International Scientific Journal of Engineering and Management (ISJEM) ISSN: 2583-6129
Volume: 05 Issue: 01 | Jan – 2026 DOI: 10.55041/ISJEM05351
An International Scholarly || Multidisciplinary || Open Access || Indexing in all major Database & Metadata

Protecting Revenue at Scale: Pre-Detecting Anomalies in High Velocity ECommerce Systems using Artificial Intelligence (AI)

Priyadarshini Jayakumar
Dennis Chan

Abstract

This paper presents a production oriented AIOps (Artificial Intelligence Operations) approach that advances conventional observability into pre-detecting anomalies early enough to prevent customer impact for large-scale digital commerce. Grounded in an Intelligent Traffic Routing program integrated with AI (Artificial Intelligence) and Automated Ops, the approach combines multi-layer health monitoring, two stage thresholding, time-aware seasonal baselines, cloud based cross stack delta localization, event aware reasoning, and confidence weighted recommendations with human in the loop, adding human approvals for high blast radius actions.

Index Terms

AIOps, anomaly pre detection, dynamic thresholds, time-series monitoring, cross stack delta analysis, incident triage, traffic routing, SRE, governance, self-healing.

I. Introduction

Digital commerce platforms are socio-technical systems: customer behavior, marketing events, and partner ecosystems interact with distributed software and cloud infrastructure. Reliability engineering in this setting must maintain low latency, correctness, and availability under volatile and asymmetric loads. Incidents frequently emerge from subtle precursors like pod restart patterns, queue aging, rising database CPU, jitter in p99 latency, or channel-specific order dips, well before any single metric crosses a hard threshold.

Traditional observability answers "what is happening now?" but often fails to answer, "what is likely to happen next?" in time to prevent customer impact. Operators compensate by adding more alerts and lower thresholds, increasing noise. AIOps addresses this by introducing automated correlation, decision workflows, and selective GenAI assistance to reduce the cognitive load of incident triage and accelerate response.

Within a Digital Commerce Platform, the Intelligent Traffic Routing (ITR) automation monitors live stack metrics, calculating deltas, and provides traffic flip recommendations with configurable thresholds and team notifications. The operational objective is explicit: recommendations must be fast actionable in under ten minutes while remaining safe and auditable via human approvals.

This paper centers on moving from observing to pre-detecting anomalies: detect deviations from time-aligned normal behavior, validate persistence and localization, correlate with live events that explain benign anomalies, and recommend mitigations with quantified confidence and explicit guardrails.

II. Background and Motivation

A. From Monitoring to Control Systems

Traffic routing in active-active and blue/green deployments can be viewed as a control system. Inputs are telemetry and business signals, the controller is the decision engine, and outputs are routing weights or mitigation toggles. The ITR program operationalizes this perspective by treating routing decisions as guardrailed actions driven by multi-signal health gates.

B. Why Static Thresholds Fail at Scale

Commerce traffic is seasonal and event driven. Static thresholds oscillate between being too sensitive (noise) and too lax (late detection). Time-aware baselines aligned to hour-of-day, and day-of-week are required to detect drift early.

C. Production Drivers from Program Review

Roadmap reviews emphasized: keep the framework lean, simplify and harden traffic flips, and expand coverage from infrastructure signals to full API monitoring and business anomalies. Integration with operational tools and incident/change correlation (e.g., ServiceNow) was highlighted to reduce manual forensics.

D. Preventing 'Overly Clever' Automation

A recurrent risk in AIOps is decision logic sprawl. Guidance explicitly cautions against brittle if/else chains, advocating global rules and independent signals that scale.

E. Where GenAI Helps and Where it does not

Program discussions clarify a deliberate boundary: the backbone is rule-based automation (logic trees, preset conditions), while GenAI is used for selective inference such as summarizing results, configuring rules, or retrieving similar historical procedures. High blast actions remain human approved. This is a pragmatic AIOps stance: use AI to accelerate operators, not to remove accountability.

