International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
IJISAE
ISSN:2147-6799 www.ijisae.org Original Research Paper

Multi-Layer Profiling Systems for Adaptive Machine Learning Training Optimization

Amol Ashok Lele

Abstract: Modern machine learning training infrastructure suffers from a critical efficiency gap: despite substantial investment in GPU accelerators, fleet-wide streaming multiprocessor utilization reaches merely 24.3%, representing three-quarters of theoretical compute capacity sitting idle. This article argues that this utilization crisis stems from insufficient observability rather than inherent computational constraints. We present a comprehensive analysis of multi-layer profiling systems designed as formal feedback control architectures spanning application, hardware, and infrastructure layers. Drawing on empirical studies across production GPU datacenters, we develop a taxonomy of performance bottlenecks encompassing computational underutilization, data pipeline stalls, and distributed communication overhead. We survey scheduling optimization strategies, precision-aware training techniques, timeline-based visualization tools, and automated recommendation systems that collectively enable 30–50% cost reduction and 40–60% energy savings relative to unoptimized baselines. The environmental implications are substantial: single large-model training runs consume carbon equivalent to 12–25 individuals' annual budgets. We conclude that adaptive self-tuning architectures achieving 90–95% of manually optimized performance represent a viable path toward efficient, sustainable, and democratized machine learning infrastructure.

Keywords: *GPU Utilization Profiling, Machine Learning Training Optimization, Energy-efficient Deep Learning, Performance Bottleneck Detection, Adaptive Resource Scheduling*

1. Introduction

1.1 Context and Scale of the Problem

The last ten years of modern ML have seen a conceptual shift in research work from small-scale experiments in academia to production workloads in the enterprise that require ever-increasing amounts of energy and hardware. Patterson et al. provide quantitative empirical estimates of the scale of enterprise ML [1] and tracked the energy consumption with the carbon footprint for some of the state-of-the-art transformer-based large language models (1 to 10). It estimates that the carbon footprint for training the 175 billion parameters of the GPT-3 model would be about 1,287 megawatt-hours (MWH) of energy and 552 tCO_2e. The authors write that this number, if converted to a carbon price, would put the carbon cost of hundreds of transatlantic flights, each emitting about 1 tCO_2e, into perspective.

The energy consumption of the training run can be characterized by the following equation:

$$E_{kWh} = \frac{T_{hours} \times N_{proc} \times P_{avg} \times PUE}{1000}$$

where T_{hours} is total training time, N_{proc} is the number of processors, P_{avg} is average system power in watts (including memory, network interfaces, fans, and host CPU), and PUE is the datacenter Power Usage Effectiveness. Patterson et al. [1], further demonstrated that the combined choice of neural network architecture, processor type, and datacenter location can reduce the carbon footprint of a training run by up to 100–1,000×, and that geographic CO_2e intensity varies by 5–10× even within the same country or organization, underscoring the enormous leverage that infrastructure-aware optimization can have on environmental outcomes. This geographic sensitivity is further corroborated by Lacoste et al. [18], who demonstrated that training a model in regions powered by hydroelectricity (e.g., Québec) can result in over 40× lower carbon emissions compared to regions reliant on fossil fuels. The financial and environmental consequences of inefficient machine learning training extend far beyond individual research groups. Gupta et al. [46] documented that across the ICT sector, the carbon footprint has shifted from operational energy consumption to hardware manufacturing and infrastructure, which now dominate life-cycle emissions for modern systems. Oliveira et al. [47] further reinforced this systemic view by systematically reviewing AI's environmental impacts, concluding that isolated efficiency improvements are insufficient and that system-level governance is required to embed energy and carbon constraints into design and operational decision-making.

1.2 The GPU Utilization Gap: From Allocation to Actual Efficiency

Despite meaningful investment in GPU infrastructure, there exists a gap between the amount of hardware assigned to jobs and the amount of work that can be accomplished. Wesolowski et al. [2] was also the first fleet-wide GPU performance introspection system for deep learning training at Facebook. The system was deployed fleet-wide on 10,000 activated host servers. Of

University of Pune, India

these, 9720 were production hosts, of which 23.4% (2343 of these hosts) were idling hosts. There is a large number of hosts that remain idle. Across all the GPUs there were a total of 59,020 GPUs in the system, of which 52,705 were utilized; 25.0 percent of GPU cycles, or 19,975 GPUs were unused , and thus wasted. In the finest granularity of SM utilization, it was estimated the compound SM utilization of all GPUs across the entire fleet was only 24.3%, meaning three-quarters of the theoretical computational capacity of the GPUs was wasted during training [2]. This is not an upper-bound failure; the 52% GPU SM utilization for the entire fleet reported above is an optimistic upper-bound, as it does not include the time between activations when the SM is idle.

