Proceedings of the 4th International Conference on Applied Physics and Mathematical Modeling
DOI: 10.54254/2753-8818/2026.38527

Toward a Context-Aware AIOps Framework for High-Energy Physics Computing Clusters: A Design Proposal with Isolation Forest Scoring and Operator-Controlled Response

Cyrus Wu

Springdale Preparatory School, New Windsor, USA
1339210231@qq.com

Abstract. High-energy physics (HEP) computing facilities combine heterogeneous compute nodes, distributed storage subsystems, multi-domain networks, job schedulers and loosely coupled services. Field-validated incident labels are scarce in practice, and legitimate shifts in the workload mix can also raise anomaly indicators. This paper puts forward a context-aware AIOps architecture; Isolation Forest ranks anomalous telemetry windows, full data lineage is kept, and human site operators hold the authority over any response. The stack has six functional tiers, that is, telemetry ingestion, data-quality governance, a temporally valid context store, multi-modal hybrid analytics, event triage and a policy-enforced response executor. Single alert triggers are merged into incident episodes, and the scores only set the review priority; they do not estimate how likely a failure is. Four deployment gates, from offline retrospective analysis to approval-gated actions, limit how far the system may act. The paper reports a design and an evaluation plan; no new operational data are analysed, and no performance gain in production is claimed.

Keywords: high-energy physics computing, AIOps, anomaly detection, Isolation Forest, human-in-the-loop operational workflows

1. Introduction

Modern high-energy physics (HEP) experiments depend on large computing systems to store, move, reconstruct and analyse data at the petabyte scale. Shared infrastructures like the Worldwide LHC Computing Grid (WLCG) combine the resources of institutions in many locations and offer them as one end-to-end service [1]. A single computing centre houses heterogeneous compute hardware, parallel storage arrays, relational and non-relational databases, scheduling middleware, authentication services and layered networks. Therefore, site operators have to interpret diagnostic signals from many domains and keep services available without interruption.

Operators mostly see a cluster through monitoring platforms; CERN gathers the metrics of WLCG sites in MONIT [2], and the Institute of High Energy Physics (IHEP) keeps its own platform [3]. A fixed threshold on one metric works until the behaviour of interest is spread over several coupled signals or moves slowly over time, and this is where machine learning has been tried, both in IHEP studies [4, 5] and in the operational intelligence work for the High-Luminosity LHC (HL-

© 2026 The Authors. This is an open access article distributed under the terms of the Creative Commons Attribution License 4.0 (https://creativecommons.org/licenses/by/4.0/).

DOI: 10.54254/2753-8818/2026.38527

LHC) [6]. Here, artificial intelligence for IT operations (AIOps) covers only methods in which an algorithm helps people notice anomalies, set priorities, reason about a diagnosis and respond after incidents.

In practice, an anomaly output is hard to judge without the operating context of that moment. Normal behaviour shifts with the node role, the mix of jobs, the running campaign, planned maintenance and the installed software version. An extreme reading may mean broken hardware, but it may also come from a planned reconfiguration or a glitch in data collection. Thus, the design treats recording the raw data, explaining it with context and allowing an action as three separate steps.

The question studied here is whether telemetry of many types, scored with Isolation Forest, can be the basis of a context-aware AIOps design for HEP clusters in which only human operators decide on remediation. To answer it, six engineering requirements are defined, and each is backed by mechanisms for handling data, adding context, scoring, collecting evidence, logging feedback and controlling permissions. In addition, the main assumptions are stated as formulas. The planned evaluation measures statistical accuracy and operational value separately, and deployment gates keep model parts that are not fully validated from doing harm.

2. Operational background and technical foundations

2.1. HEP computing and monitoring at large-scale deployments

In HEP computing, the same hardware may serve different experimental programmes or different processing phases; thus, maintenance decisions must consider how storage layers, scheduling frameworks, network fabrics and application stacks depend on each other.

The MONIT platform combines ingestion, stream processing, multi-model storage, dashboards and alarm triggers, and its data-service components stay independent [2]. IHEP's operational-analysis platform collects signals from physical infrastructure, cluster middleware, user activity and security audit trails, and data collection, streaming analytics, alert triggers, configuration management, expert knowledge bases and policy execution are joined in it [3].

Earlier IHEP work on Ganglia data handled sample curation, measurement of deviations, visual display and manual relabelling by operators [4]. A second study on Lustre I/O behaviour applied Isolation Forest with periodic model refresh and job-metadata tags, as well as a web page for checking results [5]. The efficiency gains in these papers cannot be transferred to this work directly; separate validation would be needed first. Still, both show that anomaly detection in production is an end-to-end workflow joining raw data, machine-learning models and human experts.

