EROICA: Online Performance Troubleshooting for Large-Scale Model Training

Yu Guan, Zhiyu Yin, Haoyu Chen, Sheng Cheng, Chaojie Yang, Kun Qian,
Tianyin Xu, Pengcheng Zhang, Yang Zhang, Hanyu Zhao, Yong Li, Dennis Cai, Ennan Zhai

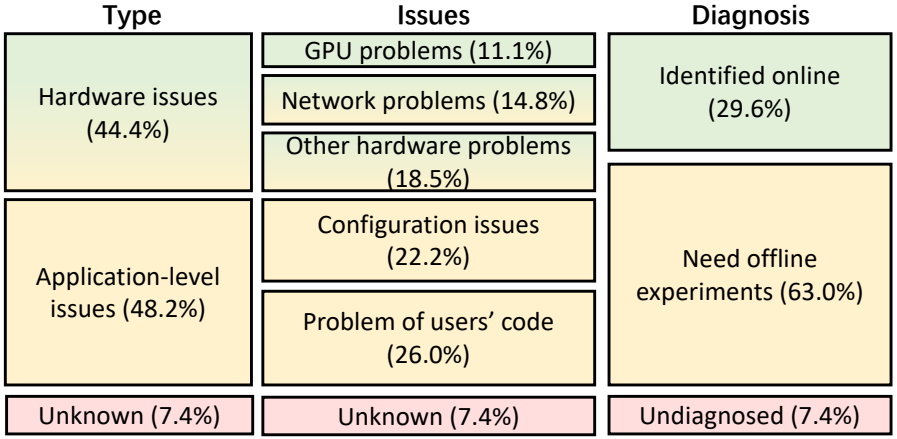
Alibaba Group & University of Illinois Urbana-Champaign

Training Performance: Complex Issues Across the Entire Stack

Anything can go wrong:

- </> User Code & Data Pipeline
- FW Training Framework (PyTorch, Megatron, Verl, Slurm, Jax)
- Kernel GPU Kernel, CPU Operators, Communication Libraries
- HW GPU, CPU, Memory, NVLink, PCIe, NIC, Disk, Network, Remote Storage

Root Causes (100K GPUs, 9 months)



Any line of code, or one hardware on a single machine, can cause problems

How to quickly find which line of code, or which hardware on which machine, is the root cause?

Monitoring Sees Wide, Profiling Sees Deep

Online Monitoring

DCGM, NCCL Profiler, EBPF

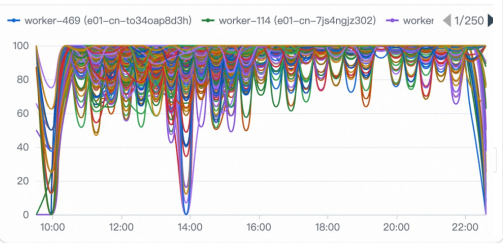
✓ Low overhead

✓ All workers, always on

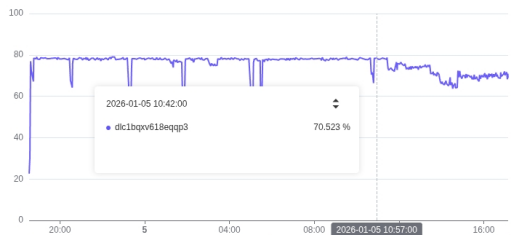
✗ Coarse-grained hardware sampling

✗ Can't see detailed code issues

WIDE coverage
but SHALLOW visibility

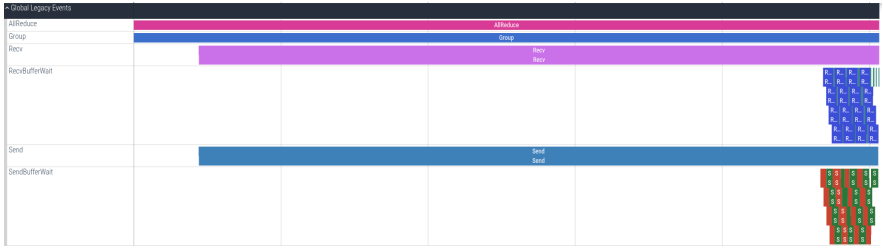


GPU Utilization



CPU Utilization

Remote Storage



Communication

```
root@jordalgo $ bpftrace -e 'tracepoint:syscalls:sys_enter_open { print((pid, str(args->
filename))); }'
Attaching 1 probe...
(4088075, /proc/sys/vm/overcommit_memory)
(4088075, /sys/kernel/mm/transparent_hugepage/enabled)
(4088134, /proc/sys/vm/overcommit_memory)
(4088134, /sys/kernel/mm/transparent_hugepage/enabled)
(4088211, /proc/sys/vm/overcommit_memory)
(4088211, /sys/kernel/mm/transparent_hugepage/enabled)
(4088245, /proc/sys/vm/overcommit_memory)
(4088245, /sys/kernel/mm/transparent_hugepage/enabled)
```