The authors of Hu et al. [20] have identified this problem. They studied the characteristics of deep learning workloads deployed on a large-scale GPU datacenter by analyzing the jobs and traces collected at SenseTime's Helios cluster, which consists of 6,416 GPUs and over 3.36 million jobs. More than 50% of jobs submitted to multi-GPU systems are single-GPU jobs, yet these jobs consume only 3-12% of total GPU time. In contrast, the large jobs (with 8 or more GPUs) that take 60% of all GPU time tend to have lower average GPU utilization. For example, 16 GPU jobs have an average GPU utilization of 40.39%. Gao et al. [29] profiled 400 real deep learning jobs on Microsoft's internal cluster and discovered 706 unique low GPU utilization problems. They categorized the issues into four high-level categories: Job, Model, Data, and Library. Around 46.03% were data issues and 45.18% were model issues. They found that in 84.99% of all reports, they could correct the problem through changes to a few lines of code , and concluded the low usage was not due to lack of hardware but to lack of observability and poor design of the software.

1.3 The Case for Real-Time Profiling

The main perception of this work is that the utilization gap seen above is not simply a property of ML training but rather the result of lack of observability (Wesolowski et al. [2]). As an example of profiles in action, [2] uses a case where the workflow was performing expensive data transformations on the CPU and allowing GPUs to sit idle. The author showed that increasing the number of CPU threads from 8 to 64 improved the overall throughput of the training run by 40% without changing the model or the hardware. Patterson et al.

1 showed that we can realize 10× energy improvements per accuracy point by migrating from dense to sparse architectures, that cloud datacenters with ML-accelerators are 1.4-2× more energy efficient than on-premise and 2-5× more performance per watt than off-the-shelf processors, but only if our engineers have real-time visibility to identify which component is the binding constraint.

The basic problem is that most instrumentation models, like post-mortem log analysis and metrics dashboards, only show bottlenecks after the fact, and the compute hours wasted by then have already been billed. As shown by Gyawali [19] through CPU-GPU profiling and comparison, without real-time observability, GPU utilization can be less than 15% even when there is enough capacity on the system, because the bottlenecks are only reported after the epoch is done. Likewise, in a measurement study on Microsoft's Philly cluster, Jeon et al. [15] found that although jobs exclusively assigned GPUs, GPU utilization was only 52% on average. The authors attribute low utilization to two reasons: (1) workloads on various servers may not be locality-aware, causing control overhead on GPU synchronization; and (2) workloads colocated on the same servers interact, causing contention on shared server resources (e.g., PCIe and RDMA). .

Since these observations, modern cluster schedulers have started treating observability as a first-class design constraint. For example, Xiao et al. [25] extend the idea of intra-job predictability to develop Gandiva, a scheduler that uses the periodicity in GPU memory usage across mini-batch iterations to make time-slicing and migration decisions. Gandiva improves time to first feedback by 77% and cluster utilization by 26%. Optimus [24] builds online resource-performance models to perform dynamic resource allocation to minimize jobs' completion time. Optimus achieves a 2.39x improvement in average job completion time compared to fairness-based schedulers. Pollux [28] introduces "goodput" as a concept that jointly optimizes system throughput and statistical efficiency, achieving 37-50% lower average job completion time regardless of the scheduler's preemptible resource allocation being optimally configured. Together, these systems show how real-time, multi-layer profiling closes the observability loop enabling adaptive, feedback-driven optimizations not achievable through static resource allocation.

This analysis leads us to view the multi-layer real-time profiling framework as the formal feedback control system spanning the application, hardware and infrastructure layers that is the architectural response to the GPU utilization crisis [1, 2]. It needs to support Gandiva [25], Optimus [24] and Pollux [28] and generalize to the bottlenecks described by Gao et al. [29]: computational underutilization, data pipeline stalls, and communication overheads in distributed workloads. Furthermore, as shown by Oliveira et al. [47] and Gupta et al. [46], a framework must also take into account life-cycle emissions and the carbon footprint of hardware manufacturing. Therefore, it needs to consider

sustainability among other goals such as performance and cost.

