REVIEW ARTICLE

Reliability Optimization for Large Language Model Training Infrastructure: Challenges, Advances, and Future Directions

Yuchang Mo[1,*], Jian Wan[1], Hao Peng[2], Ruiming Fang[3], Yuan Fan[4], Chunyu Miao[4], Mirlan Chynybaev[5], Faer Gui[6], Rengui Zhang[7], Shuying Zhai[8], Wen Wu[1], Jifeng Zhu[1], Jianyong Hu[1] and Jinbin Mu[9]

[1] School of Computer Science and Technology, Zhejiang University of Water Resources and Electric Power, Hangzhou 310018, China
[2] School of Computer Science and Technology, Zhejiang Normal University, Jinhua 321004, China
[3] School of Information Science and Engineering, Huaqiao University, Quanzhou 362021, China
[4] Hangzhou Anheng Information Technology Co., Ltd., Hangzhou, China
[5] Razzakov Kyrgyz State Technical University, Bishkek 720044, Kyrgyzstan
[6] Zhejiang Keepsoft Information Technology Co., Ltd., Hangzhou, China
[7] Zhejiang YuGong Information Technology Co., Ltd., Hangzhou, China
[8] School of Mathematical Sciences, Huaqiao University, Quanzhou 362021, China
[9] Zhejiang Institute of Hydraulics and Estuary, Hangzhou 310020, China

Abstract

The ultra-large scale and prolonged runtime of Large Language Model (LLM) training—often involving thousands of GPUs and spanning weeks—render reliability a pivotal bottleneck. Hardware failures, stragglers, and runtime issues can waste over 30% of GPU resources, delaying model rollout and driving up costs. This survey focuses on reliability optimization for LLM training systems. The discussion centers on three pillars of reliability: fault detection, fault recovery, and straggler mitigation. For each pillar, we dissect innovative mechanisms, which range from communication-aware fault detection to adaptive load balancing, and we assess their impact on critical reliability metrics such as error-induced downtime reduction, Mean Time to Failure (MTTF), and slowdown mitigation rate. Additionally, we pinpoint open challenges, such as dynamic fault prediction for mixed workloads and cross-layer reliability coordination, and outline future directions to construct more resilient LLM training systems.

Keywords: reliability optimization, large language model, training infrastructure, fault detection, failure recovery, straggler mitigation.

1 Introduction

LLM training imposes unprecedented computational demands: a 175B-parameter model may necessitate

Submitted: 27 September 2025
Accepted: 15 October 2025
Published: 02 March 2026

Vol. 2, **No.** 1, 2026.
10.62762/TSSR.2025.806733

***Corresponding author:**
✉ Yuchang Mo
yuchangmo@sina.com

Citation
Mo, Y., Wan, J., Peng, H., Fang, R., Fan, Y., Miao, C., Chynybaev, M., Gui, F., Zhang, R., Zhai, S., Wu, W., Zhu, J., Hu, J., & Mu, J. (2026). Reliability Optimization for Large Language Model Training Infrastructure: Challenges, Advances, and Future Directions. *ICCK Transactions on Systems Safety and Reliability*, 2(1), 36–53.

over 1,000 GPUs operating continuously for 2–3 months, while newer trillion-parameter models push cluster scales to exceed 10,000 GPUs [1]. At this scale, reliability failures transition from exceptions to inevitabilities: studies on Alibaba's C4 system reveal that unaddressed errors can consume 31% of total training time [2], and Meta's RSC clusters equipped with 16,000 A100 GPUs record 6.5 hardware failures per 1,000 node-days [3].

As shown in Table 1, unlike traditional deep learning (DL) workloads such as image classification using ResNet, LLMs increase reliability challenges due to three unique features [4, 5]:

Synchronous execution: In Batch Synchronous Parallel (BSP) mode, a single GPU failure or a slow network link may interrupt the entire training process. This synchronous setup means that one slow component can bottleneck thousands of otherwise healthy GPUs [6, 7].

Extended runtime: Training cycles lasting several weeks increase exposure to transient failures such as network oscillations and permanent hardware degradation such as gradual GPU wear and tear caused by prolonged high-load operations. Temporary malfunctions, although temporary, can repeatedly interrupt training, while permanent degradation usually leads to persistent malfunctions that require component replacement [8, 9].

Complex dependencies: Hybrid parallelism, which mixes data, tensor, and pipeline parallelism, creates tight links between communication and computation. These dependencies blur fault location, making it difficult to determine the root cause of the fault - for instance, an NCCL timeout may result from GPU issues, network bottlenecks, or software deadlocks [10, 11].

This survey focuses solely on reliability optimization for LLM training infrastructure. The rest of this article is organized as follows: Section 2 explores the fault detection mechanism; Section 3 delves deeply into fault recovery strategies; Section 4 examines straggler mitigation techniques; Section 5 discuss hardware-level reliability optimization of LLM training infrastructure; Section 6 gives the open challenges and future directions; Section 7 presents the conclusion.

2 Fault Detection: Identifying Anomalies in Real Time

The first step in ensuring reliability is to spot faults quickly and accurately. Traditional manual diagnosis relies on labor-intensive log analysis and user-reported issues, which may take several hours to several days, resulting in a large number of GPUs being idle and severe training delays. Modern LLM training systems address this by deploying automated, fine-grained monitoring tailored to the unique communication and computation patterns of LLM workloads [12, 13].

2.1 Communication-Aware Fault Detection

Modern LLM training systems normally will introduce a fault detection subsystem that leverages the periodicity and homogeneity of collective communication operations such as AllReduce and AllGather in LLM training to identify anomalies. Collective operations in LLM training exhibit highly predictable patterns—fixed execution intervals, consistent message sizes, and synchronized timestamps—making deviations from these patterns a clear indicator of underlying faults [14, 15].

2.1.1 Enhanced Collective Communication Library

Enhanced Collective Communication Library as an extension of NVIDIA's NCCL (NVIDIA Collective Communication Library) can be designed to log fine-grained communication metrics without disrupting ongoing training [7].

As shown in Table 2, ACCL operates across three layers, with each layer capturing key data to achieve precise fault location:

Communicator layer: It keeps track of communicator IDs, level count, and allocation. When a fault occurs, this information shows which workgroup is involved. For instance, if a tensor parallel (TP) group fails, the layer can quickly mark the queues in that group as problematic.

Operation layer: Monitor the detailed information of collection operations, including operation types such as AllReduce and ReduceScatter, data types, element counts, operation durations, and execution counts. An unexpected deviation - such as a specific level of AllReduce duration doubling - immediately marks a potential problem. This layer is particularly effective in identifying subtle performance drops that may not immediately trigger faults but escalate into discrete faults.

Transport layer: Collect low-level connection details

Table 1. Differences in reliability challenges between LLM and traditional DL training.

Challenge Dimension	Traditional DL Training	LLM Training	Core Impact
Execution Mode	Asynchronous/weakly synchronous as the main mode	Strict Batch Synchronous Parallel (BSP)	Single-point faults block the cluster, leading to high GPU idleness
Operation Cycle	Hour-level	Week-to-month level	Exposure to more transient/permanent faults, resulting in high interruption frequency
Dependency Relationship	Single parallel mode (mostly data parallelism)	Hybrid parallelism (data + tensor + pipeline parallelism)	Ambiguous fault localization, making root cause diagnosis difficult
Resource Scale	Tens to hundreds of GPUs	Thousands to tens of thousands of GPUs	Fault probability increases exponentially
Fault Cost	Small loss from training interruption	Single interruption causes loss of thousands of GPU hours	Directly increases training costs and launch delays

Table 2. Three-layer architecture design of enhanced collective communication library.

Architecture Layer	Core Monitoring Indicators	Fault Localization Capability	Performance Overhead Control
Communicator Layer	Communicator ID, level count, resource allocation status	Quickly locate the parallel group where the fault resides (e.g., TP/DP group)	No additional overhead, relying on existing communication context
Operation Layer	Collective operation type (e.g., AllReduce), execution duration, data volume	Identify precursors of performance degradation (e.g., doubled AllReduce time)	<1% of training time, with direct logging via CUDA kernels
Transport Layer	RDMA IP address, QP number, message transmission delay	Isolate network faults (e.g., RDMA link congestion, QP faults)	Lightweight statistics, without affecting data transmission bandwidth

such as RDMA IP address and QP number and message statistics including message count, size and transmission duration. This layer is critical for isolating network-related faults, such as congested RDMA links or malfunctioning QPs.

