Workload-Conditioned Health Residual Graphs for GPU Fleets

An Open Research Note on Accelerator Health Modeling,
Observability Disambiguation, and Diagnostic Orchestration

Jack Shaquer

July 2026

Draft v0.1 — shared for open discussion and collaboration

Abstract

Modern GPU fleets emit enormous volumes of telemetry — temperatures, power draw, clocks, ECC counters, link statistics, and pipeline metadata — yet operators still struggle to answer a deceptively simple question: *is this device physically deteriorating, or is it just running different work?* Static thresholds generate false positives across workload phases; generic anomaly detectors collapse rich evidence into opaque scores that are hard to act on. This note proposes a different representation: a *typed, time-varying health residual graph*, in which subsystem-specific residuals are (i) conditioned on a canonical representation of workload exposure, (ii) computed against multiple reference bases (workload-conditioned, self-history, peer-cohort, and physics-constrained), (iii) normalized by physical exposure rather than wall-clock time, and (iv) coupled to a closed-loop diagnostic selector that chooses active tests by expected information gain under operational constraints. A distinctive element is treating *observability itself* as a first-class health domain, so that device failure can be structurally disambiguated from monitoring-pipeline failure. We outline the architecture, formalize the residual construction, sketch subsystem embodiments for NVIDIA GPU fleets using DCGM/NVML/Prometheus telemetry, and close with open research questions we invite the community to take further.

1 Motivation

Accelerator reliability is now a first-order economic problem. Large training runs are gated by stragglers and silent degradation; inference fleets absorb thermal drift, HBM wear, and link flakiness that manifest as tail-latency regressions long before hard failures. Meanwhile, the monitoring stack itself (exporters, scrapers, collectors, time-series storage) fails often enough that "the GPU disappeared from Prometheus" is ambiguous between a dead device and a dead scrape job.

Three concrete problems motivate this work:

P1. Workload confounding. Raw telemetry is dominated by what the device is asked to do. A GPU switching from prefill-heavy to decode-heavy LLM traffic changes its thermal, power, and memory signature dramatically without any change in health. Thresholds and workload-blind anomaly models cannot separate "different work" from "deteriorating hardware."

P2. Observability confounding. Missing or inconsistent telemetry is routinely either imputed away or treated as device failure. Neither is correct: the monitoring path is a fallible system

in its own right and deserves its own health state.

P3. Passive-to-active gap. Passive evidence rarely resolves a hypothesis on its own. Operators need a principled way to choose the *next diagnostic action* — a field-group change, a controlled workload replay, a vendor diagnostic, a drain — balancing information gain against disruption.

2 Approach Overview

We propose maintaining, per accelerator, a **typed health residual graph**: a graph (or logically equivalent relational/event representation) whose nodes represent accelerators, HBM stacks, PCIe/NVLink links, hosts, cooling domains, workload phases, jobs, telemetry sources, RAS events, and diagnostic procedures, and whose edges represent execution, containment, cooling, connectivity, peer membership, observation, corroboration, temporal precedence, testing, and causal hypotheses.

Each node and edge carries time-varying attributes: residual magnitude, confidence, persistence, trend, exposure denominator, timestamp, model version, and provenance. The graph deliberately *preserves* topology, subsystem structure, and evidence lineage instead of collapsing everything into a single anomaly score.

The pipeline is:

Telemetry → Context & Exposure Engine → Expected-Response Models → Subsystem Residuals → Typed Residual Graph → Diagnostic Selector → Action & Feedback → (graph update)

3 Multi-Reference Residual Construction

For an observed response $y(t)$ (e.g., GPU temperature, achieved clocks, ECC increments), we compute a *stack* of residuals against distinct reference bases rather than one scalar score:

$$r_{\text{work}}(t) = y(t) - \hat{y}_{\text{workload}}(t) \tag{1}$$

$$r_{\text{self}}(t) = y(t) - \hat{y}_{\text{history}}(t) \tag{2}$$

$$r_{\text{peer}}(t) = y(t) - \hat{y}_{\text{peer}}(t) \tag{3}$$

$$r_{\text{phys}}(t) = y(t) - \hat{y}_{\text{physics}}(t) \tag{4}$$

where the references are, respectively: a workload-conditioned expectation, the device's own longitudinal baseline, a cohort of equivalently configured peers (same SKU, topology, cooling domain, firmware, workload regime), and optional physics constraints (e.g., thermal RC behavior, power–activity envelopes). Residuals may be differences, ratios, standardized deviations, quantile exceedances, or calibrated likelihoods, weighted by data availability and confidence.