The HL-LHC Operational-Intelligence projects include monitoring pipelines, smart alerts, job-failure diagnosis, error clustering, file-transfer supervision and predictive maintenance, and these functions have matured at different rates [6]. Therefore, AIOps for HEP should be modular; metric models cannot replace dedicated log-analysis pipelines, and research prototypes should never get production execution privileges.

2.2. Findings from AIOps, anomaly-detection and evaluation research

Conventional monitoring mostly relies on fixed thresholds and regular health probes. AIOps adds learned behaviour representations, statistical prioritisation, correlation of events from different sources and repeated feedback loops. Failure-management studies split fault handling into detection,

DOI: 10.54254/2753-8818/2026.38527

prediction, localisation and remediation, and every subtask needs its own kind of evidence [7]. Table 1 lists the findings of the cited studies that shape this design.

Table 1. Findings from prior work and their consequences for the design

Topic	Finding in prior work	Consequence for this design
Failure management [7]	Detection, prediction, localisation and remediation as separate tasks	Anomaly score kept apart from root cause and action
Log analysis [8]	Collection, compression, parsing, detection, forecasting and diagnosis as separate steps	Source identifiers and parser versions kept; log models only after stable parsing
Isolation Forest [9]	Unsupervised isolation by random partitioning; short paths for rare samples; subsampling	Low-cost baseline ranker for telemetry windows
Benchmarking [10]	No unsupervised detector best in every scenario (ADBench)	Comparison with simple engineering baselines
Concept drift [11]	Loss of model validity under changing data distributions	Drift monitoring; governed retraining
Time-series evaluation [12]	Data leakage and unrealistic splits in common benchmarks	Time-ordered splits; incident grouping; sealed hold-out
Range-based metrics [13]	Precision and recall for incidents with duration	Event-level, range-aware scoring
Human-AI interaction [14]	Visible system status; easy correction; lasting user control	Traceable provenance; reversible actions; reason-coded feedback
HPC anomaly detection [15]	Environment shifts; excess false positives; weak explainability	Context grouping; alert budgets; evidence bundles

3. Operational problem formulation and design requirements

The design has to catch anomalous run-time behaviour in heterogeneous HEP clusters early, keep the number of alerts low and block unvalidated automated actions. Six requirements follow from the practice in Section 2.1 and the findings in Table 1; each answers a core research question, maps to a concrete component or policy and has measurable acceptance criteria. Table 2 lists them with their technical basis and implementable controls.

Table 2. Engineering requirements and implementable controls for the proposed framework

ID	Design requirement	Technical basis	Implementable control
R1	Preserve source identity and provenance	[2, 3, 6, 8]	Event and collection time; asset ID; schema or parser version; transformation ID; raw-window pointer
R2	Add operational context before scoring	[3-6, 11, 15]	Node role; workload; campaign; software version; maintenance state; data-quality flags
R3	Combine rules with learned detectors	[4, 5, 7, 9, 10]	Data-quality rules; engineering thresholds; robust statistics; context-aware Isolation Forest
R4	Separate anomaly, incident, cause, and action	[6, 7, 9, 14]	Ranked evidence; unconfirmed status; operator triage; separate authorization state
R5	Evaluate events and operational burden	[10-13, 15]	Time-ordered splits; event precision and recall; delay; false alerts; review time; overhead

DOI: 10.54254/2753-8818/2026.38527

Table 2. (continued)

R6	Make automation staged and reversible	[3, 6, 7, 14]	Offline, shadow, advisory and approval-gated stages; audit; rollback

Each requirement must map to data fields, model behavior, interface state, evaluation measures, or authorization rules, and effectiveness remains subject to Section 5.

4. Mathematical and architectural framework

4.1. Notation and context-aware telemetry time windows

Let i represent a physical asset or logical service unit, t denote the end timestamp of a sliding analysis window, and g stand for a contextual grouping such as node role, workload profile, experimental campaign or maintenance status. Telemetry streams are converted into feature vectors incorporating signal magnitude, statistical dispersion, local temporal variation, data-missing fraction and domain-specific derived ratios, as defined in Equation (1):

$$\boldsymbol{x}_{i,t}^{(g)} = [\mu_{i,t}, \sigma_{i,t}, \Delta_{i,t}, \rho_{\text{miss}}, r_1, \ldots, r_p]^T \tag{1}$$

where p denotes the number of domain-specific derived-ratio features.

