FaultSense: Fault Localization in Large-Scale Mixture-of-Experts Model Serving Infrastructure

Harish S A*, Vignesh S*, Ashwin Kurella,† Rohan Gandhi†, Praveen Tammana*
Indian Institute of Technology Hyderabad, India* Microsoft Research, India†

Abstract

While Mixture-of-Experts (MoE) models scale LLM serving across hundreds of GPUs, their reliance on all-to-all communication makes them susceptible to gray failures (*e.g.,* GPU stragglers) which inflate serving latency without explicit errors, complicating fault localization. We present FaultSense, an application-layer approach that localizes faulty GPUs and communication paths in MoE model serving replicas without host instrumentation. FaultSense introduces a lightweight MoE probe model prototype alongside a two-phase hierarchical fault localization algorithm. Modeling the cluster as a GPU communication graph, it performs: (1) a global coverage that tests all GPUs and communication paths with a small set of probes to identify suspicious components, and then (2) a drill-down into suspicious components to localize faulty GPUs or communication paths. Our design reduces the number of diagnostic tests compared to individually testing all components (up to 20×) while maintaining low memory overheads.

CCS Concepts

• **Computer systems organization** → **Dependable and fault-tolerant systems and networks**; • **General and reference** → *Reliability*; • **Networks** → *Data center networks*; • **Computing methodologies** → Machine learning.

Keywords

Fault localization, Mixture-of-Experts, LLM serving, GPU clusters, Gray failures, Group testing, Application layer diagnostics

ACM Reference Format:
Harish S A*, Vignesh S*, Ashwin Kurella,† Rohan Gandhi†, Praveen Tammana* . 2026. FaultSense: Fault Localization in Large-Scale Mixture-of-Experts Model Serving Infrastructure. In *17th ACM SIGOPS Asia-Pacific Workshop on Systems (APSys '26), September 17–18, 2026, Bangkok, Thailand*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3838177.3841737

APSys '26, Bangkok, Thailand

ACM ISBN 979-8-4007-2912-6/2026/09
https://doi.org/10.1145/3838177.3841737

1 Introduction

The global artificial intelligence landscape has evolved significantly as Large Language Models (LLMs) [42] have become increasingly integrated into enterprise workflows [1, 5, 12]. As LLMs operate at massive scale and in latency-sensitive production pipelines, performance thresholds have narrowed. Degradations in application-level user-visible latency, such as Time-to-First-Token (TTFT) or Time-Between-Tokens (TBT), now translate directly into productivity loss and SLO violations [4, 15, 41].

Traditional infrastructure monitoring relies on low-level signals such as CPU/GPU utilization, memory usage, network throughput, and node health [8, 9, 18]. However, these metrics often fail to capture application-level performance indicators such as request latency, TTFT, and TBT in model serving replicas of GPU clusters. As a result, systems may appear healthy despite gray failures that silently degrade user-visible performance [15, 22, 25, 28]. Additionally, workload imbalance across GPUs can overload specific GPUs without triggering infrastructure-level anomalies. Consequently, application engineers lack tools and mechanisms to rapidly localize faulty GPUs or communication paths responsible for latency inflation.

We present FaultSense, an application-layer fault localization approach for large-scale Mixture-of-Experts (MoE) model serving replicas. FaultSense relies solely on application-visible signals such as request latency, TTFT, and TBT, avoiding intrusive infrastructure monitoring. Its goals are twofold: *(1) Detection:* detect gray failures (soft performance degradations) that cause latency inflation. *(2) Localization:* pinpoint the faulty GPU or communication path between GPUs responsible for the slowdown. Together, these capabilities enable application engineers to rapidly reconfigure replicas and continue operations.

Detecting and localizing faults from the application layer is challenging in modern MoE architectures (§2.1). In an MoE model, a single request is routed through many expert layers distributed across GPUs [19], often involving almost every GPU and communication path in the replica. Thus, without visibility over low-level infrastructure signals (*e.g.,* switch queues, PCIe or NVLink error statistics, routing information, etc.), it is non-trivial to attribute latency inflation to a specific GPU or communication path.

Strawman approaches present challenges (more in §3): (1) Sending test prompts through the production MoE model provides poor localization because requests traverse a large fraction (almost all) of GPUs and communication paths. (2) Pairwise network probes (*e.g.,* pings between GPU pairs) bypass actual LLM serving kernels and collective communication behavior, missing application specific bottlenecks experienced by production traffic. Moreover, exhaustive pairwise testing scales poorly. For example, a 100-GPU replica [34] requires 4950 pairwise tests $\binom{100}{2}$, introducing high diagnostic overhead. To address these challenges, FaultSense introduces a lightweight, scalable approach based on the following techniques:

First, to ensure workload fidelity, we design a lightweight probe model that co-resides with the production model. To avoid execution across all GPUs in the replica, the probe model is restricted to a single MoE layer. Carefully crafted probe prompts deterministically activate specific experts (and hence GPUs), allowing each probe to test a chosen subset of GPUs and communication paths. Probes traverse similar application-layer kernels and collective operations (*e.g.,* all-to-all) as production traffic.