CPU function and System Call

Monitoring Sees Wide, Profiling Sees Deep

Offline Profiling

Nsight Systems, Torch Profiler

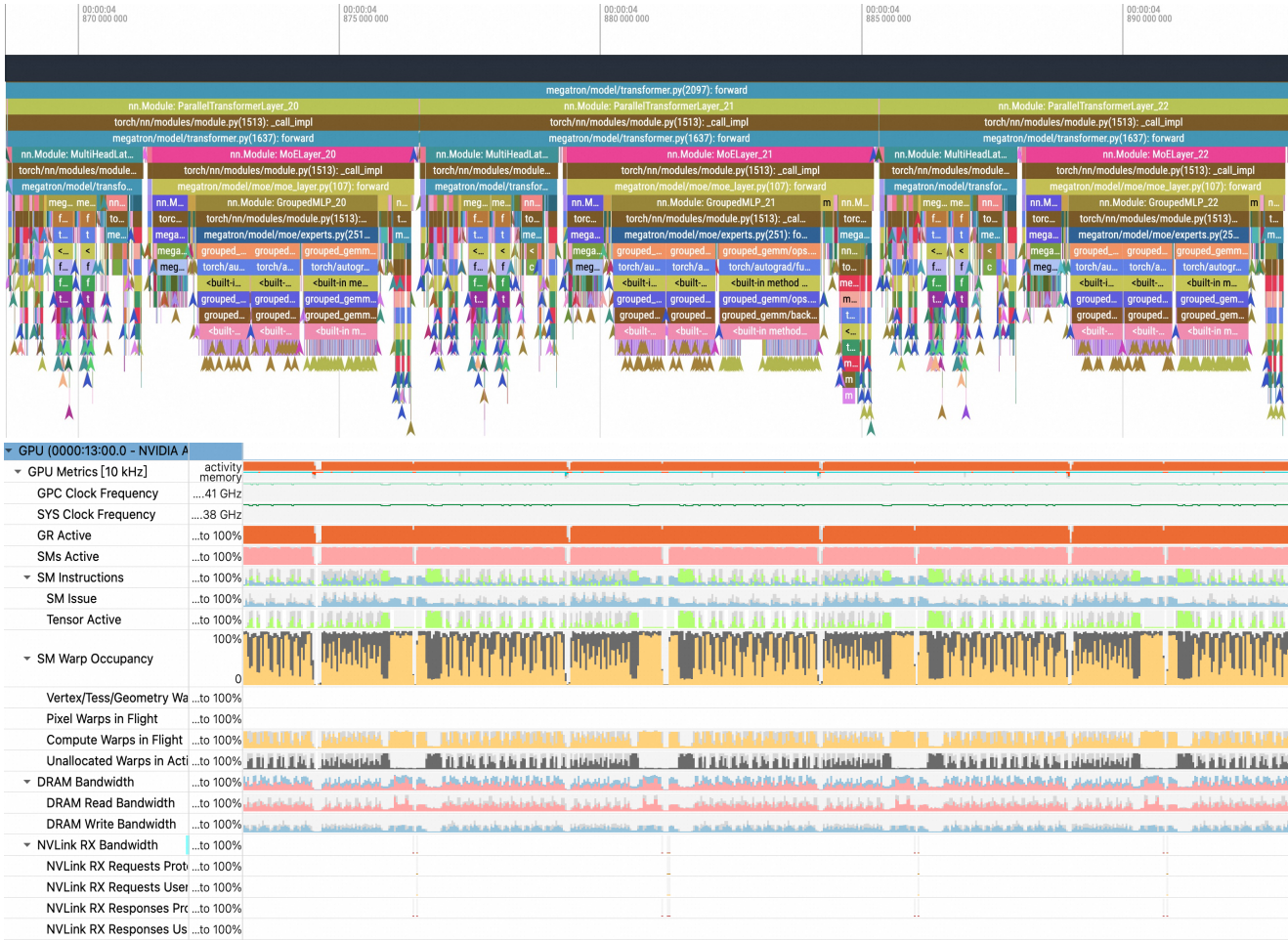
✗ ~10 TB-scale data (can not consume in realtime)

✗ Usually a few workers, a few training iterations

✓ Detailed code visibility

✓ Precise hardware tracing

DEEP visibility
but NARROW coverage



Monitoring Sees Wide, Profiling Sees Deep — Can We Have Both?

Online Monitoring

DCGM, NCCL Profiler, EBPF

✓ Low overhead

✓ All workers, always on

✗ Coarse-grained hardware sampling

✗ Can't see detailed code issues

WIDE coverage
but SHALLOW visibility

?

Offline Profiling

Nsight Systems, Torch Profiler

✗ ~10TB-scale data (can not consume in realtime)

✗ Usually a few workers

✓ Detailed code visibility

✓ Precise hardware tracing

DEEP visibility
but NARROW coverage

=

Online Profiling?

Combine both strengths

✓ Cover ALL workers

✓ Online, Low overhead

✓ Detailed code visibility

✓ Precise hardware tracing

WIDE + DEEP
How?

Challenges of Online Profiling & Troubleshooting

1 Overwhelming Data Volume

Fine-grained profiling produces **~100 MB/s per worker**

For a 10,000-GPU task: **~1 TB/s total**

- Impractical to store or transfer
- Profiling itself slows down training

2 Needle in a Haystack

Loading all data is even impractical.

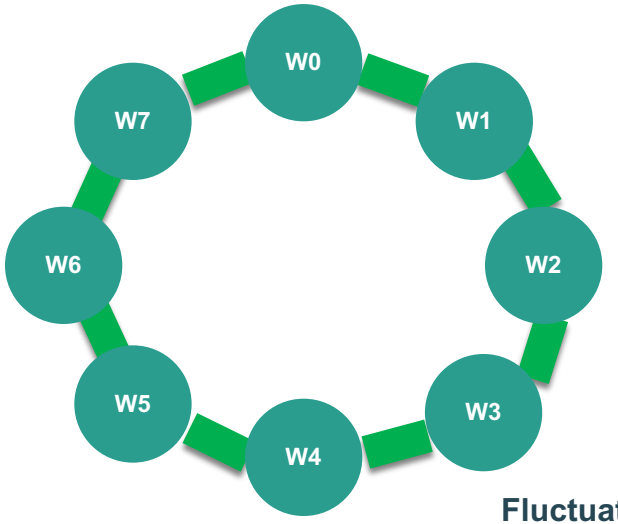
Naïve dropping / downsampling loses critical clues for troubleshooting