F. Segmented Routing and Progressive Delivery

Segmented routing assigns cohorts by geography, device, channel, behavior, or time to different stacks or regions. In practice, segmentation enables several reliability capabilities limiting blast radius of defective releases, isolating downstream dependencies, performing safe A/B experiments and canary rollouts, enabling rapid rollback by shifting only affected cohorts.

From an AIOps standpoint, segmentation also improves anomaly interpretation. When a cohort is explicitly defined, anomalies can be measured relative to that cohort's normal behavior. For example, a spike in payment failure that occurs only for a cohort routed to one stack is much easier to localize than a spike observed globally.
The ITR describes pipeline driven provisioning patterns where stacks can be created, validated under load, and torn down. This supports faster recovery and safer change validation.

Ephemeral stacks also complement pre-detection: when early warning signs appear, the system can recommend creating a fresh standby stack in parallel, reducing risk if the current stack begins degrading.

G. Toward Predictive and Proactive Operations

Pre-detection is a stepping stone to proactive operations. Once aligned baselines and event-aware models are in place, the same data can support forecasting: predict expected order volume, API rate, and dependency load for upcoming hours. Forecasting enables two practical proactive controls: capacity preparation and traffic pre-biasing.

Capacity preparation means validating that the target stacks are warmed and have headroom before known peaks. This includes DB warm-ups, pod readiness, cache priming, and search index readiness. Traffic pre-biasing means adjusting routing weights ahead of a surge to reduce the probability of saturating a single stack.

These capabilities should be introduced conservatively: forecasts should influence recommendations like considerations of shifting 10% preemptively rather than automatically changing traffic. Over time, as forecasts are validated against outcomes, the system can expand the set of safe proactive actions.

Critically, forecasting must incorporate business calendars, events, launches, promotions, and must separate expected surge from unexpected anomaly. Event monitoring feeds are therefore not just gating signals, they are training labels for forecasting models.

III. AIOps Pre-Detection Architecture

Figure 1 summarizes AIOps pre-detection architecture. The architecture fuses multi-layer telemetry with event context to produce safe and explainable recommendations.

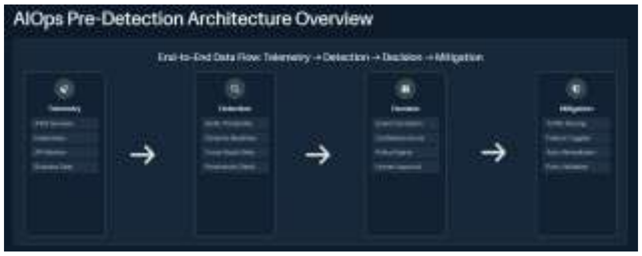
AIOps Pre-Detection Architecture Overview
End-to-End Data Flow: Telemetry → Detection → Decision → Mitigation

Figure 1. AIOps Pre-Detection Architecture

A. Live Stack Management and Scheduling

ITR relies on accurate, continuously updated knowledge of which stacks are live and receiving traffic. Live stack configuration is refreshed via cron for a specific time like every 15 minutes and stored in a central database. Each scheduled health check begins by retrieving the latest stack set and executing validations on the current active environments.

The health check system is multi-layered: Infrastructure services health, infrastructure component health, API performance checks, and data integrity validation. Different check types run at different cadences to balance detection latency and cost.

B. AWS Services Health Monitoring

AWS health checks emphasize fast, deterministic detection using static thresholds. Representative thresholds include SQS queue age, RDS contention, OpenSearch cluster resource/health thresholds, and ElastiCache capacity/eviction constraints.

Service	Key Metric(s)	Example Threshold(s)
Queues	Approximate Oldest Message Time	> 50 s
Database	CPU, Connections, Memory, Disk Queue	CPU > 25%; Connections > 4000; Memory < 3 GB; Q > 6

Open Search	CPU, JVM Memory, Storage, Health	CPU > 25%; JVM > 75%; Storage < 3 GB
Cache	CPU, Memory, Swap, Evictions	CPU > 25%; Memory < 500 MB; Swap=0; Evictions=0

Table I. Example Health checks and thresholds

C. Infrastructure, Kubernetes, and Application Monitoring

Automated Infrastructure checks include Active MQ queue health, Elastic search cluster health, consumer service health, pod health, and capacity comparisons. Pod health uses restart analysis and resource usage from Prometheus/Grafana. Restart and status anomalies are ITR-trigger eligible; CPU/memory can be informational depending on configuration.