Here, μ and σ capture signal level and variability; Δ represents local slope or temporal difference; ρ_{miss} denotes the fraction of missing entries; r_1 through r_p correspond to domain-derived ratios including read-write throughput proportion or CPU wait time against overall utilisation. Feature sets ought to be finalised before inspecting test-set incident cases. Derived ratios featuring unstable denominators shall attach validity flags instead of carrying out silent value clipping.

Telemetry signals frequently exhibit heavy-tailed distributions and shift numerical scales under distinct operational regimes. Within validated context groups, outlier-robust normalisation reduces the adverse impact of extreme training-sample observations, as defined in Equation (2):

$$z_{j,t}^{(g)} = [x_{j,t} - \text{median}_g(x_j)]/[1.4826\,\text{MAD}_g(x_j) + \varepsilon] \tag{2}$$

Here, j indexes an individual feature of the window vector of asset i; the asset index is omitted in Equation (2) for brevity. The coefficient 1.4826 aligns the median absolute deviation (MAD) with standard deviation under Gaussian reference distributions, while the small term ε prevents division-by-zero failures; the formulation makes no Gaussian distribution presumption. A feature that is almost constant, with a MAD near zero, should be dropped or left to dedicated deterministic change-trigger rules.

Pooled training data give more samples but may mix regimes that operate differently. Per-context models see more uniform data but can be unstable with few samples. Therefore, the system sets minimum sample counts and a documented fallback order, from workload-role groups to role-only groups and lastly the whole facility. Each time a fallback happens, the result gets a low-confidence label.

4.2. Isolation Forest score and threshold semantics

Isolation Forest is trained by building random isolation trees on the normalised window vector z. With the path length h(z) and the subsample size n, Equation (3) gives the normalised anomaly

DOI: 10.54254/2753-8818/2026.38527

score:

$$s(\boldsymbol{z}, n) = 2^{-E[h(\boldsymbol{z})]/c(n)}, \quad c(n) = 2H_{n-1} - 2(n-1)/n \tag{3}$$

In this score, n is the subsample size for training Isolation Forest, and H_k is the k-th harmonic number.

The term c(n) is the expected path length for normalising the score [9]. A larger s means faster isolation of a sample from the fitted reference data. Thus, s is only a statistic for ranking. It cannot be read as a fault probability without separate calibration studies on the target clusters.

Per-context thresholds turn continuous scores into binary candidate-alert flags, as Equation (4) shows:

$$a_{i,t} = 1[s(\boldsymbol{z}_{i,t}, n) \geq \tau_g] \tag{4}$$

Here, τ_g denotes the decision threshold of context group g.

The threshold τ_g can be fixed based on protected validation periods, an alert-volume budget or quantiles of trusted baseline behaviour. The final test incidents stay out of this tuning; otherwise the test would reward a threshold fitted to its own answers.

Figure 1 shows the six tiers of the design, and Table 3 lists what each tier takes in, checks and passes on.

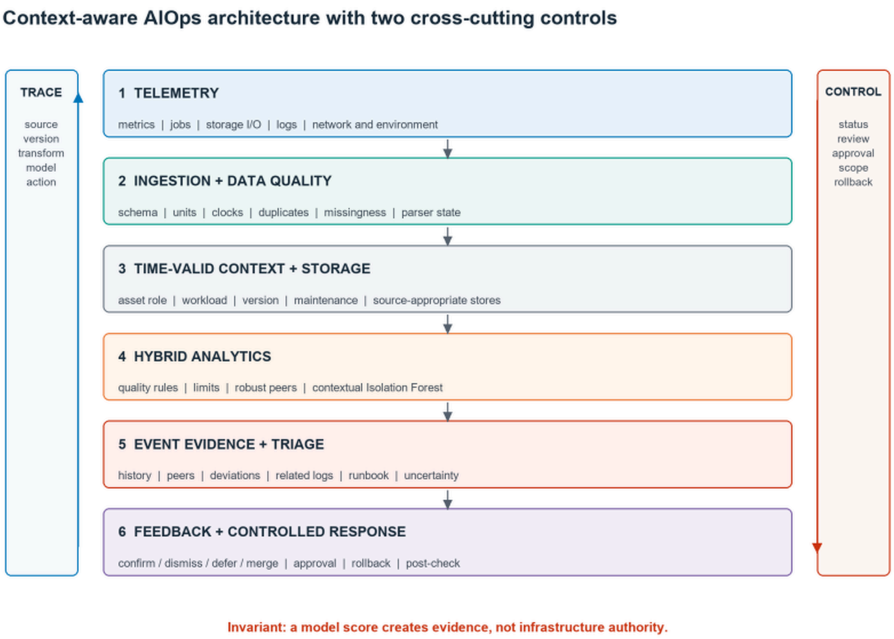


Figure 1. Six-layer context-aware AIOps framework for HEP computing clusters

4.3. Six-layer system architecture

The duties of each tier are set out in Table 3.