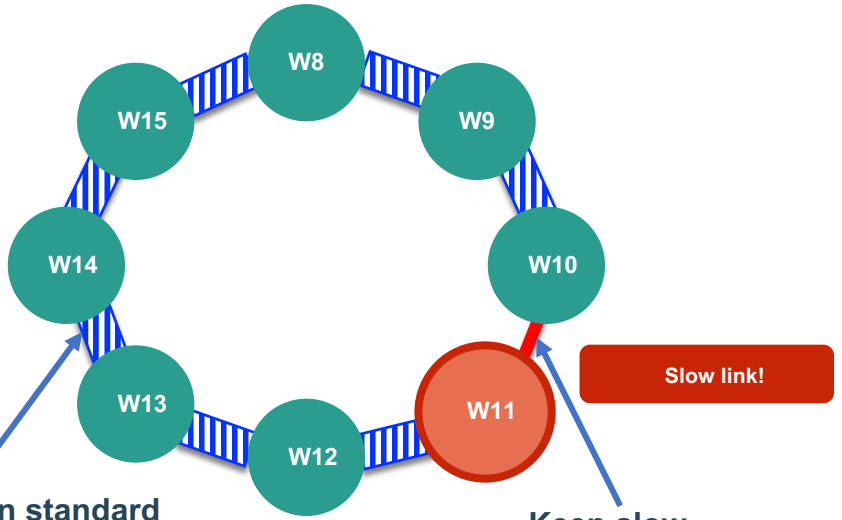
Cannot compare timestamps across workers due to clock desynchronization.

How to efficiently spot issues in massive data?

Insight: Most Workers Behave Similarly — Differences Reveal Problems

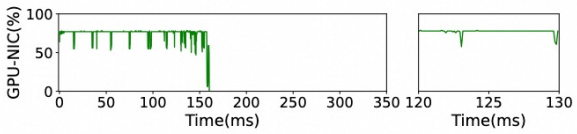
An Example of the function execution of Ring AllReduce: 16 workers, each 8 workers pass data around a ring

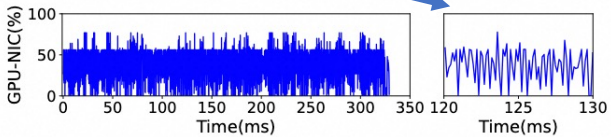


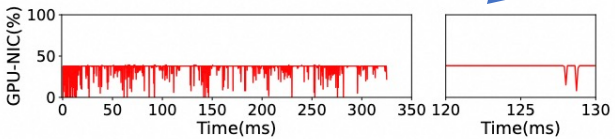


Fluctuate between standard throughput and zero

Keep slow







Insight: Describe function execution behaviors by "Behavior Fingerprint"

Average and std of bandwidth utilization as “Behavior Fingerprint”

Average	Standard deviation
W0: 85%	W0: 2%
W1: 83%	W1: 4%
W2: 86%	W2: 3%
W3: 82%	W3: 5%
W4: 84%	W4: 1%
W5: 85%	W5: 7%
W6: 87%	W6: 4%
W7: 84%	W7: 4%

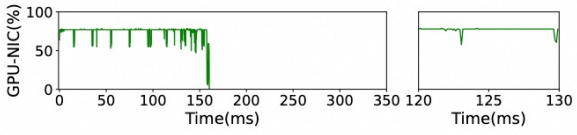
Average	Standard deviation
W8: 45%	W8: 85%
W9: 43%	W9: 83%
W10: 46%	W10: 86%
W12: 44%	W12: 84%
W13: 45%	W13: 85%
W14: 47%	W14: 87%
W15: 44%	W15: 84%

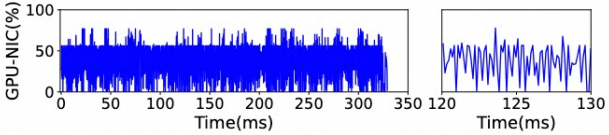
Average	Standard deviation
W11: 44%	W11: 3%

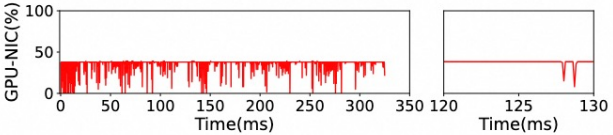
W11 immediately stands out!

Independent of absolute timestamp – No need of clock synchronization

A well-designed behavior fingerprint can tell the difference







EROICA Overview: Extract Behavior Fingerprints and Compare

1 Detect Degradation
Monitor iteration time
Trigger only when slow
Zero routine overhead

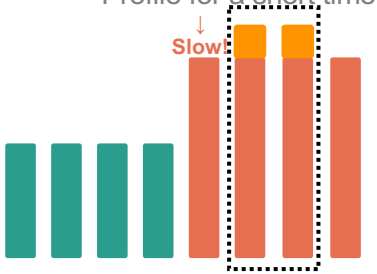
2 Profile ALL Workers
Synchronized: same iterations
GPU, NIC, memory, Python
~3GB raw data / worker

3 Fingerprint Extraction
Describe behaviors of all functions executed on all workers
3GB → 30KB per worker

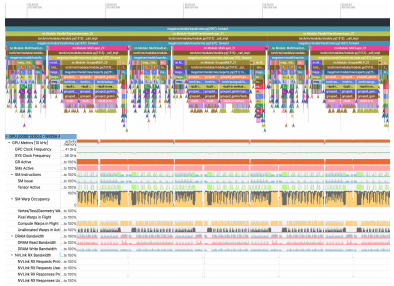
4 Compare & Localize
Cross-worker comparison
Pinpoint abnormal func + host
10K GPUs → 3 minutes