To ensure minimal overhead, Alibaba's C4 system refines all relevant CUDA kernels to log their start and completion times directly. This avoids the inaccuracy of CPU timestamps and the performance cost of CUDA events, which are usually ineffective or overly resource-intensive for fine-grained monitoring of communication kernels [2].

2.1.2 Anomaly Analysis

There are two complementary methods for detecting and classifying anomalies to ensure comprehensive coverage of communication and computing failures:

Slow iteration detection: By comparing the iteration times of all worker threads, we can identify the worker threads with significantly longer execution times. For communication binding operations, worker threads whose iteration time is 15% longer than the average time of the cluster are marked as suspicious. This threshold is calibrated based on empirical data from the production cluster, balancing the sensitivity

to actual failures and the resistance to normal performance jitter [16].

Matrix-based latency mapping: For set operations like AllReduce, a two-dimensional delay matrix can be constructed, where each element represents the communication delay between a pair of workers, i.e., rows represent the source rank and columns represent the target rank. This matrix supports precise fault location: a single high-value unit points to a specific link bottleneck, a whole row of high values indicates that the source node has a problem, and a whole column indicates that the target node has a problem. For instance, in the training work with 2400 GPUs, this matrix helps C4D quickly identify the congested RDMA link between two leaf switches, which would take several hours to manually locate [2].

In the production environment, these two complementary methods have demonstrated significant efficacy: they have reduced the fault detection time from 30 minutes to 10 seconds and the downtime caused by errors from 31.19% to 1.16% for 2400-GPU LLM training jobs. This translates to saving thousands of GPU-hours per training cycle [17].

2.2 Straggler Localization

Finding computation stragglers such as slow-performing GPUs and communication stragglers such as congested or faulty network links are important in hybrid-parallel LLM training. As shown in Table 3, modern LLM training systems normally can employ a three-phase, non-intrusive workflow to pinpoint stragglers with 99.8% accuracy, ensuring that mitigation efforts are targeted to the correct component [18].

2.2.1 Tracking: Slow Iteration Identification

The first stage of discrete positioning focuses on identifying slow iterations across workers. It monitors the iteration time of each worker thread and detects statistically significant changes in execution time. The detection algorithm proposed in [18] analyzes time-series data in real time, calculating the probability that each timestamp marks a "change-point". This algorithm is well-suited for LLM training, as it can adapt to dynamic workloads and does not require predefining fixed thresholds.

To reduce false positives, straggler localization will add a post-verification step: it compares the average iteration time before and after each detected change-point. Only change-points that result in a performance difference of 10% or more are classified as genuine slowdowns, filtering out normal jitters caused by temporary system noise such as brief CPU contention from background processes. Compared with the sliding window detection method, this verification step reduces false positives by 90%. The sliding window detection method usually triggers slight transient fluctuation alerts [19].

2.2.2 Profiling: Suspicious Group Isolation

Validating every GPU in a large1000+ GPUs cluster would be prohibitively time-consuming and disruptive. To address this, straggler localization narrows the search space by profiling parallel groups—subsets of GPUs engaged in coordinated computation or communication such as TP groups, data parallelism (DP) groups [20].

During profiling, we can inject lightweight CUDA Events into NCCL calls to measure the execution time of communication and computation within each group. These events add minimal overhead (less than 0.5% of iteration time) and provide granular data on how time is spent in each group [21].

Groups where communication time exceeds 1.1× the median communication time across all groups are labeled "suspicious." This threshold is chosen based on empirical data, as it balances the need to capture genuine stragglers with avoiding over-labeling healthy groups. Concentrating the verification on these suspicious groups can reduce the number of GPUs that require further testing by 80%, thereby significantly lowering the overhead [18].

2.2.3 Validation: Lightweight Benchmarking

The final stage of lagging location involves temporarily suspending training and running target benchmark tests on suspicious groups. This pause is designed to be non-interruptible because it is consistent with the natural synchronization points in BSP training between iterations. These benchmarks are specifically designed to distinguish between those lagging behind in computing and communication [22]:

Computation validation: Runs standard GEMM (general matrix multiplication) tests on each GPU in the suspicious group. GEMM is a core operation in transformer models [45], making it an accurate proxy for GPU computation performance. A GPU with GEMM throughput 15% lower than its peers is classified as a computation straggler.

Communication validation: For collective operations organized in ring or tree topologies, the collective

Table 3. Three-stage process of straggler localization.

Stage	Core Operation	Key Threshold/Algorithm	Performance Indicator
Tracking Stage	Real-time monitoring of worker iteration time and detection of change points	10% performance difference verification (to filter system noise)	Reduce false positive rate by 90% (compared with sliding window method)
Profiling Stage	Inject CUDA Events to monitor communication / computation time of parallel groups	Groups with communication time >1.1× median are marked as suspicious	Reduce the number of GPUs to be verified by 80%
Verification Stage	GEMM computation test + P2P communication test	Computation: GEMM throughput <15% of peers; Communication: doubled delay	99.8% localization accuracy, with overhead <1%

can be decomposed into peer-to-peer (P2P) communication tests. For a ring topology, it runs two passes: the first pass transfers data from even-rank GPUs to adjacent odd-rank GPUs, and the second pass transfers data from odd-rank to even-rank GPUs. This decomposition enables precise identification of slow links. For example, if the link between GPU 3 and GPU 4 shows twice the latency of other links in the ring, it is marked as a communication straggler.

In experiments with 64 H800 GPUs training a GPT-2 model, FALCON-DETECT proposed in [18] correctly identified 99.8% of stragglers, with an overhead of less than 1%—demonstrating its efficacy and practicality for large-scale LLM training.

2.3 Long-Term Fault Pattern Analysis

Existing research utilizes a large amount of tracking data to identify recurrent failure modes, quantify their impacts, and provide information for more resilient system design [1, 23, 24].

2.3.1 Failure Distribution and Impact

The Acme Trace in [24] analyzed 6-month data from two LLM-specific clusters, i.e., screen with 2,288 A100 GPUs and Kalos with 2,416 A100 GPUs. This study focuses on GPU operations as they form the core of LLM training and reveals several key failure trends:

Failure timing: 40% of the malfunctions occur within the first 10 minutes of job startup. These early failures were mainly attributed to configuration errors, model loading issues, or initial hardware defects. The remaining 60% of the failures occur during the training process, driven by factors such as GPU thermal suppression or the intensification of high utilization in LLM training, network oscillations or temporary link disconnections, and gradual hardware degradation [25].

Failure impact by workload type: Pretraining jobs—though only 3.2% of the total job count in Kalos—account for 94% of GPU time lost to failures. This disparity arises because pretraining jobs have significantly longer runtimes, increasing their exposure to failures. Evaluation tasks account for 92.9% of the total tasks. Due to their short duration, hardware-related failures are rarely encountered.

Root cause localization: 82.5% of failures stay on a single node or device such as CUDA errors, ECC errors, or NVLink failures. This shows that node-level isolation is effective, since faults rarely spread across nodes.

Acme Trace also showed how the environment matters. In July 2023, a record-hot month, GPU overheating led to a big jump in NVLink and ECC errors during 7B model training. This shows why environmental monitoring needs to be part of the reliability system.

2.3.2 MTTF Modeling and Lemon Node Detection

The RSC cluster study in Meta [1] compiled 11 months of failure data from two research clusters i.e., RSC-1 has 16,000 A100 GPUs for general LLM training, and RSC-2 has 8,000 A100 GPUs for vision-related LLM tasks. The focus of this research was on quantifying reliability indicators and identifying persistent failure sources, generating some key insights:

Scale-dependent MTTF: The mean time between failures (MTTF) of training assignments decreases exponentially with the number of GPUs. On RSC-1, the MTTF is 47.7 days for 8-GPU jobs, 7.9 hours for 1024-GPU jobs, and about 0.23 hours for 131,072-GPU

jobs. This is consistent with the theoretical model MTTF, in which the probability of at least one failure increases as the number of components increases. This trend highlights the urgency of improving reliability for large-scale LLM training.