DOI: 10.54254/2753-8818/2026.38527

Table 3. Duties of the six tiers and the shared storage

Tier	Takes in	Checks or keeps	Passes on
Telemetry	Compute-node metrics; scheduler job events; storage I/O; service logs; network and environmental sensors	Event time and collection time; asset ID; unit; schema revision; quality marker	Timestamped records
Ingestion and data quality	Telemetry records	Schema, units, duplicates, clock skew, data loss; collector restarts, counter resets and parser upgrades logged	Validated records; raw and intermediate outputs; scoring paused on corrupted input
Context	Asset inventory; scheduler status; campaign metadata; software configuration; maintenance calendar; trouble tickets	Joins by valid time interval; low weight for unknown or conflicting tags, manual audit still possible	Context-labelled windows
Analytics	Context-labelled windows	Engineering rules; robust peer-group statistics; context-aware Isolation Forest	Scores; candidate alerts
Decision support	Candidate alerts	Grouping by asset, service and time range; repeat counts and spans; high priority for safety violations; no score averaging before calibration and prospective validation	Incident episodes with evidence bundles (trends, peer comparison, deviating features, missing-data share, log lines, maintenance context, model metadata, runbook links)
Feedback and governed response	Operator decisions; action outcomes	Triage labels (confirm, dismiss, corrupted input, defer, merge, fix context); formal state tags; "root cause" only after operator or verified-rule confirmation	Approved actions with preconditions, scope, rollback, timeout, post-check and immutable audit trail
Storage (all tiers)	Metrics; logs and events; metadata, approvals and runbooks	Time-series stores; indexed stores; relational or document stores; append-only audit trails	Query results linked to sources, time windows, query logic and transformation versions

The detectors of the analytics tier are compared in Table 4; a log-sequence model joins them only once parsing is stable, data are sufficient, upkeep is manageable and it beats simple baselines [8].

Table 4. Detector roles, assumptions, and safeguards

Detector	Best-suited input	Main assumption	Principal risk	Required safeguard
Data-quality rule	All collected sources	Known validity conditions	Gaps or delays read as normal operation	Explicit invalid flag; downstream scoring blocked or qualified
Engineering threshold	Metric with a physical or policy limit	Operational meaning of the limit	Alert storms across workloads; weak multivariate shifts missed	Context; hysteresis; suppression; named owner
Robust peer statistic	Single metric, comparable peers	Median and MAD fit the reference group	Poor peer groups; scale near zero	Context hierarchy; minimum group size; scale checks

DOI: 10.54254/2753-8818/2026.38527

Table 4. (continued)

Isolation Forest	Windowed multivariate numeric features	Rare partitions worth a review	Score read as probability or diagnosis	Model versions; seed stability; protected thresholds; human triage
Sequential log model	Stably parsed event sequences	Meaningful templates and order	Parser drift; heavy upkeep	Parser-version tracking; fallback rules; simple-baseline comparison

4.4. Evidence assembly and operator triage

Figure 2 traces one telemetry window from feature construction to operator feedback.

