LLNL-CONF-2018702

Lawrence Livermore National Laboratory

CoDL: A Framework for Studying Cross-Component Interference in Deep Learning Training Pipelines

H Hariharan Devarajan

Sep 2026

6th Workshop on Re-envisioning Extreme-Scale I/O 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: A Framework for Studying Cross-Component Interference in Deep Learning Training Pipelines

Druva Dhakshinamoorthy
BITS Pilani, KK Birla Goa Campus
India

Ray A. O. Sinurat
University of Chicago
USA

Nikoli Dryden
Lawrence Livermore National Laboratory
USA

Arnab K. Paul
DaSHLab, BITS Pilani, KK Birla Goa Campus
India

Hariharan Devarajan
Lawrence Livermore National Laboratory
USA

Abstract

Deep learning training on HPC systems exhibits significant step-time variability—data stalls can dominate 50–85% of training time [12, 31]—that limits efficiency. Existing benchmarks and profilers isolate individual components: I/O benchmarks replace compute with sleep, compute profilers ignore I/O, and GPU tracers operate on separate timescales. This isolation prevents understanding real variability, which stems from interactions among data loading, compute, and communication competing for shared CPU resources. We present CoDL, a framework that combines component-isolation benchmarking with multi-level tracing on a unified microsecond timeline, enabling diagnosis that reframes optimization from isolated component throughput to CPU–GPU scheduling coordination. On a single 4-GPU node, evaluation of ResNet-50 shows compute duration more than doubles (+127.4%) under active data loading and NCCL AllReduce degrades 2.8×, despite zero direct GPU-kernel overlap with data loading; the dominant mechanism is data-loading workers preempting GPU-launch threads on the CPU, inflating CUDA runtime API latency by 3.3×.

CCS Concepts

• **Computing methodologies** → **Parallel computing methodologies**; • **Computer systems organization** → *Heterogeneous (hybrid) systems.*

Keywords

Deep Learning, I/O Benchmarking, Performance Interference, GPU tracing, NCCL, High Performance Computing

ACM Reference Format:
Druva Dhakshinamoorthy, Ray A. O. Sinurat, Nikoli Dryden, Arnab K. Paul, and Hariharan Devarajan. 2026. CoDL: A Framework for Studying Cross-Component Interference in Deep Learning Training Pipelines. In *The 35th International Symposium on High-Performance Parallel and Distributed Computing (HPDC '26), July 13–16, 2026, Cleveland, OH, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3806645.3816128

ACM acknowledges that this contribution was authored or co-authored by an employee, contractor, or affiliate of the United States government. As such, the United States government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for government purposes only. Request permissions from owner/author(s).
HPDC '26, Cleveland, OH, USA
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2640-8/2026/07
https://doi.org/10.1145/3806645.3816128

1 Introduction

Modern large-scale scientific workloads rely on deep learning (DL) for discoveries in domains such as bioengineering, climate modeling, and high-energy physics [5, 24]. These workloads are commonly trained on high-performance computing (HPC) systems with thousands of GPUs, petabyte-scale datasets, and frequent inter-GPU communication [13, 16]. Efficient training therefore depends on coordinated optimization across the full software and hardware stack, not just isolated kernels. Training is a pipelined system in which data loading, compute, and communication directly affect one another, so each stage must be tuned with the others in mind (e.g., input staging rate, kernel launch cadence, and collective overlap) [33].

To support tuning, the HPC and ML communities have built a rich ecosystem of benchmarks and profilers [11, 28, 37, 40, 46]. Most benchmarks fall into two groups: component-level tests that isolate one stage for targeted optimization [46], and end-to-end tests that treat training as a black box and report aggregate throughput [9, 28]. Most tools are also single-layer by design: compute profilers see framework operations and GPU kernels but miss I/O behavior; I/O tools see file-system activity but miss compute context; and GPU telemetry captures hardware events on a timeline that is hard to align with application events [26]. Such methods improve individual components but still emphasize isolated behavior over pipeline interactions (e.g., faster decode, GEMM, or collectives measured independently) [9, 31].

However, real DL variability is driven by interactions among components [17, 31], and current practice does not capture those interactions well. First, many benchmarks isolate a target stage by replacing other stages with sleep calls, which removes the shared-resource contention that exists in real training [8]. For example, data-loading workers, host-to-device transfers, and communication setup can compete for CPU time and memory paths [20, 32]; sleep-based substitution hides this interference. Second, monitoring tools report events at separate levels (application, system, or hardware) without a single correlated timeline, so causal links across layers remain unclear. To optimize real pipelines, we need methods that preserve component interactions and trace them in one unified view (e.g., aligning data-loader activity, kernel launch latency, and GPU execution in the same time window).

To address these limitations, we present CoDL, a framework for systematically studying cross-component interactions and quantifying their contribution to performance variability in large-scale

DL workloads. CoDL combines two parts: a benchmark framework with switchable data loaders and configurable compute/communication toggles, and a tracing framework that records application, library, and hardware events on a unified timeline. Together, these capabilities enable controlled interference experiments and cross-layer diagnosis in realistic training settings. Using ResNet-50 and UNet-3D, we show that cross-component interference is a dominant source of step-time variability, and that an apparent I/O bottleneck can actually be CPU scheduling contention that delays GPU kernel launches by 3.3× while kernel execution remains near full speed.

The core insight is that cross-component interference in DL pipelines manifests primarily as CPU scheduling delay on kernel launch paths rather than as direct GPU contention; the data loaders never touch the GPU but they preempt the host threads that do. Surfacing this requires two capabilities together, neither sufficient alone: (i) controllably toggling each component while preserving real-workload semantics—sleep-based emulation removes the contention being studied—and (ii) a single microsecond-resolution timeline that aligns POSIX, framework, CUPTI, and NCCL events captured from heterogeneous clock domains. The technical challenges are therefore semantics-preserving mocking at component boundaries (Sec. 3), GPU–host clock unification at ingestion (Sec. 3), and a framework-agnostic experiment schema that keeps configurations directly comparable across runs. The contributions of this paper are as follows:

(1) A **benchmark framework** with switchable data loaders and configurable compute/communication toggles that enable controlled interference experiments.
(2) A **tracing framework** that captures application, library, and hardware events on a unified timeline, enabling cross-layer interference analysis.
(3) **Empirical evidence** from ResNet-50 and UNet-3D workloads demonstrating that cross-component interference is the dominant source of training step variability.

2 Background and Motivation

2.1 DL Training Pipeline

A distributed DL training step interleaves three components on shared hardware. **Data loading** workers read samples from storage, apply data transformations, and place batches into shared memory via PyTorch's `DataLoader`. **Compute** dequeues a batch, transfers it to the GPU, and runs forward and backward passes. **Communication** synchronizes gradients across GPUs via NCCL (NVIDIA Collective Communication Library) AllReduce, overlapped with backward computation in DDP [27]. These components contend for CPU cores (data loading workers vs. CUDA runtime threads vs. NCCL proxy threads), memory bus (shared-memory writes, host-to-device staging, gradient accumulation), and PCIe bandwidth.