Drill-down behaviors of pods show high usage, fetch instance-level details, improve localization and recommendation quality.

D. Performance Monitoring and Business Signals

It is recommended to monitor API response times and order trends per channel. Analysis of traffic split percentages so that trend evaluation remains accurate under asymmetric routing and order dip events are treated as business-impact signals. These are correlated with technical telemetry to improve actionability.

E. Component Interfaces and Data Flow
The end-to-end flow can be implemented as a set of loosely coupled services: (1) a stack inventory service (cron job + DB), (2) scheduled check executors (AWS, infra, API integrity), (3) an alert ingestion API that receives triggers from monitoring systems, (4) an anomaly store that persists findings and supports recurrence detection, (5) an event store that annotates runs with deploy/perf-test/release context, (6) a decision service that assembles evidence and computes confidence, and (7) an execution service that can trigger pipelines for traffic routing changes.

Decoupling these components keeps the system maintainable and supports the program's goal of global rules. For example, new metrics can be added to the check executors without changing the decision engine, as long as they emit standardized signals.

F. Architecture Patterns for Safe Routing
Safe traffic routing depends on patterns that preserve session affinity, cache efficiency, and data correctness during shifts. Segmented routing (by geo, channel, behavior, or time) enables progressive delivery and targeted mitigation. In active-active and blue/green topologies, maintaining at least one good and available stack reduces failover time and enables rapid rollback. Cohort stickiness (e.g., via consistent hashing) mitigates cache churn and avoids breaking session-dependent flows such as carts and checkouts.

ITR operational guidance also emphasizes preventing oscillation after routing changes and using sample windows before flips to confirm persistence. These control-system features turn routing into a repeatable operational practice rather than an ad-hoc emergency lever.

G. Data Integrity Validation
Not all anomalies are performance anomalies. In commerce systems, correctness failures stale promotions, inconsistent product catalog attributes, or drift between datastores and search indexes can create revenue-impacting incidents without obvious infra symptoms. The health check design explicitly includes data integrity validation covering promotional campaign consistency, product catalog validation, and database-to-search record matching.

From a pre-detection perspective, data integrity checks benefit from event correlation and from cohort segmentation. If errors cluster in a subset of traffic like a region or channel, routing can be used to protect customers while remediation proceeds.

I. Signal Taxonomy and Evidence Bundle
To keep the framework scalable, signals should be standardized into a small set of types:
• Infrastructure capacity signals (CPU/memory saturation, disk queue, connection pool exhaustion).
• Reliability signals (5xx rate, failed dependency calls, restart loops).
• Latency signals (p95/p99 response time, queue age).
• Business outcome signals (orders/min, payment success rate) normalized by traffic split.
• Context signals (deployments, performance tests, releases).

Each signal should include an evidence bundle: the raw metric snapshot, the baseline value and deviation score, and a link to the underlying telemetry. Standardizing evidence bundles make automation explainable and supports reproducibility during post-incident reviews.

IV. ITR Decision Workflow

ITR converts uncertain signals into safe actions through gated phases: trigger ingestion, persistence validation, event correlation, delta stack identification, dependency checks, confidence scoring, human approval, and post-action validation.

Figure 2. ITR Decision Workflow

A. Trigger Sources and Recurrence Detection

The workflow may be initiated by alert triggers, health-check anomalies, Pyro anomalies, order dips, or manual invocation. Recurrence detection updates existing flows rather than creating duplicates. A configurable look-back window controls re-trigger timing.

B. Persistence and Timeline Validation

Persistence validation re-checks after a short delay (e.g., 3 minutes) and, for some failure modes, requires sustained evidence (e.g., 10–15 minutes) to avoid acting on transients. Timeline reconstruction helps distinguish early drift from stable behavior.

C. Event Correlation

Event monitoring tracks performance tests, stack releases, and business events. If anomalies align with known events, the workflow can reduce sensitivity or suppress routing actions.