5 Fixed by AI or Human
AI auto-fix suggestion
or manual intervention
Structured output for LLM

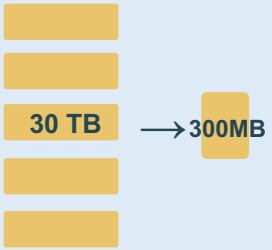
Profile for a short time



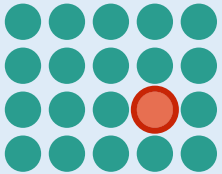
Iteration Time



Using existing tools like Kineto and NSYS

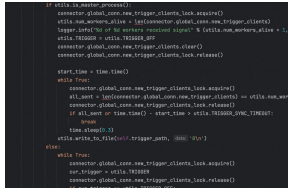


10^5× compression



Pinpoint the outlier function executions with behaviors

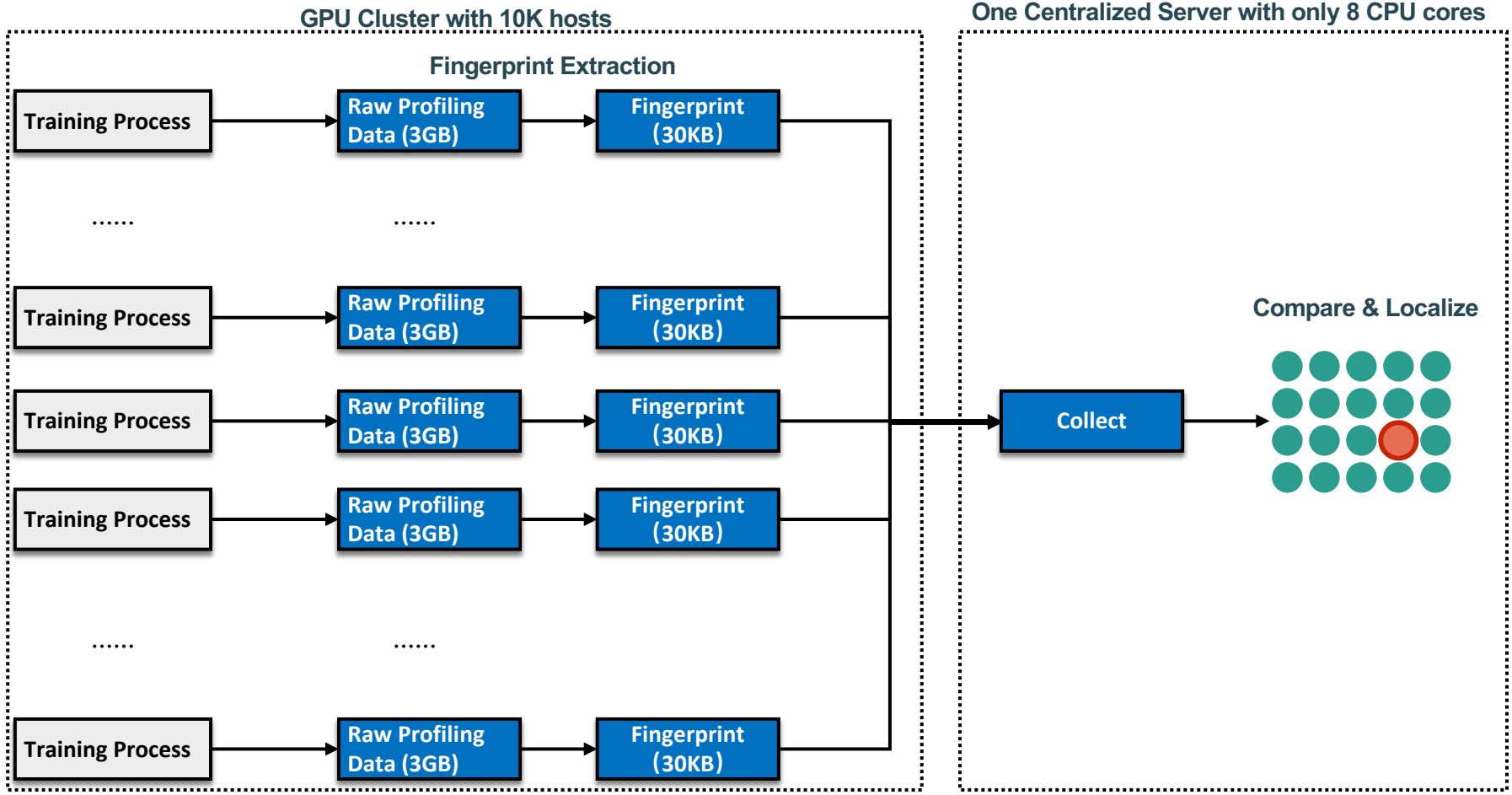
Abnormal function execution	Function runtime behaviors		
	Duration	Avg Resource util.	Resource util. std
dataloader.py: socket recv on all workers	500ms (480ms↑)	98% CPU freq	1%
Ring Allreduce on worker 7	200ms	37% PCIe Tx (38%↓)	5%
CUDA GEMM on workers {0,1,2,3}	300ms (200ms↑)	33% GPU SM (66%↓)	5%



Feed codes of training task with EROICA's output to AI

Key Design

EROICA Overview: Extract Behavior Fingerprints and Compare



Fingerprint Extraction: Which functions need Fingerprinted

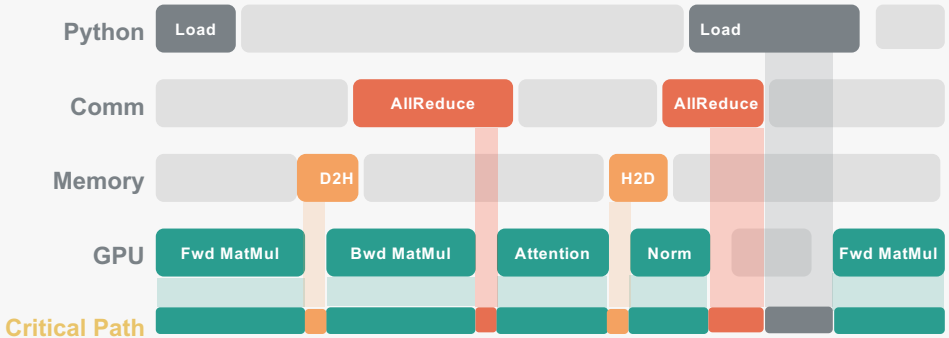
What is the Critical Path?

The function executions which block the training process, determine total iteration time.

Priority (higher = more critical)

1. **GPU Kernels** — MatMul, Attention, Norm
2. **Memory Ops** — D2H, H2D, malloc
3. **Communication** — AllReduce, Send/Recv
4. **Python** — Data loading, scheduling

Example: One Training Iteration Timeline



When GPU is busy, GPU kernels are critical.