The severity of this contention depends on the data access pattern. Workloads that issue many small reads (e.g., ResNet-50: 16,000 JPEG images per run) tend to keep data loading threads active across most of each step when the loader's worker count is sized to match prefetch demand, creating *persistent* CPU contention; the actual saturation depends on worker count, prefetch depth, and the share of per-sample work spent in CPU transforms versus device parallelism. Workloads that issue few large reads (e.g., UNet-3D: 168 NPZ volumes per run) leave data loading threads idle during much of compute under light preprocessing, creating only *intermittent* contention; when per-sample preprocessing is heavy, even a small number of reads can sustain CPU pressure [35]. This distinction predicts that many-small-reads workloads should exhibit high step-time variance while few-large-reads workloads should remain stable—a prediction confirmed in Section 4.

2.2 Motivation: Why Existing Tools Miss Interference

Training step times vary significantly. Prior work has shown that data stall time can dominate 50–85% of training time [12, 31], with preprocessing pipeline configuration alone causing 3–13× throughput differences [17]. This variability is not random noise: it stems from transient resource availability (cache state, CPU allocation, storage contention) that changes between steps.

Data loading competes with compute for shared resources. Across Google's ML fleet, 20% of jobs spend more than a third of their compute time in the input pipeline [32], and assigning too many data loading workers itself causes CPU contention that degrades rather than improves throughput [20]. These findings, echoed by studies of asynchronous I/O interference in HPC applications [43], establish that the interaction is workload-dependent and difficult to predict, motivating CoDL's methodology.

Existing benchmarks and profilers miss this interaction. DLIO [11]—the state-of-the-art I/O benchmark for DL workloads—replaces compute with a calibrated `sleep()` call, which releases the CPU during the compute window and removes the contention that drives variability in real training. Profiling tools operate at single levels of the stack: I/O tracers miss compute context, GPU profilers miss I/O activity, and merging their outputs is unreliable because different timestamp sources prevent causal alignment [26]. No existing tool captures the DL-pipeline event union—POSIX I/O, PyTorch framework operations, CUDA kernels and runtime API, and NCCL collectives—on a unified microsecond timeline while also providing controllable per-component isolation. General-purpose HPC profilers (e.g., Nsight Systems [39], HPCToolkit [1], TAU [41], Score-P [22], THAPI [4]) each cover only part of this event union and treat data loading as opaque user code, so the cross-component contention this paper measures remains attributable to either "I/O variance" or "compute jitter" depending on which tool is used.

These gaps define two requirements: (1) controlled experiments where components can be selectively enabled or disabled without removing realistic contention, and (2) a unified timeline across all stack levels so interference can be diagnosed causally. Section 3 describes how CoDL provides both, and Section 4 quantifies the interference directly: removing data loading workers reduces ResNet-50 compute time by more than half, confirming that sleep-based benchmarking misses a dominant performance effect.

3 Design

The overall goal of CoDL is to study cross-component interference and to understand and quantify its impact on DL training variability. To achieve this goal, our design has two main parts: a benchmark framework and a tracing framework. We build these by extending two state-of-the-art systems that are already tailored for DL

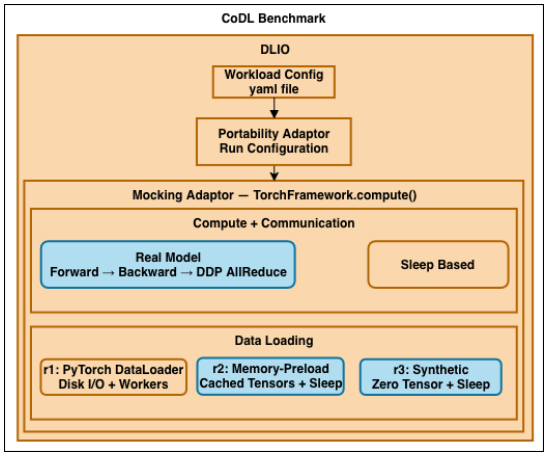


Figure 1: CoDL Benchmark Framework. Orange boxes are existing DLIO components; blue boxes are CoDL extensions.

workloads but are primarily I/O-centric: DLIO as the benchmark base [11] and DFTracer as the tracing base [10]. The extensions are designed to preserve workload semantics while enabling controlled interference experiments and unified cross-layer tracing.

3.1 Architecture Overview

The benchmark framework (Figure 1) provides switchable modes with a three-run protocol; the tracing framework (Figure 2) captures events on a unified microsecond timeline. Together they isolate each component's contribution to variability.

3.2 CoDL Benchmark Framework

The benchmark framework extends DLIO from I/O-focused benchmarking to cross-component interference benchmarking. It provides controlled toggles over data loading, compute, and communication while preserving per-step workload meaning (same model class, batch shape, and optimization path where applicable). This enables causal comparisons across runs rather than unrelated executions.

3.2.1 Mocking Adaptor. The goal of the Mocking Adaptor is to selectively mock one pipeline component while keeping the semantics of the remaining components unchanged. The main challenge is that naive mocking (for example, replacing compute with unconstrained sleep) changes resource contention and invalidates interference conclusions. We solve this by using semantics-aware mock modes: memory-preloaded real tensors for reduced-I/O runs, synthetic pinned tensors with calibrated timing for no-I/O-worker runs, and configurable real/sleep compute modes that preserve step structure. A key design consideration is where to insert mocking hooks. We choose component-boundary hooks (data ingress and compute launch boundaries) rather than operator-level hooks because boundary hooks are less brittle across framework versions and preserve portability.

Concretely, the mocking hook is inserted at the `TorchFramework.compute()` boundary. When a real model is configured, this method dispatches to `PyTorchLayers.compute()`, which executes the actual forward pass, loss computation, backward pass, and gradient synchronization. When no model is loaded, the same method falls back to `sleep(computation_time)`, preserving step structure and timing. Semantic equivalence is enforced by asserting model initialization before optimizer construction, verifying GPU device consistency before DDP wrapping, and emitting identical profiling events regardless of mode.

3.2.2 Portability Adaptor. The goal of the Portability Adaptor is to maintain a standardized interface across data loader behaviors and communication primitives. The challenge is heterogeneity: different loader modes expose different APIs, runtime assumptions, and distributed orchestration paths. We solve this with an intermediate experiment schema that maps abstract intents (loader mode, compute mode, communication mode, run protocol) to backend-specific configurations via declarative mappings, improving reproducibility and reducing duplication. The current implementation targets PyTorch; the declarative schema is designed to be extensible to other frameworks (e.g., JAX, TensorFlow) by adding new mapping backends without modifying the experiment protocol.

3.2.3 Benchmark Execution Modes. The benchmark framework currently supports three progressive I/O-isolation modes and configurable compute/communication toggles. The default mode uses PyTorch DataLoader with configurable worker count, prefetch depth, and multiprocessing context. Memory-Preload performs one-time bulk loading and serves cached pinned tensors with calibrated sleep to remove disk variance while keeping realistic tensor values. Synthetic mode repeatedly yields a pre-allocated pinned tensor and removes I/O-worker contention. Compute can run as real model execution or sleep-based emulation, and distributed communication can be enabled via DDP with NCCL.

