

Debugging Billion-Core AI Clusters: Distributed Crash Diagnosis for Heterogeneous Accelerator Systems

Mohammad Shuab Siddique¹

¹ Independent Researcher

Email ID: shuaib.siddiqi@gmail.com

Abstract: Correlated kernel panics are a common distributed system reliability issue because modern AI systems involve billions of logical processing cores on a variety of NVIDIA, TPU, AMD, and specialized accelerator platforms. This study examined Cluster Crash Diagnosis (CCD) as an architecture that uses Linux kdump technology and transforms it into a scalable, secure fleet telemetry system. The study involved two production validations, using an 8,000-node NVIDIA H100 cluster and a 512-node TPU v5e pod, respectively. CCD introduced 2 GB of crashkernel reservation, vendor sidecars, local persistence of vmcore, LZO compression, 50 Mbps transport control, exponential backoff, hashing stack with a hash function that uses SHA256, normalized Levenshtein clustering at a threshold of $\tau = 0.85$, vendor-neutral classification, and topology-aware analytics. In the H100 case, the crash rate went up from 0.5 crashes/day to 15 crashes/day after a patch to the operating system; CCD reduced diagnosis time from 6 hours to 8 minutes, a 45× faster turnaround. In the TPU case, all crashes observed were associated with the same failing top-of-rack optical switch. Overall, CCD lowered the MTTD by 70-85% while improving multi-tenant security. These results demonstrate how coordinated capture, transport, deduplication, correlation, and access controls can help enhance resilience in exascale AI operations. Future work should extend predictive topology correlation and cross-vendor validation.

Keywords: Cluster Crash Diagnosis (CCD), Heterogeneous Accelerator Systems, Linux Kdump, Distributed Crash Diagnosis, Kernel Panic Deduplication, Exascale AI Clusters

1. Introduction

Artificial Intelligence infrastructure has gone from being a computing platform designed for experimentation to exascale computing infrastructure that can coordinate billions of logical processing cores across ~10,000 to ~24,000 nodes. Heterogeneous nodes, consisting of NVIDIA H100 GPUs, Google TPUs, AMD MI300 accelerators and tensor engines, work together via NVLink, InfiniBand and RoCEv2 fabrics in large language model training. At this scale, kernel panics are not uncommon in operations. The propagation of memory corruption, PCIe faults, failures in the accelerator drivers, and interconnect degradation can trigger simultaneous impact to multiple nodes via multiple fabrics. The way to store the kernel state as a vmcore is still Linux kdump, which is the kexec boot of a capture kernel. It is important to note, however, that when 500 such nodes panic at once and try to upload 500+ vmcore files of several gigabytes,

the common storage backend can be overloaded, producing lots of diagnostic traffic competing with the remaining workload.

The biggest challenge is the gap between node-local crash diagnostics and cluster-scale heterogeneous AI appliances. The idea for conventional kdump was to focus on the failure of a single server, not a coordinated failure of several accelerators. It can take another 4–8 hours to diagnose using manual SSH, serial console, dmesg, and ad-hoc log analysis. The simultaneous vmcore transport can cause a self-inflicted storage or fabric denial-of-service (DoS) condition. Furthermore, accelerator vendors publish incompatible telemetry, and GPU/TPU registers use themselves up after reboots. Any duplicate occurrences of the same defect are incidented separately, and sensitive multi-tenant memory is exposed in raw vmcores in existing workflows. The study thus focuses on the challenge of turning local

crash capture into distributed, scalable, and secure fleet telemetry.

The study had four goals that address: identifying conventional kdump limitations in 10,000+ node heterogeneous clusters. To see Cluster Crash Diagnosis for looking at synchronicity of panic capture and decoupling of it from asynchronicity of backpressure-aware transport. To evaluate the different methods for fleet-level deduplication: SHA256 stack hashing, normalized Levenshtein similarity, and vendor-agnostic telemetry normalization. To assess the production cases to be implemented by NVIDIA H100 and TPU v5e, leveraging MTTD, crash grouping, topology correlation, and operational resilience in CCD.

The study included Linux kernel panics, kdump, and hardware and software failures when using accelerators, product cases with NVIDIA H100 and TPU v5e, as well as AMD MI300X and Cerebras CS-3, as architectural contexts. It also analyzed crash kernel reservation, VM core compression & transport, deduplication, fleet analytics, vendor-neutral JSON normalization, Zero-Trust crash-dump security, multi-tenant isolation, and exascale diagnosis. There was no crash dataset found in a raw public format, as the source is a case-study observation dataset rather than an event-level downloadable dataset.

This study is structured in various chapters. Chapter 2 discusses how kernel panics are handled, kdump,

heterogeneous accelerators, and failure probability. The data sources, case-selection technique, equations, and analytical procedures are described in Chapter 3. Chapter 4 features CCD's distributed architecture, hardware abstraction, fleet analytics, transport controls, and security design. Chapter 5 provides results and discussion of the NVIDIA and TPU validation procedures, including diagnostic efficiency and future research priorities. Chapter 6 summarizes the implications of the study for scalable, secure, and resilient crash diagnosis in exascale AI systems.

2. Literature Review

2.1 The Anatomy of a Kernel Panic

Kernel panic is the final action taken by the operating system when the system can no longer run without causing harm to state or data. Four failure classes dominate the diagnostic path in heterogeneous AI clusters. Invalid memory accesses include null-pointer accesses in optimized accelerator drivers [1]. Synchronization failures include deadlocks and race conditions, especially if the Luster or GPFS services have very heavy checkpoint traffic. Hardware uncorrectable errors refer to Machine Check Exceptions, as well as errors that start in GPU High Bandwidth Memory [2]. Bus and interconnect errors comprise Non-Maskable Interrupts, degraded PCIe links, and NVLink signal loss.

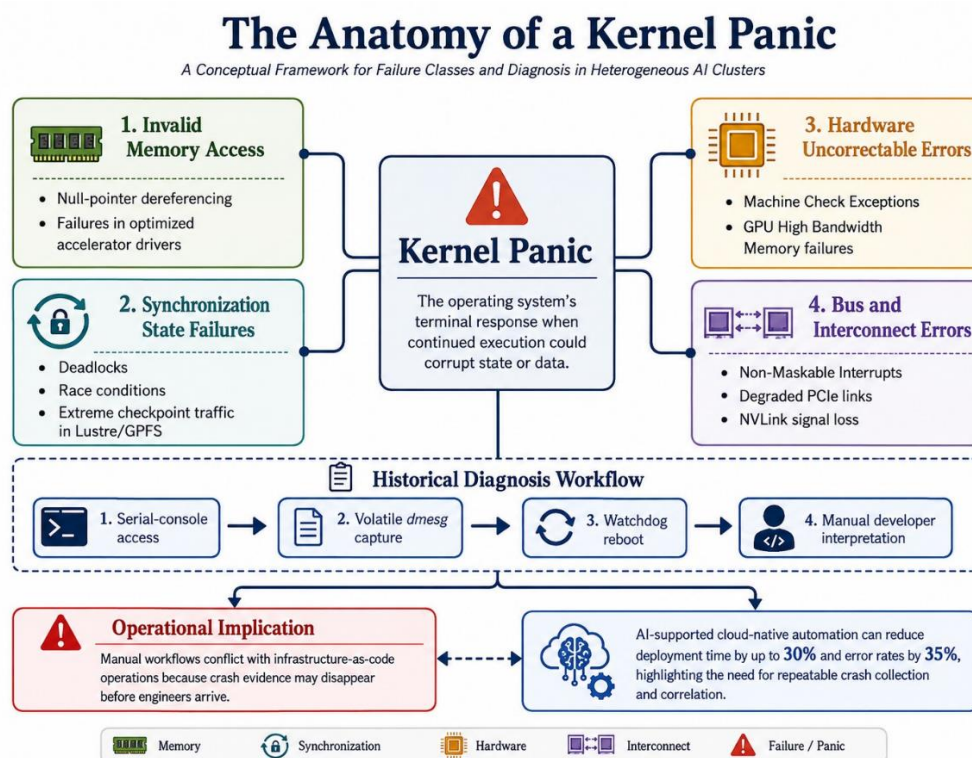


Figure 1: Conceptual framework illustrating kernel panic failure classes, diagnosis workflow, and implications in AI clusters.

