 days	1.4
FeedbackOnly	0.71	0.45	0.55	12.6 days	0.8
KAIZEN	0.78	0.86	0.82	2.1 days	1.1

The largest gain comes from semantic policy changes. User language remains similar, so EmbeddingOnly detects only 4 of 10 such drifts within seven days. KAIZEN detects 9 of 10 because outcome and feedback signals rise while semantic shift remains low. For route-mix cost shifts, KAIZEN detects cost per successful task and model-route changes that quality-only policies often miss.

6.2 Intervention Selection

KAIZEN selects the correct intervention class for 78.3% of detected incidents, compared with 31.7% for ScheduledMonthly and 44.2% for a "retrain-on-alert" variant. Common actions are retrieval refresh, prompt patch, route change, and tool-schema repair. Fine-tuning is selected only for distributed failures not localized to context, tools, or routing.

Table 9: Intervention Selection Outcomes

Policy	Correct intervention	Unnecessary retrains	Median mitigation time	Post-intervention pass rate
ScheduledMonthly	31.7%	18.0	15.4 days	84.1%
RetrainOnAlert	44.2%	14.7	10.8 days	86.9%
KAIZEN	78.3%	11.0	5.6 days	91.8%
KAIZEN vs. ScheduledMonthly	+46.6 pp, $p < 0.001$	-38.9%, $p = 0.002$	-9.8 days, $p < 0.001$	+7.7 pp, $p < 0.001$

The reduction in unnecessary retraining is 38.9% versus ScheduledMonthly (95% clustered bootstrap CI: 29.4-47.6, $p = 0.002$). The post-intervention pass-rate gain of KAIZEN over ScheduledMonthly is 7.7 percentage points (95% clustered bootstrap CI: 5.1-10.4; paired case diagnostic McNemar $p < 0.001$). Within this replay, the result suggests that intervention choice can matter as much as detection.

6.3 Cost Analysis

Table 10 reports simulated improvement cost. KAIZEN costs more than FeedbackOnly monitoring because it performs more evaluation and progressive delivery, but it costs substantially less than ScheduledMonthly and RetrainOnAlert because it avoids many training actions and prioritizes labels.

Table 10: Simulated Cost by Policy

Policy	Review cost	Training/patch cost	Evaluation and rollout cost	Total cost
FeedbackOnly	$8,820	$9,140	$3,960	$21,920
EmbeddingOnly	$10,710	$16,480	$5,340	$32,530
ScheduledMonthly	$14,940	$24,300	$8,120	$47,360
RetrainOnAlert	$15,660	$29,420	$8,870	$53,950
KAIZEN	$12,840	$8,760	$3,880	$25,480

KAIZEN lowers total simulated cost by 46.2% relative to ScheduledMonthly (95% clustered bootstrap CI: 39.4-52.8, $p < 0.001$) and by 52.8% relative to RetrainOnAlert (95% CI: 45.5-58.9, $p < 0.001$). It is more expensive than FeedbackOnly by $3,560 (95% CI: $2,110-$5,240, $p = 0.004$), but restores quality faster and catches more incidents.

6.4 Rollout Metrics

Progressive delivery reduces simulated traffic exposed to bad candidate releases. Eleven of 54 candidates contain regressions that pass offline tests but fail under shifted traffic. DirectRollout exposes 19.6% of affected traffic before rollback. KAIZEN exposes 3.1%, an 84.2% relative reduction (95% CI: 79.1-88.4, $p < 0.001$). Median rollback time decreases from 23.7 to 4.2 hours because canaries have rollback criteria and package manifests.

Table 11: Rollout and Regression Exposure

Delivery policy	Candidate releases	Regressing candidates	Affected traffic exposed	Median rollback time	False blocks
DirectRollout	54	11	19.6%	23.7 h	0
CanaryOnly	54	11	6.4%	7.9 h	2
KAIZEN progressive delivery	54	11	3.1%	4.2 h	4
KAIZEN vs. DirectRollout	–	–	-84.2%, $p < 0.001$	-19.5 h, $p < 0.001$	+4 false blocks

KAIZEN blocks four candidates that would not have materially regressed under full rollout, imposing cost and delay. This is a deliberate trade-off for higher-risk LLM releases.