3.2.4 Three-Run Protocol and Calibration. A suite runner orchestrates the three-run protocol: (1) full pipeline with disk I/O, (2) memory-preloaded data without disk I/O, and (3) synthetic data without I/O workers. Between runs, the runner updates configuration files and isolates outputs for direct comparison. Sleep calibration is derived from full-pipeline logs by extracting per-batch load and compute durations, filtering warmup, and applying a stable central estimate before replay.

The calibration procedure extracts per-batch load times and per-step compute durations from a full-pipeline run, discards warmup (3 steps) and the final step, removes outliers exceeding 3× median, and injects the means into mock runs as configuration fields.

3.3 CoDL Tracing Framework

The tracing framework extends DFTracer from I/O-centric tracing to unified multi-level tracing for causal cross-component analysis. It captures events from application I/O, framework, GPU runtime, and communication on a normalized timeline.

3.3.1 Multi-Level Event Collection. At the application level, POSIX and STDIO calls are intercepted via `LD_PRELOAD` and GOTCHA and recorded with timestamps, durations, byte counts, and paths. At the

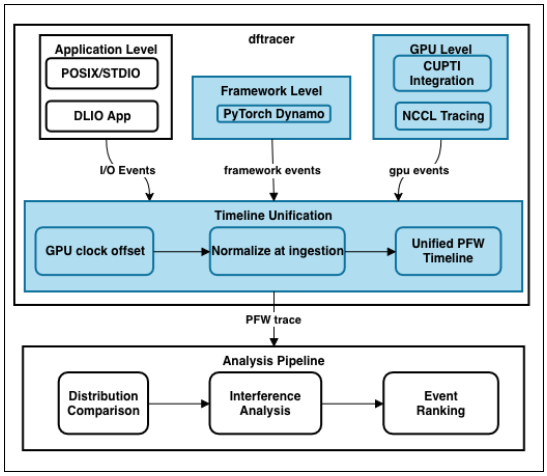


Figure 2: CoDL Tracing Framework. Blue boxes are CoDL extensions to DFTracer; white boxes are existing components. Arrows show event flow from collection through unification to analysis.

framework level, PyTorch Dynamo integration records `forward_pass` and `backward_pass` events, and explicit markers capture host-to-device transfer windows. At the GPU level, CUPTI integration captures CUDA kernels, memory-copy events, and CUDA runtime API calls. Communication events are separated from general compute by identifying NCCL kernel patterns in the trace stream.

3.3.2 Timeline Unification. The goal of timeline unification is to enable causal reasoning across asynchronous events (for example, overlapped data-loader operations, kernel launches, and NCCL collectives). The challenge is clock heterogeneity: application events use host clocks, while CUPTI events use GPU-side nanosecond clocks. We solve this by normalizing all timestamps to a common microsecond-resolution reference timeline at ingestion time, using calibrated offsets for GPU-origin timestamps and consistent per-process event ordering in PFW outputs. A key design consideration is when to normalize. We choose ingestion-time normalization instead of post-hoc alignment because it reduces merge ambiguity and limits accumulated drift during long traces.

At initialization, DFTracer records a GPU-side reference timestamp t_0 via `cuptiGetTimestamp()`. All subsequent CUPTI events (kernels, memory copies, runtime API calls) are normalized as $t_{\text{event}} = (t_{\text{gpu}} - t_0)/10^3$, converting from GPU nanoseconds to microseconds relative to tracer start. Application-level events (POSIX/STDIO interceptions and PyTorch markers) use the host clock directly. Because t_0 is captured at process initialization before any GPU work begins, all event timestamps share a consistent per-process reference point, enabling direct temporal overlap analysis across stack layers. For multi-node runs, each process maintains an independent t_0; cross-node alignment relies on synchronized job launch rather than explicit drift correction, which is sufficient for the single-node experiments in this work but represents a known limitation for multi-node deployments.

3.3.3 Analysis Pipeline. The analysis pipeline quantifies interference in three stages. Distribution comparison evaluates duration shifts across run modes after steady-state filtering and outlier removal. Interference analysis detects temporal overlap between data-loading events and target compute/communication events using a sweep-line method, then groups events by interference severity. Event ranking prioritizes operations by slowdown, wall-clock impact, and variability change (CV delta), identifying where cross-component interference has the largest practical effect.

4 Evaluation

We evaluate CoDL along four questions: whether the methodology is representative, whether the benchmark and tracing frameworks are faithful, whether interference can be measured on real workloads (UNet-3D and ResNet-50), and whether the mechanism of interference can be explained.

4.1 Methodology

All experiments run on the Matrix cluster at Lawrence Livermore National Laboratory (LLNL). Each node contains two Intel Xeon Platinum 8480+ processors (112 cores total, 3.7 GHz), 512 GB DDR5 memory, and 4 NVIDIA H100 GPUs (80 GB HBM3 each), connected via InfiniBand, with Lustre as the parallel file system. We use two representative scientific DL workloads: ResNet-50 image classification on 387×387 JPEG images and UNet-3D segmentation on 128^3 NPZ volumes. Both workloads run on 1 node with 4 GPUs under DDP with NCCL, with batch sizes of 200 (ResNet-50) and 7 (UNet-3D). ResNet-50 runs for 125 training steps (5 epochs); UNet-3D runs for 30 training steps (5 epochs). For each workload, we execute the three-run protocol from Section 3: r1 (full pipeline with disk I/O, real compute, and communication), r2 (memory-preloaded data with real compute and communication), and r3 (synthetic tensors, no I/O workers, real compute and communication). We report per-event duration distributions, slowdown across run modes, overlap/concurrency statistics, and event-level attribution indicators. Trace data is collected on a unified microsecond timeline; for ResNet-50 r1 this includes ~100,000 data-loading events, 488 compute events (across all 4 GPUs; 122 per GPU, of which 68 are steady-state after warmup and outlier filtering), 508K CUDA kernels, 6.6M CUDA runtime API calls, and 295K memory-copy operations.

Software stack. TOSS 4 (RHEL-based), CUDA 12.9 (driver 13020), NCCL 2.27.5, PyTorch ≥2.2.0, DFTracer 4.0.0 (pydftracer 1.0.11, commit `a2998d6`), CoDL commit `057cead`, DLIO commit `f145958`.

For UNet-3D r1, the trace comprises 168 data-loading events, 12 compute events (3 per GPU after filtering), 14K CUDA kernels, 186K CUDA runtime API calls, and 5.3K memory-copy operations.

4.2 Validating CoDL Framework: Benchmark and Tracing

To validate the benchmark framework, we compare CoDL with real compute enabled against a standalone PyTorch training script under matched model, batch size, optimizer settings, and device configuration. For each workload, we run the same number of steps,

Table 1: Fidelity validation: Native PyTorch vs. CoDL compute. Compute includes forward + backward + optimizer step + DDP synchronization, so it exceeds the sum of forward and backward medians.