A subsystem health state is then a fusion:

$$h_s(t) = G_s\big(r_{\text{work}}, r_{\text{self}}, r_{\text{peer}}, r_{\text{phys}},\ \text{exposure},\ \text{persistence},\ \text{corroboration}\big) \tag{5}$$

where G_s can be a rule set, state-space model, Bayesian update, sequential test, learned model, or hybrid physics/data model. The multi-reference structure is diagnostic in itself: a residual against self-history but *not* against peers suggests a fleet-wide (environmental or workload) cause; divergence from both suggests a device-specific one.

4 Exposure-Normalized Subsystem Residuals

A central design principle: normalize reliability evidence by *physical exposure*, not wall-clock time. A GPU that transferred 100× the HBM bytes should be allowed proportionally more raw ECC events before it looks anomalous.

Thermal path.

$$h_{\text{thermal}}(t) \leftarrow T_{\text{obs}}(t) - T_{\text{exp}}(P,\ \text{workload},\ \text{cooling},\ \text{peer},\ \text{history}) \tag{6}$$

A persistent rise in temperature *per unit dissipated power* implicates airflow, TIM, cold plates, recirculation, fans, or sensors — not workload.

Power–performance. Rising energy per unit of comparable activity, falling clocks at comparable power, or growing throttle residency under equivalent work:

$$h_{\text{power}}(t) \leftarrow \left[P - P_{\text{exp}}(\text{activity},\ \text{clocks},\ \text{cap})\right] \text{ and/or } \left[\text{activity} - \text{activity}_{\text{exp}}(P,\ \text{phase})\right] \tag{7}$$

Memory health. ECC and row-remap evidence normalized by memory exposure, with cumulative-counter vs. rate handling and counter-reset detection:

$$e_{\text{ECC}}(t) = \frac{\Delta\text{ECC}(t)}{\max(\text{HBM\_bytes}(t), \varepsilon)}, \qquad e_{\text{remap}}(t) = \frac{\Delta\text{row\_remaps}(t)}{\max(\text{mem\_active\_time}(t), \varepsilon)} \tag{8}$$

Interconnect health. PCIe replays, retraining, CRC-like errors, and NVLink/fabric events normalized by topology-aware traffic:

$$e_{\text{link}}(t) = \frac{\Delta\text{link\_error}(t)}{\max(\text{link\_bytes}(t), \varepsilon)} \tag{9}$$

Control plane. Unexpected clock behavior, throttle-reason codes, XID transitions, driver/firmware state, resets, and requested-vs-achieved operating-state mismatch.

Observability (first-class). Rather than imputing missing telemetry away, we compare the *expected structure* of telemetry (device UUID set, field cardinality, sample counts, scrape cadence, exporter continuity) with the observed structure, and use host continuity, scheduler state, exporter process state, and network reachability to classify: device-specific disappearance vs. exporter/scrape loss vs. host/network loss. Each classification drives a *different* graph-state update.

5 Canonical Workload Exposure and Adapters

To keep the health engine application-agnostic, application telemetry is translated by **workload adapters** into a *canonical exposure vector*: compute-active cycles, tensor activity, memory bytes, memory-active time, PCIe/NVLink traffic, energy, active duration, thermal exposure, and a workload-regime label.

- **LLM inference adapter:** prefill/decode/mixed phase, prompt and generated tokens, batch size, sequence length, KV-cache occupancy (or proxy), model configuration, parallelism role.

- **Training/HPC adapter:** forward/backward/optimizer phase, kernel class, collective communication, compute cycles, HBM and interconnect traffic.

A shift from prefill-heavy to decode-heavy traffic then changes *expected* behavior without raising hardware risk; residual drift that *persists after phase conditioning* is what signals deterioration.