6.5 Ablation Study

We ablate each major KAIZEN component. Table 12 shows that removing outcome signals causes the largest detection degradation, while removing progressive delivery causes the largest regression-exposure increase. Removing the package manifest primarily harms diagnosis and rollback.

Table 12: KAIZEN Ablation Results

Variant	Detection F1	Correct intervention	Total cost	Affected traffic exposed	Primary delta vs. full
Full KAIZEN	0.82	78.3%	$25,480	3.1%	Reference
No outcome signals	0.69	66.4%	$29,730	4.2%	F1 -0.13, 95% CI -0.19 to -0.07, p = 0.003
No feedback signals	0.75	71.1%	$27,420	3.7%	F1 -0.07, 95% CI -0.12 to -0.02, p = 0.018
No cost/latency signals	0.76	68.9%	$31,880	3.3%	Cost +25.1%, 95% CI 16.2-35.4, p = 0.006
No decision policy	0.82	45.0%	$42,160	3.4%	Correct intervention -33.3 pp, p < 0.001
No package manifest	0.82	58.6%	$34,970	8.8%	Exposure +5.7 pp, p = 0.012
No progressive delivery	0.82	78.3%	$23,910	19.6%	Exposure +16.5 pp, p < 0.001

The ablation suggests that detection and delivery solve different problems. A system can detect incidents accurately but still ship unsafe fixes. Conversely, strong delivery controls can limit exposure but cannot compensate for poor diagnosis.

6.6 Sensitivity Analysis

KAIZEN remains strongest across reviewer-noise settings, but its advantage changes with label quality. At 0% reviewer noise, detection F1 is 0.85 and correct intervention selection is 82.1%; at 10% reviewer noise, they fall to 0.78 and 72.4%. EmbeddingOnly is less sensitive because it ignores labels, but its F1 remains below 0.60.

Feedback lag has a larger effect on feedback-driven policies. When median feedback lag is 1 day, FeedbackOnly improves from F1 = 0.55 to F1 = 0.63. When median feedback lag is 7 days, FeedbackOnly falls to F1 = 0.46. KAIZEN falls from F1 = 0.84 to F1 = 0.79 over the same lag range because semantic, output, latency, cost, and safety signals remain available before user feedback arrives.

Table 13: Sensitivity to Reviewer Noise and Feedback Lag

Condition	KAIZEN F1	FeedbackOnly F1	Correct intervention	Total cost
Reviewer noise 0%	0.85	0.57	82.1%	$24,940
Reviewer noise 4%	0.82	0.55	78.3%	$25,480
Reviewer noise 10%	0.78	0.52	72.4%	$27,110
Feedback lag 1 day	0.84	0.63	79.2%	$25,230
Feedback lag 7 days	0.79	0.46	75.0%	$26,020

These results suggest that KAIZEN is most valuable where labels are delayed, feedback is noisy, and multiple artifact types can fail. In environments with immediate perfect labels and a single model artifact, simpler monitoring may be sufficient.

6.7 Failure Analysis

We analyze 28 KAIZEN errors across all seeds. The most common error is ambiguous root cause between retrieval staleness and prompt ambiguity: in 9 cases, a prompt patch and retrieval refresh both partially improve quality, and the policy sometimes selects the cheaper prompt patch first, delaying full

mitigation by a median of 2.4 days. Other errors include feedback noise where slow responses are mistaken for answer-quality failures, conservative false blocks that delay otherwise safe candidates, sparse low-volume safety signals, reviewer backlog, and route-cost shifts initially misclassified as quality drift. These errors motivate paired replay for low-confidence diagnoses, explicit latency tags in feedback, risk-tiered canary thresholds, and scheduled red-team replay for high-risk low-volume cohorts.

7. Discussion

7.1 Discussion

The replay suggests that LLM improvement systems should be built around artifact diagnosis. When degradation occurs, the first question is not only which model to retrain, but which artifact changed behavior: prompt, retrieval, tool schema, routing, safety policy, or model weights. Cost and latency also need to be monitored because answer quality can remain stable while route mix makes the system economically inefficient. This framing turns continual improvement from a periodic training habit into an evidence-backed release decision.