D. Delta Stack Identification and Eligibility Checks

Cross-stack comparisons determine whether a single stack is degraded while another is healthy. If both stacks are degraded, ITR blocks flip and escalates.

E. Test Scenarios

Initial tests validated flip recommendations for high-impact single-stack issues, early termination for transient anomalies, rejection of low-impact cases, and no-flip behavior when both stacks are degraded. Testing also revealed a false positive pattern where minor order variations drove flip suggestions, motivating refinement of severity logic.

Scenario	**Condition**	**Expected Decision**
High-impact single-stack	Order dip + high latency + high failures in one stack	Recommend flip to healthy stack
Transient anomaly	Issue resolves during persistence check	Stop; no flip
Low-impact variance	Minor dip; latency/error normal	No flip; monitor
Multi-stack issue	Both stacks show degradation	No flip; escalate
Payment failures	High payment failures; dependency healthy	Recommend self-mitigation toggle

Table II. Representative test scenarios for ITR workflows

I. Operator Dashboard and Evidence Delivery
To support fast approvals, the workflow must present evidence in a structured way. This program includes a dashboard that surfaces health checks, triggered alerts, cross-stack comparisons, and the recommendation rationale. A practical layout mirrors the decision workflow: (i) what triggered, (ii) persistence results, (iii) event context, (iv) delta stack evidence, (v) confidence score and contributing signals, and (vi) proposed action with rollback notes.

This dashboard is not cosmetic. It is the interface through which operators evaluate risk. If evidence is incomplete, the system should clearly mark the confidence as degraded and default to recommend monitoring/ escalating rather than proposing a flip. This design principle ensures the system remains safe even when some telemetry pipelines are impaired.

F. Detailed Validation Matrix

The ITR decision matrix formalizes how symptoms are validated. Representative categories include error-pattern validation - confirm abnormal error rates or total failures for specific APIs in a single stack, persistence checks- confirm sustained failure over a configured duration), dependency correlation - rule out shared downstream failures, infrastructure correlation - validate CPU/memory/network signals, and telemetry consistency (ensure logs and metrics corroborate).

This matrix matters because it constrains automation. Rather than trusting the model, the system executes explicit validation tasks, telemetry queries, telemetry traces, synthetic checks then use their outcomes as evidence in the confidence score. In effect, the AIOps agent becomes an orchestrator of checks.

G. Payment-Specific Self-Healing as a Pattern
The initial testing included a dedicated payment API flow where the system can recommend self-resolution actions instead of traffic flips. In practice, payment failure incidents have high business impact and are often time sensitive. When failures are isolated to one stack and external dependencies are healthy, flipping traffic may help. However, when the dominant stack has 100% of web traffic, a targeted mitigation such as enabling an asynchronous processing toggle can contain impact quickly while maintaining routing stability.

From an AIOps standpoint, payment flows are ideal for codifying runbook knowledge: define known symptoms (endpoint failure rate, dependency timeouts), validation steps (dependency health checks, traffic distribution), and safe mitigations (toggle enablement, throttling) before recommending routing changes.

H. Flip/No-Flip Policy

The following policy captures the runbook-inspired flip/no-flip logic, with explicit checks for external/shared dependency failures and target readiness. A Flip / No-Flip Policy (Algorithm) clearly defines the next state of action.

V. Anomaly Detection Methodology

Pre-detection combines deterministic thresholds with adaptive statistical baselines, then fuses signals into confidence-weighted recommendations. The program implements a two-stage threshold model and a time-aware anomaly analyzer based on two weeks of history aligned to the same time window.

A. Two-Stage Thresholding

Issue identification thresholds are more sensitive and populate issue lists/notifications; ITR trigger thresholds are stricter and drive the decision workflow.

