GUARD: SCALABLE STRAGGLER DETECTION AND NODE HEALTH MANAGEMENT FOR LARGE-SCALE TRAINING

Guanliang Liu [1] **Abhinandan Patni** [1] **Congzhu Lin** [1] **Zoe Zeng** [1] **Jack Wittmayer** [1] **Josh Wu** [1]
Ashvin Nihalani [1] **Binxuan Huang** [1] **Yinghong Liu** [1] **Rory Na** [1] **Anthony Ko** [1] **Alexander Zhipa** [1] **Cong Cheng** [1]
Mi Sun [1] **Vijay Rajakumar** [1] **Rejith George Joseph** [1] **Parthasarathy Govindarajen** [1]

ABSTRACT

Training frontier-scale foundation models involves coordinating tens of thousands of GPUs over multi-month runs, where even minor performance degradations can accumulate into substantial efficiency losses. Existing health-check mechanisms, such as NCCL tests or GPU burn-in, primarily focus on functional correctness and often fail to detect fail-slow behaviors that silently degrade system performance. In this paper, we present **Guard**, a scalable system for detecting stragglers and ensuring node health in large-scale training clusters. Guard combines lightweight online performance monitoring during training with an offline node-sweep mechanism that systematically evaluates and qualifies nodes before they participate in production workloads. This design enables Guard to detect both acute failures and long-running fail-slow behaviors that traditional diagnostics cannot capture. Deployed on large-scale foundation model pretraining workloads, Guard improves mean FLOPs utilization by up to **1.7**×, reduces run-to-run training step variance from **20%** to **1%**, increases mean time to failure (MTTF), and significantly reduces operational and debugging overhead. These results demonstrate that proactive straggler detection and systematic node qualification are critical for maintaining stable and efficient large-scale training.

1 INTRODUCTION

As large language model (LLM) training has grown in scale from hundreds to thousands to tens of thousands of GPUs, maintaining node health has become a key bottleneck to sustaining high throughput over an extended period of time. (Huang et al., 2019; Narayanan et al., 2019; Rajbhandari et al., 2020; Ren et al., 2021; Shoeybi et al., 2019; Jiang et al., 2024; Dubey et al., 2024; Liu et al., 2024; Google DeepMind, 2024; Rasley et al., 2020; Chowdhery et al., 2022; Ziji Shi, 2025). Foundation model training runs now employ hybrid parallelism, combining data, tensor, pipeline, and expert parallelism, across thousands of accelerators that are interconnected by hierarchical high-speed network. (Narayanan et al., 2019; Ren et al., 2021; Shoeybi et al., 2019; Chang et al., 2024; Chen et al., 2024; Gao et al., 2024; Fedus et al., 2021; NVIDIA, 2018). In such a workload, even a single underperforming node can slow down global training progress as hybrid parallelism significantly increases synchronous NCCL operations such as all-reduce and all-to-all that are bounded by the slowest participant (Jiang et al., 2024; Chang et al., 2024; Chen et al., 2024; Gao et al., 2024; NVIDIA, 2018). While fail-stop faults such as GPU crashes or hardware errors are relatively straightforward to detect and recover from, a more subtle and persistent class of failures known as "grey nodes" have emerged as a major obstacle to stable large-scale training (Huang et al., 2017; Gunawi et al., 2018; He et al., 2023; Xiong et al., 2024; Lean Wang1, 2024; Apostolos Kokolis, 2025).

Grey nodes refer to machines that pass standard health checks designed to detect hard failures but exhibit degraded performance during real workloads (Huang et al., 2017; Gunawi et al., 2018; He et al., 2023; Xiong et al., 2024; Hu et al., 2024; Shazeer et al., 2017; Lean Wang1, 2024; Zhou et al., 2023; Stojkovic et al., 2025). Unlike fail-stop faults, grey nodes do not immediately cause errors in training jobs; rather, they silently reduce the overall throughput of the job, with their impact accumulating and worsening over time (Jiang et al., 2024; Hu et al., 2024; Shazeer et al., 2017; Dubey et al., 2024; Ziji Shi, 2025; Zhou et al., 2023). In large-scale training runs, a single grey node's impact is magnified as the workload requires thousands of GPUs to synchronize frequently during each training step, causing the slowest node to gate progress(Jiang et al., 2024; Chang et al., 2024; Chen et al., 2024; Gao et al., 2024; Rombach et al., 2022; NVIDIA, 2018). Existing validation mechanisms such as NCCL tests, GPU burn-in procedures, or short stress benchmarks focus primarily on hardware functionality

[1]Store Foundational AI, Amazon, Seattle, WA, USA. Correspondence to: Guanliang Liu <guanlian@Umich.edu>.

Proceedings of the 9th MLSys Conference (Industry Track), Bellevue, WA, USA, 2026. Copyright 2026 by the author(s).

and communication correctness (NVIDIA, 2021; 2024a;b; Xiong et al., 2024; Rombach et al., 2022; NVIDIA, 2018; Hu et al., 2022). Even with improved on-chip diagnostics reducing silent data corruption and other hardware-level fault rates, these error detection mechanisms only ensure basic operability; performance-impacting faults that grey nodes exhibit under long-duration, mixed compute–communication workloads persist (He et al., 2023; Chang et al., 2024; Chen et al., 2024; Gao et al., 2024; Hu et al., 2024). Consequently, large production clusters often silently experience prolonged efficiency loss. (Huang et al., 2017; Gunawi et al., 2018; He et al., 2023; Hu et al., 2024; Shazeer et al., 2017; Lean Wang1, 2024; Lian et al., 2020).

Our production observations reveal that grey nodes can have numerous root-causes that manifest across multiple system layers. At the hardware level, thermal throttling, degraded high-bandwidth network adapters, and unstable NVLink interconnects can lead to reduced GPU frequency or increased communication latency (Jiang et al., 2024; NVIDIA, 2021; 2024a;b; Rombach et al., 2022; Hu et al., 2022). At the system level, CPU or PCIe bandwidth limitations can further exacerbate these slowdowns

(Chang et al., 2024; Chen et al., 2024; Gao et al., 2024; NVIDIA, 2018). Despite these issues being minor in isolation, they are workload- and synchronization-dependent making them difficult to proactively diagnose. When they appear in multi-week training runs, their cumulative effect results in significant inefficiencies. (He et al., 2023; Xiong et al., 2024; Hu et al., 2024; Shazeer et al., 2017; Lean Wang1, 2024). This observation highlights a critical gap in current large-scale training infrastructure: the lack of continuous, non-intrusive, and performance-sensitive node-level health monitoring mechanisms that leverage workload-specific performance metrics. (NVIDIA, 2021; Xiong et al., 2024; Shazeer et al., 2017; Rombach et al., 2022; Lean Wang1, 2024; Yulin et al., 2021).

To address these gaps, we present Guard, a system for straggler detection and node health management designed for large-scale hybrid-parallel training. as illustrated in Figure 1. Guard integrates online performance monitoring with an offline diagnostic workflow to systematically detect and localize both transient and persistent gray-node behaviors, using training-step time as the primary end-to-end performance signal.