When GPU is idle, whatever blocks the GPU becomes critical.

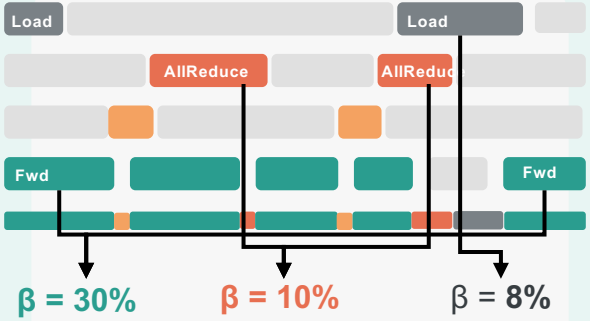
Only functions spend >1% time on the critical path get fingerprinted — typically < 20 functions per worker.

Fingerprint Extraction: How to define function behavior fingerprint

β Duration

Does the function dominate critical path?

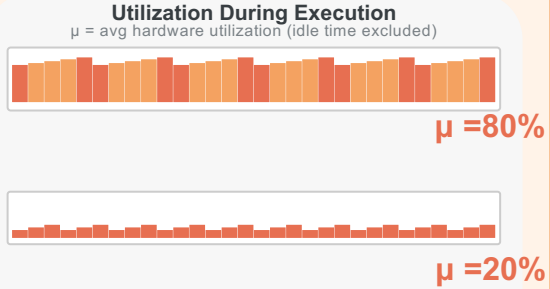
Fraction of the profiling window spent on this function's critical path.



μ Hardware Utilization

Is the hardware underperforming?

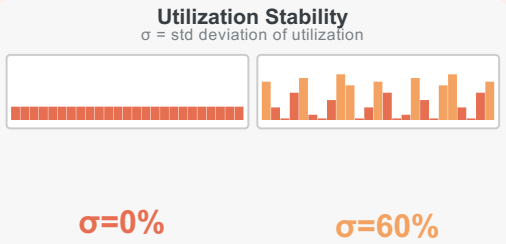
Average hardware utilization during function execution.



σ Stability

Is blocked by something else?

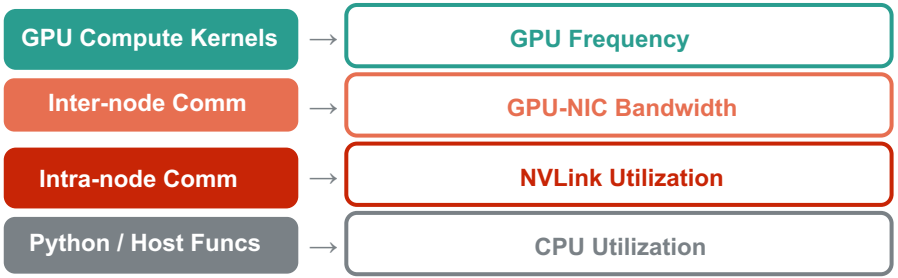
Std deviation of utilization.



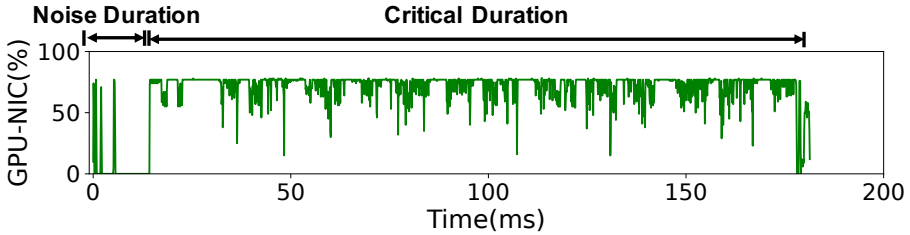
$P_{f,w} = (\beta_{f,w} , \mu_{f,w} , \sigma_{f,w})$ — Behavior fingerprint of function **f** executed on worker **w**

Fingerprint Extraction: Details

μ and σ measures the hardware that determines f's performance



Filtering idle wait time for accurate μ and σ



When a function executes multiple times in the profiling window

μ and σ : Weighted average

Across all executions of f, μ and σ are weighted by each execution's duration.