Failure cascades: A big job failure can disrupt the whole cluster. For example, when a 1024-GPU job failed due to hardware issues, it was automatically re-requested. That rescheduling pushed out 548 smaller jobs and affected more than 7,000 GPUs. These cascades occur because large, high-priority jobs often preempt low-priority jobs to ensure resource security, and repeated failures of large jobs may lead to excessive cluster chaos [17].

Lemon node identification: Approximately 0.5% of nodes labeled "lemons" exhibit persistent failure patterns—they cause 13% of large job failures despite their small numbers. These nodes are identified using a combination of metrics: the number of XID errors or GPU-specific error codes, repair tickets, job failure rates, and user-reported exclusions. Separating lemon nodes from the scheduler can reduce the failure rate of large-scale jobs by 30%.

The RSC cluster study shows why overlapping health checks matter. On RSC-1, 57% of PCIe errors happen together with XID 79 errors i.e., GPU disconnected from the bus, and 21% happen with IPMI "Critical Interrupt" events. Because of this overlap, if one check misses a fault, another can still catch it, cutting the chance of missing problems.

As shown in Table 4, current fault detection methods face trade-offs: ACCL offers high accuracy but is limited to NCCL operations, while Collie scales to 4096 GPUs but has higher overhead. Future research needs to develop "all-scenario" detectors that balance accuracy, overhead, and scalability—e.g., integrating ACCL's communication awareness with Collie's large-scale support.

3 Failure Recovery: Minimizing Wasted Computation

When a fault is found, fast recovery is needed to avoid losing weeks of training. The recovery process has two main goals: cut checkpoint overhead and speed up node replacement [26].

3.1 Asynchronous Checkpointing

For a model with 100B-parameters, the traditional synchronous checkpoint (which pauses training when the model state is written to disk) may reduce the LLM training speed by 43% [27]. This overhead arises because writing terabytes of model data including parameters, gradients and optimizer states to persistent storage takes significant time, during which all GPUs remain idle.

3.1.1 Asynchronous Checkpointing Process

As shown in Table 5, asynchronous checkpointing approach leverages underutilized host memory or CPU memory to minimize training interruption, a key insight given that CPU memory is often underutilized in GPU clusters.

A two-stage process can be designed as follows: 1) Each GPU writes its local model states including parameters, gradients, and optimizer states to the host's CPU memory via high-speed PCIe(with bandwidth up to 80GB/s for H100 GPUs. This step takes less than 10 seconds for a 123B-parameter model, as modern GPU nodes such as Kalos nodes with 2TB host memory can store multiple checkpoints in memory simultaneously. By using the host memory as an intermediate buffer, training can be resumed immediately after a dump without having to wait for data to be written to the disk. 2) A separate thread moves checkpoints from memory to the distributed file system such as HDFS in MegaScale's deployment. It runs on its own, so storage I/O does not block training. To optimize this transfer, MegaScale uses parallel I/O techniques—splitting the checkpoint into chunks and writing them to multiple HDFS nodes simultaneously—reducing storage latency by 40%.

This two-stage approach delivers dramatic improvements: it reduces checkpoint overhead by 3.6× for 7B-parameter models and 58.7× for 123B-parameter models compared to synchronous disk writes. For a 175B-parameter model training job, the checkpoint time drops from 30 minutes to 5 minutes, freeing up thousands of GPU-hours for productive training [17].

3.1.2 Checkpoint Loading Optimization

While asynchronous checkpointing reduces the time to save checkpoints, loading checkpoints remains a bottleneck for large models. Two key optimizations can be used.

Selective sharding: Checkpoints are segmented by data parallelism (DP) groups. For instance, if an assignment uses 16 DP groups, then the check points will be divided into 16 partitions, with each partition corresponding to one DP group. During recovery,

Table 4. Cross-method comparison of fault detection mechanisms.

Method	Overhead (%)	Detection Time (s)	Accuracy (%)	Scalability (Max GPUs)	Key Strengths	Key Limitations
ACCL [2]	0.8	10	99.2	2400	Communication-aware, fine-grained location	Limited to NCCL-based collective operations
FALCON-DETECT [18]	1.0	15	99.8	1024	Distinguishes compute/communication stragglers	Requires 1-minute benchmarking pause
HostPing [22]	0.5	5	98.5	512	Fast intra-host link detection	Cannot detect GPU compute faults
Collie [13]	1.2	8	97.0	4096	Scales to large clusters	High overhead for small jobs (100 GPUs)

Table 5. Comparison of asynchronous and synchronous checkpointing strategies.

Strategy Type	Implementation Principle	Checkpoint Time for 100B-Parameter Model	Training Speed Loss	Data Consistency Guarantee
Synchronous Checkpointing	Pause training to write the entire model state to disk at one time	~30 minutes	~43%	Strong consistency (no data loss)
Asynchronous Checkpointing	In-memory dumping (non-blocking) + background disk writing	~10 seconds (in-memory dumping) + ~5 minutes (background writing)	~8%	Weak consistency (risk of partial data loss, mitigated by incremental backups)

each GPU only reads the partitions related to its DP group, rather than the entire checkpoint. This reduces the amount of data that each GPU needs to load, equivalent to the number of DP groups, and reduces the usage of HDFS bandwidth by 80%.

Shared partition loading: GPUs in the same DP group share a checkpoint partition. One GPU in the group reads the partition from HDFS and broadcasts it to the other GPUs in the group via high-speed NVLink (providing a bandwidth of 400Gbps, much faster than HDFS). For a DP group with 16 GPUs, this will reduce HDFS read operations by 15 times and further accelerate checkpoint loading.

These optimizations are particularly effective for large-scale jobs: when training a 175B-parameter model for a 1024-GPU job, the checkpoint loading time is reduced from 2 hours (initial loading) to 15 minutes (optimized loading) [28].

4 Automated Node Isolation & Replacement

Manual node isolation means admins remove bad GPU nodes from the cluster by hand. This can take hours and often causes the same node to fail jobs more than once. Modern LLM training systems handle this automatically. They cut downtime by removing bad nodes fast and swapping in healthy ones [29].

4.1 Isolation Workflow

Modern LLM training systems use Kubernetes (k8s) with custom job control services to isolate bad nodes and restart jobs automatically [30, 31]. Following workflow needs no manual work:

Fault confirmation: When the system spots a fault (for example, "Node 123: NVLink error"), it sends an alert to the job redirect service. The alert lists the fault type, the level affected, and key metrics like NVLink latency and GPU temperature to confirm the issue.

Node eviction: The job service in k8s marks the bad node as "unschedulable" so no new jobs are sent to it. It then clears all running jobs on that node so it can be maintained.

Job restart: The evicted job restarts from the last good checkpoint on the healthy node pool. For every 1024 GPUs, 64 are kept as backups so there are always spare resources. These backup nodes use the same hardware and software as the main nodes, so no compatibility problems occur.

With automation, isolating and replacing a bad node

takes 5 minutes instead of 2 hours. As a result, the reinitialization overhead— the time spent restarting the job—drops from 0.6% to 0.15% of total training time.

4.1.1 Backup Node Pool

The backup node pool takes another step forward on the basis of automated replacement by maintaining a dedicated backup node pool, aiming to ensure that even super-large jobs [32] can be restored quickly. The size of the backup pool is adjusted according to the scale of the cluster: in the main cluster, for every 1024 GPUs (128 servers), MegaScale will allocate 64 backup GPUs on 8 servers [3]. These backup servers have the same hardware and software as the main server, ensuring that training can be restored without performance degradation.

The backup pool is managed dynamically: When a primary node fails, the job service picks a backup node under the same Top-of-Rack switch to keep latency low. If a TP group node fails, a backup in the same rack takes over so TP communication stays fast [33].

In practice, MegaScale's backup pool lets the system recover from node failures in under 15 minutes. It also keeps the effective training time ratio (productive runtime divided by total runtime) above 90%. This ratio is a critical reliability metric, as it quantifies how much of the total training time is spent on useful computation [3].

4.2 Fast NCCL Group Initialization

A hidden bottleneck in failure recovery is the initialization of NCCL communication groups. For large clusters (10,000+ GPUs), setting up these groups using PyTorch's default tools can take 1047 seconds (17 minutes) [36]. The delay comes from two main problems. First, PyTorch uses TCPStore, a single-threaded blocking key-value store for group setup. Second, the default process adds a global barrier after each group is created, which makes the time complexity $O(n^2)$ for n groups [34].