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%

Table 2: Cross-workload interference attribution.

Metric	ResNet-50	UNet-3D
Data loading events (r1)	16,000	168
Compute overlap with loading	100%	67%
Mean concurrent loads	13.4	2.0
Compute slowdown (r1 vs r2)	+127.4%	+6.4%
CUDA runtime API inflation	3.3×	1.5×
NCCL AllReduce slowdown	+184.1%	<5%

discard warmup, and compute median per-step durations for forward, backward, and end-to-end compute windows. Table 1 reports phase-level medians. Forward and backward medians match within 1% for both workloads, while overall compute medians differ by 10.3% for ResNet-50 and 4.8% for UNet-3D. This pattern indicates that CoDL preserves compute-phase semantics and that residual differences are dominated by constant wrapper overhead rather than altered model behavior. Crucially, this overhead is present in CoDL configurations (r1, r2, r3) equally, so interference measurements—always ratios between configurations—are unaffected.

To validate the tracing framework, data-loading and compute medians differ by less than 1% between near-equivalent configurations (DataLoader vs. Memory-Preload: 0.1%; Memory-Preload vs. Synthetic: <1%), so instrumentation perturbation is negligible relative to the 2×–3× interference effects measured later.

4.3 Measuring Interference for Real Workloads: UNet-3D and ResNet-50

We quantify compute-path degradation under loader contention by comparing compute duration distributions across r1/r2/r3 (Figure 3), then explain the mechanism using unified event timelines (Figure 4). This design separates full contention (r1), reduced I/O contention with realistic tensors (r2), and no loader-worker contention (r3). Figure 3 shows the results for both workloads.

ResNet-50 (Figure 3, left). Under full-pipeline data loading (r1), compute median is 349 ms with an IQR spanning 300–550 ms. Removing data loading workers (r2, r3) collapses the distribution to a 130–160 ms band, with the r1-vs-r3 median ratio exceeding 2× (+127.4%). The forward pass degrades by +103.4% (55.7 ms vs. 27.4 ms) and the backward pass by +140.8% (52.8 ms vs. 21.9 ms), consistent with memory-intensive gradient computation under host contention. Memory transfers slow by +87.8% (17.3 ms vs. 9.2 ms). NCCL AllReduce degrades 2.8× (2.5 ms vs. 875 μs) because proxy threads compete with data loading workers for CPU time. Binning compute events by concurrent data-loading calls shows monotonic scaling: 146 ms at 1–19 concurrent loads, 299 ms at 20–40, and 462 ms at 41+ (3.2× spread), ruling out random variance.

UNet-3D (Figure 3, right). Compute degrades by only 6.4% (median 104.5 ms vs. 98.2 ms in r2), with all phases showing similarly modest slowdowns and NCCL AllReduce unaffected. Only 8/12 compute events (67%) overlap with data loading, with concurrency of 2.0 (vs. 13.4 for ResNet-50) and peak 10 (vs. 16). UNet-3D's heavier compute (~100 ms per step) outruns the data loading cadence, leaving gaps where compute runs without contention.

4.4 Analyzing and Explaining Interference

The previous subsection establishes that interference exists; this subsection explains where and how it occurs. We first characterize overlap structure by aligning compute-event windows with loader-event windows on the unified timeline and then aggregating concurrency statistics over steady-state steps. In ResNet-50, all steady-state compute events overlap with active data loading (68/68), with a mean of 241 overlapping getitem events per compute step summed across the compute window (instantaneous mean 13.4, peak 16, i.e., all workers), and 3.99 of 4 GPUs have active data loading. This indicates sustained prefetch-driven pressure rather than occasional bursts.

CUPTI-level isolation clarifies the mechanism. In ResNet-50, traces include 508K CUDA kernels, 294K device-to-device memory copies, and 6.6M CUDA runtime API calls. We do not observe direct kernel overlap with data-loading events, but CUDA runtime API latency increases from 2.93 μs (r3) to 9.55 μs (r1), a 3.3× increase. This points to CPU-side scheduling delay on the launch path as the dominant mechanism, which is difficult to detect with single-level tools because device execution can still appear healthy.

Figure 4 visualizes this interference: data loading workers (W0–W15) run continuously, every compute block temporally overlaps with active loading, and batch delivery never has exclusive CPU access. Memory transfers and forward/backward passes execute only after data arrives, by which point the CPU is already saturated.

For UNet-3D, GPU-level traces confirm the same isolation pattern: CUDA kernels (14,108), device-to-device copies (5,232), and runtime API calls (185,628) all show zero direct overlap with data loading—matching ResNet-50. However, CUDA runtime API call duration increases from 50.3 μs (r3) to 74.1 μs (r1), a 1.5× slowdown—less severe than ResNet-50's 3.3×, consistent with the lower data loading concurrency.

Table 2 distills the cross-workload comparison. The same mechanism—CPU scheduling contention on GPU launch paths—produces an order-of-magnitude difference in impact depending on data-loading-to-compute ratio: persistent saturation (13.4 concurrent loads, 100% overlap) versus intermittent interference (2.0 concurrent loads, 67% overlap).

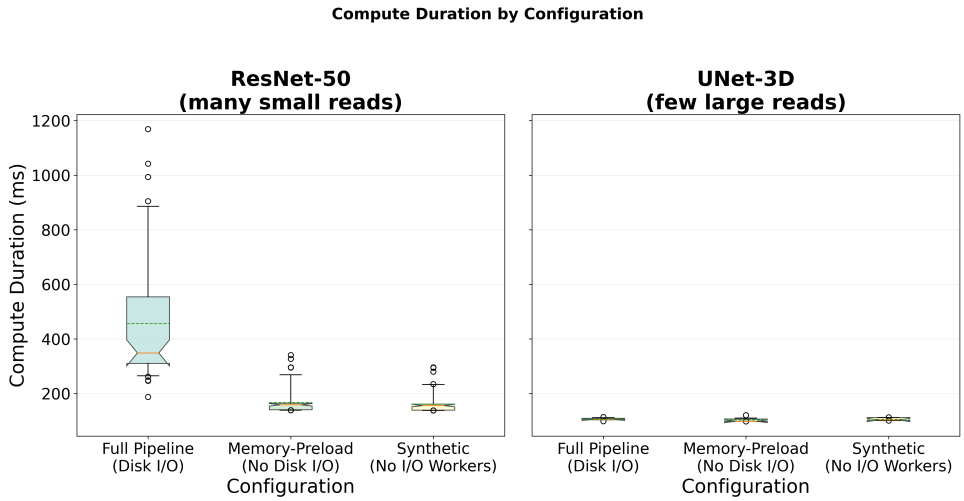


Figure 3: Compute duration by configuration (shared y-axis). ResNet-50 (left): compute spans 187–1,169 ms (std=231 ms) under full-pipeline data loading vs. 139–341 ms without workers. UNet-3D (right): near-identical distributions across all configurations (std=4.5–7.8 ms).