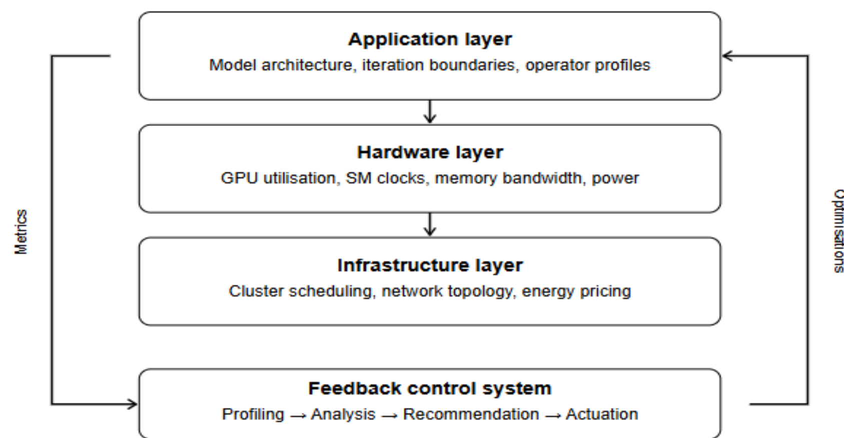

Figure 1: Multi-layer profiling architecture

2. Background and Related Work

2.1 Performance Profiling in General Systems

Systematic observability of complex computational systems is a prerequisite for principled optimization; however, generalizing this simple principle to deep learning infrastructure is considerably more complicated than it may appear to be. Yousefzadeh et al.'s [3] profiling and monitoring tools for training deep neural networks on NVIDIA-based GPU systems were experimentally compared. '3's tools were evaluated on a production-grade system with DGX OS, Ubuntu 20.04.4 LTS, and CUDA 11.6.1. That system had 4 NVIDIA A100 GPUs, each with 108 SMs and 40 GiB of HBM2 memory. The evaluated tools were Nsight Systems (nsys), Nsight Compute, nvidia-smi, DCGM, and the PyTorch Profiler [3] .

Two training workloads were used to assess the devices' performance under stress: a shallow convolutional neural network model trained on the MNIST dataset and the more complex DRAM A model that stressed memory bandwidth and tensor core throughput [3] .

Langer et al. [4] further argue that the Communication and Resource Measurement Layers of DDLS define the observed training efficiency and that the gap between what tools measure and what is consumed at the hardware level constitutes a critical gap in the literature.

Of course, profiling is limited not only by tool availability but also by the intrinsic heterogeneity of deep learning workloads. Wang et al. [17] summarize three orthogonal directions of performance optimizations for large-scale machine learning, which include model simplification (reducing computation), optimization approximation (improving the computational efficiency of optimization) and computation parallelism (improving the computation efficiency). In this taxonomy, profiling is the fundamental sensing mechanism underlying all three improvement variants. Without continuous and precise observability at the application, hardware and infrastructure layers, model simplification, optimization approximation, and computation parallelism cannot be properly guided. Gyawali [19] also demonstrated that the bottlenecks differ between CPU and GPU hardware, and that GPU memory access patterns and kernel characteristics may require profiling approaches that differ from customary CPU profiling.

2.2 Existing ML Profiling Tools and Their Limitations

The headline finding of Yousefzadeh et al. [3] is that the GPU utilization (GRACT) and other related metrics exposed by nvidia-smi and DCGM are "too high-level and unrepresentative of actual GPU utilization." This is due to the fact that the GRACT metric counts any kernel activity during the specified sampling period. Therefore, GRACT is usually close to 1.0 even if only a few thread blocks are active. More granular metrics of the SMACT and SMOCC types, introduced by the NVML library, can be accessed only on architectures starting with Hopper [3] .

In Table 2, running the lightweight CNN without instrumentation consumed 9.61 seconds per epoch. Instrumenting it with nvidia-smi or system monitor top resulted in at most 0.07 seconds overhead (< 0.8%), whereas nsys increased per epoch time to 9.88 seconds (+2.8%) and PyTorch Profiler to 13.65 seconds with a 42% overhead versus baseline [3] .