Second, because widespread faults are rare [25], exhaustive pairwise testing is inefficient. Instead, FaultSense uses group testing [16, 39] using the single MoE layer probe model. By leveraging its controlled expert activation, a single probe can test only a chosen group of GPUs and their interconnecting communication paths, allowing FaultSense to quickly verify healthy components using group testing.

Finally, to scale to large replicas, we structure these group tests into a two-phase hierarchical algorithm. A broad global coverage phase first identifies suspect regions, followed by a systematic drill-down to pinpoint the exact faulty GPU or communication path. For instance, in a 100-GPU replica, this reduces the required diagnostic probes from 4950 to ≈ 250, an ≈ 20× reduction.

Although FaultSense is currently designed for Mixture-of-Experts (MoE) architectures, its core principles generalize to dense models, where structured communication patterns similarly expose performance bottlenecks. Lastly, our goal is not to triage a specific component or find the root cause (interconnect link, NVSwitch, PCIe, etc.) of the slowdown. Instead, our goal is to localize faults so that application engineers can get actionable fault localization quickly evict faulty GPUs or reroute around degraded communication paths. In summary, our contributions are:

- FaultSense, an application-layer approach that detects and localizes faulty GPUs and communication paths in large MoE serving replicas.

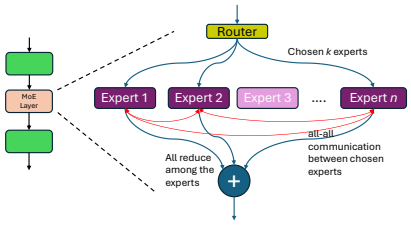


Figure 1: Routing and all-to-all communication in MoE model serving

- The design and prototyping of a lightweight single layer MoE probe model engineered to allow explicit expert selection for fault localization with low memory overheads.
- A two-phase hierarchical group testing algorithm that leverages the probe model to localize faults and scales to large replicas while reducing probe complexity by ≈ 20×.

2 Background and motivation

2.1 Background

MoE model serving. Modern LLMs increasingly adopt the Mixture-of-Experts (MoE) architecture [3, 34]. Unlike dense transformers where each layer applies attention followed by a fully connected FFN block, MoE layers use a lightweight router to select the top-k experts for each token (Figure 1). Experts are typically sharded across GPUs, and token activations are exchanged through all-to-all communication before expert outputs are aggregated and forwarded to subsequent layers. This conditional activation enables models to scale parameter count while keeping per-token computation approximately constant.

Gray failures in serving clusters. In large multi-GPU serving clusters, fail-slow or gray failures are common and often persist for long durations (*i.e.,* in the order of hours) without explicit error signals before detection [25, 43]. These failures include hardware and software degradations such as thermal throttling, memory errors, PCIe/NVLink degradation, resource contention, congestion, etc. [22]. Unlike hard failures (fail-stops [43]), gray failures leave components (*e.g.,* GPUs, communication path) operational but significantly slower, creating persistent stragglers that manifest by degrading application-level end-to-end latency metrics such as TTFT and TBT [44].

Cascading effect of gray failures (faults). Large-scale MoE replicas often span hundreds of GPUs and dozens of layers [34]. Since different layers route tokens to different experts, processing a single request may involve a large fraction (potentially all) of the GPUs in the replica. Consequently, MoE inference relies on tightly synchronized distributed execution, where a single slow GPU or degraded communication path can stall collective operations and cascade across the entire serving pipeline, increasing latency.

Application-layer visibility gap. When gray failures inflate end-to-end LLM serving latencies (*e.g.,* TTFT or TBT),

it is not easy for application teams to quickly localize the faulty GPU or communication path. Existing infrastructure-side tools [25, 43, 45] are typically built for training jobs and require privileged, per-host telemetry such as switch queues or PCIe/NVLink error statistics. In multi-tenant or managed environments, strict security boundaries keep this low-level telemetry with the infrastructure team, forcing application teams to wait on slow cross-team triage. Furthermore, standard libraries like NCCL [9] expose metrics only at the collective-communication level: a straggler caused by a GPU's slow compute appears merely as inflated collective time on the ranks that wait for it, so the telemetry cannot attribute the slowdown to a specific GPU's compute or a specific communication path. Recovering finer-grained visibility at this layer also imposes non-trivial overheads [6, 7].