6 Evidence Fusion and State Transitions

Subsystem and aggregate states such as HEALTHY, WATCH, DEGRADED, ACTIONABLE, FAILED_OR_UNOBSERVABLE, and RECOVERING transition only under persistence, acceleration, cross-domain corroboration, RAS corroboration, or peer divergence — suppressing single-sample noise. Evidence propagates over graph edges: a decode-heavy workload may explain high HBM activity yet fail to explain rising HBM temperature per unit of HBM traffic; a simultaneous throttle residual and peer divergence strengthen a thermal/memory-path hypothesis, while a fleet-wide shift weakens device-specific hypotheses in favor of environmental ones.

7 Closed-Loop Diagnostic Selection

Given the current graph state, the system generates candidate diagnostic actions — increase telemetry frequency, request additional field groups, run DCGM/vendor diagnostics, replay a controlled workload, compare a peer, check cooling/power sensors, collect driver logs, drain/migrate, or schedule inspection — and ranks them by:

expected information gain, diagnostic duration, workload disruption, safety, maintenance-window constraints, probability of confirming/rejecting a hypothesis, and cost of delay.

The diagnostic outcome is attached back to the graph as evidence, updating states and confidence. This residual-to-diagnostic loop turns passive monitoring into an active, budget-aware inference process — conceptually close to Bayesian experimental design applied to fleet operations.

8 Worked Scenarios

Gradual thermal degradation. A workload-conditioned temperature model predicts expected temperature per activity/power; a target GPU develops a persistent positive residual vs. both history and peers; throttle residency grows under comparable work while RAS and observability stay clean; the graph raises a thermal-path hypothesis and selects a low-disruption inspection; the outcome confirms or clears it.

Exposure-normalized HBM wear. ECC increments and row-remaps, normalized by bytes and memory-active time, diverge from both the device's own baseline and its cohort — distinguishing a workload-driven rise in *raw counts* from a genuine rise in *error rate per unit exposure*.

Device vs. exporter disappearance. One of eight GPU UUIDs vanishes while node, exporter, and the other seven keep reporting: classified as device-specific loss. All series vanish alongside host/exporter metrics: classified as monitoring-path failure. Different classifications, different actions.

9 Relation to Existing Work

Observed-minus-expected residual monitoring, ML anomaly detection on hardware telemetry, condition-based maintenance, and GPU traffic/error monitoring are all well-trodden (in both the literature and the patent record, e.g., classic predictive condition-monitoring residual systems and platform-health engines with failure-prediction-driven migration). The contribution proposed here is not residuals per se but their *specific composition*: a workload-conditioned, exposure-normalized, multi-reference residual stack maintained in a typed graph with first-class observability disambiguation and an information-gain-driven diagnostic loop. Whether that composition earns its complexity is precisely an empirical question — see below.

10 Open Research Questions

We share this note in the hope others will pressure-test, extend, or refute it. Questions we consider open:

1. **Graph schema.** What node/edge type system and update semantics best trade expressiveness against operability at fleet scale (10k–100k+ devices)?
2. **Reference-model choice.** When do simple conditional baselines (quantile regression on exposure features) match learned models? How should the four reference bases be weighted when they disagree?
3. **Regime inference.** How should workload regimes be inferred when explicit application labels are unavailable — purely from counter signatures?
4. **Peer-cohort construction.** How much do cohorts need to control for (topology, cooling domain, firmware, workload mix) before peer residuals become trustworthy?
5. **Multiscale persistence.** What persistence/corroboration rules across seconds–minutes, hours, and days–weeks time scales minimize false positives without missing slow drift?
6. **Observability disambiguation.** Can the device-vs-pipeline classification be made provably (or at least measurably) reliable from structural evidence alone?
7. **Diagnostic selection.** What is the right formalization of expected information gain here — and how does it compose with SLO-aware operational cost?
8. **Benchmarks.** The field lacks open, labeled GPU-degradation telemetry traces. An offline DCGM replay corpus with worked cases (thermal drift, HBM wear, device-vs-exporter loss) would enable honest comparison across methods.

11 Call for Collaboration

A minimal next step is an **offline DCGM replay prototype**: ingest historical fleet telemetry, construct the residual graph in shadow mode, and evaluate against incident tickets and RMA outcomes. If you operate GPU fleets and have telemetry, incident histories, or strong opinions about why this will or won't work, I would genuinely like to hear from you.

This note describes ideas at a conceptual stage; it makes no claims of completed empirical validation. Feedback, criticism, and pointers to prior art are all welcome.