At its core, Guard consists of two tightly coupled components. The online component continuously tracks key hardware and communication metrics—including GPU frequency, temperature, power consumption, network retransmission events, and effective bandwidth—enabling low-overhead detection of anomalous behavior without disrupting ongoing training jobs. These signals provide real-time visibility into performance degradations that are otherwise difficult to observe through traditional health checks.

Complementing the online monitoring layer, Guard includes an offline node-sweep mechanism that executes lightweight diagnostic workloads to reproduce realistic communication and synchronization patterns on a subset of nodes. This design allows Guard to isolate underperforming components, such as degraded interconnects or faulty network interfaces, while minimizing interference with production workloads. Together, the online and offline components form a unified system that enables accurate diagnosis of both transient and persistent performance issues.

Finally, we present a sample triage workflow that demonstrates how Guard integrates detection, diagnosis, and mitigation into a closed-loop operational pipeline. Through this system-level design, Guard provides a practical, scalable, and effective solution for maintaining node health and mitigating stragglers in large-scale distributed training. We have open-sources a subset of monitoring/detection tools in fkat [1], Amazon's Foundational Software Kit for AI Training.

The remainder of this paper is organized as follows. Section 2 describes our production cluster environment and the motivation for our approach. Section 3 analyzes real-world performance degradation patterns observed in industrial-scale LLM training. Section 4 introduces the online monitoring subsystem, while Section 5 presents the offline node-sweep methodology. Section 6 details the end-to-end triage workflow, and Section 7 evaluates the system through extensive experiments and discusses the broader implications of our findings.

2 Background and Motivation

Detecting and mitigating performance-degraded nodes has become fundamentally more challenging in modern large-scale training systems due to the combined effects of architectural complexity, dynamic workloads, and extreme scale.

Modern large-scale language model training relies on increasingly complex hybrid parallelism strategies—including data, tensor, pipeline, and expert parallelism—that require sustained and high-volume communication across thousands of GPUs (Huang et al., 2019; Narayanan et al., 2019; Rajbhandari et al., 2020; Ren et al., 2021; Shoeybi et al., 2019; Jiang et al., 2024; Chang et al., 2024; Chen et al., 2024; Gao et al., 2024; Fedus et al., 2021; NVIDIA, 2018). In such environments, overall training performance is determined not only by the raw computational capacity of individual accelerators, but by the efficiency of global synchronization across the entire system. As a result, a single underperforming node can slow down global progress, since all participants

[1] https://github.com/amzn/fkat

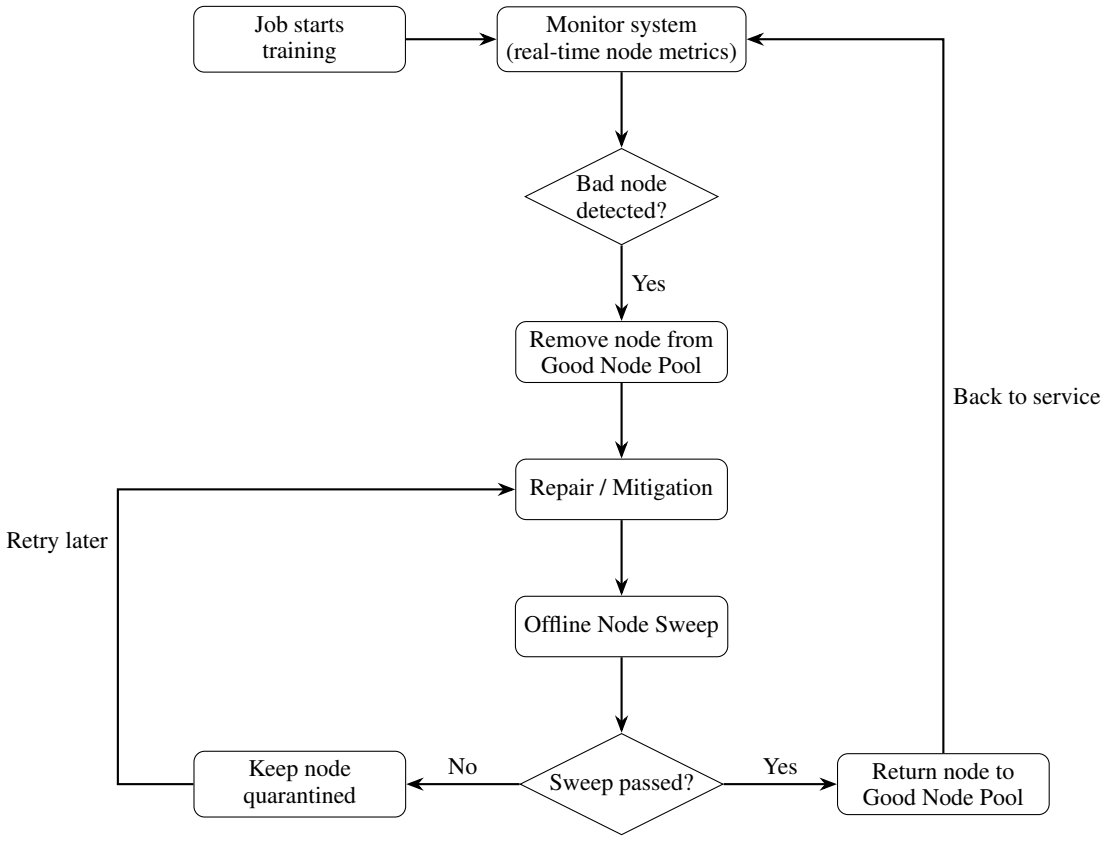

Figure 1. Automated node health management workflow. The monitor system continuously checks node metrics, isolates bad nodes, performs offline sweeps after repair, and reintegrates healthy nodes.

must wait at synchronization barriers during collective operations making system-wide performance sensitive to even small per-node slowdowns. (Jiang et al., 2024; Chang et al., 2024; Chen et al., 2024; Gao et al., 2024; Ziji Shi, 2025).

The architectural complexity of modern foundation models further exacerbates this issue. In Mixture-of-Experts (MoE) models, expert parallelism distributes subsets of experts to different GPUs, leading to inherently imbalanced and dynamically changing workloads (Shazeer et al., 2017; Dubey et al., 2024; Liu et al., 2024; Google DeepMind, 2024; Le Scao et al., 2022; Ziji Shi, 2025). Because routing decisions and token distributions vary over time, a degraded node may remain undetected during initial validation phases, only manifesting as a straggler under specific communication or workload conditions later in training where its impact compounds across repeated synchronization points (Jiang et al., 2024; Hu et al., 2024; Shazeer et al., 2017).