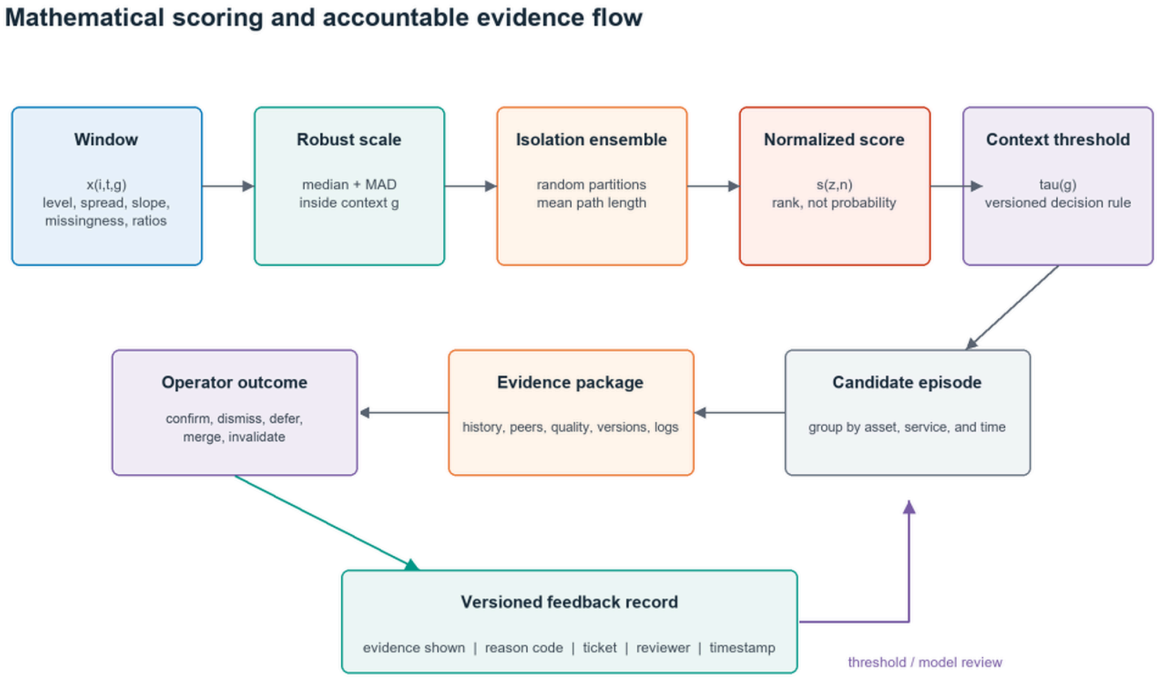


Figure 2. Mathematical scoring and accountable evidence-assembly pipeline

In Figure 2, x(i,t,g) is the window feature vector of asset i at time t in context group g, s(z,n) the normalised Isolation Forest anomaly score and τ(g) the context-dependent decision threshold; grey arrows show the forward flow, and teal and purple arrows show operator feedback.

The evidence panel should let operators see the deviating behaviour, the reference group definition, the anomaly onset time, data integrity, the affected services or job workflows and the permitted actions. The interface shows raw scores next to the thresholds, with plain labels like candidate, under review and confirmed by operator.

Figure 3 illustrates the threshold logic with synthetic, deterministic scores; it contains no IHEP data.

Figure 3 covers three operating regimes, with alerts suppressed during planned maintenance periods. The blue line is the synthetic Isolation Forest score and the orange line the context-aware threshold; red dots mark candidate-alert windows, which are triage candidates and not confirmed faults. The score ranges from 0 to 1.

DOI: 10.54254/2753-8818/2026.38527

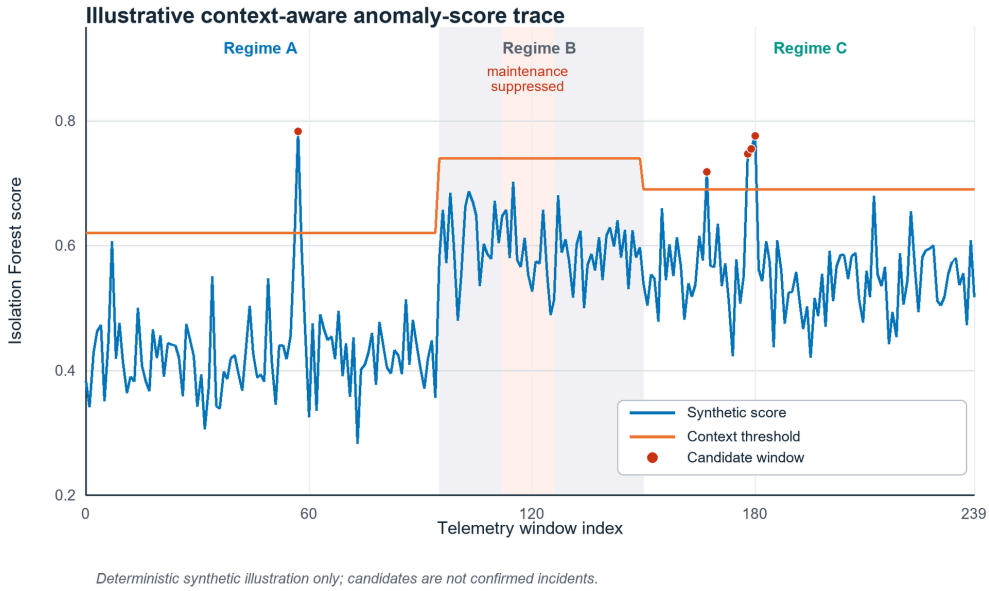


Figure 3. Illustrative context-aware anomaly-score trace using deterministic synthetic data

4.5. Drift management and governed response

Figure 4 shows the four deployment gates, and Table 5 gives the rules for drift checks, retraining, promotion and demotion.