Modern LLM training systems fix these problems with two main changes. First, they replace TCPStore with Redis. Redis is an in-memory store that does not block and supports concurrent reading and writing. Using Redis for group setup removes the blocking behavior of TCPStore. For 2048 GPUs, this reduces initialization time from 1047 seconds to 361 seconds. Second, remove extra global barriers. By default, the initialization process adds a barrier to make sure all queues finish setting up one group before moving to the next. Reordering the setup, for example initializing TP groups before DP groups, cuts down the need for these barriers. For instance, since TP groups are limited to a single node, their initialization does not require a cross-node barrier. This reduces the number of barriers from $O(n)$ to $O(1)$ and lowers the initialization time to less than 30 seconds.

These optimizations have a particularly significant impact on recovery: they reduce the time it takes to restart 10,000-GPU jobs from 20 minutes to 5 minutes, minimizing downtime [35].

4.3 Straggler Mitigation: Addressing Slow Components

Stragglers—components such as GPUs or network links that operate 10–50% slower than their peers—are a major threat to LLM training reliability. In hybrid parallel training, due to synchronization requirements, a single laggard may slow down the entire work: the fast GPU must wait for the slowest component to complete its task before proceeding to the next iteration [36]. As shown in Table 6, the mitigation mechanism focuses on dynamic load adjustment rebalancing the workload to match the functionality of components, topology reconfiguration bypassing slow components and network resilience ensuring consistent communication performance to maintain throughput [37].

4.4 Micro-Batch Adjustment

For those lagging behind in computing, the lightest and non-destructive mitigation method is to adjust the micro-batch size, so that the workload of each GPU matches its current processing capacity. This method avoids the overhead of job restarts or topology changes and is particularly effective for transient outcasts, such as GPUs that have temporarily slowed down due to thermal throttling [38].

4.4.1 Semi-Dynamic Sizing

The semi-dynamic load balancing adjusts micro-batch sizes at iteration boundaries—avoiding intrusive changes to intra-iteration execution—and uses weighted gradient aggregation to ensure model convergence [39].

The process has two main steps. First, at the end of each iteration, each GPU reports its batch processing time to the central coordinator. This step is light since it reuses timing data already collected by the training framework. Second, the central coordinator applies quadratic programming to reduce the variance

Table 6. Trade-offs of straggler mitigation strategies.

Mitigation Strategy	Straggler Type Addressed	Slowdown Reduction Rate	Performance Overhead	Applicable Cluster Scale
Micro-Batch Sizing	Transient compute stragglers	~54%	<0.3%	Small-to-large clusters (100-4096 GPUs)
Link Reassignment	Persistent network stragglers	~61.5%	~2.0%	Medium-to-large clusters (512-10240 GPUs)
Straggler Consolidation	Multiple pipeline parallelism (PP) stragglers	~40%	~1.5%	Clusters with PP mode ($\geq$256 GPUs)
Adaptive Routing	Sudden network link failures	~30%	<0.1%	All scales ($\geq$100 GPUs)

in iteration time across GPUs. This keeps the slowest GPU from becoming a bottleneck and balances the workload across all components [6]. For instance, the micro-batch allocation of a GPU with a speed 20% lower is reduced by 20%.

To maintain the convergence of the model in the case of uneven batch sizes, semi-dynamic grading weights the gradients of each GPU through micro-batch counting. This ensures that the global gradient update reflects the contribution of each GPU, preventing bias from uneven workloads. Experiments conducted on ResNet-32 (CIFAR-10) and Inception-V3 (ImageNet) show that this weighted aggregation does not reduce the model accuracy, thereby achieving the same test accuracy [39].

In a heterogeneous GPU cluster consisting of NVIDIA V100 and A100 GPUs, semi-dynamic sizing reduces iteration time by 54% compared to uniform batch sizing. This improvement arises because the slower V100 GPUs are assigned fewer micro-batches, eliminating the need for faster A100 GPUs to wait [40].

4.4.2 Adaptive Sizing for Hybrid Parallelism

Modern LLM training system extend the micro-batch adjustment idea to hybrid-parallel LLM training, where stragglers may be confined to specific parallel groups such as a slow TP group or a congested DP group [41]. The adaptive sizing mechanism is tailored to the unique characteristics of each parallelism type.

For DP groups, where each group handles part of the data, FALCON shifts micro-batches based on group speed. For instance, in a 4-DP job, if a group is 10% slower, the number of micro-batches obtained by that group will be reduced by 10%. This rebalancing reduces slowdown by 79.7%, as the faster groups no longer wait for the slow one [42].

For pipeline parallelism (PP) groups, where each group handles part of the model layers, FALCON changes the size of micro-batches to overlap straggler delays with pipeline bubbles. A pipeline bubble is an idle period when one stage finishes but has to wait for data from the stage before it. Splitting one micro-batch into smaller chunks lets faster stages keep working while waiting for the slow stage, hiding up to 40% of the delay [18].

The adaptive sizing mechanism also works when more than one DP group is slow. For example, if two of the four groups lag, it shifts micro-batches to the two faster groups but limits how many they take to avoid overload. This way no group turns into a new bottleneck, and the workload stays balanced [43].

4.5 Topology Reconfiguration

For persistent communication stragglers such as a congested network link or a faulty switch port, micro-batch adjustment is insufficient—these stragglers require a more aggressive mitigation: topology reconfiguration. A topology reconfiguration mechanism rearranges the parallelism topology to bypass slow components, effectively moving high-traffic communication away from congested links or slow nodes [44].

4.5.1 Link Reassignment

Link reassignment strategy leverages the fact that different parallelism types generate varying levels of network traffic: DP groups generate high traffic due to gradient synchronization via AllReduce, while PP groups generate low traffic due to only exchanging activations between adjacent stages. By moving crowded links from high-traffic groups to low-traffic groups, FALCON reduces link load and eliminates the straggler [18].

Here, an example shows how this strategy works: 1) Before reassignment: The congested link between node 3 and node 4 carried DP gradient sync, which needs frequent large data transfers. This pushed DP communication time up by 60% and slowed the whole job. 2) After reassignment: A role swap between node 2 and node 3 served as a traffic engineering solution. This successfully decoupled the critical DP

synchronization from the congested link, moving it to the healthy path and leaving only lightweight PP traffic on the problematic connection. This reassignment reduces the load by 60% and cuts iteration time by 61.5% for weak communication stragglers.

Link reassignment is made feasible by integration with the training framework's topology management module. FALCON dynamically updates the rank-to-node mapping used by NCCL, ensuring that communication operations use the new links without requiring a full job restart [18].

4.5.2 *Straggler Consolidation*

When multiple stragglers are present, FALCON consolidates them into the fewest possible pipeline stages. This strategy takes advantage of the fact that the PP phase runs synchronously: the performance of a phase is determined by its slowest GPU, regardless of how many slow GPUs there are in this phase [9].

Here, we give an example to show that consolidating the laggards into one stage can avoid delays between multiple stages. 1) Before consolidation: Two straggling GPUs are in separate PP stages (Stage 1 and Stage 3). Each straggler added 0.5 seconds to its stage, so the total iteration time went up by 1 second since delays stack across stages. 2) After consolidation: Both slow GPUs are grouped within Stage 2. This configuration increases the duration of Stage 2 by 0.5 seconds, but since the delay is confined to this single stage, the total iteration time increase is reduced to 0.5 seconds, the total iteration time increases by only 0.5 seconds [18].

Straggler consolidation also prioritizes moving stragglers to interior PP stages (not the first or last stages). The first and final stages typically handle additional preprocessing (for example, embedding layers) or post-processing tasks, making them more sensitive to latency. By moving the stragglers to the internal phase, FALCON minimizes their impact on the iteration time [18].

4.6 Network Resilience

Network failures, including link oscillations, congestion and switch port failures, are the main sources of communication disconnections in LLM training. The LLM training infrastructure introduces network resilience mechanisms to alleviate these issues and ensure consistent communication performance [46, 47].