5 Related Work

We situate our work relative to DL benchmarks, tracing tools, overlap-aware training systems, and storage I/O characterization.

DL Benchmarking Landscape. DLIO [11] is the closest existing benchmark to our work, providing configurable I/O benchmarking for DL workloads with multiple data formats, access patterns, and storage backends. However, compute is replaced with calibrated sleep, removing the cross-component interference our work measures. MLPerf Storage [30] packages DLIO as its workload generator for storage benchmarking and therefore inherits the same sleep-based compute emulation, so cross-component CPU/GPU interference remains invisible to both by construction; CoDL targets exactly this gap. Full-pipeline benchmarks like MLPerf Training [28] standardize convergence-based evaluation but treat the pipeline as a black box with no component visibility. Similar tools like DeepBench [46], which benchmarks individual primitives (GEMM, convolutions, AllReduce), and DAWNBench [9] / AIBench [42], which provide cost-aware and industry metrics, all measure either isolated components or aggregate performance without revealing interactions. Our framework extends DLIO with real compute and communication, enabling the first systematic measurement of *interactions* between components that existing benchmarks cannot capture.

Performance Tracing and Profiling Tools. DFTracer [10] provides analysis-friendly I/O tracing with POSIX/STDIO interception and application-level logging on a unified timeline, but cannot identify whether interference manifests in framework operations, GPU kernel scheduling, or NCCL collectives. Facility-scale tools like Darshan [7] and event-level tools like Recorder [44] capture I/O traces but lack compute or GPU visibility. PyTorch Profiler [40] instruments framework operations and GPU kernels via CUPTI but does not capture I/O activity and cannot isolate components. General-purpose HPC profilers—Nsight Systems [39] (CUDA plus optional filesystem-plugin counters), THAPI [4] (heterogeneous device APIs), HPCToolkit [1], TAU [41], Score-P [22], Caliper [6], and DCDB [34] (fleet-scale telemetry)—each cover part of the DL training stack but lack DL-framework instrumentation, treat data loading as opaque user code, and do not provide controllable component isolation. CoDL is the first to put the full DL-pipeline event union—POSIX file I/O, PyTorch Dynamo operators, CUPTI kernels and runtime API, and NCCL collectives—onto one microsecond timeline and combine it with controllable component-isolation experiments, which is what makes the kernel-launch CPU contention identified in Sec. 4 attributable across layers. Concretely, our work extends DFTracer with framework-level (PyTorch Dynamo [2]), GPU (CUPTI [37]), and communication (NCCL [18]) tracing on a unified microsecond timeline.

Overlap-Aware Training and CPU Contention Mitigation. BytePS [21] offloads gradient summation to spare CPUs to mitigate compute-communication contention, while Bagua [14] provides flexible communication overlapping and hierarchical reduction. PyTorch DDP's gradient bucketing [27] overlaps AllReduce with backward propagation, and NVIDIA's CUDA Multi-Process Service (MPS) [36] and NCCL affinity configuration [29] enable CPU-GPU co-scheduling. GPU-side data loading systems such as NVIDIA DALI [38] and FFCV [25] mitigate CPU contention by moving preprocessing off the CPU entirely. These systems demonstrate community recognition of CPU contention as a practical problem; CoDL's measurement infrastructure enables evaluating such mitigations but does not itself propose one.

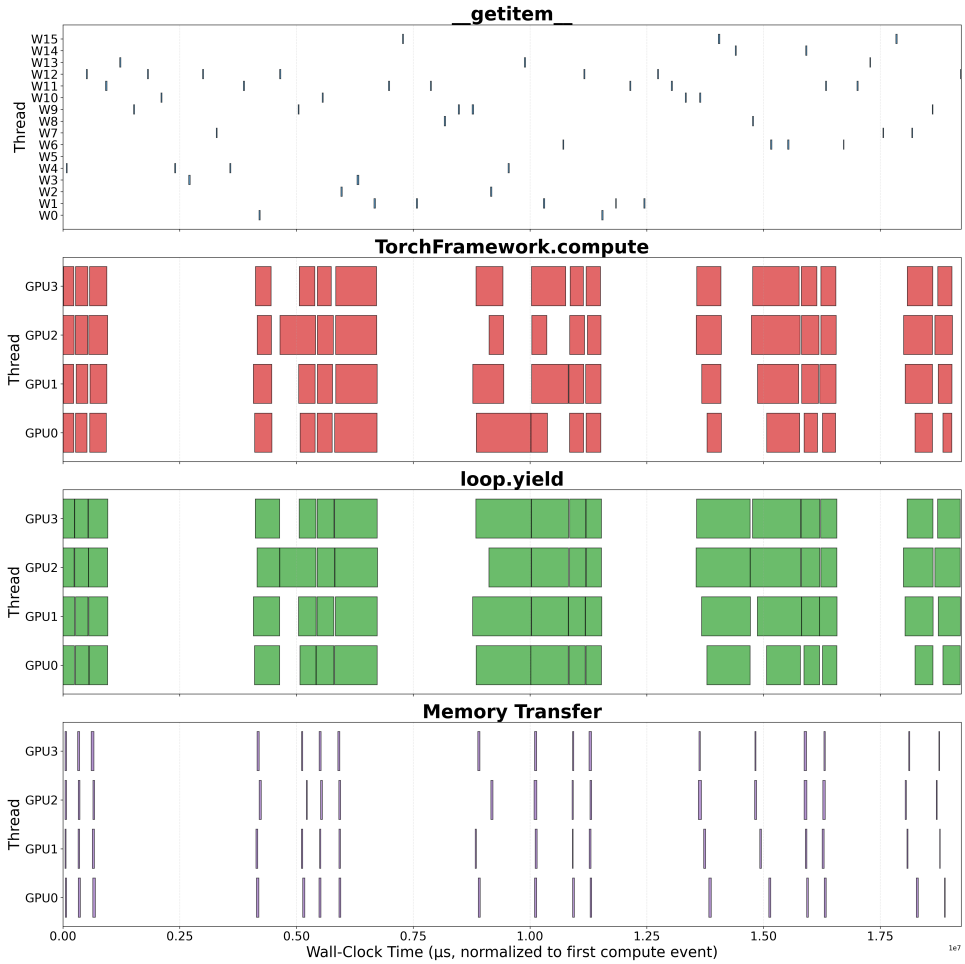


Figure 4: ResNet-50 unified event timeline. Top to bottom: data loading workers (W0–W15), compute (GPU0–GPU3), batch yield, memory transfers. Time is normalized to first compute event. Every compute block temporally overlaps with active data loading, confirming persistent CPU contention.

Storage Characterization and System Parameters. Storage-side factors—stripe configuration, client-side caching, and file layouts—affect I/O performance on parallel file systems [7, 44]. Dryden et al. [12] show that I/O can consume up to 85% of training time at scale and propose clairvoyant prefetching to mitigate it. Our three-run protocol isolates storage effects, but systematic study of how Lustre configuration affects interference magnitude remains future work.

