LLNL-CFPRES-2021076

Lawrence Livermore National Laboratory

CoDL: Studying Cross-Component Interference

D Dhakshinamoorthy, R Sinurat, H Hariharan Devarajan, N Dryden, A Paul

Jul 2026

6th Workshop on Re-envisioning Extreme-Scale IO for Emerging Hybrid HPC Workloads

Cleveland, OH, United States of America

2026-07-13 through 2026-07-13

Disclaimer

This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

CoDL: Studying Cross-Component Interference

Interference in Deep Learning Training Pipelines: Data Loading, Compute, and Communication

Druva Dhakshinamoorthy, Ray A. O. Sinurat, Nikoli Dryden, Arnab K. Paul, Hariharan Devarajan

BITS Pilani | University of Chicago | Lawrence Livermore National Laboratory

REX-IO '26 Workshop

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-CFPRES-2021076

Unpredictable Step Durations

- **The variance gap:** in ResNet-50, compute duration spans 187 to 1,169 ms across steps.
- **Variability is systemic:** preprocessing pipelines alone can cause 3 to 13x slowdowns.
- **It matters:** you can't separate a real speedup from noise, so tuning is misled and throughput-based models break.

Where does this variability come from?

Shared Pipeline Components

Data Loading

Disk I/O and CPU-based preprocessing workers running in the background.

Compute

GPU forward and backward passes, launched by host CPU threads.

Communication

Network-bound synchronization (NCCL AllReduce) for distributed training.

Contention for shared CPU and memory -> variability in performance

The Blind Spots of Existing Tools

- **I/O benchmarks (DLIO):** replace compute with sleep(), releasing the CPU and erasing real contention.
- **Compute profilers:** generally blind to asynchronous I/O background-worker activity.
- **GPU tracers:** live on separate clocks from host events, making alignment impossible.

Existing tools cannot be used to study contention.

What Sleep-Based Benchmarks Hide

ResNet-50 *many small reads*

Per-step compute barely changes: 22.9 vs 23.2 ms. But the full pipeline runs far longer under real contention.

3.8x

total step-time gap DLIO's sleep hides

UNet-3D *few large reads*

Even per-step compute diverges (110.7 vs 147.2 ms), from memory-bus pressure a sleep never creates.

+33%

compute gap sleep never sees

Sleep releases the CPU, so DLIO reports I/O under artificially clean conditions.

CoDL

Benchmark Framework

keep the contention real

- Real compute and communication, not sleep(), so CPU contention survives.
- Toggle each component on and off to isolate its effect.

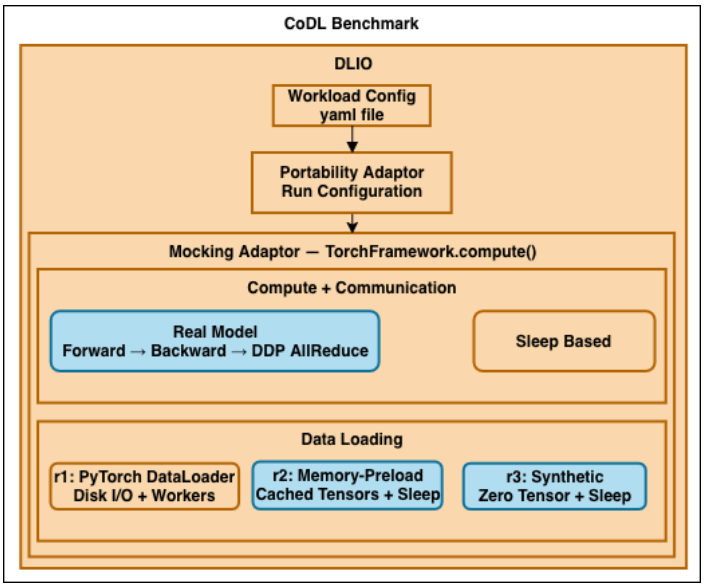
Tracing Framework

make it observable

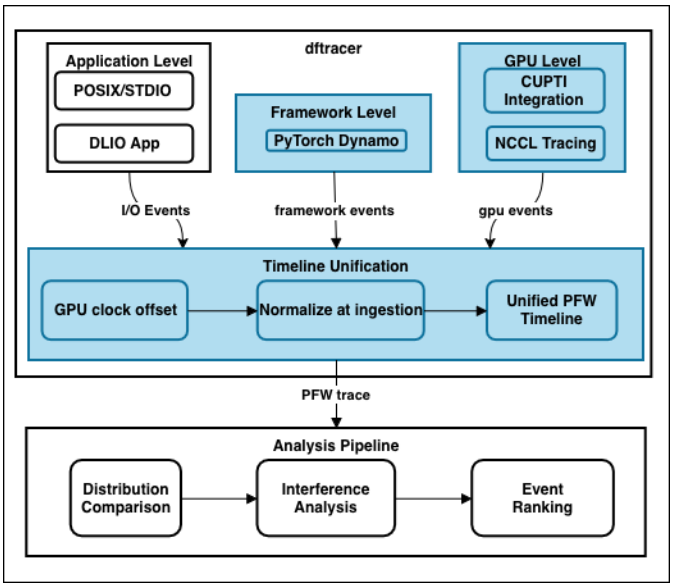
- One trace across I/O workers, framework, and GPU.
- Host and GPU events normalized onto a single microsecond clock.