FaultSense bridges this visibility gap from the application layer. By generating controlled probe workloads (probe prompts) and analyzing their end-to-end latencies, FaultSense rapidly pinpoints degraded components without requiring infrastructure-level access. This enables application teams to immediately reroute traffic and sustain LLM serving, complementing rather than replacing infrastructure side hardware diagnosis.

2.2 Problem statement and requirements

Problem statement. The goal of FaultSense is to detect and localize faulty GPUs and communication paths between them responsible for elevated serving latency (*e.g.,* TTFT, TBT) in large-scale MoE serving replicas from the application-layer without relying on privileged infrastructure telemetry or modifications to the serving stack. We identify the following requirements for such an approach:

[R1] ***Fault-agnostic detection:*** The system must detect both hard failures and gray failures that inflate end-to-end latency (TTFT/TBT) without relying on explicit error signals or extensive log analysis [27].

[R2] ***Low instrumentation:*** To ensure deployability across heterogeneous hardware (*e.g.,* NVIDIA [23], AMD [37]) and serving stacks (*e.g.,* vLLM [14], HuggingFace TGI [13]), the framework must operate without kernel hooks, shim layers, or modifications to the serving runtime.

[R3] ***Agile, Precise and Lightweight localization:*** The system must rapidly distinguish healthy from faulty components and precisely localize the degraded GPU or communication path [2] while imposing minimal overhead on production serving.

3 FaultSense's approach and challenges

Our approach. FaultSense uses active application-layer probing. Operating entirely within the application layer's (*e.g.,* a tenant's) permission boundaries, it injects carefully constructed probes to localize faults. Unlike passive monitoring, our probing approach allows testing specific groups of GPUs and communication paths to strategically localize faults. However, designing such probes present challenges:

1. Production model probes mask faults: Sending test requests (prompts) through the active production model is ineffective for localization. As mentioned in §2.1, modern MoE models route tokens through dozens of layers, causing a single request to scatter across nearly every GPU in the replica. If such a test prompt's latency increases (or stalls), its highly entangled execution path makes it impossible to attribute the delay to a specific GPU or communication path.

2. Individual diagnostic tests lack workload fidelity: To bypass the entangled execution paths, one could test GPUs and communication paths independently using compute workloads and network or collective traffic. However, neither captures the coupled compute-communication behavior of MoE inference. That is, a collective-only probe can miss a compute-slow GPU, while a compute-only probe leaves degraded communication paths untested [22, 33]. Combining both in an external probe still fails to reproduce how the serving stack interleaves expert computation with all-to-all communication, where individually healthy components may become stragglers under contention. Faithfully reproducing this production execution path would require modifying the runtime, violating **[R2]**.

3. Application-aware pairwise testing does not scale: Achieving fidelity requires application-aware point-to-point probes between GPU pairs. However, this forces GPUs to load diagnostic models (for each unique pairs of GPUs) to test their compute kernels and communication paths between them. In a 100-GPU replica, this approach requires $\binom{100}{2} = 4950$ pairwise probes leading to high compute and memory overheads. It is essentially an upper bound in probe count that tests every path once. Alternatives like sampling may be cheaper, but they can skip communication paths and could also miss the degraded components. FaultSense instead belongs to the adaptive, graph-aware group-testing class [39]: a single probe covers many paths at once, so Phase 1 tests every path at least once while issuing far fewer probes, and Phase 2 spends adaptively only on suspects (§4.3).

4 System design

4.1 Design choices in FaultSense

1. Lightweight probe model: FaultSense co-locates a single-layer MoE probe model with the production model and runs it on the unmodified serving stack, so each probe drives the similar expert-compute and all-to-all collectives, on the same GPU streams as production traffic without any runtime changes (**[R2]**). Custom routing lets crafted prompts (probe

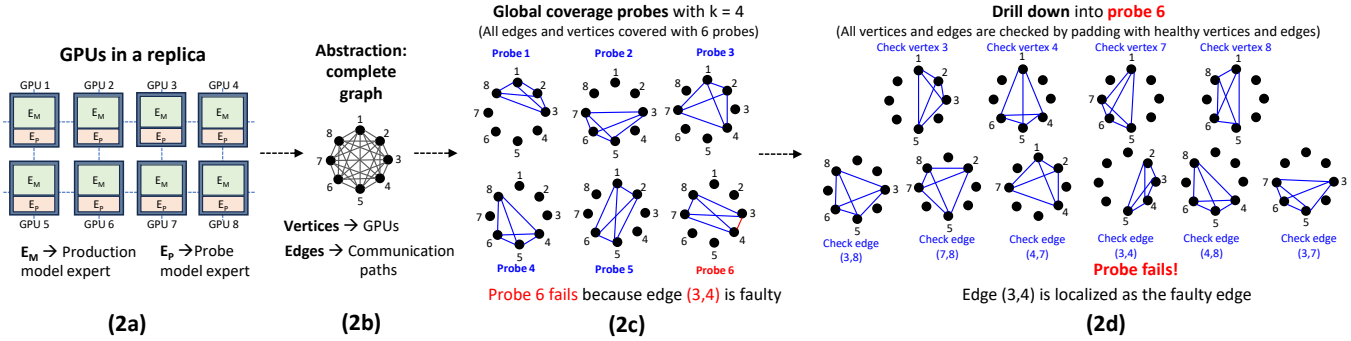


Figure 2: Global coverage and Adaptive drill-down probing §4.3 example with n=8 and k=4