Training Variability and Data Pipeline Analysis. Mohan et al. [31] document that data preprocessing creates fetch and prep stalls that cause significant step-time variability. Isenko et al. [17] show that preprocessing pipelines contain hidden trade-offs causing 3–13× slowdowns. Gujarati et al. [15] demonstrate that individual DNN operations are deterministic but system-level choices introduce variability. Large-scale cluster analyses [16, 19] document queueing delays, locality effects, and resource contention in production GPU clusters. These studies identify variability symptoms; our work provides measurement infrastructure to diagnose the cross-component interference mechanism.

Cross-Layer Profiling and Disaggregated Training. XSP [26] proposes across-stack profiling to correlate framework, library, and kernel events, noting that "existing profiling tools are disjoint and only focus on profiling within a particular level of the stack." Zhao et al. [45] model data storage and ingestion contention at datacenter

Table 3: What CoDL extends over DLIO and DFTracer (top); capability gap vs. other DL profilers (bottom). ✓=present, –=absent.

(a) Extensions over base systems

Component	Base (DLIO / DFTracer)	CoDL addition
Compute backend	Sleep-emulated [11]	Real PyTorch model exec. at component boundary
Communication	Absent	NCCL DDP toggle, AllReduce events
Component isolation	I/O modes only	Mocking adaptor: 3-run protocol (r1/r2/r3)
Configuration	Per-tool YAML	Portability adaptor: framework-agnostic schema
Framework events	Absent	PyTorch Dynamo `fwd`/`bwd` markers
GPU events	Absent	CUPTI kernels, memcopies, runtime API
Comm. trace	Absent	NCCL kernel-pattern identification
Timeline	Host clock only [10]	GPU–host clock unification at ingest (t_0)

(b) Capability vs. other DL profilers

Capability	*DFTracer*	*PyT. Prof.*	*Darshan*	**CoDL**
I/O Tracing	✓	–	✓	✓
Compute Tracing	–	✓	–	✓
GPU Kernel Tracing	–	✓	–	✓
NCCL Tracing	–	–	–	✓
Component Isolation	–	–	–	✓
Unified μs timescale	Partial	Partial	–	✓

scale. On the disaggregation side, tf.data service [3] demonstrates the value of separating data preprocessing from training compute, and Plumber [23] decomposes ML input pipelines into analyzable stages. Chen et al. [8] study GPU multiplexing interference in shared clusters. Our work complements these efforts by providing a unified measurement framework that can quantify interference rather than assuming or mitigating it.

Positioning. Table 3 summarizes the capability gap. Existing benchmarks provide isolation (DLIO) or full-pipeline measurement (MLPerf) but not both. Existing profilers trace one or two levels but not across all levels on a unified timeline. Overlap-aware systems mitigate contention but lack measurement infrastructure to quantify it. CoDL is the first to combine switchable data loaders and compute/communication toggles with application-through-GPU tracing on a unified timeline.

6 Conclusion

We presented CoDL, extending DLIO and DFTracer to systematically quantify cross-component interference in DL training pipelines through controlled experiments that isolate I/O, compute, and communication. Our evaluation on ResNet-50 and UNet-3D revealed that interference propagates through CPU-side scheduling contention rather than direct GPU contention: (1) compute duration more than doubles under data loading interference in ResNet-50 (+127.4%), while UNet-3D shows only 6.4% degradation—the difference traced to access pattern: many small reads create persistent contention (13.4 concurrent loads), while few large reads create intermittent overlap (2.0 concurrent loads); (2) NCCL AllReduce degrades 2.8× in ResNet-50 yet GPU kernels show zero direct overlap with data loading; (3) CUDA runtime API latency increases 3.3×, revealing that data loading workers delay the CPU threads that *launch* GPU kernels rather than the kernels themselves. These findings demonstrate that single-component benchmark optimization (e.g., I/O throughput in isolation) can be misleading if it ignores CPU contention, and that effective DL system optimization requires understanding interference structure across all pipeline components—which only becomes visible on unified multi-level timelines.

Limitations and Future Work. CoDL has four scope limits we plan to address. (1) *Single-node scope.* All experiments use 4 GPUs on one node; inter-node network contention is not exercised, and our timeline unification relies on synchronized job launch rather than explicit cross-node clock alignment (Sec. 3). We expect contention to amplify with multi-node runs and plan to extend timeline unification with PTP/NTP-bounded skew correction and NCCL handshake timestamps as alignment anchors. (2) *Parallelism scope.* CoDL is validated only on DDP; tensor, pipeline, context, and expert parallelism (MoE) introduce different interference patterns (e.g., pipeline bubbles, all-to-all contention) that the mocking adaptor and tracing layer must learn to isolate. (3) *Workload scope.* The evaluation covers many-small-reads (ResNet-50) and few-large-reads (UNet-3D) image workloads, but does not cover compute-bound LLM training (checkpoint writes, large-tensor staging), inference serving, or workloads with heavy CPU-side preprocessing where data loading dominates regardless of read size [35]. (4) *Diagnosis, not mitigation.* CoDL identifies interference but does not prescribe a fix. Our measurements suggest concrete directions—CPU pinning of data loading workers away from kernel-launch threads, NUMA-aware worker placement, and NCCL stream prioritization—which we plan to evaluate using CoDL itself as the measurement substrate.

Acknowledgments

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-CONF-2018702). This material is based upon work supported by LLNL LDRD 25-ERD-042. This work was supported in part by BITS Pilani CDRF grant C1/23/173, BioCyTiH grant BBF/BITS(G)/FY2022-23/BCPS-123/24-25/R1, and SERB SRG grant SRG/2023/002445

References

[1] L. Adhianto, S. Banerjee, M. Fagan, M. Krentel, G. Marin, J. Mellor-Crummey, and N. R. Tallent. 2010. HPCTOOLKIT: tools for performance analysis of optimized parallel programs http://hpctoolkit.org. *Concurr. Comput. : Pract. Exper.* 22, 6 (April 2010), 685–701.