4.6.1 *Adaptive Routing*

Adaptive routing enables their InfiniBand networks to route dynamically around congested or faulty links [48]. Adaptive routing runs at the network level. It changes packet paths based in real time and spreads traffic across all paths to avoid bottlenecks. Its main benefits for LLM training include:

Bandwidth preservation: When a link fails, AR keeps about 90% of peak bandwidth, while static routing (e.g., ECMP) holds only 50%. For instance, if a link in the topology fails, the AR will redirect the traffic to another path, so that the data can continue to move.

Reduced throughput variation: AR has reduced the difference in communication throughput between jobs by 40%. In multi-tenant clusters, this consistency is critical—LLM training jobs require predictable communication performance to avoid unexpected stragglers.

Compatibility with LLM traffic patterns: LLM training generates "elephant flows" (few large data transfers, such as AllReduce), which are well-suited for AR. AR can precisely route each elephant flow to a less congested path, whereas static routing often concentrates flows on a small number of links.

Meta's experiments show that AR is particularly effective for large-scale jobs: a 1024-GPU job training a Llama model experienced 30% fewer communication-induced stragglers with AR enabled [1].

4.6.2 *Traffic Engineering*

A traffic engineering subsystem shall be designed to optimize network traffic for LLM training's unique flow characteristics. LLM training generates a small number of large elephant flows rather than many small mice flows, making it feasible to plan routes for each flow individually— a strategy that is impractical in traditional data centers with diverse traffic [49]. The key mechanisms include:

Path allocation at task startup: Before a training job starts, traffic engineering subsystem requests communication paths from a global controller. The controller uses a full-grid path detection mechanism through randomly selected servers to identify reliable paths and avoid faulty links. For example, if a link between two leaf switches is detected as faulty during probing, it is excluded from path allocation.

Load balancing across RDMA QPs: traffic engineering subsystem balances the number of RDMA Queue Pairs

Table 7. Cross-method comparison of straggler mitigation strategies.

Strategy	Slowdown Reduction (%)	Overhead (%)	Applicability	SLA Compliance (%)	Key Strengths	Key Limitations
Semi-Dynamic Sizing [39]	54	0.3	Transient compute stragglers	99.5	No job restart, preserves convergence	Ineffective for persistent faults
Link Reassignment [18]	61.5	2.0	Persistent communication stragglers	98.8	Bypasses congested links without restart	Requires topology management integration
Adaptive Routing [1]	30	0.1	Network link failures	99.0	No software changes, network-level optimization	Cannot address compute stragglers
Straggler Consolidation [18]	40	1.5	Multiple PP stragglers	99.2	Reduces stacked delays across stages	Limited to pipeline parallelism

(QPs), which are used for end-to-end communication, across available paths. This stops any one path from overloading. For example, C4P sends traffic from the same NIC evenly to both ports of a bonded link so the load stays balanced [2].

Dynamic path adjustment: During training, the traffic engineering subsystem tracks how long messages take on each path. If a path gets over 20% slower, showing congestion, C4P redirects the next flows to a faster path. This way traffic always goes through the most efficient route [2].

In experiments with AllReduce operations (a core communication primitive in LLM training), traffic engineering subsystem can achieve an effective bandwidth of 360Gbps, compared to less than 240Gbps for the baseline (without traffic engineering). This improvement eliminates network-induced stragglers and ensures that communication does not become a bottleneck in training [50].

Table 7 reveals that no single strategy addresses all straggler types: semi-dynamic sizing works for transient compute issues but not persistent faults, while adaptive routing only fixes network problems. Future work needs to develop "hybrid mitigation" frameworks that automatically select the optimal strategy based on straggler type, e.g., using link reassignment for persistent network stragglers and micro-batch adjustment for transient compute stragglers.

5 Physical Structure Reliability

The physical structure of LLM training clusters—including GPU hardware, network topology, and power supply systems—directly determines baseline reliability. Hardware failures, such as GPU overheating, network link congestion, and power outages, account for 68% of LLM training interruptions, as shown by the Acme Trace in [24], making physical structure optimization a prerequisite for end-to-end reliability.

5.1 GPU Cooling and Thermal Management

Extended LLM training causes GPU junction temperatures to rise to 85–95°C, increasing ECC errors by 3× and NVLink failure rates by 2.5× [25]. Modern clusters adopt two key cooling optimizations: 1) Liquid Cooling Systems: Meta's RSC clusters use direct liquid cooling (DLC) for A100/H100 GPUs, reducing junction temperatures by 15–20°C compared to air cooling. DLC maintains GPU temperatures below 80°C even at 100% utilization, cutting thermal-induced failures by 40%. For example, a 1024-GPU job training Llama 3 (70B parameters) experienced only 2 NVLink errors with DLC, compared to 8 errors with air cooling. 2) Dynamic Thermal Throttling: Alibaba's C4 clusters deploy AI-driven thermal control, which adjusts GPU clock speeds based on real-time temperature and workload. When temperatures exceed 82°C, the system reduces clock speeds by 5% (instead of the default 15% throttling), balancing thermal safety and performance loss. This optimization reduces training slowdown from 12% to 3% for 7B-parameter model training [2].

5.2 Redundant Network Topology

Network physical links, such as RDMA cables, switch ports, are vulnerable to wear and tear, with a mean time between failures (MTBF) of 10,000 hours for InfiniBand links [47]. To enhance resilience, clusters adopt multi-path redundancy: 1) Dual Links Topology: MegaScale clusters use a leaf-spine architecture where each GPU node connects to 2 leaf switches via dual RDMA links (400Gbps each). If one link fails, traffic is automatically rerouted through the backup link, preserving 95% of bandwidth [3]. For example, a 2048-GPU job training a 175B-parameter model experienced only a 2% iteration time increase during a link failure, compared to a 30% increase with

single links. 2) Switch Port Redundancy: Critical spine switches are equipped with 2× redundant ports. If a port fails due to electrical damage, the system activates a backup port within 100ms, avoiding network partitions. Meta's RSC-1 cluster reduced network-induced job failures by 25% using this mechanism.

5.3 Power Supply Resilience

Power fluctuations, such as voltage sags, brief outages, cause 12% of LLM training interruptions [24]. Clusters implement hierarchical power protection: 1) Use Uninterruptible Power Supply (UPS) for GPU racks: Each rack of 8 GPU servers is equipped with a 15kVA UPS, providing 10 minutes of backup power during outages. This allows the system to save a checkpoint and gracefully shut down, avoiding data loss. 2) Use Redundant Power Supplies (RPS) for servers: Each GPU server uses 2× 1600W power supplies in active-active mode. If one supply fails, the other takes over instantaneously, preventing server shutdown. Alibaba's Kalos cluster reduced power-induced failures by 60% with RPS.

6 Open Challenges and Future Directions

Although significant advancements in reliability optimization for LLM training infrastructure, there are still some open challenges that require further research to support the next generation of ultra-large-scale LLMs.

6.1 Dynamic Fault Prediction

The current reliability systems are mainly reactive. They only detect faults after they occur, but lack the ability to predict transient faults such as the upcoming GPU thermal suppression event due to temperature rise, or gradual hardware degradation such as the performance loss of NVLink after several weeks of operation. By leveraging predictive analytics to identify risky nodes, the system can drain workloads before a failure. Future research is likely to focus on:

Multi-modal sensor fusion: The model will integrate hardware information such as GPU temperature and power with software metrics for predicting faults. One example is that, when the GPU temperature becomes very fast higher and the iteration time also becomes a little bit slower steadily, this situation can warn about a thermal throttling event.

LLM-aided log analysis: LLMs scan unstructured logs such as NCCL timeouts and CUDA error logs to catch early signs of faults. Acme Trace [2] shows that many failures come with subtle log patterns beforehand, such as repeated "soft" ECC errors before a "hard" ECC error. LLMs are good at spotting these patterns.

Transfer learning for fault models: Build fault prediction models that can transfer across GPU types such as A100 and H100 and different cluster setups. Current models are usually cluster-specific, which limits their applicability to new environments.