Figure 1 shows the relationships among the four predominant failure classes in heterogeneous AI clusters, the diagnostic process, and their operational implications in this architecture. It focuses on invalid memory access, synchronization failure, uncorrectable hardware errors, and bus and interconnect errors, and highlights the need for automated, repeatable crash collection and correlation at scale, followed by serial console inspection, dmesg capture, watchdog reboot, and manual interpretation.

Previously, operators had to add and connect serial consoles, manually grab the dmesg output before watchdog reboot, and manually analyze evidence [3]. This kind of activity makes it difficult to apply automated actions to infrastructure-as-code, as crash data could be lost before engineers make it to the scene. Automated decision pipelines (AI) can cut deployment time by 30% and error rates by 35%, highlighting the operational benefits of AI-powered cloud automation [4; 5]. It is therefore applicable to crash triage as well, where repeatable collection and correlation can replace manual node-by-node investigation.

2.2 Kdump and Memory Reservations at Exascale

Kdump is a solution that keeps the state of a terminal kernel in memory prior to its crash, and instead launches a secondary kernel for capture after the crash boot. GRUB_CMDLINE_LINUX crashkernel is the main parameter in the Linux kernel that is used to reserve RAM first. When a panic occurs, kexec is invoked, skipping the BIOS/UEFI start-up process, booting the capture kernel, and presenting the memory belonging to the failed kernel as an ELF-formatted file in */proc/vmcore*, allowing makedumpfile to compress the memory in the same way [6]. While a quite typical enterprise configuration reserves 128-256 MB, the heterogeneous configuration reviewed here demands 1-2 GB and is deemed sufficient for accelerator nodes with 2 GB.

This distinction can also impact systems with 8 NVIDIA H100 GPUs, as MMIO mappings are typically hundreds of gigabytes large. Without enough memory in capture-kernels, this could lead to OOM errors causing loss of the diagnostic record [7]. Using LZO compression

can compress the volume of the created vmcores by about 40-60%. In high-value authentication systems, the literature highlights the importance of secure storage and secure transmission of sensitive data, further consolidating the need to protect memory captures [8]. Similarly, researchers consider Zero Trust very important for critical infrastructure, and it can enable controlled access to crash artifacts under production failure conditions [9]. This criterion is of paramount importance when automated recovery encompasses evidence while maintaining continuity of surviving training workloads.

2.3 The Heterogeneous Accelerator Landscape

It adds the complexity of heterogeneous acceleration, as compute scale, memory organization, software stacks, and interconnects vary greatly between vendors and may make diagnosis difficult. NVIDIA Hopper H100 offers more than 14,000 CUDA cores and NVLink 4.0 bandwidth of 900 GB/s, and 80 GB HBM3 per GPU, and failure cases are provided via CUDA drivers and MIG contexts [10]. Google TPU v5e combines matrix-multiply units, 16-32 GB HBM, an Inter-Chip Interconnect network, the XLA compiler, and failure domains via mesh routing. AMD Instinct MI300X offers 304 compute units, 192 GB of HBM3, AMD Infinity Fabric, and ROCm/HIP software, whereas Cerebras CS-3 comprises around 900,000 on-chip cores, 1.2 TB of on-wafer SRAM, SwarmX fabric, and a wafer-scale SDK [11; 12].

Different hardware state probes and failure interpretation are required for these differences. Further accelerating the trend of divergence between interconnect and memory design specifications for accelerators, Google's newer Ironwood TPU uses 192 GB HBM [13]. In reality, NVIDIA NVML/NVLink diagnostics and TPU ICI telemetry, ROCm/HIP diagnostics, and wafer-scale telemetry are not exactly equivalent in some cases and are not native to each other. By doing so, scattered and separate triage pipelines create a fragmentation of fleet observability and encourage uniformity of triage schemas across massive, multi-vendor production fleets.

Table 1: Comparison of Compute, Memory, Interconnect, and Failure Architectures

Architecture	Compute Scale	Memory Layout	Interconnect	Failure Domain
NVIDIA Hopper (H100)	14,000+ CUDA Cores	80GB HBM3 per GPU	NVLink 4.0 (900 GB/s)	CUDA Driver, MIG context
Google TPU v5e	Matrix Multiply Units	16–32GB HBM	Inter-Chip Interconnect (ICI)	XLA Compiler, Mesh routing
AMD Instinct MI300X	304 Compute Units	192GB HBM3	Infinity Fabric	ROCm stack, HIP runtime
Cerebras CS-3	900,000 Cores on-chip	1.2TB SRAM (On-Wafer)	SwarmX Fabric	Wafer-scale SDK

Table 1 compares four accelerator architectures that offer compute scale, a range of memory layouts, different interconnect technologies, and vendor-specific support for particular failure domains within a distributed AI computing system.

2.4 Statistical Probability of Failure at Scale

At fleet scale, reliability is governed by probability rather than isolated exceptions. Having reviewed this model, it is assumed that the node failures are exponentially distributed with $P(t) = 1 - e^{-\lambda t}$, where $P(t)$ is the probability of failure by time t , and λ is the node failure rate parameter; N is the number of nodes that are considered in the fleet.

$$P(t) = 1 - e^{-\lambda t}$$

Even with the individual nodes looking very reliable, a 10,000-node environment is exposed continuously. The example provided in the source claims it can have more than a 0.01% node failure rate per day, and the manuscript also indicates that one out of every two, or more, training runs with more than 10,000 GPUs can result in job-stopping failures [14]. The impact is greater with operations, as manually investigating may take about 4 to 8 hours to establish the root cause. For public systems supported by AI, accountability, transparency, and reliable governance are just as important as efficiency, and these factors are crucial when automated infrastructure decisions have economic consequences for valuable shared resources [15]. The focus of engineering then moves from just fixing what is broken to quickly locating causal domains, maintaining evidence, and having a

remediation audit trail across the cluster.

2.5 Limitations and Research Gaps in the Literature

There are currently five practice gaps. First, node-level crash capture is not trivially scalable to cases where hundreds of nodes are not crashed simultaneously. Second, for traditional kdump, there is no cluster-level backpressure to control concurrent transport of vmcore. Analysis is also split across GPU, TPU, and accelerator stacks by vendor. Minor differences in addresses, symbols, or compiler-generated stack frames can cause what is one defect manifests as multiple incidents.