prompts) deterministically activate chosen experts. By pinning one expert per GPU, each probe targets a selected subset of GPUs and their communication paths for fault localization (§4.4). MoE is essential, since only its sparse top-k router gives the per-GPU selectivity that a dense model activating every GPU cannot, and a single layer is the smallest unit reproducing the full router, all-to-all, and expert-compute pattern, keeping memory negligible (§5.2).

2. Two-phase hierarchical group testing: Because hardware faults in GPU clusters are typically sparse [25], FaultSense leverages adaptive group testing [16, 39] to avoid the overhead of exhaustive pairwise probing. Since the probe model's MoE router sends tokens to only a fixed number of experts (k), a single probe automatically tests a group of k GPUs and all interconnecting communication paths simultaneously. If the probe latency remains normal, the entire group is verified as healthy (more in §4.3).

Scope and limitations. FaultSense is an application-driven fault localization approach and does not attempt to root-cause underlying hardware failures. We also leave highly transient software faults (*e.g.,* KV cache misses or router imbalance) to future work, focusing instead on persistent gray failures that degrade user-visible latency.

4.2 Topology abstraction

To develop the two phase algorithm, we abstract the physical serving replica as an undirected complete graph $G = (V, E)$. This abstraction (Figure 2a and Figure 2b) captures both compute and communication of a replica while remaining agnostic to the underlying hardware topology.

Vertices: Each vertex $v \in V$ represents a physical GPU participating in the replica. For an n-GPU replica, $|V| = n$. We place one expert per GPU, so experts correspond one-to-one with vertices.

Edges: Each edge $(u, v) \in E$ represents the logical communication path between GPUs u and v. The edge is a logical link between two GPUs whereas physically many network links could be involved. From the perspective of MoE all-to-all communication, this effectively forms a fully connected logical mesh, *i.e.,* , the complete graph G. Hence, $|E| = n(n-1)/2$.

Probe: One probe invokes k experts ($k \leq n$) and, in turn, the k GPUs hosting them. Under the graph abstraction, a probe corresponds to testing k vertices and all $k(k-1)/2$ edges among them.

Actionability of the abstraction. An edge abstracts the shared physical substrate (*e.g.,* switches, NICs, PCIe/NVLink) a path traverses, since FaultSense operates at the application layer rather than root-causing hardware. A diagnosis is thus actionable. That is, a slow vertex or edge tells an operator which GPU to evict or path to reroute around. A shared bottleneck may surface as several edges slowing together. FaultSense reports this co-failing set, and an operator with the topology can intersect them to find the shared resource, while quick application-layer mitigation stays available.

Algorithm 1 Global coverage algorithm

Input: (1) Graph $G(V, E)$; (2) Probe size k
Initialise:
  Uncovered edges: $U \leftarrow E$
  Probe set: $P \leftarrow \emptyset$
while $U \neq \emptyset$ **do**
  Select vertex s with maximum uncovered incident edges
  Initialize probe $p \leftarrow \{s\}$
  while $|p| < k$ **do**
    Select vertex v^* with maximum uncovered edges to vertices already in p
    $p \leftarrow p \cup \{v^*\}$
  end while
  Remove $\{(u, v) : u, v \in p\}$ from U
  $P \leftarrow P \cup \{p\}$
end while
return P

4.3 Two phase fault localization

Phase 1: Global coverage probing. Each probe tests k vertices (GPUs) and all interconnecting edges (communication paths). To minimize diagnostic overhead and production interference, FaultSense must cover all n vertices and $n(n-1)/2$ edges using minimum number of probes. Finding this optimal set is NP-hard [21]. Thus, FaultSense uses a greedy algorithm (Algorithm 1). It iteratively constructs

size-k probes by first selecting the vertex with the most uncovered incident edges, then greedily adding vertices that maximize edge coverage until all edges are tested (*e.g.,* covering $n = 8, k = 4$ requires just 6 probes in Figure 2c). The algorithm runs in $O(n^2k)$ and is a one-time offline operation.