[2] Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, Geeta Chauhan, Anjali Chourdia, Will Constable, Alban Desmaison, Zachary DeVito, Elias Ellison, Will Feng, Jiong Gong, Michael Gschwind, Brian Hirsh, Sherlock Huang, Kshiteej

Kalambarkar, Laurent Kirsch, Michael Lazos, Mario Lezcano, Yanbo Liang, Jason Liang, Yinghai Lu, C. K. Luk, Bert Maher, Yunjie Pan, Christian Puhrsch, Matthias Reso, Mark Saroufim, Marcos Yukio Siraichi, Helen Suk, Shunting Zhang, Michael Suo, Phil Tillet, Xu Zhao, Eikan Wang, Keren Zhou, Richard Zou, Xiaodong Wang, Ajit Mathews, William Wen, Gregory Chanan, Peng Wu, and Soumith Chintala. 2024. PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (La Jolla, CA, USA) *(ASPLOS '24)*. Association for Computing Machinery, New York, NY, USA, 929–947. doi:10.1145/3620665.3640366

[3] Andrew Audibert, Yang Chen, Dan Graur, Ana Klimovic, et al. 2023. tf.data service: A Case for Disaggregating ML Input Data Processing. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC)*. doi:10.1145/3620678.3624666

[4] Solomon Bekele, Aurelio Vivas, Thomas Applencourt, Servesh Muralidharan, Bryce Allen, Kazutomo Yoshii, Swann Perarnau, and Brice Videau. 2025. THAPI: Tracing Heterogeneous APIs. In *Euro-Par 2025: Parallel Processing: 31st European Conference on Parallel and Distributed Processing, Dresden, Germany, August 25–29, 2025, Proceedings, Part I* (Dresden, Germany). Springer-Verlag, Berlin, Heidelberg, 54–70. doi:10.1007/978-3-031-99854-6_4

[5] Tal Ben-Nun and Torsten Hoefler. 2019. Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis. *Comput. Surveys* 52, 4 (2019), 1–43. doi:10.1145/3320060

[6] D Boehme, T Gamblin, D Beckingsale, P Bremer, A Gimenez, M LeGendre, O Pearce, and M Schulz. 2016. Caliper: Performance Introspection for HPC Software Stacks. Lawrence Livermore National Laboratory (LLNL), Livermore, CA (United States). doi:10.1109/SC.2016.46

[7] Philip Carns, Robert Latham, Robert Ross, Kamil Iskra, Samuel Lang, and Katherine Riley. 2009. 24/7 Characterization of Petascale I/O Workloads. In *2009 IEEE International Conference on Cluster Computing and Workshops (CLUSTER)*. 1–10. doi:10.1109/CLUSTR.2009.5289150

[8] Wenyan Chen, Zizhao Mo, Huanle Xu, Kejiang Ye, and Chengzhong Xu. 2023. Interference-aware Multiplexing for Deep Learning in GPU Clusters: A Middleware Approach. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. doi:10.1145/3581784.3607060

[9] Cody Coleman, Deepak Narayanan, Daniel Kang, Tian Zhao, Jian Zhang, Luigi Nardi, Peter Bailis, Kunle Olukotun, Christopher Ré, and Matei Zaharia. 2017. DAWNBench: An End-to-End Deep Learning Benchmark and Competition. In *NeurIPS ML Systems Workshop*.

[10] Hariharan Devarajan, Loïc Pottier, Kaushik Velusamy, Huihuo Zheng, Izzet Yildirim, Olga Kogiou, Weikuan Yu, Anthony Kougkas, Xian-He Sun, Jae Seung Yeom, and Kathryn Mohror. 2024. DFTracer: An Analysis-Friendly Data Flow Tracer for AI-Driven Workflows. In *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–24. doi:10.1109/SC41406.2024.00023

[11] Hariharan Devarajan, Huihuo Zheng, Anthony Kougkas, Xian-He Sun, and Venkatram Vishwanath. 2021. DLIO: A Data-Centric Benchmark for Scientific Deep Learning Applications. In *IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. 81–91. doi:10.1109/CCGrid51090.2021.00018

[12] Nikoli Dryden, Roman Böhringer, Tal Ben-Nun, and Torsten Hoefler. 2021. Clairvoyant Prefetching for Distributed Machine Learning I/O. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. 1–15. doi:10.1145/3458817.3476181

[13] Steven Farrell, Murali Emani, Jacob Balma, Lukas Drescher, Aleksandr Drozd, Andreas Fink, Geoffrey Fox, David Kanter, Thorsten Kurth, Peter Mattson, et al. 2021. MLPerf HPC: A Holistic Benchmark Suite for Scientific Machine Learning on HPC Systems. In *Workshop on Machine Learning in High Performance Computing Environments (MLHPC)*. IEEE.

[14] Shaoduo Gan, Jiawei Jiang, Binhang Yuan, Ce Zhang, Xiangru Lian, Rui Wang, Jianbin Chang, Chengjun Liu, Hongmei Shi, Shengzhuo Zhang, Xianghong Li, Tengxu Sun, Sen Yang, and Ji Liu. 2021. Bagua: scaling up distributed learning with system relaxations. *Proc. VLDB Endow.* 15, 4 (Dec. 2021), 804–813. doi:10.14778/3503585.3503590

[15] Arpan Gujarati, Reza Karimi, Safya Alzayat, Wei Hao, Antoine Kaufmann, Ymir Vigfusson, and Jonathan Mace. 2020. Serving DNNs like Clockwork: Performance Predictability from the Bottom Up. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.

[16] Qinghao Hu, Peng Sun, Shengen Yan, Yonggang Wen, and Tianwei Zhang. 2021. Characterization and Prediction of Deep Learning Workloads in Large-Scale GPU Datacenters. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. doi:10.1145/3458817.3476223

[17] Alexander Isenko, Ruben Mayer, Jeffrey Jedele, and Hans-Arno Jacobsen. 2022. Where Is My Training Bottleneck? Hidden Trade-Offs in Deep Learning Preprocessing Pipelines. In *Proceedings of the International Conference on Management of Data (SIGMOD)*. doi:10.1145/3514221.3517848

[18] Sylvain Jeaugey. 2017. NCCL 2.0: Optimized Primitives for Collective Multi-GPU Communication. In *GPU Technology Conference (GTC)*. NVIDIA. https://developer.nvidia.com/nccl.

[19] Myeongjae Jeon, Shivaram Venkataraman, Amar Phanishayee, Junjie Qian, Wencong Xiao, and Fan Yang. 2019. Analysis of Large-Scale Multi-Tenant GPU Clusters for DNN Training Workloads. In *Proceedings of USENIX Annual Technical Conference (ATC)*. 947–960.

[20] Danlin Jia, Geng Yuan, Yiming Xie, Xue Lin, and Ningfang Mi. 2024. A DataLoader Tunable Knob to Shorten GPU Idleness for Distributed Deep Learning. *ACM Transactions on Architecture and Code Optimization (TACO)* 21, 4 (2024). doi:10.1145/3680546

[21] Yimin Jiang, Yibo Zhu, Chang Lan, Bairen Yi, Yong Cui, and Chuanxiong Guo. 2020. A unified architecture for accelerating distributed DNN training in heterogeneous GPU/CPU clusters. In *Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation (OSDI'20)*. USENIX Association, USA, Article 26, 17 pages.

[22] Andreas Knüpfer, Christian Feld, Dieter Mey, Scott Biersdorff, Kai Diethelm, Dominic Eschweiler, Markus Geimer, Michael Gerndt, Daniel Lorenz, Allen Malony, Wolfgang Nagel, Yury Oleynik, Peter Philippen, Pavel Saviankou, Dirk Schmidl, Sameer Shende, Ronny Tschüter, Michael Wagner, Bert Wesarg, and Felix Wolf. 2012. Score-P: A Joint Performance Measurement Run-Time Infrastructure for Periscope, Scalasca, TAU, and Vampir. 79–91. doi:10.1007/978-3-642-31476-6_7

[23] Michael Kuchnik, Ana Klimovic, Jiri Simsa, Virginia Smith, and George Amvrosiadis. 2022. Plumber: Diagnosing and Removing Performance Bottlenecks in Machine Learning Data Pipelines. In *Proceedings of Machine Learning and Systems (MLSys)*.