β : Summation

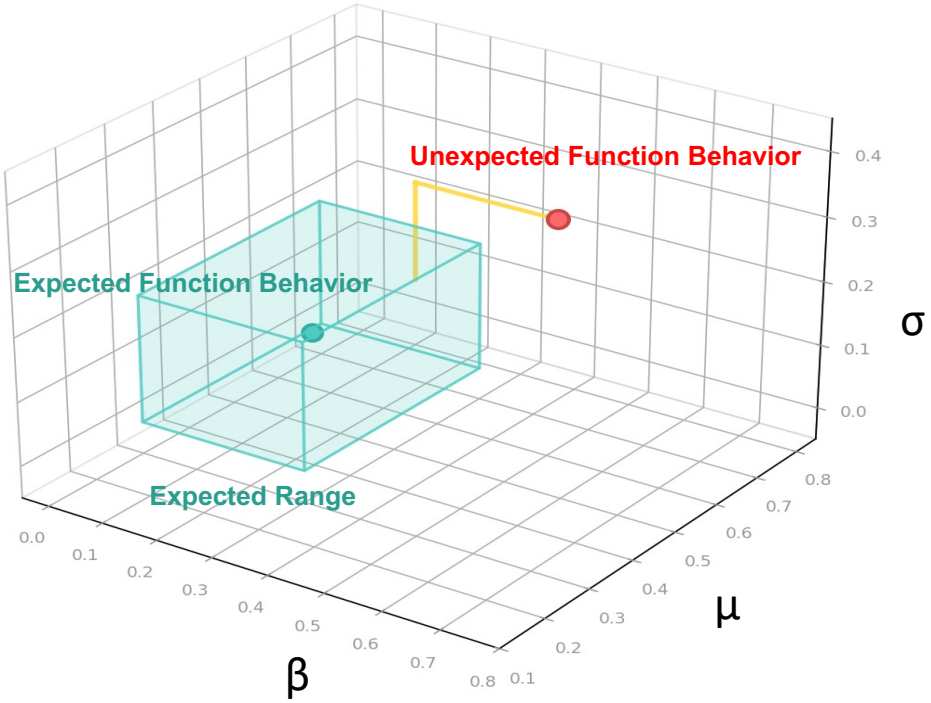
Compare & Localize

Workers ↓	NCCL AllReduce	MatMul Fwd	MatMul Bwd	Attention	…More Functions	Data Loader Recv	Python GC
W0	(0.45,0.85,0.02)	(0.30,0.88,0.03)	(0.28,0.86,0.04)	(0.15,0.72,0.05)		(0.08,0.55,0.10)	
W1		(0.30,0.87,0.02)	(0.28,0.85,0.03)	(0.15,0.73,0.04)		(0.08,0.54,0.09)	
W2	(0.43,0.86,0.02)	(0.30,0.88,0.03)	(0.28,0.86,0.04)	(0.15,0.72,0.05)		(0.08,0.55,0.10)	(0.01,0.45,0.15)
W3	(0.62,0.38,0.06)	(0.30,0.87,0.03)	(0.28,0.86,0.04)	(0.15,0.72,0.05)		(0.08,0.55,0.10)	(0.01,0.44,0.14)
W4	(0.44,0.84,0.03)	(0.30,0.92,0.02)	(0.28,0.86,0.04)	(0.15,0.72,0.05)		(0.25,0.35,0.20)	(0.01,0.45,0.15)
⋮ (9,994 more workers)							
W9999	(0.44,0.84,0.03)	(0.44,0.84,0.03)	(0.44,0.84,0.03)	(0.44,0.84,0.03)		(0.44,0.84,0.03)	(0.44,0.84,0.03)

How to define abnormal behaviors?

Compare & Localize: Behaviors that differ from expectation

- Some function executions have expected behaviors
- Define Expected Range for different types of function
  - Beta < 1% for any python functions
  - Beta < 30% for any communication functions
- Behaviors out of Expected Range are regarded abnormal



Compare & Localize: Behaviors that differ from peers

- The 3D behavior (β, μ, σ) of the same function across workers should cluster or form a regular distribution.
- **Outliers** indicate **abnormal** behavior.
- Based on Manhattan distance, EROICA determines whether each point is an outlier (refer to the paper for details).

To make comparison between workers, first we make a max normalization to $P_{f,w}$:

$$\hat{P_{f,w}} = \left(\frac{\beta_{f,w}}{\max_{w \in W} \beta_{f,w}}, \frac{\mu_{f,w}}{\max_{w \in W} \mu_{f,w}}, \frac{\sigma_{f,w}}{\max_{w \in W} \sigma_{f,w}}\right) \quad (8)$$

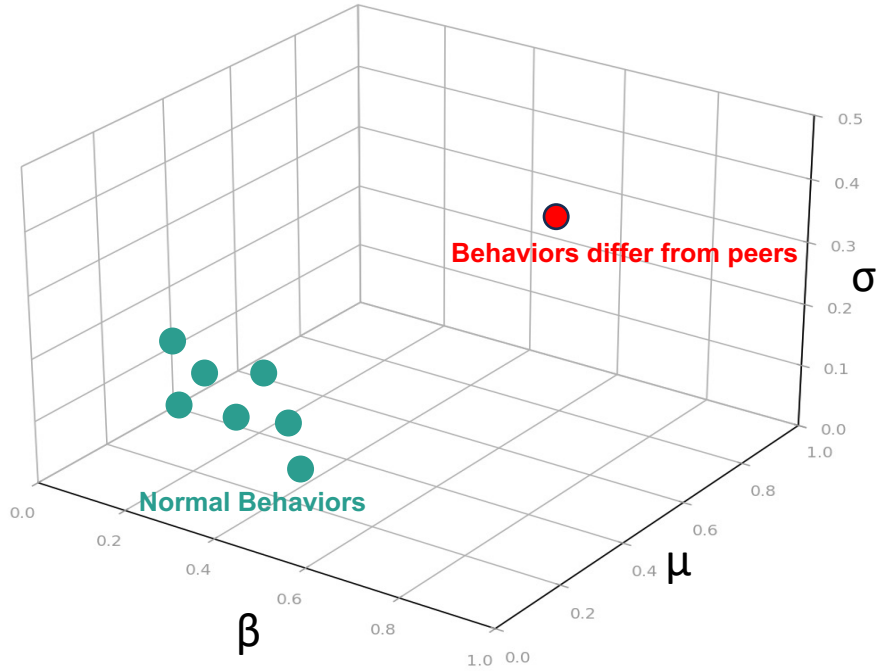
Then we define $\Delta_{f,w}$ to indicate how many workers have different patterns of f's execution from worker w:

$$\Delta_{f,w} = \frac{\sum_{w' \in W_N} I(\hat{P_{f,w}}, \hat{P_{f,w'}})}{N} \quad (9)$$

where W_N is a subset of N workers randomly sampled from the set of all workers W. In practical we set $N = \min(100, |W|)$. I is a function indicating whether the distance between two vectors are below δ:

$$I(x,y) = \begin{cases} 0 & \text{if } d_{\text{Manhattan}}(x,y) < \delta, \\ 1 & \text{otherwise.} \end{cases} \quad (10)$$

In practical, we set δ=0.4 based on our experience.