Tenant data, model state, credentials, or cryptographic material can also be included in raw VM cores, exposing privacy and security issues. These limits make it important to have an integrated architecture that leverages trustworthy capture, congestion-aware transport, vendor-neutral normalization, mathematical deduplication, fleet-level correlation, and security controls. To address this gap, CCD has been created as a distributed telemetry system, specific to crash diagnosis and not just server debugging.

3. Methods and Techniques

3.1 Data Collection Methods

The study adopted a case study, which was retrospective in nature and comparative in production, to discuss crash diagnosis in two accelerator conditions as described in the source manuscript, which are of different natures. Evidence gathered from operational validation records is not based on an event-level analysis

in the public domain. Consequently, the primary analytical sample was $n = 2$ cases, purposively selected to represent differing accelerator technologies and failure mechanisms. In Case 1, an 8k node NVIDIA H100 cluster is found to suffer software and kernel-driver regression due to an operating-system patch [16]. Fifteen post-patch crashes were evaluated, and a cumulative increase in daily crash rate was observed from 0.5 crashes/day to 15 crashes/day; the crash-generated compilers produced 12 seemingly unique stack traces, all of which were offset-related.

In Case 2, a 512-node Google TPU v5e pod gave rise to around two ICI mesh-routing panics a week owing to degrading physical interconnects [17]. The total number of TPU crash events was not reported in the manuscript and was not inferred. The total number of documented nodes in the various cases together is 8,512. The sampling approach prioritized operational contrast over statistical population inference. Reliability evidence at a larger scale was used to contextualize the design as well: researchers have analyzed 11 months of such failures from two production ML clusters, spanning roughly 4M jobs and over 150M A100 GPU-hours, to illustrate the necessity that failure analysis needs to remain feasible at scales much larger than those for training [18].

3.2 Data Analysis

These crash records were analyzed in four successive steps: In the first step, the exact panic signatures were created from the panic string and a Normalized Call Frame as the result of cryptographic hashing.

$$H_{stack} = \text{SHA256}(\text{panic_string} || \text{normalized_call_frames})$$

Similar signatures were clustered together to minimize processing of similar failures. The resulting H_{stack} was considered a determinate challenge key for exact duplicates, while near-duplicates containing the same data were addressed in the fuzzy stage. It was a two-stage process that distinguished exact equality from similarity of meaning and minimized the chance of address noise, symbol formatting, or compiler-created offsets causing a single failure to show as multiple incidences. Fuzzy stack-trace analysis was implemented using normalized Levenshtein similarity to prevent separate incidents

caused by address changes and differences in compiler inlining.

$$\text{Similarity} = 1 - \frac{\text{Levenshtein}(S_1, S_2)}{\max(|S_1|, |S_2|)} \geq \tau$$

$$\tau = 0.85$$

Records that had a similarity of at least $\tau = 0.85$ were grouped in the same conceptual cluster. Vendor-specific errors were also standardized to a common signature to enter a common analytical vocabulary, such as *hardware_error/interconnect* for GPU and TPU errors. Additionally, clustered crashes were compared with respect to their Physical and Topological causality, Driver version, Accelerator cohort, and Kernel version. The method additionally kept the conventional crash-heritage course. The researchers document that *kdump* loads a second kernel in reserved memory using *kexec*, saves a memory snapshot of the stopped kernel as *vmcore* for later processing, and crash utilities can extract the memory snapshot to obtain the kernel stack and system-state information [19]. This is the mechanism through which CCD grouping and correlation worked and to which low-level evidence was attached.

3.3 Experimental Design and Production Validation

The study utilized a comparative production-validation design and not simulation. In the NVIDIA case, this was compared to the legacy approach to manual diagnosis, automated ingestion, stack grouping, kernel-version correlation, and Mean Time to Diagnosis (MTTD) [20]. The evaluation concluded that they have little reason to believe that the 15 crashes and 12 apparent trace variants identified in the study could be grouped into one common failure cluster and correlated to the software patched brain region. In the TPU case, the baseline procedure was to manually parse proprietary telemetry, CCD-normalize ICI error codes, and correlate the affected nodes and physical topology [21]. Table 2 compares the production-validation design, summarizing NVIDIA versus TPU, basic approaches and evaluation goals, crash-capture prerequisites, and reported operational outcomes across various environments.

Table 2: Comparative Production-Validation design, diagnostic procedures, evaluation criteria, and recorded failure outcomes of accelerators.

Validation Case	Baseline Approach	CCD Approach	Evaluation Focus	Reported Outcome
NVIDIA H100 Kernel Regression	Manual diagnosis and individual crash inspection	Automated ingestion, stack grouping, kernel-version correlation, and root-cause analysis	Whether 15 crashes and 12 apparent stack-trace variants could be consolidated and linked to the patched kernel cohort	Variant traces were evaluated as a common failure cluster associated with the software patch
TPU Optical Mesh Degradation	Manual parsing of proprietary TPU telemetry	ICI error normalization and physical-topology correlation	Whether repeated mesh-routing failures converged on a common infrastructure component	Failures were traced to an implicated optical switch, and replacement was evaluated against recurrence
Crash-Capture Prerequisite	Conventional node-level crash handling	Kdump with reserved capture-kernel memory	Ensuring reproducible vmcore capture before production diagnosis	crashkernel= memory reservation was treated as a required precondition for validation
Validation Principle	Simulation or synthetic fault generation not used	Comparative production-case analysis	Preserve source-observed operational evidence	No synthetic crash counts, artificial failure injections, or unreported TPU event totals were introduced

Validation studied whether surprising cycles of network failures due to mesh-routing converged to a common infrastructure element, and whether failure of the suggested optical switch eliminated the observed panic pattern [22]. No synthetic crash counts or synthetic failure injections were added, nor were there any unreported TPU event totals added. It turned out that kdump configuration was considered an operational prerequisite since the capture kernel demands memory reservations at boot time. Researchers state that crashkernel memory is reserved during boot and that the size of a specified memory area can be configured in the `crashkernel=` kernel parameter during boot in a reproducible manner before administering the production system [23].

3.4 Technical Evaluation Metrics and Transport Model

For technical evaluation, operational metrics were used that were related to diagnosis speed, event consolidation, and transport safety. The metrics that were measured were MTTD, crash-event frequency, number of apparent variants of crash-log file stack-traces, deduplication outcome, topology-correlation percentage, vmcore transport behavior, compression efficiency, and preservation of shared storage and fabric capacity [24; 25]. In transport analysis, no measurements were added for the source parameters. This was achieved with a 50 GB vmcore and by limiting each node's upload rate to around 50 Mbps in the cascade scenario. If central storage returned an HTTP 429 Too Many Requests response or excessive I/O wait was detected, asynchronous transmission entered an exponential backoff with jitter.

$$\text{Wait Time} = \text{Base} \times (2^{\text{attempt}}) + \text{Jitter}$$

Crashing files were still kept at home until the transfer could be resumed. LZO compression was assumed to shrink the size of the vmcore subsequently by 40-60%, as stated in the source manuscript. The design was evaluated to determine if these controls were effective in preventing any undesired competition with the diagnostic traffic and the gradient-synchronization activities of the surviving nodes. Therefore, transport effectiveness was defined as controlled bandwidth and retry behavior, local durability, and reduced dump volume, not as an invented benchmark of throughput. Then, all the results were compared between the two accelerator cases to understand whether CCD could increase the level of diagnostic consolidation with reasonable cluster operational safeguards in practice.