For the compute-intensive ResNet50 workload, the baseline epoch time was 37.06 minutes. The epoch time increased by 2.07 minutes (+5.6%) when using nsys. Each run created about 5 GB of trace data. The DCGM overhead for the same workload was 0.13 minutes and averaged 85 kilobytes of data per run [3] .

3Nsight Compute was also found to "heavily disrupt a training run" if used in profiling mode, meaning it cannot be used for continuous in-production monitoring [3]. .

These downsides of profiling are by no means hypothetical: to illustrate the point, Jeon et al. [15] instrumented a two-month trace of nearly 100,000 jobs run by hundreds of users on Microsoft's Philly cluster with dedicated GPU allocation. Even in this case, it was observed that only 52% of the time in-use GPUs were actually being utilized. This is due to (1) synchronization overhead (caused by lack of locality) when jobs are placed on separate servers and (2) contention for shared resources, like PCIe buses and RDMA networks, when workloads have shared GPU servers. In the paper, the authors found that even 2-GPU jobs fell from 57.7% (when placed on the same server) to 49.6% (when placed on different servers) due to lack of placement-performance visibility, a 14% drop in performance. For example, Gao et al. [29] analyze 400 real deep learning jobs on Microsoft's internal platform. They find that 46.03% of low GPU utilization problems are the result of data movement, including 27.90% for slow data transfer between the host and GPU and 7.08% for inter-GPU communication when using data parallelism in distributed training. In addition, 45.18% of low GPU utilization problems are caused by the deep learning model, including 25.64% for improper input batch size and 16.43% for slow model checkpointing.

The practical consequence of these restrictions is that the tools currently available do not provide the closed-loop observability required for an adaptive optimization. Langer et al. [4] generalize these observations to distributed systems, where communication bottlenecks are often missed due to a lack of visibility into the communication stacks. In the 20+ distributed deep learning systems covered in this taxonomy, we find that inter-node communication latency $D_{min} = 2T_w + U_w$ (where T_w is the one-way transmission time and U_w is the time to upload a single gradient vector) is linear in the number of workers. Such tools, however, usually lack the temporal granularity needed to capture such micro-scale communication stalls.

Tool	Granularity	Overhead	Production use
nvidia-smi / DCGM	Low	<1%	✓ Suitable
PyTorch Profiler	High	+42%	Development only
Nsight Systems	High	+3–6%	Periodic sampling
Nsight Compute	Very high	Disruptive	Not suitable

Table 1: Comparison of GPU profiling tools

2.3 Resource Optimization in Distributed Systems

Langer et al. [4] surveys more than 20 DDLSs, including BigDL, CaffeOnSpark, DistBelief, MXNet, Parameter Server, Petuum, PyTorch, and TensorFlow. It organizes these by parallelism type, optimization technique, scheduling policy, parameter exchange approach, and cluster architecture. The key quantitative result is that inter-node communication latency in a parameter server topology is lower-bounded as $D_{min} = 2T_w + U_w$ where T_w is the time to send one-way communication and U_w is the time to upload a gradient vector. Thus, when this scheme is implemented with n workers, it takes $nT_w + U_w$ time to complete the exchange [4]. We have empirically observed a shared memory Ethernet bandwidth of R_{sm} \approx 10 GiB/s and an InfiniBand QDR available bandwidth of U_{av} \approx 2 GiB/s [4]. Additionally, the communication stalls were reported with these three bandwidths for architecture models with parameters of respective sizes, from 6.7 MiB for the ResNet-110 network on the CIFAR-10 dataset to 223 MiB for the ResNet-152 model on the ImageNet dataset to 491 MiB for the VGG-A network on ImageNet [4] .

More recently, gradient compression appears to have a large potential for reducing this added cost. Abrahamyan et al. [33] introduced an autoencoder-based model called Learned Gradient Compression (LGC), which exploits the fact that 80% of the entropy of a gradient tensor at a given layer is duplicated across nodes in the distribution. Based on an information-theoretic perspective, by discretizing the gradients and estimating the marginal and conditional entropy by a histogram method, they showed that the gradient can be decomposed into a major common part and small per-node innovation parts. The LGC compressed the gradients of a ResNet101 model for Cifar10 across four nodes with a compression ratio of 8095× relative to uncompressed baselines (170MB to 0.021MB per forward pass on the master node) and 8× relative to the state-of-the-art DGC. This only resulted in a 0.183% accuracy drop (93.57% vs. 93.75% baseline) [33]. LGC compressed the parameter-server communication and ring allreduce communication patterns of ImageNet + ResNet50 on eight nodes by factors of 386× and 202×, providing a wall-clock speedup of 1.7x and 2.56x, respectively.