Phase 2: Adaptive drill-down probing. Phase 1 classifies probes as *healthy* (succeeded) or *suspect* (slow/failed). It is possible that two unrelated faults caused two probes to fail where common vertices or edges are not the culprit. Thus, merely intersecting suspect groups is insufficient for localization. Instead, the drill-down procedure (Algorithm 2) systematically isolates faults. It constructs size-k probes that combine a single suspect vertex (or edge) exclusively with known healthy vertices. Any subsequent failure definitively isolates the fault to that suspect edge/vertex (*e.g.,* isolating suspect group 6 requires 10 localized probes in Figure 2d). This algorithm requires at most $\frac{k(k-1)}{2} + k$ probes per suspect group, yielding a complexity of $O(C_s \cdot k^2)$ for C_s suspect groups. Since both k (*e.g.,* 4-8) and C_s are small in practice, this step remains lightweight.

4.4 Probes in FaultSense

Constructing the probe model: To deterministically activate specific experts, we construct a single-layer probe model from OLMoE-1B-7B [17] by retaining only its first MoE layer. We suppress self-attention by zeroing its output weights, allowing injected embeddings to pass directly to the MoE router. We then replace the learned gating matrix with an identity matrix, making the i^{th} embedding dimension directly select expert i. This enables arbitrary expert combinations to be activated using custom sparse embeddings without modifying the serving stack or runtime satisfying **[R2]**. For example, under top-4 routing, the embedding $[0, 1, 0, 1, 0, 1, 1, 0, \dots]$ provided as input to the model deterministically activates experts $\{1, 3, 5, 6\}$.

Determining probe outcomes: Hard failures (*e.g.,* fail-stops) are detected via probe timeouts. To detect gray failures irrespective of root cause (to satisfy **[R1]**), FaultSense uses a relative latency baseline instead of static thresholds. Let T_i denote the latency of probe i and $\tilde{T}$ the median latency across Phase 1 probes. A probe is marked suspect if $T_i > \alpha\tilde{T}$, where α is configurable.

The median suits the sparse-fault regime targeted by group testing (§4.1). Since faults affect only a minority of probes [25], $\tilde{T}$ remains dominated by healthy probes. Relative thresholding also adapts to uniform latency shifts from load or dynamic batching because $\tilde{T}$ and $\alpha\tilde{T}$ shift together. Transient congestion may still trigger a suspect flag, but unlike persistent gray failures that can last for hours [25, 44, 45], it is short-lived. Phase 2 drill-down filters such anomalies without adding verification rounds to Phase 1. Quantifying false-positive and false-negative rates across α, load, and fault count is left to future work.

Algorithm 2 Adaptive drill-down algorithm

```
Input: (1) Graph G(V, E); (2) Probe size k;
    (3) Healthy vertices V_h; (4) Healthy edges E_h
Initialize:
    Suspected vertices: V_s ← V − V_h
    Suspected edges: E_s ← E − E_h
    Probes: P_L ← ∅
for each v ∈ V_s and (u, v) ∈ E_s do
    Vertex: p ← {v}; Edge: p ← {u, v}
    while |p| < k do
        Select w ∈ V_h \ p uniformly at random
        such that (w, v) ∈ E_h for all v ∈ p
        p ← p ∪ {w}
    end while
    P_L ← P_L ∪ {p}
end for
return P_L
```

5 Experiments

Through our experiments, we address three key questions: (1) How does FaultSense's probe scalability compare with naive pairwise testing? (2) What is the probe model's memory overhead? (3) Does probe traffic affect production model latency?

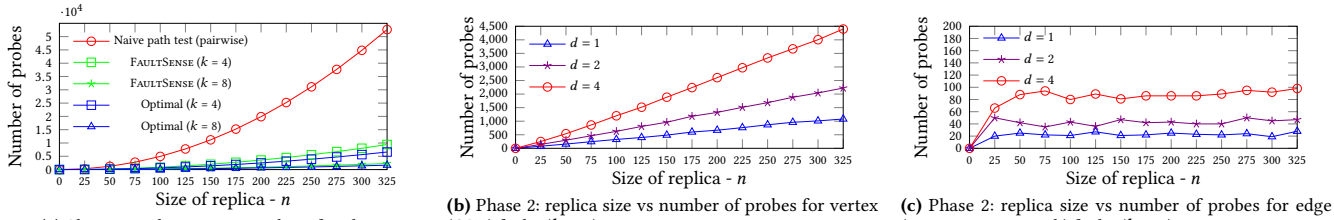
5.1 Experimental setup and methodology

Simulation: To evaluate fault localization at scale, we simulate a complete graph topology (§4.2) on an 80-core Intel Xeon server. We examine the number of probes required across phases to accurately determine the faulty component. Inspired by production MoE replicas [34], we vary the replica size n from 25 to 325 GPUs. We evaluate probe sizes $k \in \{4, 8\}$ against $d \in \{1, 2, 4\}$ concurrent failures that occur uniformly at random on either a vertex (GPU) or edge (communication path) or both.