4. CCD Architecture, Abstraction, and Security

4.1 Layer 1: Node-Level Telemetry and Agent Hooking

Cluster Crash Diagnosis was designed so that the telemetry system overview was multi-layered, with local panic handling separated from reporting to fleet analysis. In Layer 1, 'ccd-node-agent' ran as a privileged process on all nodes and tested the node crash readiness prior to crashes. Pre-flight checks identified a 2 GB 'crashkernel' reservation, the current GRUB configuration, compatibility of the current kernel with the capture image, and integrity of the 'kexec' payload. These checks prevented production panic leading to either a capture-kernel failure or symbol mismatch. Vendor sidecars-maintained accelerator state prior to hard reboot, which otherwise was lost.

The NVIDIA nodes ran `nvidia-smi -q -d ERROR`, and the TPU nodes accessed ICI error registers and used `nvidia-smi` to get volatile hardware information and store it in a JSON file. Next, the capture kernel booted itself via `kexec`, exported the problem kernel via `/proc/vmcore`, fed it to `makedumpfile`, passed the result to the LZO compression, and saved the result in `/var/crash/`. The design provided automated recovery instead of manual exception handling. A study of researchers found that self-healing

automation cut down on locator-related failures by 40-60% and maintenance efforts by 30-50%, demonstrating the role of automated corrective actions in minimizing repetitive human intervention in changing software environments [26].

4.2 Layer 2: Backpressure-Aware Transport

Large crash artifacts were moved with Layer 2 post-local capture. The design assumed that several hundred nodes can generate vmcores at the same time, and that these will cause excessive pressure on Luster or NFS metadata servers if all of the nodes upload immediately. Thus CCD stripped capture from transport and policed each artifact where it was created until network and storage environments allowed for transport [27]. The per-node diagnostic bandwidth was limited to ~ 50 Mbps. Through HTTP 429 Too Many Requests responses and increasing I/O wait, central backpressure was discovered, followed by exponential back-off with jitter for retries. The transport model retained the relationship shown below.

$$\text{Wait Time} = \text{Base} * (2^{\text{attempt}}) + \text{Jitter}$$

In terms of operations, this also helped to reduce the number of times an upload was retried; it was deferred rather than retried, while also ensuring that storage throttling did not result in a loss of diagnostic evidence, as data was stored locally first until it returned to normal.

Both local persistence and randomizing of the timing of retries were utilized to dampen synchronized retransmission bursts. This separation proved important, especially without sacrificing the gradient synchronization functionality of distributed training, which keeps surviving nodes alive using fabric [28]. The design focused on ongoing workload traffic rather than transitions to bulk crashes. CDD did not assume unlimited storage capacity; instead, it included congestion signals directly in the crash pipeline to enable transport behavior to adjust to the infrastructure on the ground.

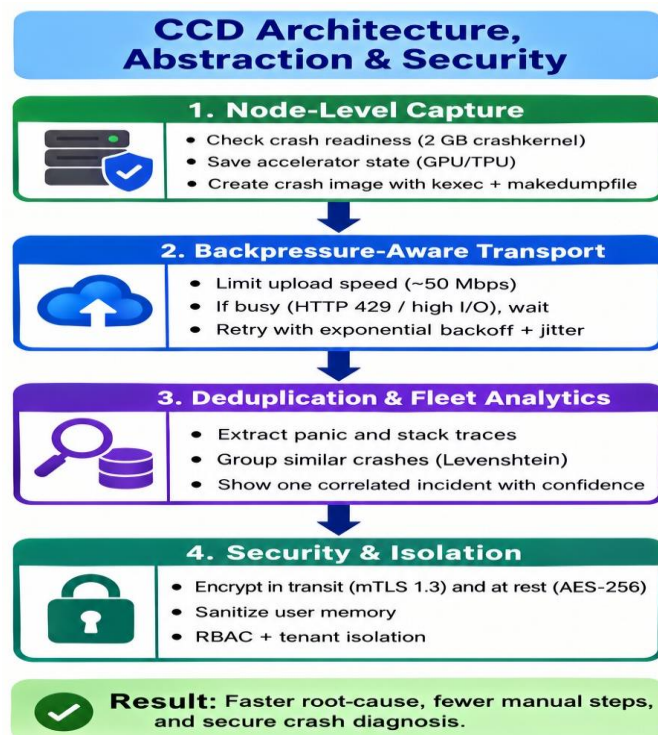


Figure 2: An overview of CCD architecture highlighting node capture, transport, analytics, security, and isolation controls.

Figure 2 shows the Cluster Crash Diagnosis, which shows Crash Information through 4 co-ordinated stages. Node-level capture manages vmcore and accelerator metadata, transport provides safe, asynchronous data delivery, analytics detects and eliminates duplication and related failures, and security secures data and limits access to sensitive information. The final result of this workflow is faster issue resolution, less downtime, and better protection for sensitive data across a variety of AI cluster environments.

4.3 Layer 3 and Layer 4: Deduplication, Indexing, Fleet Analytics

Crash artifacts were scaled to fleet-level events in layers 3 and 4. The Kubernetes Indexer Pods received VM cores and JSON sidecars, extracted panic strings and stacks, calculated a SHA256 signature for each VM core and clustered it based on the Levenshtein distance, and attributed the cluster identifier [29]. During this processing, multiple sets of causal groups were created that divided several node-specific records into smaller, causal groups. The metadata was normalized, loaded onto Elasticsearch, and analyzed in Grafana at the fleet level.

A representative incident was reported, and 500 nodes were identified with 99% confidence, with a timeout in the credit queue matching the kernel version 6.6.0. Davis AI is impressive for its ability to correlate across metrics, logs,

events, topology, and context, accelerating root cause determination and enabling more actionable remediation [30]. CCD's webhooks could also temporarily take over Kubernetes scheduling for affected cohorts of nodes during the investigation process.

4.4 Vendor-Agnostic Schema, Security, Privacy, and Multi-Tenant Isolation

Each crash was represented by a typed, JSON-encoded abstraction layer that was independent of any vendor and contained such pieces of information as node identity, time of crash, crash type, description of hardware data inventory, orchestration context, and deduplication data. The schema enabled vendor-specific context to be retained in a correlation pipeline when an NVIDIA PCI bus failure or TPU interconnect fault occurs, or when another accelerator event occurs [31]. For instance, the classification *'hardware_error/interconnect'* can exist alongside an NVIDIA H100-SXM model, driver version 550.40, a degraded interconnect state, job identifier, tenant identifier, stack hash, and Levenshtein cluster ID.

Security controls considered raw vmcores to be potentially sensitive memory content, including proprietary model weights, personally identifiable information, cryptographic keys, and credentials such as

authentication tokens. The transport used mTLS version 1.3, and the stored dumps were encrypted using AES-256-GCM with individual keys employed for each tenant, which a KMS managed [32]. Prior to human access sanitization based on `task_struct`, user-space pages were identified, and zeros were placed in those pages, that is, as long as kernel pages were not required for diagnostic purposes. Normalized metadata given to routine users via RBAC, with raw vmcore access audited via break glass procedure.