Layer	Sensitivity	Operational Role
Issue Identification	Higher	Issue list; Teams notifications; context building
ITR Trigger	Lower (stricter)	Webhook trigger; decision workflow; possible flip

Table III. Two-stage thresholds balance monitoring breadth and incident response

B. Time-Aware Baseline Computation

$\mu_t = (1/N) \sum_{i=1..N} X_{t-i\Delta}$, $A_t = |X_t - \mu_t| / (\sigma_t + \varepsilon)$. Flag if $A_t > k$ or if $X_t > \theta_{max}$. This matches the documented min/max plus dynamic threshold behavior.

C. Cross-Stack Delta Analysis

$D_t = |X_t^A - X_t^B| / (\sigma_{A,B,t} + \varepsilon)$. High D_t indicates asymmetry, strengthening localization and reducing incorrect flips when both stacks are degraded.

D. Persistence Validation and Freeze Windows

Require anomalies to persist across multiple intervals; after action, enforce freeze windows to avoid oscillation.

E. Confidence Weighting

$C = \sum w_j s_j$, with $\sum w_j = 1$; recommend action when $C \geq \tau$ and eligibility gates pass. The roadmap calls for confidence visibility and manual overrides.

G. Multi-Signal Quorum Rules

Operationally, high-confidence incidents often express as a quorum of symptoms rather than a single metric. The ITR draft highlights encoding multi-signal quorums such as p99 latency increase + 5xx increase + CPU increase over a sample window to reduce false positives.

A practical formulation is to define a quorum function Q over normalized signals s_j:
$Q = 1$ if ($s_{latency} > a$) AND ($s_{errors} > b$) AND ($s_{capacity} > c$) for N consecutive intervals.

Quorum rules can be layered: a soft quorum for issue identification to surface weak signals and a strict quorum for incident triggers to avoid flip spam.

F. Practical Statistical Choices for Operations

Operational telemetry is noisy and often non-Gaussian. While standard deviation is common, robust alternatives can reduce sensitivity to outliers: σ_t can be estimated with MAD (median absolute deviation) or with a trimmed standard deviation. For streaming use cases, an exponentially weighted moving average (EWMA) baseline can complement seasonal baselines to capture sudden shifts:
$\hat{\mu}_t = \alpha X_t + (1-\alpha) \hat{\mu}_{t-1}$

Combining seasonal baselines (same-time history) with EWMA enables faster adaptation during genuine regime shifts like new traffic levels after a launch while still detecting anomalous drift. In all cases, parameters (k, α, τ) should be tuned per metric class and separated by stage (issue identification vs incident triggering) to avoid alert fatigue.

H. Data Storage, Normalization, and Feature Engineering

Pre-detection depends on consistent historical data. The rollout plan explicitly includes storing metric data for two weeks, normalizing current metrics, and saving them for future trend calculations. In practice, this becomes a lightweight feature store for operations, enabling:

- Time-aligned baselines (same hour/day).
- Stack-to-stack comparisons for each metric.
- Traffic-split normalization for business metrics.
- Post-action analysis (did flips improve SLOs).

Normalization should standardize units that define consistent windows (5-min, 15-min, 30-min) and annotate features with traffic share and event flags so that models do not misinterpret expected changes as anomalies.

J. Alert Noise Filtering and Actionability

The program explicitly tracks alert noise filtering as a goal, with an ambitious target to reduce false positives significantly while maintaining coverage of critical alerts. Noise reduction is not just a machine learning problem; it is a product problem. Alerts should be actionable by default each trigger should map to a validation matrix and a finite set of mitigations. When an alert cannot reasonably lead to a decision, it should be surfaced as informational rather than as a routing trigger.

A practical approach is to maintain an allow-list of triggers and expand only after measuring precision.

K. Mitigation Taxonomy and Policy Selection
Pre-detection is valuable only if it leads to timely mitigations. In practice, mitigations fall into three classes:
1) Routing mitigations: change traffic weights, flip traffic away from an unhealthy stack, or temporarily isolate cohorts.
2) Control mitigations: enable/disable feature flags, switch to asynchronous processing, throttle non-critical traffic, or activate circuit breakers.
3) Remediation mitigations: rollback a deployment, restart a component, scale resources, or rotate credentials.