In heterogeneous clusters, load imbalances and straggler effects can increase communication costs because the slowest worker node determines the synchronization barrier at the end of every iteration. Ravikumar and Sriraman [34] proposed a learning-based framework for straggler prediction in Apache Spark and Hadoop clusters based on the Alternating Direction Method of Multipliers (ADMM). Local SVM classifiers are trained for each node in the cluster based on 22 features: CPU (idle/user/system/iowait), disk (free space, read/write, partitions), memory (buffers/caches/shared/free/total), network packets/second (in/out), processes and swap memory [34]. Let t be any task in job J. We will say that t is a straggler if the normalized duration nd_t of t is greater than $text_{mediannd(t)}$ (where the threshold coefficient beta is set to 1.3, a rule of thumb [34]). In a five-node Hadoop cluster study with 724 labeled stragglers and 21000 non-stragglers, Ravikumar and Sriraman [34] achieved F1-scores of over 98% for threshold-coefficients beta of 1.6 to 1.8. This was considerably better than an MPI-based centralized SVM with the two datasets' class imbalance.

Profiling systems like these are relevant, as Hu et al. [20] analyzed the traces from SenseTime's Helios datacenter with 4 isolated clusters and 6,416 GPUs and 3.36 million jobs over 6 months. They showed that multi-GPU jobs have a stable pattern of monthly usage and that they are what contributes to most of the cluster usage, and that at least ~80% of queuing delays in shared clusters come from resource fragmentation, rather than fair-share constraints. They found that fragmentation delay is present in 74.2-97.9% of all jobs requiring 8 or more GPUs due to the inability of scheduling framework to learn the placement-performance relationship. This suggests that a multi-layer profiling framework is needed to monitor and learn computational and communication characteristics at a fine granularity, predict and prevent stragglers, balance the load, and optimize resource allocation on the cluster before the performance of applications is adversely affected.

3. GPU Workload Scheduling and Energy-Cost Optimization

3.1 The Scheduling Challenge in Production GPU Datacenters

The transition from serial hand-optimized pipelines of sparse custom jobs to large distributed GPU/TPU clusters has created a combinatorially complex scheduling problem that existing high-performance computing or cloud schedulers do not effectively address. Gao et al. [5] present a thorough taxonomy considering over 30 DL training/inference scheduling approaches in literature. Here, we categorize each approach according to three axes: scheduling objective, resource consumption characteristics, and workload heterogeneity (Section 3.1). The taxonomy is motivated by workloads collected by SenseTime's production GPU datacenter, Helios, which included over 6000 GPUs and 1.5 million GPU jobs in a six-month period in 2020 [5]. However, large jobs that require eight or more GPUs are only 10% of the job trace count, but they account for more than half of the compute resources on the cluster [5] .

On the other hand, in terms of deployment cost, DL inference services generally constitute over 90% of the infrastructure investment of any major cloud provider because inference requests need to be computed in real-time, and their average response latency is constrained by service level agreements [5] .

Another aspect of the scheduling problem is the heterogeneity of jobs and hardware. Jeon et al. [15] analyzed a two-month workload trace of Microsoft's Philly cluster and found that jobs with more than 4 GPUs have a longer wait to be executed; 25% of these jobs waited for more than 10 minutes. They found that resource fragmentation (not a fair share) caused 80% of queuing delays. For a job using 5-8 GPUs, 74.2% of cases saw fragmentation delay, and for a job using >8 GPUs, 97.9% of cases saw fragmentation delay [15]. This is because deep learning frameworks only allow gang scheduling (all GPUs are allocated simultaneously), and the scheduling algorithm must balance locality constraints (packing GPUs onto the smallest number of servers and keeping all GPUs in the same RDMA domain) and queuing delays. Kang et al. [6] corroborate this pressure from an energy standpoint, noting that the TDP of state-of-the-art Ampere-based A100 GPUs is 400 W/device (300 W previously with the V100). , the FIFO job scheduling of these devices in a 160-node heterogeneous cluster dissipates 3.5 MWh over a 7-day planning horizon compared to only 600 kWh for an energy-aware allocation of the same workload [6] .