6.2 Cross-Layer Reliability Coordination

The current reliability mechanism operates in the shaft: fault detection implemented in the communication layer, fault recovery handled by the job scheduler, and straggler mitigation managed by the parallel layer operate independently with minimal coordination. Lack of integration can lead to suboptimal decisions - for instance, detected GPU discretization may trigger micro-batch tuning (mitigation), but if the discretization is caused by permanent hardware damage, it will not trigger node replacement (recovery). To solve this problem, future systems should:

Establish a central reliability controller: One controller collects data from detection, recovery, and mitigation modules to decide the next step. For example, if it shows a permanent GPU failure and long recovery time, the controller will pick node replacement over micro-batch tuning.

Define standard APIs: Build APIs so that each layer can communicate with each other. For instance, the communication layer can utilize an API to alert to link congestion, and the parallel layer can invoke it to trigger topological changes.

Adaptive mitigation selection: Select the mitigation strategy based on the cause of deceleration. For instance, a 10% slowdown caused by temporary CPU contention can be adjusted in micro-batch size, while a 50% slowdown due to network link failure can be achieved by replacing nodes.

6.3 Reliability for Mixed Workloads

Current reliability optimization is mainly designed for pre-training work, but it neglects the unique requirements of fine-tuning and evaluation work, which are key components of the LLM development pipeline. Acme Trace [2] emphasizes that evaluation jobs accounting for 92.9% of the total jobs have the longest queuing delay due to their low priority, and

their running time is short. Therefore, rapid recovery is crucial for avoiding delayed model feedback from developers. Future work should focus on:

Priority-aware recovery: Accelerate the recovery of evaluation jobs by using in-memory checkpoints rather than disk-based checkpoints and reserving a small dedicated resource pool for evaluation. This will reduce the time required to restart the assessment job from several minutes to just a few seconds.

Shared fault isolation: Ensure that errors in the pre-training work do not affect the evaluation work. For instance, isolate pre-training and evaluation jobs to separate the node pool, or use network virtualization to prevent communication congestion caused by the impact of pre-training jobs on evaluation.

Lightweight reliability mechanisms for small jobs: Develop low-cost fault detection and mitigation for small-scale assessment work. The current mechanisms such as FALCON-DETECT [18] are optimized for large-scale jobs and may impose excessive overhead on small-scale jobs.

6.4 Energy-Efficient Reliability

As LLM training clusters expand to over 100,000 GPUs, the energy consumption of reliability mechanisms (such as backup nodes and redundant communication paths) has become a significant issue. For instance, for a cluster of 100,000 GPUs, a 5% backup pool means that 5,000 GPUs are idle and consuming power. Future research should explore following energy-saving strategies:

Dynamic backup node activation: When a node fails, the backup node is quickly activated. This requires rapid power management (for example, waking up the GPU from low-power mode within a few seconds) to avoid delayed recovery.

Energy-aware fault mitigation: Prioritize mitigation strategies with lower energy consumption. For instance, micro-batch adjustment consumes less energy than node replacement (which requires powering the new node and transmitting data).

Thermal-aware scheduling: The scheduler prevents the GPU from overheating and balances the temperature by distributing workloads across nodes. This reduces the reliance on the cooling system and minimizes malfunctions caused by overheating.

6.5 Reliability Challenges in LLM Inference/Service Deployment

LLM inference such as chatbots and code generation faces unique reliability challenges compared to training: it requires low latency ($\leq$100ms for real-time services), high availability (99.99% SLA), and support for dynamic workloads—all while sharing resources across multiple tenants.

6.6 Multi-Tenancy Isolation

Inference clusters often serve multiple tenants such as a cloud provider hosting both customer service chatbots and code assistants, where one tenant's workload surge can degrade others' performance. Some possible challenges include: 1) A tenant with sudden traffic may occupy 80% of GPU memory, causing other tenants' queries to queue. For example, a code assistant tenant's peak traffic delayed a customer service chatbot's response time from 50ms to 500ms [1]. 2) A faulty query such as a prompt causing infinite token generation can crash a GPU, disrupting all tenants sharing that GPU.

Two potential solutions can be used. 1) Use technologies like NVIDIA MIG (Multi-Instance GPU) to split a single H100 GPU into 7 isolated instances, each with dedicated memory and compute resources. This ensures a tenant's surge only affects its own instance [38]. 2) Deploy per-tenant process sandboxes and query timeouts. If a query exceeds the timeout, it is terminated without affecting other tenants.

6.6.1 Workload Fluctuations

Inference workloads are highly dynamic—e.g., a news summarization service may see 5× traffic spikes during morning/evening hours. Some possible challenges include: 1) Sudden traffic increases cause queueing, violating latency SLAs. For example, a chatbot service's latency rose from 80ms to 250ms during a morning spike, missing its 100ms SLA [3]. 2) Over-provisioning to handle spikes leads to 40% idle GPU time during off-peak hours [38].

As a potential solution, we can use LSTM models to predict traffic 15 minutes in advance such as predicting morning spikes and activate backup GPUs dynamically. AlpaServe reduced latency violations by 70% using this method [20]. Another possible solution is to adjust batch sizes based on traffic—larger batches (32 queries/batch) during peaks to utilize GPU resources, smaller batches (4 queries/batch) during off-peak to minimize latency.

6.6.2 Online Updates

LLM services require frequent model updates suchc as fine-tuning for new tasks and bug fixes, but updating models without downtime is challenging. Some possible challenges include: 1) Traditional updates require stopping the old model and starting the new one, causing 5–10 minutes of downtime [5]. 2) During updates, some queries may be processed by the old model and others by the new one, leading to inconsistent outputs (e.g., different summarizations for the same text).

Here, we propose two potential solutions. 1) Roll out the new model to 10% of traffic first, monitor performance (latency, accuracy), and gradually scale to 100% if no issues arise. This avoids full-service downtime [9]. 2) Run old and new models in parallel during updates, routing queries to both and returning results from the old model until the new one is validated. This ensures consistency while enabling updates.

7 Conclusion

This survey specifically focuses on the reliability optimization for LLM training infrastructure and integrates insights from recent technological research. The discussion focused on the three core pillars of reliability: fault detection, fault recovery, and distributed mitigation. The main conclusions of this survey include:

Communication-aware fault detection is highly effective for LLM training, and collective operations provide predictable signals for anomaly detection. These mechanisms have reduced the fault detection time from several hours to just a few seconds, minimizing the idle time of GPU utilization.

Asynchronous checkpointing greatly improves large model training. It cuts checkpoint overhead by up to 58.7× and avoids the 43% slowdown seen with synchronous methods.

Dynamic mitigation strategies address stragglers without restarting the job and reduces deceleration by 54–60.1% in experiments.

Long-term trace analysis provides key insights into failure modes to achieve proactive optimization (for example, lemon node isolation, environmental monitoring).

As LLMS continue to expand into trillions of parameter models and over 100,000 GPU clusters, reliability will remain a challenge. The flexible strategy analyzed in this survey makes the construction of LLM training systems more cost-effective and fault-tolerant, promoting the development of more powerful artificial intelligence models.

Data Availability Statement

Not applicable.

Funding

This work was supported by the Joint Fund of Zhejiang Provincial Natural Science Foundation of China under Grant LGEZ26F030002, and Scientific Research Foundation of Zhejiang University of Water Resources and Electric Power under Grant JBGS2025009.

Conflicts of Interest

Yuan Fan and Chunyu Miao are affiliated with the Hangzhou Anheng Information Technology Co., Ltd., Hangzhou, China; Faer Gui is affiliated with the Zhejiang Keepsoft Information Technology Co., Ltd., Hangzhou, China; Rengui Zhang is affiliated with the Zhejiang YuGong Information Technology Co., Ltd., Hangzhou, China. The authors declare that these affiliations had no influence on the study design, data collection, analysis, interpretation, or the decision to publish, and that no other competing interests exist.

AI Use Statement

The authors declare that no generative AI was used in the preparation of this manuscript.

Ethical Approval and Consent to Participate

Not applicable.

References

[1] Kokolis, A., Kuchnik, M., Hoffman, J., Kumar, A., Malani, P., Ma, F., ... & Wu, C. J. (2025, March). Revisiting reliability in large-scale machine learning research clusters. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)* (pp. 1259-1274). IEEE. [CrossRef]

[2] Dong, J., Luo, B., Zhang, J., Zhang, P., Feng, F., Zhu, Y., ... & Fu, B. (2024). Boosting large-scale parallel training efficiency with c4: A communication-driven approach. *arXiv preprint arXiv:2406.04594*.