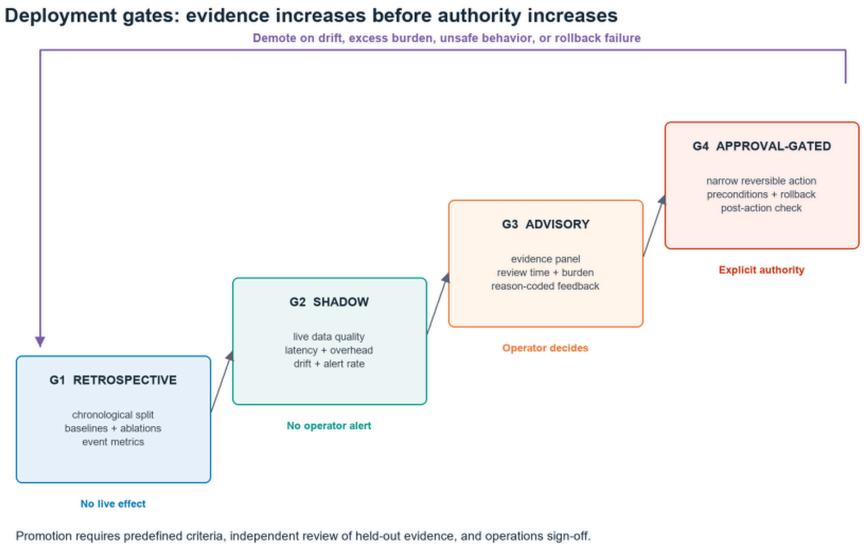


Figure 4. Prospective deployment and authority gates

DOI: 10.54254/2753-8818/2026.38527

Table 5. Drift, retraining and deployment-gate rules

Stage	Rule	Purpose
Drift and retraining	Feature and score distributions, alert frequency and operator outcomes per group; alert surge without more verified incidents checked against thresholds, grouping and data quality; formal retraining policy; validity period, contamination check, comparison on protected subgroups; model card; rollback	Controlled model change
G1 retrospective, G2 shadow	G1 pipeline on historical data, no change to production; G2 full pipeline on live telemetry, candidates hidden from operators; latency, missingness, drift, candidate rate and resource use measured	Failures from live event order, schema changes or late collection
G3 advisory	Evidence and runbooks shown; confirmations, dismissal reasons, review time, reviewer agreement, merged episodes and thin evidence logged; feedback not ground truth by default	Operator workload and bias
G4 approval-gated	Narrow reversible actions; preconditions, execution, rollback, end state and side effects checked; destructive, cross-service and irreversible actions outside the framework	Safe, reversible automation
Promotion and demotion	Promotion after preset acceptance criteria, independent review of held-out evidence and formal operations sign-off; demotion on model drift, excessive alert burden, unsafe behaviour or failed rollback	Authority tied to evidence

5. Prospective evaluation protocol

5.1. Data handling, comparison and staged deployment

Table 6 collects the evaluation rules, from the data inventory to the threshold comparison.

Table 6. Evaluation protocol rules

Stage	Rule	Purpose
Data inventory	Per-source owner, sampling or event mechanism, retention, units, schema history, expected gaps, privacy or access limits; shared asset IDs and time keys	Joins across sources; each raw source auditable
Labels	Trouble tickets, monitoring snapshots, maintenance logs and operator reviews under a labelling specification; ambiguous intervals marked uncertain	Ground truth without forced classes
Time split	Early normal-rich periods for training; later periods for feature selection and threshold calibration; sealed final hold-out	No look-ahead
Grouping	All windows of one incident, maintenance activity or recurring alert episode in one partition; one difficult subgroup reserved	No leakage; stress test
Training data	Known incidents and invalid periods removed; sensitivity to plausible contamination reported; causal missing-value handling; missing share as a feature	Contamination control
Candidate records	Threshold version, context group ID, training period, feature specification, seed and software release; ranking checked over several seeds	Reproducible candidates; less trust in single-seed borderline cases
Baselines	Engineering rules; robust peer-group statistics; context-free global model; stronger methods where data allow	Practical value beyond weak baselines

DOI: 10.54254/2753-8818/2026.38527

Table 6. (continued)

Ablation	Context grouping, robust normalisation, missing-value features, domain ratios, multi-seed ensemble, maintenance suppression and episode aggregation removed one at a time	Components kept only for clear gains
Matching	Episode counted as a match on time overlap with a confirmed incident, within a tolerance fixed before testing	Event-level scoring
Threshold comparison	Fixed alert budget; fixed statistical target; per-service tolerance for misses and false alerts recorded	Explicit operational choice

5.2. Event-oriented metrics

Event-level precision and recall follow Equation (5):

$$P_e = N_{\text{matched}}/N_{\text{alert}}, \quad R_e = N_{\text{detected}}/N_{\text{confirmed}} \tag{5}$$

$N_{matched}$: number of alert episodes matched to confirmed incidents; N_{alert}: total alert episodes generated; $N_{detected}$: confirmed incidents successfully detected; $N_{confirmed}$: total ground-truth confirmed incidents.