Behaviors of the same function execution across all workers

Compare & Localize: Structured Output

Abnormal function execution	Function runtime behaviors		
	Duration	Avg Resource util.	Resource util. std
dataloader.py: socket recv on all workers	500ms (480ms↑)	98% CPU freq	1%
Ring Allreduce on worker 7	200ms	37% PCIe Tx (38%↓)	5%
CUDA GEMM on workers {0,1,2,3}	300ms (200ms↑)	33% GPU SM (66%↓)	5%

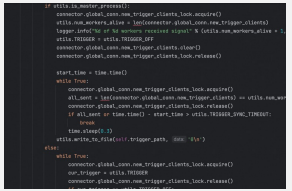
EROICA's output

First try AI auto-fix

The training task has following performance issues:

Abnormal function execution	Function runtime behaviors		
	Duration	Avg Resource util.	Resource util. std
dataloader.py: socket recv on all workers	500ms (480ms↑)	98% CPU freq	1%
Ring Allreduce on worker 7	200ms	37% PCIe Tx (38%↓)	5%
CUDA GEMM on workers {0,1,2,3}	300ms (200ms↑)	33% GPU SM (66%↓)	5%

The relevant code is here:



AI Agent:
ACTION 1
ACTION 2
...

If AI failed to auto-fix

Fixed by human

Evaluation in Production

1.5 Years, ~100K-GPU Clusters

Production GPU cluster deployment

6,144 GPUs

The largest training task EROICA troubleshooted

80+ Cases

Existing tools failed to diagnose

97.5%

Diagnosis success rate

Root Cause Breakdown

Table 2: 80 serious performance issues identified by EROICA (ONLY includes those not identified by our existing systems)

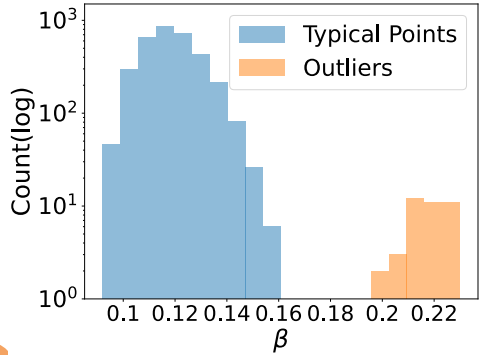
Category	Root cause	Number of LMT cases
Hardware issues	GPU	2
	CPU	2
	Network	6
Misconfigurations	PyTorch	4
	Communication	6
	Dataloader	5
Low-efficiency code of users		45

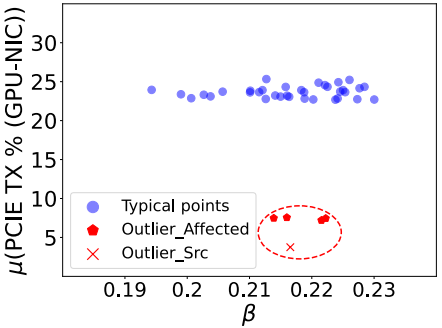
- Diverse GPUs: NVIDIA, AMD
- Diverse Host architecture: x86, arm
- Diverse Training Framework: Torch, Megatron, Verl, Torch-lightning, Slurm, Jax

Product homepage: https://www.alibabacloud.com/help/en/pai/user-guide/perftracker-online-performance-analysis-diagnostic-tool

Case Study: Mixed Hardware-Software Problems

A video generation model training on **3,400** GPUs has unexpected throughput drops.





1 SendRecv latency: 40 workers differ from the rest → 2 One worker has different μ and σ from other 39

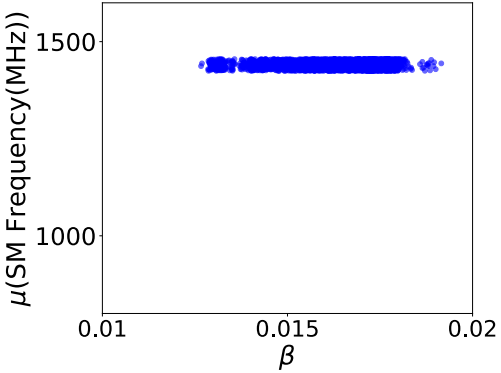
2 → 3 The worker's NIC is downgraded ✓

3 → 4 Network throughput varies across 3,360 workers

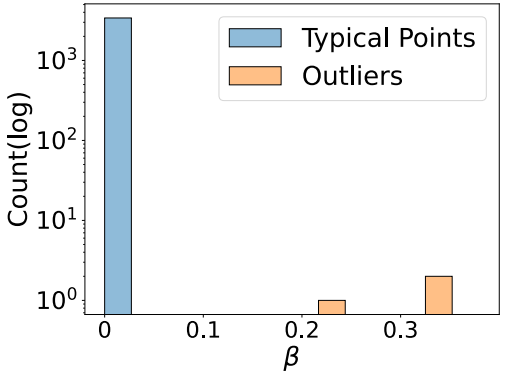
4 → 5 Affinity-based flow scheduling is not deployed on this cluster (manual verification) ✓

Case Study: Mixed Hardware-Software Problems

A video generation model training on **3,400** GPUs has unexpected throughput drops.



Example of a GPU Kernel Function
(chunk_cat_cuda_kernel<float,c10::BFloat16>)



The β value of pin_memory

1 **Similar GPU freq but different execution duration**

↓

2 **Imbalanced input workload across workers** ✓

3 **Three workers (out of 3,400) has long pin_memory**

↓

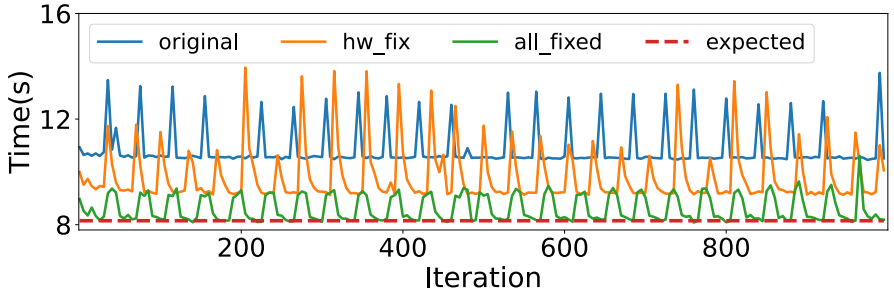
4 **Too many Dataloader processes (manual verification)** ✓

Case Study: Mixed Hardware-Software Problems

A video generation model training on **3,400** GPUs has unexpected throughput drops.