GPU testbed: We evaluate memory footprint and probe traffic interference of the probe model on the production model on a 4-GPU setup (each GPU in a different host) connected via 1Gbps Ethernet. The cluster contains three RTX A5000s [11], one RTX4500 ADA [10] each with 24GB VRAM and serves MoE models using expert parallelism orchestrated by vLLM [30] v0.21.0 and Ray [36] v2.55.1.

5.2 Results and discussion

Probe scalability. FaultSense achieves near-optimal scalability while reducing diagnostic overhead. In Phase 1, FaultSense's probe counts follow the theoretical minimum (Figure 3a), requiring over 22× fewer probes than exhaustive pairwise testing for a 300-GPU replica. Phase 2 probe overhead (Figure 3b, Figure 3c) scales with the number of faults rather than replica size, as it targets only suspect components. Overall, FaultSense yields an $\approx$ 20× reduction in total probes (Figure 4b). FaultSense successfully localized all faults in simulation.

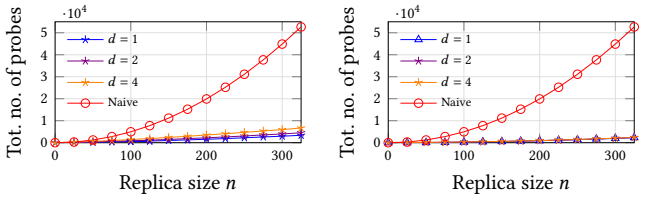


(a) Phase 1: replica size vs number of probes

(b) Phase 2: replica size vs number of probes for vertex (GPU) faults ($k = 8$)

(c) Phase 2: replica size vs number of probes for edge (communication path) faults ($k = 8$)

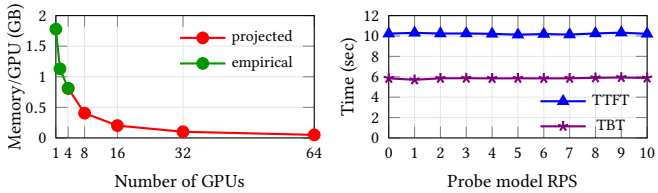
Figure 3: Replica size vs phase-wise number of probes



(a) Vertex (GPU) faults

(b) Edge (path) faults

Figure 4: Replica size vs total number of probes (phase 1 + 2) for ($k = 8$)



(a) Memory overhead

(b) Probe traffic interference

Figure 5: Evaluation of FaultSense on GPU testbed

However, an evaluation of localization accuracy on a large-scale GPU cluster testbed is left to future work. Additionally, since a single FaultSense probe can contain multiple pairwise operations, performing a communication-cost normalized comparison is also left to future work.

Memory overhead. We evaluate the per GPU memory footprint of FaultSense's co-located probe model under expert parallelism (Figure 5a). Empirical measurements show that memory occupied by the probe model per GPU scales inversely with cluster size, dropping from 1820MB on a single GPU to 834MB when sharded across 4 GPUs. Projecting this trend to production scale replicas, the per GPU footprint becomes negligible. This confirms that FaultSense can safely co-reside on the infrastructure without starving the production model of memory.

Probe traffic interference. To verify that active probing does not degrade the production model's (Deepseek-moe-16b-chat [24]) performance, we instantiate both the probe and production model on the 4-GPU testbed and subject the production model to a continuous baseline load of 5 RPS. We then scale the injected probe traffic from 0 up to 10 RPS and monitor the impact on the latency metrics of the production model. As shown in Figure 5b, the production model's Time-to-First-Token (TTFT) and Time-Between-Tokens (TBT) remain stable. Despite the frequency of probes, we observe very negligible TTFT inflation (at max $\approx$ 1%) and almost no increase in TBT between 0 to 10 RPS. This indicates that FaultSense's lightweight probes can execute concurrently without disrupting active user traffic partially satisfying **[R3]**.

6 Related work

Infrastructure-level fault diagnostics. Prior systems for GPU clusters [22, 25, 28, 29, 43, 45] and datacenter networks [20, 26, 32, 35, 38, 40] rely on privileged telemetry such as host metrics, kernel traces, switch logs, and explicit topology knowledge to localize faults. FaultSense complements these infrastructure centric tools by enabling fault localization purely from the application layer.

Resilient model serving. Works like [31, 44] mitigate latency impacts during gray failures, but they assume faulty components have already been identified via infrastructure signals. FaultSense complements them by providing a lightweight, application-level diagnostic signal to trigger these recovery mechanisms.

7 Conclusion and future work

We present FaultSense, a novel approach to localize faults in model serving GPU cluster replicas from the application layer. We prototype a single layer MoE probe model that can test specific subsets of GPUs and communication paths, enabling targeted diagnostics from the application layer without requiring privileged telemetry or infrastructure instrumentation. Our immediate future work will evaluate FaultSense's fault localization accuracy on a large-scale GPU cluster under realistic workloads.