Equation (6) gives the recall-oriented F_β and the false-alert burden FANPD; range-aware overlap is reported as well [13].

$$F_\beta = (1+\beta^2)P_eR_e/(\beta^2P_e + R_e), \quad \text{FANPD} = N_{\text{false}}/(N_{\text{nodes}} \times D) \tag{6}$$

N_{false}: number of false-positive alert episodes; N_{nodes}: total monitored node count; D: observation duration measured in days.

β>1 can be adopted when missed incidents incur heavier costs than extra manual reviews; its numerical value is locked before examining final test results. FANPD enables burden comparison across different cluster scales and observation durations. Table 7 lists the minimal set of evaluation indicators to be documented.

Table 7. Minimum prospective evaluation measures

Measure	Unit or definition	Why it matters	Reporting rule
Event precision and recall	Matched alert and incident episodes	Prevents long incidents or dense sampling from dominating the result	State overlap and tolerance rules and report uncertainty for ambiguous labels
F_β	Harmonic combination of event precision and recall	States the selected trade-off between missed incidents and review burden	Declare beta in advance and do not tune it after inspecting the final test
FANPD	False-alert episodes per node-day	Converts false positives into operator workload	Report by node role, service, campaign, and day where possible
Detection delay	Time from incident onset to first valid alert	Separates early detection from late detection	Report the median and upper tail and include missed incidents
Review time	Operator minutes per alert episode	Measures triage cost and the usefulness of the evidence panel	Compare interfaces only with similar cases and reviewers
Score and ranking stability	Variation across seeds, windows, and retraining dates	Identifies fragile candidates and model drift	Report rank correlation and disagreement at the threshold

DOI: 10.54254/2753-8818/2026.38527

Table 7. (continued)

System overhead	CPU, memory, storage, and latency	Prevents monitoring from interfering with scientific workloads	Measure ingestion, scoring, and interface paths separately

Table 7 uses F_β for the event-level F-score weighted by beta and FANPD for False-Alert episodes per Node-Day.

6. Discussion

6.1. Expected gains and trade-offs

Table 8 sets the main design choices against their expected gains and costs; none of these effects has been measured yet.

Table 8. Design choices, expected gains, and costs

Design choice	Expected gain	Cost or limit
Full data lineage	Incidents traced back to raw windows	Not discussed
Context grouping	Fewer false alarms on rare but legitimate workloads; more homogeneous data	Fewer training samples per group
More features	Complex interactions captured	Higher drift risk; harder interpretation
Aggressive thresholds	Higher incident recall	Heavier manual review
Hybrid rules and models	Safety-critical rule checks kept	Not discussed
Structured operator feedback	Basis for revising thresholds and models	Possible disagreement or confirmation bias
More automation	Less repetitive human work	Larger damage from bad input or model defects
Staged deployment gates	Trade-offs visible; short-term impact limited	Basic conflicts not removed

6.2. Limitations and validity threats

Ideas from four fields meet in this framework (HEP computing, high-performance computing, software reliability engineering and human-AI interaction). The fields differ, however, in telemetry, workloads, incident definitions and permission limits. The six requirements are engineering judgement still to be tested at facilities, and not findings confirmed across sites.

This paper has not analysed any new operational data from the field. Real performance depends on data quality, incident frequency, grouping quality, label completeness, workload features and local site policies. Therefore, moving the design to another computing facility needs separate validation on site.

For some services, Isolation Forest may do worse than rule-based detectors or other machine-learning methods. Also, trouble tickets and operator feedback can be incomplete. The evaluation protocol above lowers these risks, but prospective evaluation in real use is still needed.

DOI: 10.54254/2753-8818/2026.38527

7. Conclusion

The paper proposes a context-aware AIOps architecture for heterogeneous high-energy physics computing clusters. Raw telemetry passes through a multi-step data quality screening, is joined with time-valid context metadata, and is then analysed by deterministic rules, robust statistical estimators and Isolation Forest. The next step is an offline prototype on public or synthetic metrics with fixed seeds and released code; work with real facility data would need its own data-access approval, security audits and operator involvement.