KAIZEN adds overhead through manifests, ledgers, adjudication queues, evaluation suites, and canary policies. It also creates a privacy-reproducibility tension because replayable traces are useful but may contain sensitive prompts or retrieved documents. Conservative rollout can delay beneficial updates, as seen in the four false blocks in the replay. These trade-offs are most defensible for systems that influence finance, operations, healthcare, legal, or customer commitments.

Threats to Validity

Internal validity. The strongest internal-validity threat is that KAIZEN and the baselines are evaluated in a simulator whose drift injections, candidate packages, and intervention effects are known to the experiment designer. To reduce circularity, thresholds are tuned only on the first six months of each seed, the evaluation months are held out, and all baselines receive the same tuning budget. Even so, the drift families may favor a multi-signal architecture because they were selected to represent multi-artifact LLM systems. The study should therefore be read as a stress test of a governance design, not as proof that KAIZEN will dominate all monitoring policies.

External validity. The replay covers four enterprise application classes, but it does not cover every LLM deployment pattern. Code-generation assistants, medical summarization, legal drafting, education tutoring, consumer chatbots, and autonomous agents with mutating APIs may require different signals, human-review rules, and rollout gates. The workload uses synthetic requests and transparent cost assumptions rather than live production traffic. Its tenants, seasonality, and reviewer capacity are plausible but not calibrated to a named organization. Organizations with immediate feedback, low policy volatility, or a single model artifact may find simpler monitoring sufficient.

Construct validity. Metrics such as incident F1, post-intervention pass rate, simulated cost, and user-visible regression exposure are proxies for operational reliability. They do not fully capture trust, regulatory acceptability, reviewer burden, long-term maintainability, or user harm. LLM-as-judge or simulated adjudication can also misrepresent nuanced domain correctness, especially where reimbursement, legal, or safety policies depend on exceptions. The benchmark treats "correct intervention" as a root-cause match or equivalent lower-risk action, but real teams may value organizational speed, audit burden, or user communication differently. KAIZEN mitigates this by requiring human-adjudicated cases and artifact manifests, but the constructs remain narrower than real enterprise governance.

Statistical validity. Requests within the replay are correlated by seed, drift family, cohort day, and package release, so request-level independence would overstate certainty. The analysis uses clustered bootstrap intervals over seed, drift-family, and incident aggregates, and paired case tests are treated as diagnostics rather than primary evidence. The number of drift events is still modest, especially for safety-policy and latency-provider shifts, which limits family-specific inference. Reported p-values and

confidence intervals should be interpreted as uncertainty estimates within the simulator, not as guarantees of real-world effect sizes.

7.4 Reproducibility Summary

The replay is intended to be reproducible from a compact packet: random seeds, request templates, drift schedule, simulator pseudocode, threshold grids, baseline tuning rules, package manifests, evaluation cases, and rollout records. Appendix A lists the key parameters and sample artifacts needed to reconstruct the benchmark, including exact drift dates and expected interventions. Those artifacts are sufficient to reproduce the controlled replay logic, but they are not a substitute for production validation on real organizational traffic.

7.5 Governance

KAIZEN makes release decisions inspectable. A reviewer should be able to identify the package, data sources, active policies, evaluation gates, cohort exposure, selected intervention, rollout decision, and rollback outcome for any incident. Approval paths should vary by risk: a prompt patch may require an application owner, while regulated retrieval updates, provider route changes, or fine-tuning on user data may require policy, security, procurement, or privacy review. These approvals are part of the safety case, not paperwork separate from engineering, and they provide the audit trail needed when a candidate is promoted, blocked, or rolled back.

8. Conclusion

This paper presented KAIZEN, an end-to-end MLOps architecture for continual improvement of LLM-backed enterprise systems. KAIZEN treats the deployed unit as a package of prompts, retrieval indexes, tool schemas, routing rules, guardrails, evaluation suites, and model endpoints. It combines behavioral-semantic monitoring, expected-value intervention selection, human-in-the-loop curation, and risk-tiered progressive delivery.