[3] Jiang, Z., Lin, H., Zhong, Y., Huang, Q., Chen, Y., Zhang, Z., ... & Liu, X. (2024). MegaScale: Scaling large language model training to more than 10,000 GPUs. In

21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24) (pp. 745-760).

[4] Narayanan, D., Shoeybi, M., Casper, J., LeGresley, P., Patwary, M., Korthikanti, V., ... & Zaharia, M. (2021, November). Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the international conference for high performance computing, networking, storage and analysis* (pp. 1-15). [CrossRef]

[5] Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M. A., Lacroix, T., ... & Lample, G. (2023). Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

[6] Levy, S., Ferreira, K. B., DeBardeleben, N., Siddiqua, T., Sridharan, V., & Baseman, E. (2018, November). Lessons learned from memory errors observed over the lifetime of cielo. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis* (pp. 554-565). IEEE. [CrossRef]

[7] Qian, K., Xi, Y., Cao, J., Gao, J., Xu, Y., Guan, Y., ... & Cai, D. (2024, August). Alibaba hpn: A data center network for large language model training. In *Proceedings of the ACM SIGCOMM 2024 Conference* (pp. 691-706). [CrossRef]

[8] Smith, S., Patwary, M., Norick, B., LeGresley, P., Rajbhandari, S., Casper, J., ... & Catanzaro, B. (2022). Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model. *arXiv preprint arXiv:2201.11990*.

[9] Rasley, J., Rajbhandari, S., Ruwase, O., & He, Y. (2020, August). Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining* (pp. 3505-3506). [CrossRef]

[10] Yi, X., Zhang, S., Luo, Z., Long, G., Diao, L., Wu, C., ... & Lin, W. (2020, November). Optimizing distributed training deployment in heterogeneous GPU clusters. In *Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies* (pp. 93-107). [CrossRef]

[11] Villalobos, P., Sevilla, J., Besiroglu, T., Heim, L., Ho, A., & Hobbhahn, M. (2022). Machine learning model sizes and the parameter gap. *arXiv preprint arXiv:2207.02852*.

[12] Xiong, Y., Jiang, Y., Yang, Z., Qu, L., Zhao, G., Liu, S., ... & Zhou, L. (2024). SuperBench: Improving Cloud AI Infrastructure Reliability with Proactive Validation. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)* (pp. 835-850).

[13] Kong, X., Zhu, Y., Zhou, H., Jiang, Z., Ye, J., Guo, C., & Zhuo, D. (2022). Collie: Finding performance anomalies in RDMA subsystems. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)* (pp. 287-305).

[14] Zhu, Y., Eran, H., Firestone, D., Guo, C., Lipshteyn, M., Liron, Y., ... & Zhang, M. (2015). Congestion control for large-scale RDMA deployments. *ACM SIGCOMM Computer Communication Review, 45*(4), 523-536. [CrossRef]

[15] Zhang, S., Zhao, Y., Xiong, X., Sun, Y., Nie, X., Zhang, J., ... & Pei, D. (2024, July). Illuminating the gray zone: Non-intrusive gray failure localization in server operating systems. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering* (pp. 126-137). [CrossRef]

[16] He, Y., Hutton, M., Chan, S., De Gruijl, R., Govindaraju, R., Patil, N., & Li, Y. (2023, June). Understanding and mitigating hardware failures in deep learning training systems. In *Proceedings of the 50th Annual International Symposium on Computer Architecture* (pp. 1-16). [CrossRef]

[17] Wang, Z., Jia, Z., Zheng, S., Zhang, Z., Fu, X., Ng, T. E., & Wang, Y. (2023, October). Gemini: Fast failure recovery in distributed training with in-memory checkpoints. In *Proceedings of the 29th Symposium on Operating Systems Principles* (pp. 364-381). [CrossRef]

[18] Wu, T., Wang, W., Yu, Y., Yang, S., Wu, W., Duan, Q., ... & Zhang, L. (2024). FALCON: Pinpointing and mitigating stragglers for large-scale hybrid-parallel training. *arXiv preprint arXiv:2410.12588*.

[19] Korthikanti, V. A., Casper, J., Lym, S., McAfee, L., Andersch, M., Shoeybi, M., & Catanzaro, B. (2023). Reducing activation recomputation in large transformer models. *Proceedings of Machine Learning and Systems, 5*, 341-353.

[20] Maeng, K., Bharuka, S., Gao, I., Jeffrey, M., Saraph, V., Su, B. Y., ... & Wu, C. J. (2021). Understanding and improving failure tolerant training for deep learning recommendation with partial recovery. *Proceedings of Machine Learning and Systems, 3*, 637-651.

[21] Zhou, K., Hao, Y., Mellor-Crummey, J., Meng, X., & Liu, X. (2020). Gvprof: A value profiler for gpu-based clusters. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis* (pp. 1–16). IEEE. [CrossRef]

[22] Liu, K., Jiang, Z., Zhang, J., Wei, H., Zhong, X., Tan, L., ... & Huang, T. (2023). Hostping: Diagnosing intra-host network bottlenecks in RDMA servers. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)* (pp. 15-29).

[23] Zhong, Y., Sheng, G., Liu, J., Yuan, J., & Wu, C. (2023). Swift: Expedited failure recovery for large-scale dnn training. In *Proceedings of the 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming* (pp. 447–449). [CrossRef]

[24] Hu, Q., Ye, Z., Wang, Z., Wang, G., Zhang, M., Chen, Q., ... & Zhang, T. (2024). Characterization of large language model development in the datacenter. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)* (pp. 709-729).

[25] Taherin, A., Patel, T., Georgakoudis, G., Laguna, I., & Tiwari, D. (2021, June). Examining failures and repairs

on supercomputers with multi-gpu compute nodes. In *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)* (pp. 305-313). IEEE. [CrossRef]

[26] Jang, I., Yang, Z., Zhang, Z., Jin, X., & Chowdhury, M. (2023, October). Oobleck: Resilient distributed training of large models using pipeline templates. In *Proceedings of the 29th Symposium on Operating Systems Principles* (pp. 382-395). [CrossRef]

[27] Bautista-Gomez, L., Benoit, A., Di, S., Herault, T., Robert, Y., & Sun, H. (2024). A survey on checkpointing strategies: Should we always checkpoint à la Young/Daly?. *Future Generation Computer Systems, 161*, 315-328. [CrossRef]

[28] Young, J. W. (1974). A first order approximation to the optimum checkpoint interval. *Communications of the ACM, 17*(9), 530-531. [CrossRef]

[29] Zimmer, C., Maxwell, D., McNally, S., Atchley, S., & Vazhkudai, S. S. (2018, November). Gpu age-aware scheduling to improve the reliability of leadership jobs on titan. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis* (pp. 83-93). IEEE. [CrossRef]

[30] Agudelo-España, D., Gomez-Gonzalez, S., Bauer, S., Schölkopf, B., & Peters, J. (2020, August). Bayesian online prediction of change points. In *Conference on uncertainty in artificial intelligence* (pp. 320-329). PMLR.

[31] Burns, B., Grant, B., Oppenheimer, D., Brewer, E., & Wilkes, J. (2016). Borg, omega, and kubernetes. Communications of the ACM, 59(5), 50-57. [CrossRef]

[32] Yu, P., & Chowdhury, M. (2020). Fine-grained GPU sharing primitives for deep learning applications. *Proceedings of Machine Learning and Systems, 2*, 98-111.

[33] Li, J., Xu, H., Zhu, Y., Liu, Z., Guo, C., & Wang, C. (2023, May). Lyra: Elastic scheduling for deep learning clusters. In *Proceedings of the Eighteenth European Conference on Computer Systems* (pp. 835-850). [CrossRef]

[34] Thorpe, J., Zhao, P., Eyolfson, J., Qiao, Y., Jia, Z., Zhang, M., ... & Xu, G. H. (2023). Bamboo: Making preemptible instances resilient for affordable training of large DNNs. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)* (pp. 497-513).

[35] Xiao, W., Ren, S., Li, Y., Zhang, Y., Hou, P., Li, Z., ... & Jia, Y. (2020). {AntMan}: Dynamic scaling on {GPU} clusters for deep learning. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)* (pp. 533-548).