A policy engine should select the least risky mitigation that is consistent with the evidence. For example, if a single API endpoint regresses after a deployment, rolling back the deployment or toggling a feature is often safer than shifting traffic globally. If a stack exhibits systemic failures and a healthy stack exists, routing is appropriate.

Policy selection should also consider time-to-effect. Feature toggles may take effect within seconds, while scaling or provisioning may take minutes. Routing can be immediate but has side effects (cold caches, affinity). Capturing these constraints explicitly in the decision engine makes recommendations more realistic and increases operator trust.

I. Operational Evaluation and Post-Action Learning
A mature pre-detection system should learn from outcomes. Each ITR run should record: trigger signals, validated evidence, recommended action, approval decision, execution outcome, and post-action SLO changes. This enables offline evaluation of precision/recall for triggers and helps refine weights w_j and thresholds (k, τ). It also supports governance reporting like how often the system recommended a flip, how often operators agreed, and what business impact was avoided.

A simple starting point is to compute post-action deltas for key metrics: Δp99, Δ5xx, and Δorders, over fixed windows like 10 minutes pre, 20 minutes post. If a flip consistently improves these, confidence in the policy increases. If it does not, policies should be revised or gated more tightly.

VI. Static vs Time-Aware Thresholds

Static thresholds are essential for known hard limits, but time-aware baselines enable earlier detection of drift in seasonal metrics. Figure 3 illustrates how dynamic bands can detect early departures before a static threshold is crossed.

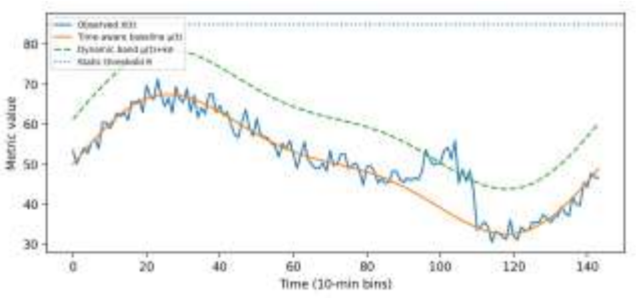

Figure 3. Time-aware baseline vs static threshold; dynamic bands detect early drift.

A. Tradeoffs and Calibration
Baselines must be long enough to capture seasonality yet short enough to adapt. Two weeks of aligned history is a practical compromise. Sensitivity parameters should differ between issue identification and incident triggers.

B. Traffic Split Normalization
Trend evaluation should incorporate traffic split and channel context. Testing non-50/50 splits (e.g., 20/80) validates that anomaly logic remains robust under asymmetric routing.

VII. Operational Safety and Governance

AIOps succeed only if it is safe and trusted. ITR adopts a human-in-the-loop posture for traffic flips while using automation to compress time-to-decision.

A. Human-in-the-Loop Approvals
Interactive approval requests in Teams support rapid operator decisions; optional multi-level approvals scale oversight.

B. Rollback, Validation, and Freeze Windows
Mitigation actions require rollback capability and post-action validation against SLO signals.

C. Explainability and Operator UX
Dashboards and structured evidence improve trust and speed. UX improvements and dashboard finalization are explicit roadmap items.

D. Change and Incident Correlation
Integration with ServiceNow can link anomalies to recent changes and incidents to accelerate RCA.

E. SLO-Driven Gating and Error Budgets
In mature SRE practice, routing decisions should be tied to SLOs and error budgets. Traffic shifts can be gated on SLO deltas (e.g., p99 latency and 5xx error rate) over a sample window, and rollouts can be paused when burn rates exceed budget. Encoding SLO gates into ITR's decision engine improves consistency and enables auditability: operators can see exactly which SLOs justified a flip and which guardrails prevented one.

Where test errors or expected anomalies, event correlation can apply an 'expected error' mask so that SLO gates reflect true customer impact, not synthetic load artifacts.

F. Risk Management for Automated Mitigation
Traffic flips and feature toggles are powerful but risky. Common risks include: (i) shifting load to a stack without sufficient capacity, (ii) introducing cold-cache latency spikes, (iii) breaking stateful flows without session affinity, and (iv) masking the underlying issue and delaying root cause remediation.