In addition to model-level complexity, large production clusters experience continuous background fluctuations at both the hardware and system levels. Thermal variations, power-management adjustments, and transient congestion in inter-node communication paths can degrade performance without triggering explicit hardware alarms (He et al., 2023; NVIDIA, 2021; 2024a;b; Xiong et al., 2024; Rombach et al., 2022; Hu et al., 2022). These soft degradations may persist for extended periods, silently eroding training throughput in specific workloads and thereby remaining invisible to conventional monitoring mechanisms (Xiong et al., 2024; Hu et al., 2024; Lean Wang1, 2024).

Conventional node validation techniques—such as short-duration communication benchmarks or GPU burn-in diagnostics—are primarily designed to verify functional correctness under idealized conditions (NVIDIA, 2021; 2024a;b; Rombach et al., 2022; NVIDIA, 2018). However, these tests fail to capture performance degradations that are workload-dependent, intermittent, or only observable under long-running, mixed compute–communication workloads (Chang et al., 2024; Chen et al., 2024; Gao et al., 2024; Hu et al., 2024; Yulin et al., 2021). As a result, nodes that pass standard validation may later exhibit degraded behavior when exposed to realistic training traffic.

As training scales further, the operational impact of such undetected inefficiencies becomes substantial. Even modest per-node performance deviations on the order of a few per-

cent can accumulate across thousands of GPUs and millions of training steps, translating into significant increases in time-to-solution and infrastructure cost (Jiang et al., 2024; Hu et al., 2024; Shazeer et al., 2017; Dubey et al., 2024). Maintaining a consistently healthy cluster therefore requires mechanisms that go beyond detecting fail-stop failures, and instead continuously identify and mitigate silent or gradual performance degradation at scale (Huang et al., 2017; Gunawi et al., 2018; He et al., 2023; Lean Wang1, 2024; Yulin et al., 2021; Apostolos Kokolis, 2025).

3 Job Slowdown Analysis

Large-scale distributed training frequently suffers from performance degradations that do not cause job failures but significantly reduce throughput. In synchronization-heavy workloads, overall progress is bounded by the slowest participants, causing even small node-level slowdowns to be amplified at the job level. Unlike fail-stop faults, these degradations often evade traditional health checks and persist silently for long periods, making them particularly costly in multi-week training runs.

Across production workloads, we observe two recurring slowdown behaviors. First, a subset of nodes consistently exhibits slower performance than the cluster baseline, leading to a persistent inflation of training step time. Second, transient step-time spikes appear intermittently, often correlated with communication instability or resource contention. While these spikes may seem minor in isolation, they compound significantly at scale and can trigger cascading slowdowns or timeouts in large hybrid-parallel jobs.

Our investigation reveals that these slowdowns arise from multiple system layers. On the CPU side, insufficient allocation or bandwidth limitations delay data movement and communication coordination. Figure 4 shows that an dynamic CPU schedulers can increase training speed variation by up to TK% even when GPU hardware and utilization remain unchanged. This highlights that GPU utilization alone is insufficient for assessing node-level performance.

Communication-related issues further exacerbate slowdown behavior. When a high-bandwidth interconnect adapter degrades or becomes unavailable, modern communication libraries such as NCCL may transparently reroute traffic through alternative paths to preserve functional correctness. Although this avoids immediate job failure, it reduces effective bandwidth and introduces routing asymmetry. As shown in Figure 6, resolving such degraded communication paths reduces training step time from 8.7 s to 8.4 s. In large-scale or MoE workloads with repeated synchronization phases, these sub-second differences accumulate into substantial efficiency loss.

GPU-side performance degradation represents another major contributor. Thermal throttling, power delivery instability, and aging hardware can reduce sustained clock frequency or effective throughput without triggering explicit errors. A single underperforming GPU can slow down collective operations for its entire parallel group, propagating localized degradation into cluster-wide straggler behavior.

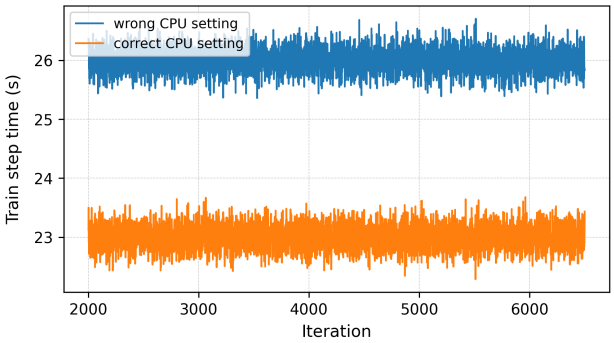

Figure 2. different CPU settings' training step time

Figure 3. Training step time reduction after removing a degraded communication path. Small per-step differences compound significantly at scale.

These issues are rarely detected by traditional node validation methods such as GPU burn-in tests or short NCCL communication checks. Burn-in tests emphasize functional correctness over sustained throughput, while NCCL tests prioritize connectivity and tolerate performance loss through transparent rerouting. As a result, nodes that pass standard validation may still behave as grey nodes in production, consistently lagging behind peers and degrading overall training efficiency.

Together, these observations motivate the need for continuous, performance-aware monitoring during training and independent offline verification before nodes are returned to service. The following sections introduce an online monitoring system that detects performance anomalies in real time

(Section 4), an offline node sweep mechanism that validates node health under realistic workloads(Section 5).

3.1 CPU Setting Limitations

CPU configuration plays a critical role in large-scale training performance and involves two key aspects: CPU allocation per GPU worker and CPU frequency settings. The optimal CPU allocation is highly workload-dependent, varying across training paradigms such as dense models and Mixture-of-Experts (MoE), while overly aggressive frequency tuning can negatively impact overall system stability.

In large-scale foundation model training, CPUs are responsible for several performance-critical tasks, including data loading, checkpointing, and coordinating intra-node GPU communication. As training scales, CPU bandwidth increasingly becomes a limiting factor. Once CPU resources approach their bandwidth limits, these essential operations can become bottlenecked, leading to cascading performance degradation and reduced overall training throughput.

Specifically, CPU bandwidth limitations can affect:

- **Data Loading**: Slower preprocessing and delivery of training samples to GPUs;
- **Checkpoint Operations**: Increased latency in saving and restoring model states;
- **Inter-GPU Communication**: Reduced efficiency in coordinating and transferring data among GPUs.

In our production experiments, we identify two key CPU configuration parameters that significantly impact training performance. The first is CPU frequency. By default, most servers enable dynamic frequency scaling; however, we observe that disabling dynamic scaling and using a fixed CPU frequency can lead to higher and more stable training throughput. The second parameter is the number of CPU cores allocated to each training pod. This requirement is highly model-dependent: models with heavier communication patterns, such as Mixture-of-Experts (MoE), require substantially more CPU resources than dense models to achieve optimal performance.

When the combined workload exceeds CPU bandwidth capacity, these operations experience increased latency, directly impacting the overall training throughput. This bottleneck becomes particularly pronounced in distributed training scenarios where efficient communication and data movement are crucial for maintaining training efficiency. the detail result is shown in the figure 1. without correct CPU setting it could be a 20% train speed slow down.