Figure 3 shows the schema of a JSON crash event that CCD uses to convert heterogeneous accelerator crashes to a vendor-agnostic form. It stores node identity, timestamp, root cause classification, severity, panic details, CPU and accelerator inventory, driver and interconnect status, orchestration identity, and deduplication metadata. The schema allows NVIDIA and other accelerator events to feed into a unified correlation pipeline, retaining the fleet context, hardware, workload, and crash-signature data to enable fleet-level analysis and diagnosis.

```
{
  "crash_event": {
    "node_id": "ai-train-gpu-1024",
    "timestamp": "2025-10-15T08:33:12Z",
    "classification": {
      "root_cause_class": "hardware_error/interconnect",
      "severity": "critical",
      "panic_string": "NMI: PCI Bus Error (Fatal)"
    },
    "hardware_inventory": {
      "cpu": {"model": "AMD EPYC 9004", "cores": 128},
      "accelerators": [
        {
          "type": "gpu",
          "vendor": "nvidia",
          "model": "H100-SXM",
          "driver_version": "550.40",
          "interconnect_status": "degraded"
        }
      ]
    },
    "orchestration": {
      "job_id": "llm_pretrain_run_7",
      "tenant_id": "research_nlp"
    },
    "deduplication": {
      "stack_hash": "a9f3b2c1...",
      "levenshtein_cluster_id": 42
    }
  }
}
```

Figure 3: Vendor-agnostic JSON crash-event schema integrating hardware, orchestration, classification, and deduplication metadata for CCD analysis.

Researchers believe that there are many concerns to address when deploying shared AIGC infrastructure such as tenant data isolation, access control, resource separation, and performance interference, necessitating the need for simultaneous confidentiality and workload isolation [33]. In reality, SRE teams with routine access to metadata about failures from a system that has tenants will see only normalized data, not unrestricted access to memory used by other applications. These controls were GDPR- and HIPAA-focused, with a crash pipeline aligned. As such, the architecture addressed capture, transport, normalization, deduplication, fleet correlation, and security all in a distributed architecture, optimized to

apply to a variety of clustered accelerators located in different types of production environments.

5. Results and Discussion

5.1 NVIDIA H100 Kernel Regression Results

When an OS patch was added to an 8,000-node H100 cluster, NVIDIA production validation resulted in a dramatic change in the failure rate. The observed rate rose from 0.5 crashes/day to 15 crashes/day, or a 30× increase. Downstream engineers handled it manually, using SSH and then took dmesg output and cross-referenced NVLink logs [34]. Compiler offsets caused 12

seemingly unrelated stack traces, resulting in 6 hours of diagnosis. CCD signed off on all 15 crashes, normalized the traces, and consolidated the 12 apparent variants into a single conceptual cluster. Table 3 compares the

traditional and CCD workflows for regression with NVIDIA H100, including crash rates, diagnostic tools, stack trace consolidation, root cause analysis, MTTD reduction, remediation, and operational recovery.

Table 3: NVIDIA H100 kernel regression results showing crash rates, diagnosis methods, MTTD, and remediation outcomes.

Metric	Traditional Workflow	CCD Workflow	Observed Result
Cluster size	8,000 H100 nodes	8,000 H100 nodes	Same production environment
Crash rate before patch	0.5 crashes/day	0.5 crashes/day	Baseline condition
Crash rate after patch	15 crashes/day	15 crashes/day	30× increase
Crash sample analyzed	Individual manual inspection	Automated ingestion of all crashes	15 crashes analyzed
Stack-trace variants	12 traces appeared unrelated	12 variants grouped together	1 conceptual crash cluster
Diagnostic method	SSH, dmesg, NVLink log cross-referencing	Trace normalization and kernel-version correlation	Automated fleet-level correlation
Identified root cause	Delayed manual identification	GPU-driver memory allocation incompatibility with new kernel scheduler	Patch identified as common factor
MTTD Remediation	6 hours Manual investigation before action	8 minutes Kernel patch rollback	45× improvement Faster operational recovery

In a version correlation step, the cluster was then linked to the patched kernel. It is reported that this is due to the lack of support for the GPU-driver memory allocation method in the new kernel scheduler. Rollback was followed by an 8-minute diagnosis, or a 45× MTTD improvement. One study looked at the use of machine learning for governance in ultra-low-latency financial networks, where automated correlation and swift action control are paramount compared to manual auditing at a slower pace [35]. The overall comparison proved the practicality of fleet-level software version correlation.

5.2 TPU Optical Mesh Degradation Results

The second was a 512-node TPU v5e pod running a large workload of TPU vision-model training. On average, about twice a week, ICI mesh-routing panics took place. The resolution took around 4 days per incident and was

made by traditional parsing of millions of proprietary TPU telemetry lines by support engineers. CCD standardized the error code to match typical diagnostics for TPU-specific hardware errors to the generic *Hardware Error/Interconnect* classification and mapped correlating nodes to the actual data-center topology [36]. That same spatial analysis revealed that 100% of the crashes observed were on nodes associated with a single top-of-rack optical switch. Panics were eliminated by replacing the switch, and poor optics was found to be a common intermittent physical fault.

This refutation showed that it was not possible to standardize the hardware context that is useful to the vendor. Rather, it occurred when failures in proprietary TPU weak spots had some way slipped through a hole in the analytical filter for GPU failures. Similarly, the

researchers compared CCD's approach of employing normalized telemetry and dashboards to diagnose infrastructure more quickly to the use of embedded analytics in the centralized operation to generate decision-

support outputs [37]. It also provided instant physical topology regions as evidence of health diagnostics.

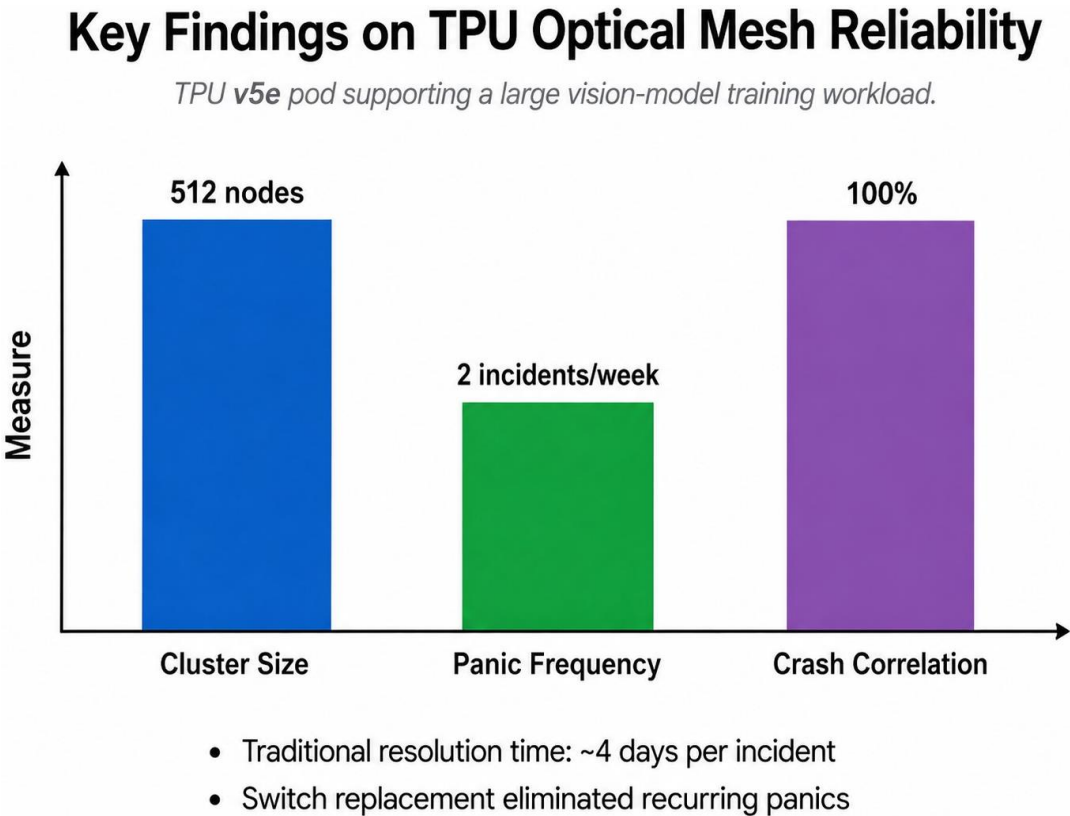


Figure 4: Key findings on TPU optical mesh degradation, panic frequency, crash correlation, and resolution.