ITR mitigates these through eligibility checks (healthy target and headroom), post-flip validation, freeze windows, and explicit recommendations that include rollback guidance. Operational governance also benefits from role-based access control to limit who can approve or execute flips, aligning with the program's role-based framework integration.

G. Compliance, Audit Trails, and Incident Artifacts
In regulated enterprise environments, incident automation must be auditable. Every recommendation should log: the rule triggered, metric snapshots, baseline computations, event context, the proposed action, the approver identity, and the final outcome. This audit trail supports post-incident reviews, compliance controls, and continuous improvement. It also enables attribution when multiple teams rely on a shared AIOps layer.

Where integrations like incident creation, change requests exist, the system should attach the evidence bundle to the ticket so that incident commanders can quickly assess the why behind automated guidance.

VIII. Lessons Learned / Production Insights

1) Ship a robust MVP before broadening scope.
Start with globally applicable rules and reliable traffic flips, then expand.

2) Cross-stack comparison is a force multiplier.
It localizes faults and blocks unsafe flips when both stacks degrade.

3) Persistence checks prevent flip-happy automation.
Re-check after short delays and require sustained evidence for certain failure classes.

4) Self-mitigation toggles can be safer than traffic flips.
Payment-specific mitigations demonstrate lower-blast containment when dependencies are healthy.

5) Warm-up coverage matters.
Add DB warm-up and pod readiness checks to avoid routing to unprepared stacks.

6) Reduce latency with parallelism.
Parallel checks target a 5–10-minute completion time.

7) Guardrails scale better than complex logic.
Favor global rules and independent signals.

8) Case-derived anti-patterns and mitigations
Experience shows that not every incident should be mitigated by flipping traffic. Examples include external/DNS failures where flipping does not remove the shared dependency, or situations where flipping breaks session affinity or causes cold-cache amplification. A safe ITR system therefore includes explicit 'no-flip' policies, dependency checks, and warm-up gates.

Conversely, internal stack-local issues such as crash loops, mis-sized resource pools, or localized DB contention can be effectively mitigated by routing to a healthy stack provided the target is warmed, has headroom, and passes SLO gates.

These anti-patterns motivate a key design principle: routing is a mitigation tool, not the default response. The pre-detection engine should recommend the smallest safe action (self-mitigation toggle, rollout pause, targeted rollback) before proposing high-blast actions.

9) Metrics that matter in practice
The ITR draft categorizes primary decision metrics, 4xx/5xx, p95/p99 latency; AWS resource utilization; queue signals and secondary/context signals.

Operationally, the key is not the size of the metric list, but the consistency of the evaluation windows, the quality of baselines, and the discipline of actionability. Metrics should be included only if they can change a decision or strengthen confidence.

10) Change management and communications are operational multipliers
Pre-detection systems fail when they are hard to use. The program explicitly tracks UX improvements clear message formatting, summaries, and dashboards that explain how recommendations were derived. Well-designed communications reduce time-to-trust: incident commanders can quickly understand whether a recommendation is based on one metric spike or a validated multi-signal quorum.

Notifications should also be role aware. For example, database anomalies may require paging a DB on call, while order dips require involvement of business stakeholders. ITR's planned stakeholder notification system ensures the right responders are engaged when routing actions are proposed.

Finally, coupling recommendations with change artifacts (incident ticket, change request) creates continuity: the same evidence bundle used to recommend a flip should become the starting point for root cause analysis and postmortem documentation.

IX. Future Enhancements

A. Expand trigger coverage
Shifting from infra-only to full API and business anomaly monitoring.

B. Improving warm-up and readiness gates
Adding DB warm-up and pod readiness checks.

C. Multivariate anomaly detection and causal inference
Adopting multivariate models and causal hints while preserving explainability.

D. Operational evaluation metrics
Measuring MTTR reduction, noise reduction, time-to-recommendation, and business outcomes.

E. GenAI Enhancements with Guardrails
Future iterations can safely expand GenAI usage in three areas: (1) evidence summarization for operators (what changed, where, and why the system is confident), (2) query generation (e.g., Splunk query templates parameterized by stack and API), and (3) retrieval of similar historical incidents and SOP steps from the vector database.