Acknowledgments

We thank Jaideep Singh Kaler for technical support provided during the early stages of this work. This work is supported by the Prime Minister's Research Fellowship (PMRF), India and ZF India Technology Center.

References

[1] [n. d.]. Google Gemini. https://gemini.google.com/.

[2] [n. d.]. InfiniBand. https://www.nvidia.com/en-in/networking/products/infiniband/.

[3] [n. d.]. Llama MoE models. https://ai.meta.com/blog/llama-4-multimodal-intelligence/.

[4] [n. d.]. LLM performance. https://redis.io/blog/how-to-improve-llm-ux-speed-latency-and-caching/.

[5] [n. d.]. Microsoft M365. https://m365.cloud.microsoft/.

[6] [n. d.]. NCCL overhead. https://www.olcf.ornl.gov/wp-content/uploads/OLCF_AI_Training_0417_2024.pdf.

[7] [n. d.]. NCCL overhead 2. https://docs.cloud.google.com/ai-hypercomputer/docs/nccl/collect-and-understand.

[8] [n. d.]. NVIDIA DCGM. https://developer.nvidia.com/dcgm.

[9] [n. d.]. Nvidia NCCL library. https://developer.nvidia.com/nccl.

[10] [n. d.]. NVIDIA RTX A4500 Ada Generation Graphics Card. https://www.nvidia.com/en-in/products/workstations/rtx-4500/.

[11] [n. d.]. NVIDIA RTX A5000 Ada Generation Graphics Card. https://www.nvidia.com/en-in/products/workstations/rtx-a5000/.

[12] [n. d.]. OpenAI ChatGPT. https://chatgpt.com/.

[13] [n. d.]. Text Generation Inference Toolkit. https://huggingface.co/docs/text-generation-inference/en/index.

[14] [n. d.]. vLLM model serving framework. https://docs.vllm.ai/en/latest/.

[15] Amey Agrawal, Anmol Agarwal, Nitin Kedia, Jayashree Mohan, Souvik Kundu, Nipun Kwatra, Ramachandran Ramjee, and Alexey Tumanov. 2024. Etalon: holistic performance evaluation framework for llm inference systems. *arXiv preprint arXiv:2407.07000* (2024).

[16] Matthew Aldridge, Oliver Johnson, and Jonathan Scarlett. 2019. Group testing: an information theory perspective. *Foundations and Trends® in Communications and Information Theory* 15, 3-4 (2019), 196–392.

[17] Allen Institute for AI. 2024. OLMoE-1B-7B-0924-Instruct. https://huggingface.co/allenai/OLMoE-1B-7B-0924-Instruct. Accessed: 2026-05-21.

[18] Behnaz Arzani, Selim Ciraci, Luiz Chamon, Yibo Zhu, Hongqiang Harry Liu, Jitu Padhye, Boon Thau Loo, and Geoff Outhred. 2018. 007: Democratically finding the cause of packet drops. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. 419–435.

[19] Weilin Cai, Juyong Jiang, Fan Wang, Jing Tang, Sunghun Kim, and Jiayi Huang. 2024. A survey on mixture of experts. *Authorea Preprints* (2024).

[20] Yan Chen, David Bindel, and Randy H Katz. 2003. Tomography-based overlay network monitoring. In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*. 216–231.

[21] Derek G Corneil and Jean Fonlupt. 1988. The complexity of generalized clique covering. *Discrete Applied Mathematics* 22, 2 (1988), 109–118.

[22] Weihao Cui, Ji Zhang, Han Zhao, Chao Liu, Jian Sha, Bo Sang, Bingsheng He, Minyi Guo, and Quan Chen. 2026. {FLARE}: Anomaly Diagnostics for Divergent {LLM} Training in {GPU} Clusters of {Thousand-Plus} Scale. In *23rd USENIX Symposium on Networked Systems Design and Implementation (NSDI 26)*. 1021–1035.

[23] William J Dally, Stephen W Keckler, and David B Kirk. 2021. Evolution of the graphics processing unit (GPU). *IEEE Micro* 41, 6 (2021), 42–51.

[24] DeepSeek-AI. 2024. DeepSeek-MoE-16B-Chat. https://huggingface.co/deepseek-ai/deepseek-moe-16b-chat. Accessed: 2026-05-21.

[25] Yangtao Deng, Xiang Shi, Zhuo Jiang, Xingjian Zhang, Lei Zhang, Zhang Zhang, Bo Li, Zuquan Song, Hang Zhu, Gaohong Liu, et al. 2025. Minder: Faulty machine detection for large-scale distributed model training. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*. 505–521.