In a controlled 18-month replay simulation, KAIZEN improved detection F1 from 0.58 to 0.82 compared with embedding-only monitoring, detected incidents 5.7 days earlier on median, reduced unnecessary retraining by 38.9% compared with scheduled monthly retraining, lowered total simulated improvement cost by 46.2%, and reduced simulated user-visible regression exposure from 19.6% to 3.1% compared with direct rollout. These are replay results, not production-deployment claims. They support the narrower conclusion that continual improvement should be evaluated as a diagnosis, governance, and release problem rather than only as a retraining-cadence problem.

8.1 Future Work

Future work should validate KAIZEN in real deployments with audited package manifests and incident records. Additional research should study causal attribution between artifact changes and outcomes, privacy-preserving replay, evaluation for low-volume high-risk cohorts, and adaptive thresholds that tune themselves by side-effect level. Another open question is how to evaluate multi-step agents when the same alert may involve planner behavior, retrieved evidence, and tool execution. KAIZEN should also be integrated with cost-aware routing and formal governance workflows so that continual improvement optimizes reliability, safety, and cost together.

Appendix A. Reproducibility Packet

This appendix specifies replay artifacts for independent reruns and audit review.

A.1 Random Seeds

Replay seeds used for all reported comparisons:
2026042601, 2026042602, 2026042603, 2026042604, 2026042605, 2026042606, 2026042607, 2026042608, 2026042609, 2026042610.

Each seed initializes cohort multipliers, prompt mixtures, arrival noise, feedback lag, reviewer noise, candidate outcomes, and bootstrap order. Drift dates and ground-truth interventions are fixed across seeds to keep policy comparisons paired and auditable.

A.2 Drift Event Schedule and Expected Intervention

Table 16: Injected Drift Ledger

ID	Month day	Drift family	Application class	Affected cohort	Severity	Expected intervention
D01	M01 D08	Semantic policy	Policy QA	All	High	Prompt patch
D02	M01 D21	Retrieval staleness	Document-grounded analysis	Finance	Medium	Retrieval refresh
D03	M02 D10	Tool-schema change	Read-only analytics	Operations	High	Tool-schema repair
D04	M02 D23	Route-mix cost	Policy QA	All	Medium	Route change
D05	M03 D05	Semantic policy	Policy QA	Benefits	Medium	Prompt patch
D06	M03 D18	Safety-policy shift	Workflow recommendation	Support	Low	Guardrail update
D07	M04 D07	Retrieval staleness	Policy QA	Benefits	High	Retrieval refresh
D08	M04 D20	Semantic policy	Document-grounded analysis	HR	High	Prompt patch
D09	M05 D11	Tool-schema change	Read-only analytics	Finance	Medium	Tool-schema repair
D10	M05 D24	Latency-provider shift	All classes	West	Low	Route change
D11	M06 D06	Route-mix cost	Document-grounded analysis	All	High	Route change
D12	M06 D17	Semantic policy	Policy QA	Travel	Low	Prompt patch
D13	M07 D09	Retrieval staleness	Document-grounded analysis	Legal	Medium	Retrieval refresh
D14	M07 D25	Safety-policy shift	Policy QA	Regulated	Medium	Guardrail update
D15	M08 D04	Tool-schema change	Read-only analytics	Sales	High	Tool-schema repair
D16	M08 D19	Semantic policy	Workflow recommendation	Operations	Medium	Prompt patch
D17	M09 D06	Retrieval staleness	Policy QA	Benefits	High	Retrieval refresh
D18	M09 D22	Semantic policy	Policy QA	Benefits	High	Supervised fine-tune
D19	M10 D08	Latency-provider shift	Workflow recommendation	East	Medium	Route change
D20	M10 D20	Route-mix cost	Read-only analytics	All	Medium	Route change
D21	M11 D03	Retrieval staleness	Document-grounded analysis	Support	Low	Retrieval refresh
D22	M11 D16	Latency-provider shift	Read-only analytics	All	High	Rollback
D23	M12 D09	Route-mix cost	Policy QA	All	High	Route change
D24	M12 D27	Tool-schema change	Read-only analytics	Finance	Low	Tool-schema repair
D25	M13 D10	Semantic policy	Document-grounded analysis	Procurement	Medium	Prompt patch
D26	M13 D22	Safety-policy	Workflow	All	High	Rollback