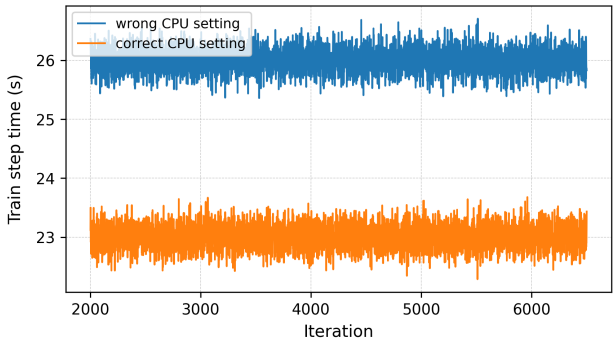

Figure 4. performance difference in different CPU setting

3.2 Slow communication

Communication inefficiencies represent another critical challenge in production training environments. While the current NCCL routing architecture assigns dedicated network adapters to each GPU, the system's fallback mechanism can mask underlying hardware issues. When a GPU's network adapter fails, NCCL automatically reroutes communication through another GPU's adapter (typically GPU 0) without triggering hardware failure alerts. For instance, if GPU 5's adapter malfunctions, its communication will be redirected through GPU 0's network connection.

This fallback mechanism, while preventing immediate job failures, introduces significant performance degradation across the training cluster. The impact is particularly severe in Mixture of Experts (MoE) model training, where communication patterns are more complex. MoE architectures require two critical synchronization points: token dispatch and result combination, both involving communication across the entire Expert parallelism group. When a node operates sub-optimally, it creates bottlenecks at these synchronization points.

The performance impact compounds significantly in modern architectures. For example, in a model with 32 MoE layers, these communication inefficiencies accumulate across each layer, resulting in substantial overall training slowdown. This cascading effect demonstrates why detecting and addressing sub-optimal communication patterns is crucial for maintaining training efficiency at scale.

Table 1 shows an abnormal network status. Even when one network adapter is down, the NCCL library automatically reroutes communication through another available adapter to maintain functionality. However, this failover leads to abnormally high workloads on the remaining network adapter, effectively doubling the network traffic as illustrated in Figure 2. Consequently, each training step becomes slower by approximately 0.3 seconds, as shown in Figure 3.

Table 1. Comparison between normal and abnormal GPU–NIC mappings.

	GPU0 (Normal)	GPU7 (Abnormal)
Expected NIC	0	7
Actual NIC	0	0 (misrouted)
Status	Normal	Faulty (adapter down)

3.3 GPU Calculation Performance Degradation

GPU performance degradation represents one of the most common yet subtle causes of node-level slowdowns in large-scale distributed training. In production environments, multiple hardware and environmental factors, such as thermal throttling, unstable power delivery, and component aging, can lead to sustained performance loss on individual GPUs even when the node passes standard validation tests.

Thermal throttling is a primary contributor to GPU performance degradation. During intensive and prolonged training workloads, GPUs operate near their thermal design limits. When the temperature exceeds predefined thresholds, the hardware automatically reduces core clock frequency or power draw to prevent overheating. These protective mechanisms, while preserving device safety, directly reduce computational throughput. As shown in Table2 2, an increase in GPU temperature from 50°C to 77°C can result in a core frequency drop from 1.93 GHz to 1.38 GHz, leading to significant slowdowns during synchronization-heavy training steps.

In addition to temperature effects, power delivery instability is another critical factor influencing GPU performance. Modern GPUs rely on consistent and high-quality power supply to sustain peak clock frequencies. Voltage fluctuations, current limits imposed by faulty power distribution units (PDUs), or degraded power cables can cause the GPU to enter conservative performance states. In our production observations, nodes with slightly lower power draw (10–15% below cluster average) exhibited noticeably reduced FLOPS utilization despite showing normal utilization rates and frequencies, suggesting that power constraints can silently limit performance.

Memory subsystem efficiency also plays an important role. Aging hardware or marginal memory modules may experience increased access latency or ECC corrections, reducing effective memory bandwidth and causing stalls in compute pipelines. These issues are rarely detected by burn-in tests, which typically run for short durations and emphasize arithmetic correctness rather than sustained data movement.

Moreover, environmental and mechanical factors—such as dust accumulation, uneven airflow across chassis, or partially degraded cooling fans—can create persistent temperature imbalances between GPUs within the same node. Even a single GPU operating below its nominal frequency can slow down collective operations like AllReduce and AllGather, propagating the slowdown to all participating devices in the cluster.

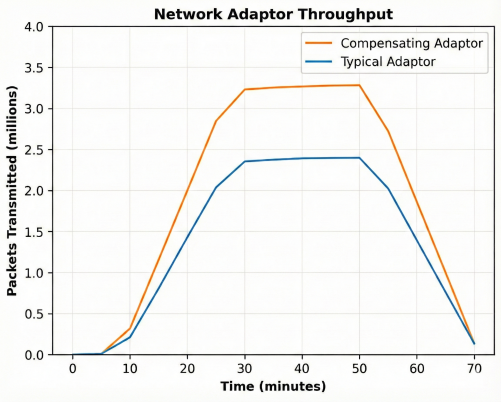

Figure 5. Abnormal network packages transmitted metric.

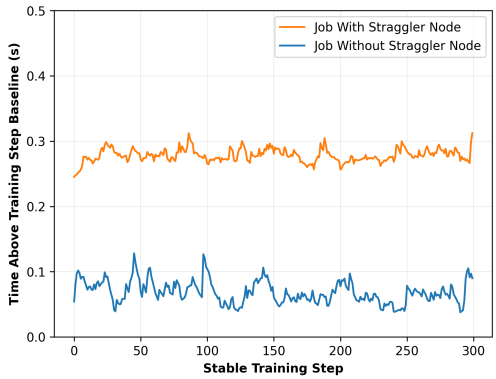

Figure 6. Training step time reduction from 8.7s to 8.4s.

In summary, GPU computation degradation can arise from a combination of thermal, electrical, and memory-level inefficiencies. While each factor may appear minor in isolation, their cumulative effects can cause severe straggler behavior in distributed training. Continuous monitoring of temperature, power draw, frequency stability, and memory error rates is therefore essential for early detection and mitigation of GPU-level degradation before it impacts cluster-wide performance.

4 Online Node Health Monitoring System

Traditional node health checks in large-scale training systems focus primarily on functional correctness, such as GPU availability or binary link status. However, our production experience shows that many of the most damaging perfor-

Table 2. Relationship between GPU temperature and clock frequency. Higher temperatures trigger protective downclocking, reducing effective throughput.

GPU	TEMPERATURE (°C)	CORE FREQUENCY (GHZ)
GPU0	50	1.93
GPU1	60	1.93
GPU2	69	1.78
GPU3	77	1.38

mance issues do not cause explicit failures. Instead, nodes remain operational while silently delivering reduced effective throughput, becoming stragglers in synchronization-heavy workloads.