[26] Chuanxiong Guo, Lihua Yuan, Dong Xiang, Yingnong Dang, Ray Huang, Dave Maltz, Zhaoyi Liu, Vin Wang, Bin Pang, Hua Chen, et al. 2015. Pingmesh: A large-scale system for data center network latency measurement and analysis. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. 139–152.

[27] Zhihan Jiang, Junjie Huang, Guangba Yu, Zhuangbin Chen, Yichen Li, Renyi Zhong, Cong Feng, Yongqiang Yang, Zengyin Yang, and Michael Lyu. 2025. L4: Diagnosing large-scale llm training failures via automated log analysis. In *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering*. 51–63.

[28] Zhihan Jiang, Rui Ren, Guangba Yu, Yulun Wu, Wenwei Gu, Yichen Li, Yujie Huang, Cong Feng, Zengyin Yang, Yongqiang Yang, et al. 2025. LLMPrism: Black-box Performance Diagnosis for Production LLM Training Platforms. In *2025 55th Annual IEEE/IFIP International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S)*. IEEE, 1–7.

[29] Jakob Krebs, Dimitry Gavrilenko, Daniel Amir, Shir Landau Feibish, and Mark Silberstein. 2025. FlowPulse: Catching Network Failures in ML Clusters. In *Proceedings of the 24th ACM Workshop on Hot Topics in Networks*. 139–148.

[30] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*. 611–626.

[31] Myungjin Lee, Akshay Jajoo, and Ramana Rao Kompella. 2024. Enabling Elastic Model Serving with MultiWorld. *arXiv preprint arXiv:2407.08980* (2024).

[32] Huikang Li, Yi Gao, Wei Dong, and Chun Chen. 2017. Taming both predictable and unpredictable link failures for network tomography. In *Proceedings of the ACM Turing 50th Celebration Conference-China*. 1–10.

[33] Wenxiang Lin, Xinglin Pan, Lin Zhang, Shaohuai Shi, Xuan Wang, and Xiaowen Chu. 2025. HierMoE: Accelerating MoE Training with Hierarchical Token Deduplication and Expert Swap. *arXiv preprint arXiv:2508.09591* (2025).

[34] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. 2024. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437* (2024).

[35] Liang Ma, Ting He, Kin K Leung, Ananthram Swami, and Don Towsley. 2013. Identifiability of link metrics based on end-to-end path measurements. In *Proceedings of the 2013 conference on Internet measurement conference*. 391–404.

[36] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I Jordan, et al. 2018. Ray: A distributed framework for emerging {AI} applications. In *13th USENIX symposium on operating systems design and implementation (OSDI 18)*. 561–577.

[37] Nathan Otterness and J Anderson. 2020. AMD GPUs as an alternative to NVIDIA for supporting real-time workloads. In *Proceedings of the 32nd Euromicro Conference on Real-Time Systems*.

[38] Yanghua Peng, Ji Yang, Chuan Wu, Chuanxiong Guo, Chengchen Hu, and Zongpeng Li. 2017. {deTector}: a topology-aware monitoring system for data center networks. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. 55–68.

[39] Saurabh Sihag, Ali Tajer, and Urbashi Mitra. 2021. Adaptive graph-constrained group testing. *IEEE Transactions on Signal Processing* 70 (2021), 381–396.

[40] Cheng Tan, Ze Jin, Chuanxiong Guo, Tianrong Zhang, Haitao Wu, Karl Deng, Dongming Bi, and Dong Xiang. 2019. {NetBouncer}: Active device and link failure localization in data center networks. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. 599–614.

[41] Felicia Fang-Yi Tan, Moritz Alexander Messerschmidt, Wen Yin, and Oded Nov. 2026. The Impact of Response Latency and Task Type on

Human-LLM Interaction and Perception. In *ACM CHI*.

[42] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. [n. d.]. Attention is all you need. In *NeurIPS 2017*.

[43] Tianyuan Wu, Wei Wang, Yinghao Yu, Siran Yang, Wenchao Wu, Qinkai Duan, Guodong Yang, Jiamang Wang, Lin Qu, and Liping Zhang. 2025. {GREYHOUND}: Hunting {Fail-Slows} in {Hybrid-Parallel} Training at Scale. In *2025 USENIX Annual Technical Conference (USENIX ATC 25)*. 731–747.

[44] Ziyi Xu, Zhiqiang Xie, Swapnil Gandhi, and Christos Kozyrakis. 2025. FailSafe: High-performance Resilient Serving. *arXiv preprint arXiv:2511.14116* (2025).

[45] Zhiyi Yao, Pengbo Hu, Congcong Miao, Xuya Jia, Zuning Liang, Yuedong Xu, Chunzhi He, Hao Lu, Mingzhuo Chen, Xiang Li, et al. 2025. Holmes: Localizing irregularities in {LLM} training with mega-scale {GPU} clusters. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*. 523–540.