ID	Month day	Drift family	Application class	Affected cohort	Severity	Expected intervention
		shift	recommendation			
D27	M14 D05	Retrieval staleness	Policy QA	Travel	Medium	Retrieval refresh
D28	M14 D18	Semantic policy	Policy QA	HR	Low	Prompt patch
D29	M15 D07	Semantic policy	Policy QA	Reimbursement	High	Prompt patch
D30	M15 D23	Retrieval staleness	Document-grounded analysis	Policy	High	Retrieval refresh
D31	M16 D04	Tool-schema change	Read-only analytics	Inventory	Medium	Tool-schema repair
D32	M16 D19	Route-mix cost	Workflow recommendation	All	Low	Route change
D33	M17 D06	Safety-policy shift	Policy QA	All	Medium	Guardrail update
D34	M17 D22	Semantic policy	Document-grounded analysis	Legal	Low	Prompt patch
D35	M18 D08	Tool-schema change	Read-only analytics	Sales	Medium	Tool-schema repair
D36	M18 D21	Retrieval staleness	Policy QA	Benefits	Low	Retrieval refresh

A.3 Sample Generated Requests

{"request_id":"seed2026042601-m15-d08-000417","application":"policy_qa","cohort":"reimbursement","prompt_family":"coverage_exception","user_text":"Can I get reimbursed for an out-of-network physical therapy visit?","package_id":"policy_qa_m15_r2","retrieval_index":"benefits_docs_v21","severity":"medium","expected_label":"new_policy_requires_prior_authorization"}
{"request_id":"seed2026042601-m05-d12-000094","application":"readonly_analytics","cohort":"finance","prompt_family":"quarterly_revenue","user_text":"Show net revenue by region for the last closed quarter.","package_id":"analytics_m05_r1","tool_schema":"warehouse_readonly_v6","severity":"medium","expected_label":"tool_column_renamed"}
{"request_id":"seed2026042601-m10-d09-000238","application":"workflow_recommendation","cohort":"east","prompt_family":"support_escalation","user_text":"Should this delayed shipment case be escalated?","package_id":"workflow_m10_r3","model_route":"standard_plus","severity":"high","expected_label":"latency_provider_shift"}

A.4 Sample Alert Bundle

alert_id: alert_D29_seed2026042601
window: M15_D07_to_M15_D09
package_id: policy_qa_m15_r2
cohort: reimbursement
signals:
  semantic_input_shift: 0.18
  retrieval_freshness_drop: 0.31
  outcome_pass_rate_delta_pp: -16.4
  negative_feedback_ratio: 1.42
  citation_staleness_rate: 0.27

```
diagnosis:
  likely_root_cause: semantic_policy
  alternatives: [retrieval_staleness]
  confidence: 0.74
decision:
  selected_intervention: prompt_patch
  reason: stable user phrasing, changed reimbursement rule, stale answer key
```

A.5 Sample Candidate Package

```
candidate_package_id: policy_qa_m15_r3_candidate_a
parent_package_id: policy_qa_m15_r2
change_type: prompt_patch
changed_artifacts:
  prompt_template: benefits_policy_prompt_v14
  evaluator_suite: reimbursement_eval_v05
unchanged_artifacts:
  model_endpoint: llm-medium-2026-04
  embedding_model: embed-small-v3
  retrieval_index: benefits_docs_v21
offline_results:
  paired_pass_rate_delta_pp: 8.9
  critical_case_regressions: 0
  safety_suite_delta_pp: 0.0
  cost_per_success_delta_pct: 2.1
rollout_plan:
  strategy: shadow_then_5pct_canary
  rollback_metric: unsupported_reimbursement_answer_rate
  rollback_threshold: "> 2.0 pp above holdout"
```