[36] Chen, Y., Xie, C., Ma, M., Gu, J., Peng, Y., Lin, H., ... & Zhu, Y. (2022). Sapipe: Staleness-aware pipeline for data parallel dnn training. *Advances in neural information processing systems, 35*, 17981-17993.

[37] Zhang, R., Xiao, W., Zhang, H., Liu, Y., Lin, H., & Yang, M. (2020, June). An empirical study on program failures of deep learning jobs. In *Proceedings of the ACM/IEEE 42nd international conference on software engineering* (pp. 1159-1170). [CrossRef]

[38] Li, Z., Zheng, L., Zhong, Y., Liu, V., Sheng, Y., Jin, X., ... & Stoica, I. (2023). AlpaServe: Statistical multiplexing with model parallelism for deep learning serving. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)* (pp. 663-679).

[39] Chen, C., Weng, Q., Wang, W., Li, B., & Li, B. (2020, October). Semi-dynamic load balancing: Efficient distributed learning in non-dedicated environments. In *Proceedings of the 11th ACM Symposium on Cloud Computing* (pp. 431-446). [CrossRef]

[40] You, Y., Li, J., Reddi, S., Hseu, J., Kumar, S., Bhojanapalli, S., ... & Hsieh, C. J. (2019). Large batch optimization for deep learning: Training bert in 76 minutes. *arXiv preprint arXiv:1904.00962*.

[41] Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., & Catanzaro, B. (2019). Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*.

[42] Huang, Y., Cheng, Y., Bapna, A., Firat, O., Chen, D., Chen, M., ... & Wu, Y. (2019). Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems, 32*.

[43] Narayanan, D., Harlap, A., Phanishayee, A., Seshadri, V., Devanur, N. R., Ganger, G. R., ... & Zaharia, M. (2019, October). PipeDream: Generalized pipeline parallelism for DNN training. In *Proceedings of the 27th ACM symposium on operating systems principles* (pp. 1-15). [CrossRef]

[44] Ren, J., Rajbhandari, S., Aminabadi, R. Y., Ruwase, O., Yang, S., Zhang, M., ... & He, Y. (2021). Zero-offload: Democratizing billion-scale model training. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)* (pp. 551-564).

[45] Li, Z., Wallace, E., Shen, S., Lin, K., Keutzer, K., Klein, D., & Gonzalez, J. (2020, November). Train big, then compress: Rethinking model size for efficient training and inference of transformers. In *International Conference on machine learning* (pp. 5958-5968). PMLR.

[46] Zhang, C., Ma, L., Xue, J., Shi, Y., Miao, Z., Yang, F., ... & Yang, M. (2023). Cocktailer: Analyzing and optimizing dynamic control flow in deep learning. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)* (pp. 681-699).

[47] Tan, C., Jin, Z., Guo, C., Zhang, T., Wu, H., Deng, K., ... & Xiang, D. (2019). NetBouncer: Active device and link failure localization in data center networks. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)* (pp. 599-614).

[48] Kumar, G., Dukkipati, N., Jang, K., Wassel, H. M., Wu, X., Montazeri, B., ... & Vahdat, A. (2020, July). Swift: Delay is simple and effective for congestion control in the datacenter. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication*

on the applications, technologies, architectures, and protocols for computer communication (pp. 514-528). [CrossRef]

[49] Alistarh, D., Grubic, D., Li, J., Tomioka, R., & Vojnovic, M. (2017). QSGD: Communication-efficient SGD via gradient quantization and encoding. *Advances in neural information processing systems, 30*.

[50] Barham, P., Chowdhery, A., Dean, J., Ghemawat, S., Hand, S., Hurt, D., ... & Wu, Y. (2022). Pathways: Asynchronous distributed dataflow for ml. *Proceedings of Machine Learning and Systems, 4*, 430-449.

Yuchang Mo received the B.E. (2002), M.S. (2004), and Ph.D. (2008) degrees in Computer Science from Harbin Institute of Technology, Harbin, China. He is currently a Distinguished Professor with the School of Computer Science and Technology, Zhejiang University of Water Resources and Electric Power, Hangzhou, China. He was a Visiting Scholar with the Department of Electrical and Computer Engineering, University of Massachusetts (UMass) Dartmouth, USA from September 2012 to August 2013. Prof. Mo was the Program Chair of the 5th Conference on Dependable Computing in China. He is currently a Senior Expert at the Reliability Research Society of China and a Senior Expert at the Technical Committee on Fault Tolerant Computing of China. His current research focuses on the reliability analysis of complex systems and networks using combinatorial models. His research has been supported by the National Science Foundation of China. (Email: yuchangmo@sina.com)

Jian Wan received the Ph.D. degree in computer application technology from Zhejiang University, Zhejiang, China, in 1996. His research interests include bioinformatics, service computing, and cloud computing. He is currently a Professor with the Zhejiang University of Water Resources and Electric Power, Hangzhou, China. (Email: jianwan@hotmail.com)

Hao Peng received the PhD degree in computer science and technology from Shanghai Jiaotong University in 2012. He is currently a full professor with the School of Computer Science and Technology, Zhejiang Normal University. His main research interests include AI security, IoT security, software and system security, data-driven security, Big Data mining and analysis. He has published more than 100 papers in prestigious journal and conferences. He served as program committee for more than ten international conferences. (Email: hpeng@sina.cn)

Ruiming Fang received Ph.D. degrees from Southeast University in 2002. Presently, he is a professor at Huaqiao University. His fields of interest are support vector machine and its application ine electrical engineering. (Email: fangrm@sina.com)

Yuan Fan, CEO of the Hangzhou Anheng Information Technology. His research interests include data security, network security, clouding computing. (Email: yuanfan@sina.com)

Chunyu Miao, SVP of the Hangzhou Anheng Information Technology. His research interests include data security, network security, security education. (Email: chunyumiao@sina.com)

Mirlan Chynybaev is the Rector of Kyrgyz State Technical University (KSTU) in Bishkek, Kyrgyzstan, a position he has held since December 2020. He earned his Master's degree in Applied Mechanics from KSTU in 2003 and later obtained a Ph.D. in Physics and Mathematics in 2008. Dr. Chynybaev specializes in geomechanics and has contributed to research on service quality in higher education institutions in Kyrgyzstan.

Faer Gui, CEO of the Zhejiang keepsoft Information Technology. His research interests include data modeling, IoT, digital twin, clouding computing. (Email: faergui@sina.com)

Rengui Zhang, CEO of the Zhejiang YuGong Information Technology. His research interests include data science, data mining, clouding computing. (Email: renguizhang@sina.com)

Shuying Zhai received Ph.D. degrees from Xinjiang University in 2014. Presently, she is a professor at Huaqiao University. His fields of interest are numerical calculation of differential equations, scientific calculation and numerical analysis. (Email: syzmath@sina.com)

Wen Wu received the B.Sc. degree in electronic and information engineering from Wuhan College of Arts and Science, Wuhan, China, in 2016, and the M.Sc. degree in computer applied technology from Hubei University, Wuhan, China, in 2019. He received the Ph.D. degree in computer science and technology from Hangzhou Dianzi University, Hangzhou, China, in 2025. From 2019 to 2022, he was a Lecturer in the Department of Computer Science at the Xinjiang Institute of Technology, Xinjiang, China. Since 2025, he has been with the School of Computer Science and Technology, Zhejiang University of Water Resources and Electric Power, Hangzhou, China, as a Faculty Member. In addition, since 2024, he has been a Research Assistant with the City University of Hong Kong, Hong Kong, China. His research interests include deep learning and computer vision. (Email: wuwen@sina.cn)

Jifeng Zhu was born in Shaoxing, Zhejiang, China. He received the B.S. degree from the School of Physics, Huazhong University of Science and Technology, the M.S. degree from the School of Information Engineering, China Jiliang University, and the Ph.D. degree from the School of Electronics and Information, Hangzhou Dianzi University. In 2024, he conducted research at Dublin City University, Ireland, as a Visiting Ph.D. Student. He is currently a lecturer with the School of Computer Science and Technology, Zhejiang University of Water Resources and Electric Power. His research interests include image segmentation, object detection, saliency detection, and visual object tracking in underwater environments. (Email: zhujifeng@sina.cn)