1. **Fix network issues: training throughput +10.3%**
2. **Fix pin_memory issues: training throughput +3.6%**
3. **Load balance for video input: training throughput +20.5%**

34.4% End-to-end performance improved



Conclusion

1. Function Behavior Fingerprint: Bridging the gap between "always watching" and "deeply understanding”, the two classical performance trouble-shooting approaches

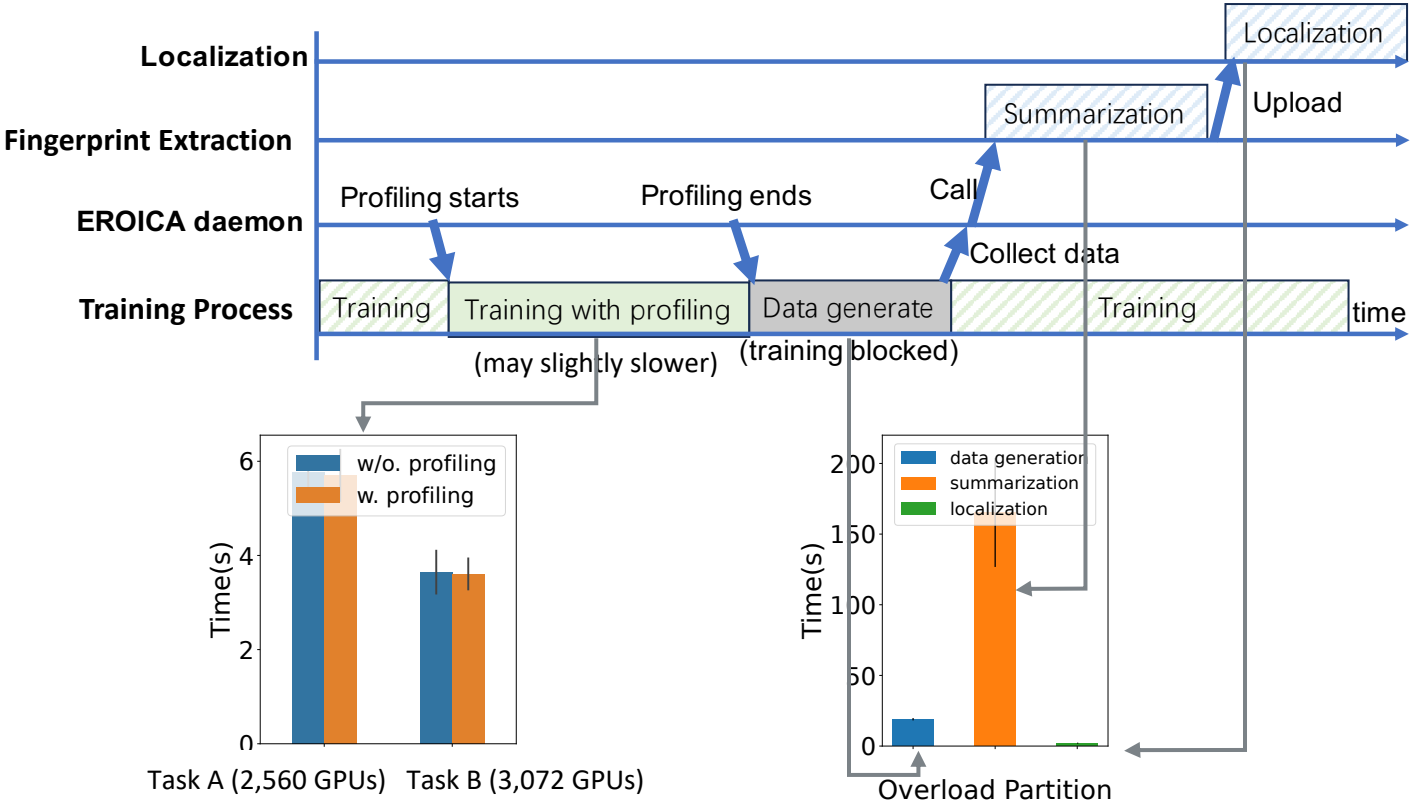
2. The first try of AI-assisted performance trouble-shooting for large model training

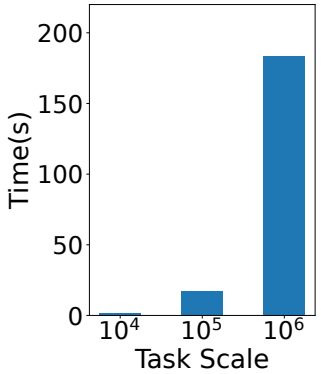
3. Production-proven since 2024, not a story

Thank You!

Questions & Discussion

Evaluation: Overhead





Capability to troubleshoot even larger tasks

An in-depth analysis of profiling overhead with model parameters is presented in the paper

Behind EROICA: Implementation Challenges in Production

- Profiling Synchronization across Workers
  - Challenge: Workers may receive the profiling starting signal at different time.
  - Solution: TCP-based periodic sync; workers wait for a target step to start profiling simultaneously.
- Memory Leakage
  - Challenge: Residual hooks after profiling slow down training.
  - Solution: Modified profiling logic to clean up hooks post-collection.
- Hardware Metrics Access
  - Challenge: Customer containers lack hardware read permissions.
  - Solution: Privileged Sidecar collects metrics → Shared Dir → Customer containers .

Behind EROICA: Implementation Challenges in Production

- Conflict with other hardware monitors
  - Challenge: Some hardware metrics can be only subscribed by one process.
  - Solution: Co-existence mechanism, other monitors pauses during EROICA collection, then resumes.
- Fault Tolerance
  - Challenge: Missing data from individual workers skews analysis.
  - Solution: Supports root cause localization with fingerprint of partial worker.
- Profiling Cost Optimization
  - Optimized underlying Torch Profiler; reduced overhead by 33%.