To address this gap, we design an online node health monitoring system that continuously tracks performance-correlated hardware and network signals during real training runs. Rather than relying on short validation tests or static thresholds, the system is grounded in degradation patterns repeatedly observed in production.

4.1 Proposed Metrics for Continuous Node Health Monitoring

Based on observations from our production-scale training environment, we identify several hardware and system-level indicators that strongly correlate with node stability and training efficiency. These metrics are continuously collected from all active nodes during training and jointly analyzed to detect anomalies that traditional validation methods fail to capture.

GPU Temperature. Sustained high GPU temperatures can trigger thermal throttling mechanisms, directly reducing computational performance. While short burn-in tests may not expose this behavior, long-running workloads often reveal persistent thermal imbalances due to cooling inefficiencies. Continuous temperature monitoring allows early detection of thermal anomalies, preventing long-term degradation in sustained workloads.

GPU Utilization. GPU utilization represents how effectively GPU compute resources are being used. Persistently low utilization can indicate CPU–GPU imbalance, software inefficiency, or internal bottlenecks such as limited memory bandwidth. Monitoring utilization trends helps identify nodes underperforming relative to cluster averages.

GPU Clock Frequency. Clock frequency serves as a direct indicator of GPU performance state. Frequency drops caused by thermal throttling, power capping, or hardware aging can silently reduce node throughput. Tracking per-GPU frequency stability in real time enables detection of subtle slowdowns that are invisible in coarse-grained cluster metrics.

GPU Power Draw. Power draw provides an indirect measure of GPU workload intensity. Under normal training loads, stable power consumption is expected. Deviations—especially low power draw despite high utilization—often suggest voltage regulation issues or failing hardware components. In our production observations, nodes showing normal utilization but abnormally low power draw were consistently correlated with reduced FLOPS utilization, validating this as a strong health signal.

Network device Error Count. Network device Error error metrics, including packet retransmissions, link retries, and link flaps, reveal low-level network instability. These transient issues rarely appear in short validation tests but can cause synchronization stalls or communication timeouts in long-running distributed training. Monitoring error counts helps identify potential cable degradation or malfunctioning network adapters early.

Network device Transmission Rate (Gb/s). Transmission rate reflects the effective throughput of network communication. Persistent reductions in transmission rate—without corresponding error spikes—may indicate degraded cables, misconfigured MTU, or thermal throttling of the network interface. This metric provides a holistic view of inter-node communication efficiency.

Network device Status. Network device may intermittently reset due to hardware faults or driver-level issues. Although NCCL automatically reroutes traffic to maintain connectivity, such fallback paths usually introduce additional latency or bandwidth reduction. Continuous monitoring of link status allows operators to proactively identify and isolate affected nodes before performance degradation propagates.

4.2 Online Monitoring Strategy

Motivated by the observations above, our online monitoring system continuously tracks a compact yet informative set of hardware- and network-level metrics that directly reflect effective training performance. These metrics include GPU temperature, clock frequency, power draw, interconnect error counters, and effective communication throughput. Rather than relying on fixed absolute thresholds, all metrics are evaluated relative to peer nodes participating in the same training job, allowing the system to naturally adapt to workload characteristics and hardware heterogeneity.

To ensure robustness, nodes are flagged only when multiple performance indicators exhibit sustained deviation from the peer baseline across consecutive evaluation windows. This multi-signal and temporal filtering strategy effectively suppresses false positives caused by transient fluctuations, while still enabling early detection of persistent performance degradation. Once a node is flagged, it is removed from the healthy node pool and scheduled for offline verification,

ensuring that online monitoring remains lightweight and non-intrusive. Final validation is performed through the offline node sweep mechanism described in Section X.

Importantly, our system does not classify nodes as faulty based solely on hardware error counters. Instead, training step time—representing user-visible performance—is treated as the primary signal, with hardware metrics serving as supporting indicators. Based on the severity of observed performance impact, we apply a tiered response policy that balances mitigation urgency with operational disruption:

- **No observable impact.** If training throughput remains unaffected, the node is marked as *pending verification*. The running job is left unchanged, and the node continues to participate while being monitored more closely.
- **Moderate, sustained slowdown ($\sim$10%).** When a measurable but tolerable slowdown is detected, the issue is considered actionable but non-urgent. Mitigation is deferred to the next checkpoint to confirm the diagnosis while avoiding unnecessary job interruption.
- **Severe degradation or stalls ($\geq$20%).** When performance degradation is substantial or progress stalls, the node is deemed harmful to training. The job is immediately restarted with a healthy replacement node, and the affected node is removed from service for remediation.

This tiered strategy prevents overreaction to transient anomalies while ensuring timely intervention when performance degradation materially impacts large-scale training efficiency.

5 Offline Node Health Verification Method

5.1 Why Traditional Node Health Checks Are Insufficient

Traditional node validation methods such as GPU burn-in tests and NCCL communication tests are widely used in large-scale distributed training systems to verify basic hardware functionality before admitting nodes into production clusters. However, these methods are primarily designed to detect hard failures and short-term instability, rather than sustained performance degradation under realistic training workloads.

GPU burn-in tests focus on exercising compute units using synthetic arithmetic and memory-intensive kernels over short durations. While effective at detecting faults such as overheating or defective cores, they do not sufficiently stress performance-critical subsystems including memory bandwidth, NVLink interconnects, and PCIe paths. As a result, nodes may pass burn-in validation while still exhibiting degraded intra-node communication or asymmetric GPU performance under sustained load.

Similarly, NCCL tests validate the functional correctness of collective communication primitives (e.g., AllReduce, Broadcast) but typically run for only a few seconds under idealized conditions. Moreover, NCCL's fault-tolerance mechanisms can transparently reroute traffic around degraded links, allowing tests to pass while masking reduced bandwidth, routing asymmetry, or network-level instability. Such effects often emerge only during prolonged, synchronization-heavy distributed training.

In practice, these limitations lead to the emergence of *grey nodes*: machines that do not fail outright but consistently lag behind others, inflating step time and reducing overall cluster throughput. To address this gap, we adopt a closed-loop node health management framework that combines continuous online monitoring with an offline verification step. Online monitoring conservatively detects suspicious behavior during real workloads, after which the node is removed from the healthy pool. An offline *node sweep job* is then triggered to perform end-to-end performance validation before the node is allowed to re-enter production, ensuring both safety and diagnostic accuracy.

5.2 Single-Node Sweep: Intra-Node Performance Validation

The single-node sweep targets performance degradations within a single node that frequently evade traditional validation. It is designed to expose sustained throughput loss and communication asymmetry while remaining lightweight enough to run independently of full training jobs.

Specifically, the sweep evaluates:

- **Per-GPU compute throughput**, ensuring consistent sustained FLOPS across all GPUs.
- **Intra-node interconnect bandwidth**, validating NVLink connectivity and symmetry through pairwise GPU communication tests.

Unlike conventional burn-in tests that emphasize functional correctness, the single-node sweep explicitly measures sustained throughput and communication efficiency. This design makes subtle degradations—such as uneven GPU throttling or partially degraded NVLink paths—directly observable.