Figure 4 highlights results from the TPU optical mesh degradation case, including the 512-node TPU v5e pod, ~2 ICI mesh-routing panic incidents/week, and 100% crash correlation with one top-of-rack optical switch. It also highlights the traditional 4-day resolution period and demonstrates that the switch replacement eliminated repeat panics across the affected training environment.

5.3 Cross-Case Discussion and Operational Implications

In both cases, CCD generated operational gains by decomposing the architectures, not just with a single algorithm. The source reported an overall MTTD reduction from legacy approaches of 70-85 percent. The

spatial correlation for the H100 case dropped from 6 hrs to 8 minutes, while for the TPU case, a 100% spatial correlation could be reached with one switch cohort [38]. The fuzzy deduplication threshold was 0.85, and LZO compression got the vmcore size down by ~40-60%. A successful flow included capturing nodes, locally storing them, moving nodes with control, deduplicating nodes, normalizing nodes, fleet correlating nodes, and finally displaying in the SRE dashboard. Table 4 provides a cross-comparison of CCD performance to identify the benefits of MTTD reduction, deduplication, shared diagnostic workflows, failure correlation, vmcore compression, and operational benefits in the NVIDIA H100 and NVIDIA TPU v5e environments.

Table 4: Cross-case comparison of CCD performance, diagnostic efficiency, deduplication, correlation, compression, and operational resilience outcomes.

Evaluation Metric	NVIDIA H100 Case	TPU v5e Case	Cross-Case Implication
MTTD performance	Reduced from 6 hours to 8 minutes	Traditional diagnosis required extended	Overall CCD MTTD reduction of 70–85%

		manual investigation	
Failure correlation	Kernel-version correlation linked crashes to the patched software cohort	100% of observed crashes correlated with one switch cohort	Fleet correlation accelerated root-cause identification
Deduplication	Multiple stack variants consolidated into one conceptual cluster	Vendor-specific ICI errors normalized for common analysis	Fuzzy matching used $\tau = 0.85$
Vmcore handling	Automated capture and controlled transport supported analysis	Centralized telemetry reduced manual parsing	LZO compression reduced vmcore size by 40–60%
Operational workflow	Capture → persistence → transport → normalization → correlation	Same analytical sequence applied across TPU telemetry	Common cross-vendor diagnostic pipeline
Operational benefits	Faster rollback and reduced alert volume	Faster infrastructure fault localization	Better fabric protection, evidence preservation, and scheduler-level mitigation

This sequence lowered the volume of alerts, fast-tracked rollback decisions, safeguarded fabric capacity, did not give up evidence of volatile accelerator, and facilitated scheduler-level mitigation. Volume-related studies focused on edge-computing approaches for real-time order-flow optimization to ensure time-sensitive processing of operational events is done nearer to its generation point [39]. CCD also eliminated uncalled-for delay by segregating the initial trapping of local evidence from subsequent centralized analysis. There was no redundancy in the Levenshtein accuracy percentage. Capture and transport, combined with analytics and response across the operational environment, resulted in operational resilience.

5.4 Future Research Recommendation

Future research should test Cluster Crash Diagnosis in larger-scale heterogeneous fleets, without modifying the architecture used in the source study. Current evidence includes an 8,000-node NVIDIA H100 environment and a 512-node TPU v5e pod [40]. In the future, researchers can expect more evaluations featuring mixed NVIDIA, AMD MI300X, TPU, and Cerebras deployments in a vendor-agnostic schema. Studies should be conducted to establish the stability of normalized RCClasses for a substantial change in proprietary error format, driver stack, memory organization, and interconnect technology.

The other priority should be to look into crashkernel sizing under extreme MMIO to the accelerator. The actual design detailed in the documentation called for a 2 GB reservation; blocks of the standard 128–256 MB size may not be enough. Controlled tests should be conducted to compare the success of vmcore capture using the various supported reservation sizes and different numbers of accelerators in each test, using the same methodology and under real failure conditions [41]. This applies to transport behavior in correlated panic storms as well, which is more broadly to be validated.

The CCD includes about 50 Mbps node-level clamping, 429 backpressures, exponential retry delays, jitter, and local persistence. Future experiments should gauge available storage capacity, the completion of the diagnostic transfer, and the performance of the remaining training fabric when hundreds of nodes produce many vmcores simultaneously. Existing signatures for SHA256 and a Levenshtein threshold of 0.85 should be tested against other, bigger crash corpora, other kernels and compiler configurations, and other vendor drivers [42]. The number and rate of false merges and false splits should be reported separately. Under realistic multi-tenant threat scenarios, security studies should also be done on the sanitization of task_struct, AES-256-GCM, mTLS 1.3, and audited break-glass

access with per-tenant KMS isolation.

Topology-aware correlation needs to move forward towards prediction. AI and cloud have been highlighted as technologies that enable scalable infrastructures and real-time data-shipping, further validating the importance of predictive operational intelligence [43]. Thus, as CCD evaluations evolve in the future, they should examine recurring spatial patterns to identify failures in switches, links, HBM modules, PCIe paths, or software cohorts ahead of large-scale impact while still maintaining other important longitudinal evaluation metrics such as MTTD, crash frequency, correlation confidence, upload behavior, and capture success.

6. Conclusion

Debugging artificial intelligence infrastructure with billions of cores is, in essence, a reliability problem in a system of distributed components rather than a suite of isolated node failures, the study concluded. With tens of thousands of nodes involved in an accelerator cluster, and manually or synchronously diagnosing cluster crashes, correlated crash scenarios become increasingly incompatible, potentially resulting in simultaneous VM cores, consumption of shared cluster storage, loss of diagnostic information, and slowing recovery once the cluster is in operation. The results showed that it was indeed necessary to migrate Linux kdump to a distributed telemetry architecture that could be deployed under diverse and high failure-pressure conditions.

Cluster Crash Diagnosis did this using the coordinated layers within the architecture. Reliable capture depended on a 2 GB crashkernel reserving area, properly configuring GRUB and kexec parameters, vendor sidecars, access to the/proc/vmcore file, makedumpfile processing, and LZO compression. In the transport layer, local persistence, around 50 Mbps node-level bandwidth control, and awareness of HTTP 429 and exponential backoff and jitter minimized the chances of diagnostic traffic eclipsing existing workloads in importance during training. The analytics layer combined similar and adjacent fail codes using the Levenshtein clustering method with a threshold of 0.85 into equivalent fail codes and grouped NVIDIA and TPU telemetry into a vendor-neutral classification into a single fleet-level workflow via SHA256 crash signatures. Large amounts of node evidence were next turned into actionable incidents by Elasticsearch, Grafana, node topology correlation, and webhooks for the scheduler.