[24] Thorsten Kurth, Sean Treichler, Joshua Romero, Mayur Mudigonda, Nathan Luehr, Everett Phillips, Ankur Mahesh, Michael Matheson, Jack Deslippe, Massimiliano Fatica, Prabhat, and Michael Houston. 2018. Exascale Deep Learning for Climate Analytics. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC)*. doi:10.1109/SC.2018.00054

[25] Guillaume Leclerc, Andrew Ilyas, Logan Engstrom, Sung Min Park, Hadi Salman, and Aleksander Madry. 2023. FFCV: Accelerating Training by Removing Data Bottlenecks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

[26] Cheng Li, Abdul Dakkak, Jinjun Xiong, Wei Wei, Lingjie Xu, and Wen-mei Hwu. 2020. XSP: Across-Stack Profiling and Analysis of Machine Learning Models on GPUs. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. doi:10.1109/IPDPS47924.2020.00042

[27] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and Soumith Chintala. 2020. PyTorch distributed: experiences on accelerating data parallel training. *Proc. VLDB Endow.* 13, 12, 3005–3018. doi:10.14778/3415478.3415530

[28] Peter Mattson, Christine Cheng, Gregory Diamos, Cody Coleman, Paulius Micikevicius, David Patterson, Hanlin Tang, Gu-Yeon Wei, Peter Bailis, Victor Bittorf, David Brooks, Dehao Chen, Debo Dutta, Udit Gupta, Kim Hazelwood, Andy Hock, Xinyuan Huang, Daniel Kang, David Kanter, Naveen Kumar, Jeffery Liao, Deepak Narayanan, Tayo Oguntebi, Gennady Pekhimenko, Lillian Pentecost, Vijay Janapa Reddi, Taylor Robie, Tom St John, Carole-Jean Wu, Lingjie Xu, Cliff Young, and Matei Zaharia. 2020. MLPerf Training Benchmark. In *Proceedings of Machine Learning and Systems*, I. Dhillon, D. Papailiopoulos, and V. Sze (Eds.), Vol. 2. 336–349. https://proceedings.mlsys.org/paper_files/paper/2020/file/411e39b117e885341f25efb8912945f7-Paper.pdf

[29] Microsoft Azure HPC Team. 2024. Optimizing AI Workloads on Azure: CPU Pinning via NCCL Topology File. https://techcommunity.microsoft.com/blog/azurehighperformancecomputingblog/optimizing-ai-workloads-on-azure-cpu-pinning-via-nccl-topology-file/4371810.

[30] MLCommons. 2024. MLPerf Storage Benchmark Suite. https://mlcommons.org/benchmarks/storage/. Built on top of DLIO as the workload generator.

[31] Jayashree Mohan, Amar Phanishayee, Ashish Raniwala, and Vijay Chidambaram. 2021. Analyzing and Mitigating Data Stalls in DNN Training. *Proceedings of the VLDB Endowment* 14, 5 (2021), 771–784. doi:10.14778/3446095.3446100

[32] Derek G. Murray, Jiri Simsa, Ana Klimovic, and Ihor Indyk. 2021. tf.data: A Machine Learning Data Processing Framework. *Proceedings of the VLDB Endowment* 14, 12 (2021), 2945–2958. doi:10.14778/3476311.3476374

[33] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R. Devanur, Gregory R. Ganger, Phillip B. Gibbons, and Matei Zaharia. 2019. PipeDream: Generalized Pipeline Parallelism for DNN Training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP)*. doi:10.1145/3341301.3359646

[34] Alessio Netti, Micha Müller, Carla Guillen, Michael Ott, Daniele Tafani, Gence Ozer, and Martin Schulz. 2020. DCDB Wintermute: Enabling Online and Holistic Operational Data Analytics on HPC Systems. In *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing* (Stockholm, Sweden) *(HPDC '20)*. Association for Computing Machinery, New York, NY, USA, 101–112. doi:10.1145/3369583.3392674

[35] Rahma Nouaji, Stella Bitchebe, Ricardo Macedo, and Oana Balmau. 2026. MinatoLoader: Accelerating Machine Learning Training Through Efficient Data Preprocessing. In *Proceedings of the 21st European Conference on Computer Systems*

(McEwan Hall/The University of Edinburgh, Edinburgh, Scotland UK) *(EUROSYS '26)*. Association for Computing Machinery, New York, NY, USA, 2037–2053. doi:10.1145/3767295.3769376

[36] NVIDIA Corporation. 2024. CUDA Multi-Process Service. https://docs.nvidia.com/deploy/mps/index.html.

[37] NVIDIA Corporation. 2024. CUPTI: CUDA Profiling Tools Interface. https://developer.nvidia.com/cupti.

[38] NVIDIA Corporation. 2024. NVIDIA DALI: Data Loading Library. https://developer.nvidia.com/dali.

[39] NVIDIA Corporation. 2024. NVIDIA Nsight Systems. https://developer.nvidia.com/nsight-systems.

[40] PyTorch Contributors. 2023. PyTorch Profiler. https://pytorch.org/tutorials/recipes/recipes/profiler_recipe.html.

[41] Sameer S. Shende and Allen D. Malony. 2006. The Tau Parallel Performance System. *The International Journal of High Performance Computing Applications* 20, 2 (2006), 287–311. arXiv:https://doi.org/10.1177/1094342006064482 doi:10.1177/1094342006064482

[42] Fei Tang, Wanling Gao, Jianfeng Zhan, et al. 2020. AIBench Training: Balanced Industry-Standard AI Training Benchmarking. (2020). arXiv:2004.14690 doi:10.48550/arXiv.2004.14690

[43] Shu-Mei Tseng, Bogdan Nicolae, Franck Cappello, and Aparna Chandramowlishwaran. 2021. Demystifying asynchronous I/O Interference in HPC applications. *Int. J. High Perform. Comput. Appl.* 35, 4 (July 2021), 391–412. doi:10.1177/10943420211016511

[44] Chen Wang, Jinghan Sun, Marc Snir, Kathryn Mohror, and Elsa Gonsiorowski. 2020. Recorder 2.0: Efficient Parallel I/O Tracing and Analysis. In *IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 1–8. doi:10.1109/IPDPSW50202.2020.00176

[45] Mark Zhao, Niket Agarwal, Aarti Basant, Buğra Gedik, et al. 2022. Understanding Data Storage and Ingestion for Large-Scale Deep Recommendation Model Training. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*. 1042–1057. doi:10.1145/3470496.3533044

[46] Hongyu Zhu, Mohamed Akrout, Bojian Zheng, Andrew Pelegris, Anand Jayarajan, Amar Phanishayee, Bianca Schroeder, and Gennady Pekhimenko. 2018. Benchmarking and Analyzing Deep Neural Network Training. In *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 88–100.