Figure 7 shows representative results in which GPUs within the same node exhibit measurable performance divergence despite passing burn-in tests. Such asymmetries often am-

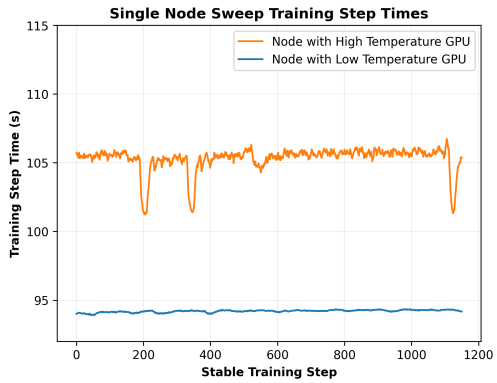

Figure 7. Single-node sweep results showing intra-node performance divergence across GPUs that pass traditional validation.

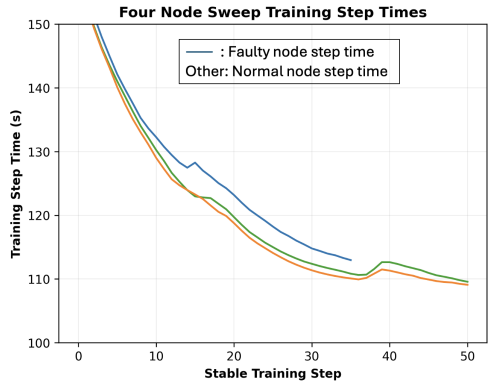

Figure 8. Multi-node (2-node) sweep showing step-time inflation caused by inter-node communication degradation.

plify during collective operations, leading to persistent straggler behavior in distributed training.

5.3 Multi-Node Sweep: Inter-Node Communication Validation

While single-node validation captures local degradation, many stragglers originate from inter-node communication issues, including degraded high-bandwidth network links or transparent rerouting by the communication library. To detect these failures, we perform a multi-node sweep that stresses cross-node collective communication under controlled conditions.

We evaluate configurations with 2, 4, and 8 nodes and find that most communication-related degradations are already detectable in the 2-node setting. Degraded links or misrouted traffic manifest immediately as elevated latency or reduced bandwidth, making minimal-scale sweeps surprisingly effective. Although larger configurations increase sensitivity, they offer diminishing returns relative to their cost.

Figure 8 illustrates how including a faulty node in a 2-node sweep leads to consistent step-time inflation. This behavior scales predictably as more nodes are added, as shown at the cluster level in Figure 9, motivating our default use of 2-node sweeps as a cost-effective verification mechanism.

5.4 Node Sweep Results and Operational Use

The node sweep evaluates end-to-end performance under sustained load, succeeding where traditional NCCL tests fail. Because it does not rely on short-lived correctness checks, the sweep makes bandwidth loss, routing asymmetry, thermal throttling, and intermittent network instability directly observable.

Sweep results are interpreted conservatively. Nodes that pass both single-node and multi-node sweeps are returned to the healthy pool, while nodes that fail remain quarantined for further repair or re-imaging. In practice, a sweep duration of 1–2 hours is sufficient to detect persistent compute and communication degradation.

Rather than periodically sweeping all nodes, we adopt an event-driven strategy in which sweeps are triggered by anomalies detected through online monitoring or following repair actions. This approach minimizes validation overhead while preventing degraded nodes from silently re-entering production workloads.

6 Detection Parameters and Node Triage Workflow

Given the typical fixed compute budget that clusters exhibit, node replacement is a key part to achieving a high utilization. With traditional fail-stop faults, the remediation process usually involves replacing hardware, however given that grey node failures can have multiple root causes that are not immediately visible, a more detailed triage workflow

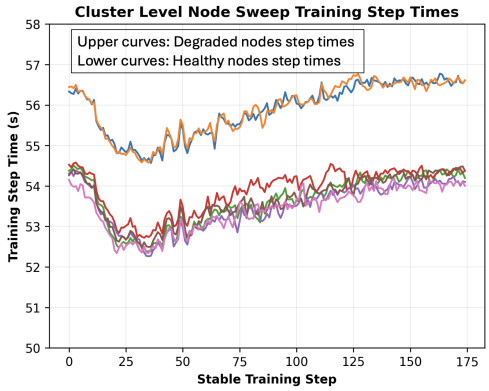

Figure 9. Cluster-level node sweep demonstrating scalability of the offline validation approach as faulty nodes are introduced.

Table 3. Grey Node Classification Rates

METRIC	TK PERCENTAGE	SAMPLE SIZE
FALSE POSITIVE RATE (FPR)	12.4%	124 OUT OF 1000 NEGATIVE SAMPLES
FALSE NEGATIVE RATE (FNR)	7.8%	78 OUT OF 1000 POSITIVE SAMPLES