That incorporation also addressed the core mismatch discovered at the start of the study. CCD was not the only thing that preserved evidence of failures, though; in addition, CCD controlled the flow of crash artifacts, CMS- and I-collapsed unique symptoms, and presented failures to engineers at a layer where fleet operators made decisions. Maximum production cases showed the demonstrated instrumented value of this design. The number of crashes in the NVIDIA H100 environment, which includes 8,000 nodes, jumped from 0.5 to 15 crashes per day following a patch to the operating system.

Previously, only 6 hours of hand analysis was needed to process all the crashes; CCD took all 15 crashes, consolidated 12 distinct stack traces into a single conceptual one, correlated the issue with the patched kernel, and reduced diagnosis time to just 8 minutes - a factor of 45. For the 512-node TPU v5e case, CCD normalized the ICI mesh-routing failures and deemed that 100% of nodes observed at the time of the crash were connected with 1 failing top-of-rack optical switch. The switch has now been replaced, and the panics have been avoided.

The total MTTD for both cases was reduced by 70–85% from the legacy methods. Also, CCD introduced operational velocity and safeguarded sensitive crash information with mTLS 1.3, AES-256-GCM, tenant-specific KMS keys, task_struct-based sanitization, RBAC, and audited break-glass entry. The study concludes that scalable crash diagnosis requires capture, normalization, transport, correlation, deduplication, and security and should function as a seamless solution. With increasing variations in AI infrastructure available, automated and secure crash telemetry will evolve from a convenience into a necessity for operational use.

References

1. P. Metzger, A. T. Markettos, E. T. Napierała, M. Naylor, R. N. Watson, and T. M. Jones, "Deprivileging low-level GPU drivers efficiently with user-space processes and CHERI compartments," in *Proc. 2025 ACM SIGSAC Conf. Computer and Communications Security*, Nov. 2025, pp. 2699–2713. <https://dl.acm.org/doi/pdf/10.1145/3719027.3765036>
2. T. Breitenbach, S. M. Divakar, L. Rasbach, and P. Jahnke, "ML-driven risk estimation for memory failure in a data center environment with convolutional neural networks, self-supervised data labeling and distribution-based

- model drift determination," *J. Parallel Distrib. Comput.*, vol. 185, Art. no. 104800, 2024. <https://www.sciencedirect.com/science/article/pii/S0743731523001703>
3. A. Mathai, C. Huang, P. Maniatis, A. Nogikh, F. Ivančić, J. Yang, and B. Ray, "KGYM: A platform and dataset to benchmark large language models on Linux kernel crash resolution," *Advances in Neural Information Processing Systems*, vol. 37, pp. 78053–78078, 2024. https://proceedings.neurips.cc/paper_files/paper/2024/file/8e9ed2a28af7d9085180e3817b2c9a57-Paper-Datasets_and_Benchmarks_Track.pdf
 4. M. Baqar, S. Naqvi, and R. Khanda, "AI augmented CI/CD pipelines: From code commit to production with autonomous decisions," in *Proc. 2025 3rd Int. Conf. Foundation and Large Language Models (FLLM)*, Nov. 2025, pp. 1041–1048. <https://arxiv.org/pdf/2508.11867>
 5. S. K. RaoVanama, "AI-augmented CI/CD pipeline optimization for scalable cloud-native deployment," *Int. J. Artificial Intelligence, Data Science, and Machine Learning*, vol. 5, no. 4, pp. 175–187, 2024. <https://ijaidsm.org/index.php/ijaidsm/article/download/368/338>
 6. Y. Dehfouli and A. Habibi Lashkari, "Memory analysis for malware detection: A comprehensive survey using the OSCAR methodology," *ACM Comput. Surv.*, vol. 58, no. 4, pp. 1–58, 2025. https://scholar.google.com/scholar?output=instlink&q=info:nYhjUuDr3GQJ:scholar.google.com/&hl=en&as_sdt=0,5&cillfp=9643302646581153144&oi=lle
 7. IBM, "Dump collection failed due to out of memory (OOM) error," *IBM Support*, 2026. <https://www.ibm.com/support/pages/node/7060322>
 8. G. Malik, "Biometric authentication—Risks and advancements in biometric security systems," *J. Computer Science and Technology Studies*, vol. 6, no. 3, pp. 159–180, 2024. <https://alkindipublisher.com/index.php/jcsts/article/download/9131/7816>
 9. P. Ananthula, S. Sepuri, M. Sethi, and H. Goel, "The strategic value of AI-driven physical identity and access management: Establishing cyber physical zero trust in critical infrastructure," in *Proc. 2026 4th Int. Conf. Self Sustainable Artificial Intelligence Systems (ICSSAS)*, May 2026, pp. 292–300. <https://ieeexplore.ieee.org/abstract/document/11559385/>
 10. W. Luo, R. Fan, Z. Li, D. Du, Q. Wang, and X. Chu, "Benchmarking and dissecting the NVIDIA Hopper GPU architecture," in *Proc. 2024 IEEE Int. Parallel and Distributed Processing Symp. (IPDPS)*, May 2024, pp. 656–667. <https://arxiv.org/pdf/2402.13499>
 11. M. Ozkan *et al.*, "Performance, efficiency, and cost analysis of wafer-scale AI accelerators vs. single-chip GPUs," *Device*, 2025. [https://www.cell.com/device/fulltext/S2666-9986\(25\)00147-4](https://www.cell.com/device/fulltext/S2666-9986(25)00147-4)
 12. A. Smith and V. K. Alla, "AMD Instinct MI300X: A generative AI accelerator and platform architecture," *IEEE Micro*, vol. 45, no. 3, pp. 41–48, 2025. https://hc2024.hotchips.org/assets/program/conference/day1/23_HC2024.AMD.MI300X.ASmith%28MI300X%29.v1.Final.20240817.pdf
 13. CloudOptimo, "Google TPU Ironwood: Revolutionizing AI inference at scale," 2025. <https://www.cloudoptimo.com/blog/google-tpu-ironwood-revolutionizing-ai-inference-at-scale/>
 14. Duan *et al.*, "Efficient training of large language models on distributed infrastructures: A survey," *Vicinagearth*, vol. 3, no.1, Art. no. 9, 2026. <https://link.springer.com/content/pdf/10.1007/s44336-026-00038-z.pdf>
 15. S. Gondi, S. Sepuri, and P. Narayan, "AI ethics for public financial systems: A cross-sector perspective," *Int. J. Applied Mathematics*, vol. 38, pp. 2212–2233, 2025.
 16. M. Chow *et al.*, "ServiceLab: Preventing tiny performance regressions at hyperscale through pre-production testing," in *Proc. 18th USENIX Symp. Operating Systems Design and Implementation (OSDI 24)*, 2024, pp. 545–562. <https://www.usenix.org/system/files/osdi24-chow.pdf>
 17. N. Jouppi *et al.*, "TPU v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings," in *Proc. 50th Annu. Int. Symp. Computer Architecture*, Jun. 2023, pp. 1–14. <https://dl.acm.org/doi/pdf/10.1145/3579371.3589350>
 18. A. Kokolis, M. Kuchnik, J. Hoffman, A. Kumar, P. Malani, F. Ma, Z. DeVito, S. Sengupta, K. Saladi, and C.-J. Wu, "Revisiting reliability in large-scale machine learning research clusters," *arXiv preprint arXiv:2410.21680*, 2024. <https://arxiv.org/pdf/2410.21680>
 19. Oracle Linux Documentation, "Collecting crash dumps using Kdump utility," 2025. <https://docs.oracle.com/en-us/iaas/oracle-linux/oci/diagnostics-kdump.htm>
 20. S. Tao, B. Zhang, and Q. Zhang, "ECHO: Enhancing Linux kernel fuzzing via call stack-aware crash deduplication," *Electronics*, vol. 14, no. 14, Art. no. 2914, 2025. <https://www.mdpi.com/2079-9292/14/14/2914>
 21. S. Zhang *et al.*, "Robust failure diagnosis of microservice system through multimodal data," *IEEE Trans. Services*