However, GenAI outputs must be bound by deterministic checks: recommendations should cite the exact signals and gates that justify action and never bypass approval or post-action validation. This preserves the AI assists humans' posture that the program reinforced.

F. Advanced Modeling Roadmap
Beyond baseline-and-quorum detection, future enhancements can explore: (i) multivariate anomaly models (autoencoders or probabilistic graphical models) that detect coupled drifts, (ii) causal discovery to separate symptom networks from root causes, and (iii) reinforcement learning to select among mitigations (flip vs rollback vs throttle) subject to safety constraints.

These methods must still be bound by operational guardrails.

G. Pre-Biasing Traffic During Surges
Another forward-looking capability is pre-biasing traffic weights in anticipation of known surges (device launches, promotions). This can be done using forecasting models over historical traffic and order trends, combined with stack capacity signals. The result is proactive resilience: reduce risk before the surge, not after alerts fire.

H. Performance and Reliability of the AIOps System Itself
Finally, the AIOps system must be reliable: if Splunk queries fail, if the event feed is stale, or if the orchestration pipeline is down, the system should degrade gracefully and notify operators. The rollout plan should include resiliency and fallback plans to handle such failures.

Operationally, this requires health checks for the AIOps pipeline, clear error reporting, and safe defaults.

K. Platform Scalability and High Availability for AIOps
As AIOps becomes operationally critical, it must itself be highly available. Practical implementation must include running multiple GO instances with load balancing, which prevents a single orchestrator instance from becoming a bottleneck or single point of failure.

In addition, scheduling load should be controlled: health checks running every 15 minutes across multiple stacks can create telemetry pressure. Efficient caching of stack inventory, shared query templates, and adaptive scheduling like increasing frequency during active incidents, decrease during stable periods can reduce cost while improving responsiveness.

Finally, security considerations grow with capability: execution endpoints that can trigger traffic flips must enforce authentication, authorization, and least privilege. Audit logs should be immutable and queryable, and the system should support dry-run modes so that new rules can be tested safely in production before being enabled for action.

X. Conclusion

Pre-detecting anomalies is a practical discipline that blends time-series methods with operational control and governance. The AIOps approach mentioned in this paper demonstrates a pragmatic path: multi-layer health monitoring, time-aware baselines, cross-stack delta localization, event-aware reasoning, and confidence-weighted recommendations routed through human approvals. This shifts operations from simply observing failures to pre-detecting and preventing them, protecting customer experience while keeping operational complexity sustainable.

References

[1] G. Scharf, P. J. P. C. Mendes, and A. B. Bondi, "AIOps: Real-world challenges and research innovations," *Proc. IEEE/ACM Int. Conf. Software Eng. (ICSE)*, pp. 1544–1555, 2023. Doi: 10.1109/ICSE48619.2023.00136

[2] P. Wang *et al.*, "CloudRanger: Root cause identification for cloud native systems," in *Proc. 18th IEEE/ACM Int. Symp. Cluster, Cloud and Grid Comput. (CCGrid)*, 2018, pp. 492–502. doi: 10.1109/CCGrid.2018.00071

[3] D. J. Dean *et al.*, "PerfGuard: Deploying ML-based anomaly detection in practice," in *Proc. ACM Symp. Cloud Comput. (SoCC)*, 2022, pp. 388–402. doi: 10.1145/3542929.3563481

[4] P Jayakumar, "Transformation of Telecom Infrastructure Provisioning from Reactive to Proactive, Intelligent System" in *International Journal of Emerging Trends in Computer Science and Information Technology*, 2025

[5] P Jayakumar, and UST Team "Redefining Incident Triage and Recovery with Agentic AI" in *https://www.ust.com/en/insights/redefining-incident-triage-and-recovery-with-agentic-ai,* 2025

[6] J. Cao *et al.*, "Per-request customization of cloud-based services using adaptive traffic routing," in *Proc. IEEE Int. Conf. Autonomic Comput. (ICAC)*, 2013, pp. 51–60. doi: 10.1109/ICAC.2013.29