References

[1] Bird, I. (2011). Computing for the Large Hadron Collider. *Annual Review of Nuclear and Particle Science, 61*(1), 99–118. https://doi.org/10.1146/annurev-nucl-102010-130059

[2] Aimar, A., Aguado Corman, A., Andrade, P., Delgado Fernandez, J., Garrido Bear, B., Karavakis, E., Kulikowski, D. M., & Magnoni, L. (2019). MONIT: Monitoring the CERN data centres and the WLCG infrastructure. *EPJ Web of Conferences, 214*, Article 08031. https://doi.org/10.1051/epjconf/201921408031

[3] Hu, Q., Wang, L., Zheng, W., & Jiang, X. (2022). Operation and maintenance analysis platform at IHEP. *Proceedings of Science, ISGC2022*, Article 011. https://doi.org/10.22323/1.415.0011

[4] Chen, J., Wang, L., & Hu, Q. (2020). Machine learning-based anomaly detection of Ganglia monitoring data in HEP data center. *EPJ Web of Conferences, 245*, Article 07061. https://doi.org/10.1051/epjconf/202024507061

[5] Wang, L., Hu, Q., & Chen, J. (2023). Anomaly detection of I/O behaviours in HEP computing cluster based on unsupervised machine learning. *Journal of Physics: Conference Series, 2438*(1), Article 012010. https://doi.org/10.1088/1742-6596/2438/1/012010

[6] Di Girolamo, A., Legger, F., Paparrigopoulos, P., Schovancová, J., Beermann, T., Boehler, M., Bonacorsi, D., Clissa, L., Decker de Sousa, L., Diotalevi, T., Giommi, L., Grigorieva, M., Giordano, D., Hohn, D., Javůrek, T., Jezequel, S., Kuznetsov, V., Lassnig, M., Mageirakos, V., . . . Tuckus, N. (2022). Preparing distributed computing operations for the HL-LHC era with operational intelligence. *Frontiers in Big Data, 4*, Article 753409. https://doi.org/10.3389/fdata.2021.753409

[7] Notaro, P., Cardoso, J., & Gerndt, M. (2021). A survey of AIOps methods for failure management. *ACM Transactions on Intelligent Systems and Technology, 12*(6), Article 81. https://doi.org/10.1145/3483424

[8] He, S., He, P., Chen, Z., Yang, T., Su, Y., & Lyu, M. R. (2021). A survey on automated log analysis for reliability engineering. *ACM Computing Surveys, 54*(6), Article 130. https://doi.org/10.1145/3460345

[9] Liu, F. T., Ting, K. M., & Zhou, Z.-H. (2012). Isolation-based anomaly detection. *ACM Transactions on Knowledge Discovery from Data, 6*(1), Article 3. https://doi.org/10.1145/2133360.2133363

[10] Han, S., Hu, X., Huang, H., Jiang, M., & Zhao, Y. (2022). ADBench: Anomaly detection benchmark. *Advances in Neural Information Processing Systems, 35*, 32142–32159. https://doi.org/10.52202/068431-2329

[11] Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., & Bouchachia, A. (2014). A survey on concept drift adaptation. *ACM Computing Surveys, 46*(4), Article 44. https://doi.org/10.1145/2523813

[12] Wu, R., & Keogh, E. J. (2023). Current time series anomaly detection benchmarks are flawed and are creating the illusion of progress. *IEEE Transactions on Knowledge and Data Engineering, 35*(3), 2421–2429. https://doi.org/10.1109/TKDE.2021.3112126

[13] Tatbul, N., Lee, T. J., Zdonik, S., Alam, M., & Gottschlich, J. (2018). Precision and recall for time series. *Advances in Neural Information Processing Systems, 31*. https://proceedings.neurips.cc/paper/2018/hash/8f468c873a32bb0619eaeb2050ba45d1-Abstract.html

[14] Amershi, S., Weld, D., Vorvoreanu, M., Fourney, A., Nushi, B., Collisson, P., Suh, J., Iqbal, S., Bennett, P. N., Inkpen, K., Teevan, J., Kikin-Gil, R., & Horvitz, E. (2019). Guidelines for human-AI interaction. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Paper 3, pp. 1–13). Association for Computing Machinery. https://doi.org/10.1145/3290605.3300233

[15] Lu, Y., Zhang, W., Yang, L., Dai, Y., Wu, H., Wu, Z., Tian, H., Li, Y., Chen, J., Liu, C., & Dong, Y. (2026). A survey of anomaly detection in HPC systems using machine learning. *CCF Transactions on High Performance Computing, 8*(3), 352–376. https://doi.org/10.1007/s42514-025-00266-7