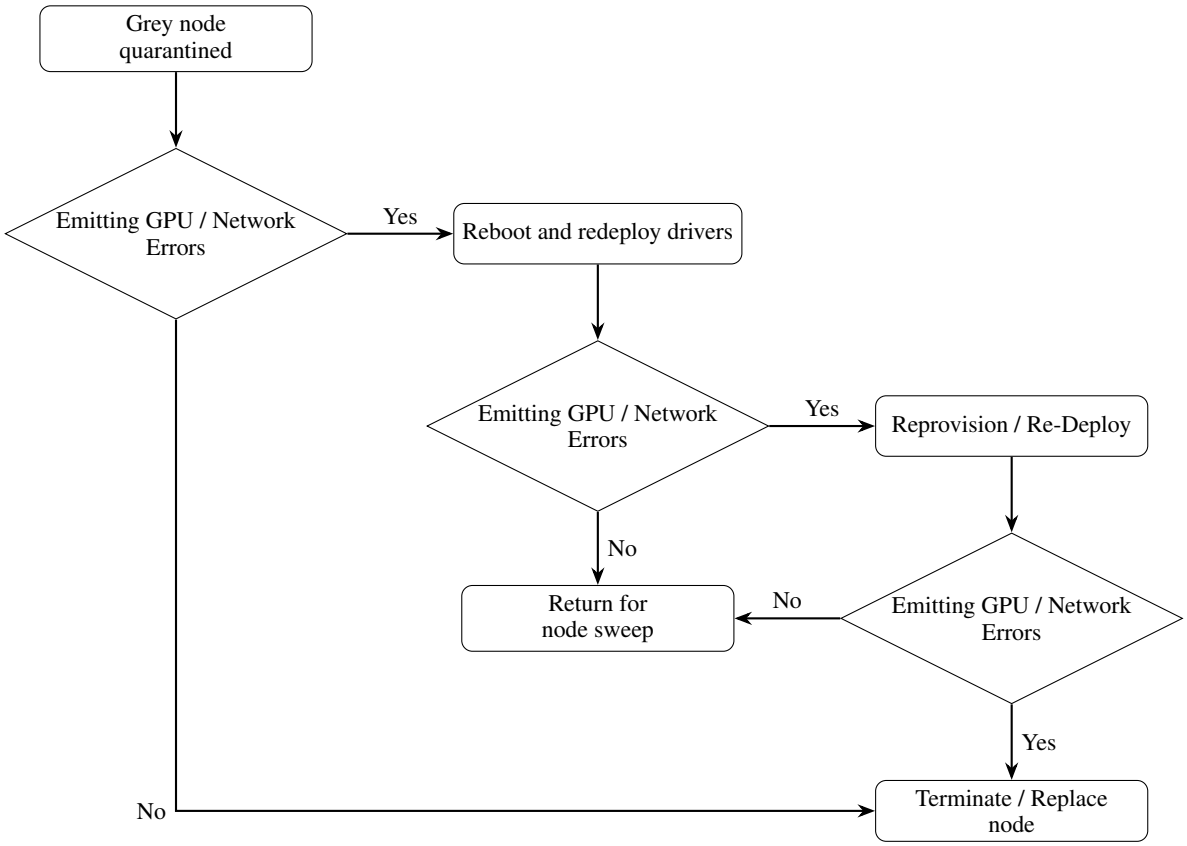

Figure 10. Bad Node Remediation Workflow

is needed. To triage grey nodes, we created a multi tiered triage workflow that can be easily automated and is focused on driving down wasted compute. The workflow is exists as part of the above mentioned automated node health management workflow, specifically within the repair/mitigation section. One important note is that if a specific node was repeatedly being added to triage workflow within a certain time-period, a human manually marked it for termination without going through the entire triage process. Through the empirical observations we determined that three times within a week was sufficient evidence that a node was terminally bad and required replacement, however this may change depending on cluster configurations.

6.1 False Positive and False Negative Rates

The effectiveness of the grey node detection and triage workflow is highly dependent on the accuracy of the underlying classification signals used to trigger remediation actions. False positives result in healthy nodes being unnecessarily removed from service and subjected to remediation or replacement, while false negatives allow degraded nodes to continue consuming compute and negatively impacting workload performance. As such, both error modes represent a direct cost to overall cluster efficiency and must be carefully balanced.

In practice, these error rates informed the design of the tiered triage workflow shown in Table 3, where early remediation stages are intentionally lightweight and reversible. By combining moderately conservative detection thresholds with staged escalation and explicit health re-evaluation, the system limits the operational impact of misclassification while maintaining strong protection against sustained performance degradation.

6.2 Sample Node Triage Workflow

Figure.10 demonstrates a simplified node triage workflow that attempts to remediate grey nodes in a tiered manner, beginning with early termination for nodes that do not exhibit actionable error signals and progressively escalating remediation for those that do. Nodes emitting GPU or network errors are subjected to increasingly invasive recovery steps, with health checks after each stage determining whether the node can safely return to the general sweep pipeline. The workflow ultimately resolves in one of two terminal states, either returning the node for further validation once errors clear or terminating and replacing the node when repeated remediation fails, thereby limiting wasted compute and operational churn.

7 RESULTS

7.1 Cluster Setup

Our experiments were conducted on a high-performance GPU cluster equipped with thousands of GPUs deployed in a traditional 8 GPU node setup. Intra-node communication is handled by NVLink with inter-node communication handled by external network cards. Hardware metrics for the online system are scrapped through a combination of NVIDIA DCGM and custom sidecar processes with custom polling frequencies between 30 seconds and 1 minute. The cluster supports both long-term foundation model pretraining and short-term post-training and inference workloads. The interconnect fabric is based on a high-bandwidth network card, which enables high-throughput and low-latency communication across nodes.

The software environment is built on a distributed training framework with optimized communication primitives, supporting mixed-precision training with FP16 and BF16 and model parallelism strategies such as tensor and pipeline parallelism.

7.2 Cluster-Level Results

The main experimental results are derived from large-scale foundation model pretraining conducted over several months across thousands of GPUs. The model was trained on a diverse corpus containing trillions of tokens spanning multiple domains, including natural language, programming code, and scientific text. We evaluate the proposed system through both quantitative metrics and qualitative observations to assess its impact on large-scale training efficiency and reliability.Qualitatively, the system maintains high reliability during prolonged pretraining runs. Even under large-scale workloads, faulty nodes are effectively detected, isolated, and recovered through the combined online monitoring and offline node sweep workflow, minimizing manual maintenance overhead.Furthermore, as illustrated

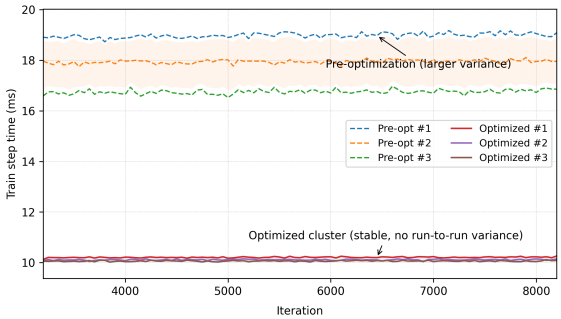

Figure 11. Comparison of run-to-run variance in training step time before and after applying the proposed node health monitoring system.

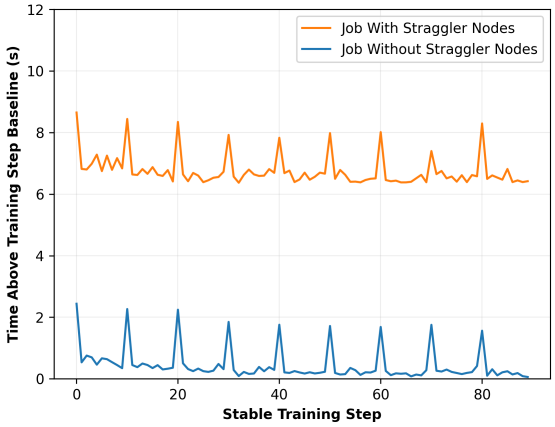

Figure 12. Training step time reduction of 7 seconds after applying node health monitoring and selection strategies.

in Figure 11, the run-to-run variance in training step time is reduced from 20% to only 1%, indicating significantly more stable cluster behavior.

In addition to improved stability, the proposed health monitoring and node selection strategies significantly reduce training latency. As shown in Figure 12, the average training step time decreases from 17 seconds to 10 seconds, saving approximately 7 seconds per step and resulting in an overall **70% improvement in training efficiency**.

To better understand the contribution of each component, we further conduct an ablation study that incrementally enables online monitoring and offline node sweep As summarized in Table 4. Enabling online monitoring alone improves the average MTTF by 14% (from 8.1h to 9.2h), reduces the average human intervention interval by 40% (from 2.0h to 1.2h), and increases MFU from 10% to 14%. Introducing the enhanced offline node sweep yields an additional 82% improvement in average MTTF (from 9.2h to 16.7h), further reduces human intervention time by 58% (from 1.2h to 0.5h), and increases MFU from 14% to 17%.