- Comput.*, vol. 16, no. 6, pp. 3851–3864, 2023.
<https://arxiv.org/pdf/2302.10512>
22. Z. Zhao, B. Wu, and L. Guo, "Integrated failure localization of optical switches and links in all-optical DCNs using DM-trails based on bidirectional voltage constraint," *J. Lightwave Technol.*, vol. 43, no. 8, pp. 3524–3535, 2025.
<https://ieeexplore.ieee.org/abstract/document/10840248/>
23. Red Hat Enterprise Linux, "Chapter 14: Configuring Kdump on the command line," in *RHEL 9 Managing, Monitoring and Updating the Kernel*, 2025.
https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/9/html/managing_monitoring_and_updating_the_kernel/installing-kdump_assembly_managing-kernel-command-line-parameters-with-uki
24. S. Cheng *et al.*, "Thorough characterization and analysis of large transformer model training at-scale," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 8, no. 1, pp. 1–25, 2024.
<https://dl.acm.org/doi/pdf/10.1145/3639034>
25. Z. Jiang *et al.*, "L4: Diagnosing large-scale LLM training failures via automated log analysis," in *Proc. 33rd ACM Int. Conf. Foundations of Software Engineering*, Jun. 2025, pp. 51–63.
<https://dl.acm.org/doi/pdf/10.1145/3696630.3728531>
26. S. Grover, "Self-healing web automation: An empirical comparison of traditional Selenium frameworks and Healenium stack," *Int. J. Applied Mathematics*, vol. 38, no. 12s, pp. 428–438, 2025.
<https://ijamjournal.org/ijam/publication/index.php/ijam/article/download/1391/1265>
27. M. J. Gossman, B. Nicolae, and J. C. Calhoun, "Scalable I/O aggregation for asynchronous multi-level checkpointing," *Future Gener. Comput. Syst.*, vol. 160, pp. 420–432, 2024.
<https://www.sciencedirect.com/science/article/am/pii/S0167739X24002929>
28. X. Han *et al.*, "vClos: Network contention aware scheduling for distributed machine learning tasks in multi-tenant GPU clusters," *Comput. Netw.*, vol. 268, Art. no. 111285, 2025.
https://jhc.sjtu.edu.cn/~shizhenzhao/computer_network_2025.pdf
29. S. Pavlenko and P. Kuliabko, "Deduplication of error reports in software malfunction: Algorithms for comparing call stacks," *Bulletin of Cherkasy State Technological University: Technical Sciences*, vol. 28, no. 4, pp. 59–69, 2023.
<https://er.chdtu.edu.ua/bitstream/ChSTU/4887/1/8.pdf>
30. W. Beer, "Transform your operations with Davis AI root cause analysis," *Dynatrace*, 2024.
<https://www.dynatrace.com/news/blog/transform-your-operations-with-davis-ai-root-cause-analysis/>
31. S. Bekele *et al.*, "THAPI: Tracing heterogeneous APIs," in *Proc. European Conf. Parallel Processing*, Cham, Switzerland: Springer Nature, Aug. 2025, pp. 54–70.
<https://arxiv.org/pdf/2504.03683>
32. J. K. Lastre, Y. Ko, H. Kwon, and I. You, "Evaluating Transport Layer Security 1.3 optimization strategies for 5G cross-border roaming: A comprehensive security and performance analysis," *Sensors*, vol. 25, no. 19, Art. no. 6144, 2025.
<https://www.mdpi.com/1424-8220/25/19/6144>
33. Ruijie, "Multi-tenant isolation technology in AIGC networks—Data security and performance stability," 2024.
<https://www.ruijie.com/en-global/support/tech-gallery/multi-tenant-isolation-technology-in-aigc-networks-data-security-and-performance-stability>
34. J. Dong *et al.*, "Evolution of Aegis: Fault diagnosis for AI model training service in production," in *Proc. 22nd USENIX Symp. Networked Systems Design and Implementation (NSDI 25)*, 2025, pp. 865–881.
<https://www.usenix.org/system/files/nsdi25-dong.pdf>
35. A. C. Jha, P. Gannavarapu, and S. Durgam, "Machine learning-driven security system for next-generation ultra-low-latency financial network governance," 2025.
<https://ieeexplore.ieee.org/abstract/document/11374574>
36. Y. Sun *et al.*, "ClusterRCA: An end-to-end approach for network fault localization and classification for HPC system," in *Proc. 2025 IEEE 36th Int. Symp. Software Reliability Engineering (ISSRE)*, Oct. 2025, pp. 347–358.
<https://arxiv.org/pdf/2506.20673>
37. K. S. Chadha, "Predictive risk modeling in P&C insurance using Guidewire DataHub and Power BI embedded analytics," *Int. J. Networks and Security*, vol. 5, no. 2, pp. 1–29, 2025.
38. J. Krebs, D. Gavrilenko, D. Amir, S. Landau Feibish, and M. Silberstein, "FlowPulse: Catching network failures in ML clusters," in *Proc. 24th ACM Workshop Hot Topics in Networks*, Nov. 2025, pp. 139–148.
<https://dl.acm.org/doi/pdf/10.1145/3772356.3772384>
39. N. Kumar, "Edge computing patterns for real-time order flow optimization," *Int. J. Applied Mathematics*, 2025.
<https://www.papjournals.com/index.php/edm/article/download/785/540>
40. A. Wongpanich *et al.*, "Machine learning fleet efficiency: Improving TPU systems at scale with ML productivity goodput," *Proc. Machine Learning and Systems*, vol. 8, pp. 1291–1303, 2026.
https://proceedings.mlsys.org/paper_files/paper/2026/file/fbe2b2f74a2ece8070d8fb073717bda6-Paper-Conference.pdf

41. L. Rzepka, J. Ottmann, R. Stoykova, F. Freiling, and H. Baier, "A scenario-based quality assessment of memory acquisition tools and its investigative implications," *Forensic Sci. Int.: Digital Investigation*, vol. 52, Art. no. 301868, 2025. <https://www.sciencedirect.com/science/article/pii/S2666281725000071>
42. E. Shibaev, D. Sushentsev, Y. Golubev, and A. Khvorov, "Stack trace deduplication: Faster, more accurately, and in more realistic scenarios," in *Proc. 2025 IEEE Int. Conf. Software Analysis, Evolution and Reengineering (SANER)*, Mar. 2025, pp. 511–521. <https://arxiv.org/pdf/2412.14802>
43. S. Rangu, "Enterprise digital transformation in financial services: Emerging trends and technologies," *Computer Fraud & Security*, 2025. <https://computerfraudsecurity.com/index.php/journal/article/view/786>