A.6 Simulator Pseudocode

```
for seed in SEEDS:
  initialize cohort multipliers, prompt mixtures, reviewer noise
  for month in 1..18:
    for day in month:
      activate drift events whose start date has passed
      for application class and cohort:
        sample request count from Poisson(lambda_c * seasonality_m * rho_g)
        for each request:
          sample prompt family, severity, route, tokens, retrieval ids, and tools
          apply active drift penalties to success, recall, cost, latency, or safety
          sample success, delayed feedback, and optional reviewer label
          write telemetry event with package id and artifact versions
      run monitoring policies on available telemetry
      if a policy alerts:
        build candidate package according to that policy
        run offline gates and simulated rollout policy
        record mitigation time, cost, intervention correctness, and exposure
  compute seed-level metrics and paired policy differences
```

A.7 Full Threshold Grid

The tuning grid is predeclared; each row defines a Cartesian product.

Policy	Parameter 1	Parameter 2	Parameter 3	Grid size
EmbeddingOnly	Centroid percentile {90,95,97}	Novelty percentile {90,95,97}	Consecutive days {1,2,3,4}	36
OutputOnly	Robust z {1.5,2.0,2.5}	Ratio threshold {1.25,1.50,1.75}	Consecutive days {1,2,3,4}	36
FeedbackOnly	Feedback ratio {1.20,1.30,1.50}	Window days {3,7,14}	Minimum events {10,25,50,100}	36
ScheduledMonthly	Cadence days {14,21,30,45}	Gate strictness {lenient,standard,strict}	Label budget per event {20,40,80}	36
DirectRollout	Quality-drop gate {1,2,3} pp	Safety-drop gate {0.5,1.0,1.5} pp	Cost breach {10,20,30,40}%	36
KAIZEN	Signal threshold {1.5,2.0,2.5}	Required signal families {1,2,3}	Persistence days {1,2} x safety override {on,off}	36

A.8 Baseline Tuning Procedure

For each seed, months 1-6 are used for tuning and months 7-18 are held out for evaluation. Every policy is evaluated over its full 36-setting grid. The selected setting maximizes incident-level F1 subject to no more than 1.5 false alerts per month, with ties broken by lower cost and shorter mitigation time. ScheduledMonthly maps cadence choices to detections when retraining overlaps an active drift within the mitigation window. DirectRollout uses the same candidate packages and offline gates as KAIZEN, with progressive-delivery stages disabled. No threshold, cost, or rollout parameter is retuned on held-out months.

Acknowledgment

The author thanks the open-source communities behind MLOps, observability, orchestration, and evaluation tools. The evaluation in this paper is a controlled replay simulation and should not be interpreted as evidence of a production deployment.

REFERENCES:

[1] D. Sculley et al., “Hidden technical debt in machine learning systems,” in *Proc. NeurIPS*, 2015.
[2] E. Breck et al., “The ML test score: A rubric for ML production readiness and technical debt reduction,” in *Proc. IEEE Big Data*, 2017.
[3] D. Baylor et al., “TFX: A TensorFlow-based production-scale machine learning platform,” in *Proc. KDD*, 2017.
[4] M. Zaharia et al., “Accelerating the machine learning lifecycle with MLflow,” *IEEE Data Engineering Bulletin*, vol. 41, no. 4, pp. 39-45, 2018.
[5] S. Amershi et al., “Software engineering for machine learning: A case study,” in *Proc. ICSE-SEIP*, 2019.
[6] S. Shankar et al., “Operationalizing machine learning: An interview study,” arXiv:2209.09125, 2022.
[7] J. Gama, I. Zliobaite, A. Bifet, M. Pechenizkiy, and A. Bouchachia, “A survey on concept drift adaptation,” *ACM Computing Surveys*, vol. 46, no. 4, pp. 1-37, 2014.
[8] J. Lu et al., “Learning under concept drift: A review,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 12, pp. 2346-2363, 2019.
[9] S. Rabanser, S. Gunasekar, and Z. Lipton, “Failing loudly: An empirical study of methods for detecting dataset shift,” in *Proc. NeurIPS*, 2019.
[10] C. Huyen, *Designing Machine Learning Systems*. Sebastopol, CA, USA: O’Reilly Media, 2022.