Built as extensions to DLIO (benchmark) and DFTracer (tracer).

CoDL Architecture - Benchmark



CoDL Architecture - Tracing



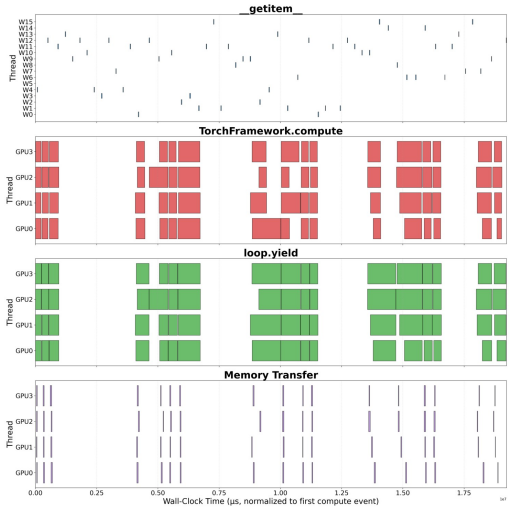
Is the Benchmark Faithful?

Workload	Event	Native	CoDL	Diff
ResNet-50	Forward pass	7.7 ms	7.8 ms	1.3%
	Backward pass	15.2 ms	15.2 ms	0.0%
	Compute	23.1 ms	25.4 ms	10.3%
UNet-3D	Forward pass	28.0 ms	28.1 ms	0.4%
	Backward pass	82.3 ms	82.1 ms	0.2%
	Compute	110.3 ms	115.7 ms	4.8%

Diff = |CoDL - Native| / Native, on per-step median durations.

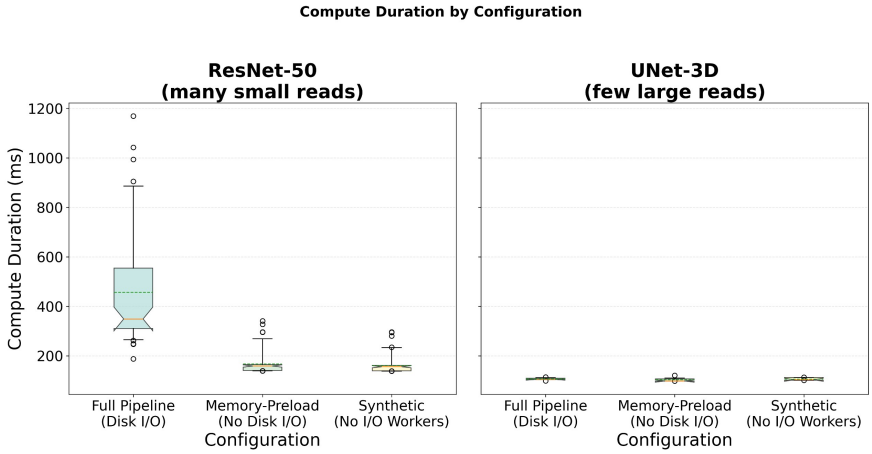
Forward and backward match within 1%. The compute gap (up to 10%) is constant overhead, present equally in all three runs, so it doesn't distort the comparisons.

Results on a Unified Timeline



ResNet-50: every compute block overlaps active data loading. 68 of 68 events, mean 13.4 concurrent loads.

Impact on Compute



ResNet-50 compute more than doubles under full data loading (median 349 ms), and collapses to a 130 to 160 ms band without loader workers. UNet-3D barely moves.

But the boxplot says that it happens, not why.

It's the Launch Path, Not the GPU

- **The contention is real:** every compute window overlaps active data loading (the timeline we just saw).
- **But the delay is on the launch path:** CUDA runtime-API latency rises from 2.9 to 9.6 microseconds, a factor of 3.3x, not in kernel execution.
- **So the loaders never touch the GPU:** they preempt the host threads that launch its work.

An apparent I/O bottleneck is really CPU scheduling delay, invisible to any single-layer tool.

What This Means

- **The bottleneck isn't I/O bandwidth:** it is the host threads that launch GPU work, so they must keep getting CPU.
- **Likely fixes:** pin or prioritize the launch threads, and throttle or isolate the loader workers.
- **Measure first:** interference is workload-specific, and CoDL shows which knob actually matters.

It turns "my training is slow" into "here is exactly where, and why."

Takeaways

- **Interference is a dominant source** of DL training step-time variability.
- **The data access pattern shapes severity:** many-small reads hurt, few-large reads don't (though workload-dependent).
- **The mechanism is host-side launch contention,** not GPU contention, only visible on a unified timeline.

Takeaway: optimizing one component in isolation can mislead performance engineering.

Questions?

Thank you!