Table 4. Ablation study of system components

Method	Avg. MTTF (h)	Avg. Human Interval (h)	Avg. MFU	Detects HW Degradation
NCCL / Burnin Tests Only	6.6	5.6	5%	No
NCCL / Burnin + Node Sweep	8.1	2.0	10%	No
NCCL / Burnin + Online Monitoring + Node Sweep	9.2	1.2	14%	Yes
NCCL / Burnin + Online Monitoring + Enhanced Node Sweep	**16.7**	**0.5**	**17%**	Yes

These results demonstrate that the combination of online monitoring and offline node sweep is essential for identifying both acute failures and long-running fail-slow nodes whose degradations may not manifest as explicit hardware errors but can significantly impact training step time. Excluding such nodes accounts for a substantial portion of the MFU and MTTF improvements observed at scale.

7.3 Implications at Larger Scale

As training scale increases, the impact of grey nodes on overall efficiency becomes increasingly amplified. In large-scale deployments with thousands of GPUs and frequent synchronization points, even minor node-level performance degradation can propagate through collective operations and lead to disproportionate slowdowns at the job level. This amplification effect makes timely detection and mitigation of grey nodes substantially more critical at larger scales.

Importantly, the proposed online monitoring and offline node sweep mechanisms are designed to remain effective as system scale grows. The online component continuously operates with low overhead across the cluster, enabling scalable detection of performance anomalies, while the offline node sweep validates isolated nodes independently of overall cluster size. Together, this design allows the framework to scale to larger training systems while maintaining stable performance and manageable operational cost.

8 Conclusion

This paper presents a practical, production-grade system for node health monitoring and straggler detection in large-scale foundation model training. The proposed system continuously monitors critical hardware and network signals during pretraining across thousands of GPUs, and complements this online monitoring with an efficient offline node-sweep component to systematically identify performance-degrading nodes, including edge cases that traditional NCCL tests and GPU burn-in procedures often fail to expose.

Through extensive evaluations on real-world, production-scale training workloads, the system achieves up to 70% improvement in training step efficiency, reduces run-to-run performance variance from 20% to 1%, and substantially improves overall cluster reliability and utilization. These results demonstrate that a system-level, closed-loop approach—combining continuous monitoring with targeted validation—can effectively mitigate fail-slow behaviors and significantly enhance both the performance and operational stability of large-scale GPU clusters. The proposed system is scalable, lightweight, and readily deployable in modern foundation model training environments.

References

Apostolos Kokolis, Michael Kuchnik, J. H. A. K. P. M. F. M. Z. D. S. S. K. S. C.-J. W. Revisiting reliability in large-scale machine learning research clusters. In *Arxiv*, 2025.

Chang, L. et al. FLUX: Fast software-based communication overlap on GPUs through kernel fusion. *arXiv preprint arXiv:2406.06858*, 2024.

Chen, C., Zhang, Z., et al. Centauri: Efficient scheduling for communication-computation overlap in large model training. In *ASPLOS*, 2024.

Chowdhery, A. et al. PaLM: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

Dubey, A. et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Fedus, W., Zoph, B., and Shazeer, N. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *arXiv preprint arXiv:2101.03961*, 2021.

Gao, Y., Zhang, H., et al. An empirical study on low GPU utilization of deep learning jobs. In *ICSE*, 2024.

Google DeepMind. Gemini 1.5 technical report, 2024.

Gunawi, H. S., Hao, M., Suminto, R. O., et al. Fail-slow at scale: Evidence of hardware performance faults in large production systems. *ACM Transactions on Storage*, 2018.

He, Y., Xu, Y., et al. Understanding and mitigating hardware failures in deep learning training systems. In *ISCA*, 2023.

Hu, H., Jiang, C., Zhong, Y., Peng, Y., Wu, C., Zhu, Y., Lin, H., and Guo, C. dpro: A generic profiling and optimization system for expediting distributed dnn training. 2022.

Hu, Q., Yu, Z., et al. Characterization of large language model development in the datacenter. In *NSDI*, 2024.

Huang, P., Jin, C., et al. Gray failure: The achilles' heel of cloud-scale systems. In *HotOS*, 2017.

Huang, Y., Cheng, Y., Bapna, A., et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. In *NeurIPS*, 2019.

Jiang, Z., Gao, M., Lin, Y., et al. Megascale: Scaling large language model training to more than 10,000 GPUs. In *NSDI*, 2024.

Le Scao, T. et al. BLOOM: A 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*, 2022.

Lean Wang1, Huazuo Gao, C. Z. X. S. D. D. Auxiliary-loss-free load balancing strategy for mixture-of-experts. *https://arxiv.org/html/2408.15664v1*, 2024.

Lian, X. et al. Understanding stragglers in distributed machine learning. *arXiv preprint arXiv:2002.06765*, 2020.

Liu, A. et al. Deepseek-V3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Narayanan, D., Harlap, A., Phanishayee, A., et al. Pipedream: Generalized pipeline parallelism for DNN training. In *SOSP*, 2019.

NVIDIA. NVLink: Scalable interconnect for GPU clusters, 2018.

NVIDIA. NVIDIA data center GPU manager (dcgm). https://developer.nvidia.com/dcgm, 2021.

NVIDIA. NVIDIA extended utility diagnostics (eud), 2024a.

NVIDIA. NVIDIA xid errors reference, 2024b.

Rajbhandari, S., Ruwase, O., Rasley, J., and He, Y. Zero: Memory optimizations toward training trillion parameter models. In *SC*, 2020.

Rasley, J., Ruwase, O., et al. Deepspeed: System optimizations for training deep learning models at scale. In *SC*, 2020.

Ren, J., Rajbhandari, S., Aminabadi, R. Y., et al. Zero-offload: Democratizing billion-scale model training. In *USENIX ATC*, 2021.

Rombach, R. et al. High-resolution image synthesis with latent diffusion models. In *CVPR*, 2022.

Shazeer, N. et al. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

Shoeybi, M., Patwary, M., Puri, R., et al. Megatron-LM: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

Stojkovic, J., Zhang, C., Goiri, I., Choukse, E., Qiu, H., Fonseca, R., Torrellas, J., and Bianchini, R. Tapas: Thermal- and power-aware scheduling for llm inference in cloud platforms. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2025.

Xiong, Y., Liu, Y., et al. Superbench: Improving cloud AI infrastructure reliability with proactive validation. In *USENIX ATC*, 2024.

Yulin, J. N. Z. J. X. H. W. X. S. et al. Benchmark test of high performance computing cluster based on hpcc. In *ICCECE*, 2021.

Zhou, Y. et al. Towards practical monitoring for large-scale deep learning clusters. In *MLSys*, 2023.

Ziji Shi, Le Jiang, A. W. J. Z. C. W. Y. L. X. X. W. L. J. L. Tapas: Fast and automatic derivation of tensor parallel strategies for large neural networks. In *ICPP*, 2025.