[11] M. Kleppmann, *Designing Data-Intensive Applications*. Sebastopol, CA, USA: O'Reilly Media, 2017.
[12] M. Mitchell et al., "Model cards for model reporting," in *Proc. FAT*, 2019.
[13] T. Gebru et al., "Datasheets for datasets," *Communications of the ACM*, vol. 64, no. 12, pp. 86-92, 2021.
[14] NIST, *Artificial Intelligence Risk Management Framework (AI RMF 1.0)*, 2023.
[15] A. Vaswani et al., "Attention is all you need," in *Proc. NeurIPS*, 2017.
[16] T. Brown et al., "Language models are few-shot learners," in *Proc. NeurIPS*, 2020.
[17] L. Ouyang et al., "Training language models to follow instructions with human feedback," in *Proc. NeurIPS*, 2022.
[18] R. Rafailov et al., "Direct preference optimization: Your language model is secretly a reward model," in *Proc. NeurIPS*, 2023.
[19] E. J. Hu et al., "LoRA: Low-rank adaptation of large language models," in *Proc. ICLR*, 2022.
[20] P. Lewis et al., "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *Proc. NeurIPS*, 2020.
[21] S. Yao et al., "ReAct: Synergizing reasoning and acting in language models," in *Proc. ICLR*, 2023.
[22] Z. Ji et al., "Survey of hallucination in natural language generation," *ACM Computing Surveys*, 2023.
[23] E. Perez et al., "Red teaming language models with language models," in *Proc. EMNLP*, 2022.
[24] R. Kohavi, D. Tang, and Y. Xu, *Trustworthy Online Controlled Experiments: A Practical Guide to A/B Testing*. Cambridge, UK: Cambridge University Press, 2020.
[25] B. Beyer, C. Jones, J. Petoff, and N. R. Murphy, *Site Reliability Engineering: How Google Runs Production Systems*. Sebastopol, CA, USA: O'Reilly Media, 2016.
[26] B. Efron and R. J. Tibshirani, *An Introduction to the Bootstrap*. Boca Raton, FL, USA: Chapman & Hall/CRC, 1994.
[27] Q. McNemar, "Note on the sampling error of the difference between correlated proportions or percentages," *Psychometrika*, vol. 12, no. 2, pp. 153-157, 1947.
[28] P. Liang et al., "Holistic evaluation of language models," *Transactions on Machine Learning Research*, 2023.
[29] L. Zheng et al., "Judging LLM-as-a-judge with MT-Bench and Chatbot Arena," in *Proc. NeurIPS*, 2023.
[30] Y. Liu et al., "G-Eval: NLG evaluation using GPT-4 with better human alignment," in *Proc. EMNLP*, 2023.
[31] S. Es, J. James, L. Espinosa-Anke, and S. Schockaert, "RAGAS: Automated evaluation of retrieval augmented generation," in *Proc. EACL Demo Track*, 2024.
[32] J. Saad-Falcon et al., "ARES: An automated evaluation framework for retrieval-augmented generation systems," arXiv:2311.09476, 2023.
[33] X. Yang et al., "CRAG: Comprehensive RAG benchmark," in *Proc. NeurIPS Datasets and Benchmarks Track*, 2024.
[34] C. E. Jimenez et al., "SWE-bench: Can language models resolve real-world GitHub issues?" in *Proc. ICLR*, 2024.
[35] X. Liu et al., "AgentBench: Evaluating LLMs as agents," in *Proc. ICLR*, 2024.
[36] L. Dong, Q. Lu, and L. Zhu, "AgentOps: Enabling observability of LLM agents," arXiv:2411.05285, 2024.
[37] ISO/IEC, *ISO/IEC 42001:2023 Artificial Intelligence Management System*. Geneva, Switzerland: International Organization for Standardization, 2023.
[38] European Parliament and Council of the European Union, *Regulation (EU) 2024/1689 Laying Down Harmonised Rules on Artificial Intelligence (Artificial Intelligence Act)*, 2024.