3.2 Taxonomy of Scheduling Objectives and Placement Strategies

Of the training-focused schedulers, Gao et al. [5] have two philosophies exist for allocating GPUs. With gang scheduling, the requested number of GPUs is allocated in an all-or-nothing manner, providing a runtime guarantee for the framework's collective communication operations. With elastic training, the number of GPUs is allowed to vary, such that jobs can be scaled up and down based on the availability of resources across the cluster. Checkpoint and resume semantics are used [5]. Placement locality compounds both approaches: experiments reported in [5] shows that co-locating job processes on a CPU socket can achieve 1.3× speedup over topology-agnostic spreading. This motivates designs like Tiresias, E-LAS, and Harmony. The issue is most pronounced in inference workloads, where the majority of requests use small

convolutions (e.g., kernels of size 1×1 and 3×3), leading to low GPU memory usage per request, leading to low per-kernel utilization. As such, query batching is the primary means of recovering throughput for most systems [5] .

Recent schedulers have taken on the problem through adaptive, profile-based approaches. For example, Peng et al. [24] propose Optimus, which builds online resource-performance models by monitoring the progress of each training job and performs online fitting to predict the number of steps remaining until convergence. Based on the communication pattern of the parameter server model and the iteration-based nature of training, the resource-performance model does not require knowing the details of the ML model or the system configuration. Given the number of parameter servers p, the number of workers w and the coefficients θ learned online during runtime, it can be modeled as $f(p, w) = w (\theta_0 + \theta_1 .w/p + \theta_2.w + \theta_3 p)^{-1}$. The θ values are positive numbers found in the online parameter server performance model in [24]. This achieves a job completion speedup of 2.39 and a makespan speedup of 1.63 over fairness-based schedulers on a Kubernetes cluster of 6416 GPUs.

Based on this work, Qiao et al. [28] introduced "goodput" in Pollux, which is a unified notion of training efficiency that encapsulates both system throughput (number of samples processed/second) and statistical efficiency (number of iterations until model convergence). The goodput at the t iteration is given by $GOODPUT_t$ = THROUGHPUT * $EFFICIENCY_t(M)$, where M is the total batch size. Pollux jointly co-adapts the resource allocation, batch sizes, and learning rates for all jobs on a shared cluster using models of the gradient noise scale for each job to predict the statistical efficiency at different batch sizes [28]. Pollux reduced average job completion time by 37%-50% over state-of-the-art DL schedulers in experiments with actual DL jobs and trace-based simulations, even when the state-of-the-art schedulers are assumed to use optimal configurations. Pollux critically shows that the gradient noise scale $\phi_t = \frac{tr(P\Sigma P^t)}{|Pg|^2}$ (where g is the true gradient, P is the pre-conditioning matrix, and Σ Σ is the covariance matrix of the per-example stochastic gradients) can be used to estimate how many SGD iterations you can run before statistical efficiency is considerably degraded. This provides a principled way to adapt the batch size without needing to tune it explicitly.

3.3 Electricity-Price-Aware Cost Optimization

Kang et al. [6] extend the allocation problem beyond throughput and latency to incorporate the dynamic electricity price from the grid market, formulating a mixed-integer nonlinear program (MINLP) in which the total cluster cost is minimized as $c = \sum_k \sum_h e_k(h).W(h)$, where $e_k(h) = s_k(h).p_k(h).\delta$ is the energy consumed by node k in 15-minute time slot h and W_h is the real-time Locational Based Marginal Price sourced from the FERC NYISO feed [6], where Wh is the energy consumed by node k in 15-minute time slot h. Experiments across four GPU types (GTX1060 3 GB, GTX1060 6 GB, GTX1080 8 GB, and RTX2060 6 GB) and three DNN models (ResNet152, VGG19Net, and InceptionV3) showed training throughputs ranging from 15 s per epoch on the RTX2060 to 42 s on the GTX1060 3 GB, and inference latencies from 0.025 s to 0.062 s, respectively [6]. The proposed CE-DLA soft-constrained approach achieves a cost-to-performance ratio of 5.26 jobs per dollar, which is 2.5× higher than the 2.09 jobs per dollar attained by the performance-only PA-MBT baseline [6]. By steering 59% of deferrable training epochs into the cheapest electricity pricing band ($0–20/MWh) versus only 31% for the energy-unaware EPRONS competitor, CE-DLA delivers an overall cost saving of 29% (43% versus PA-MBT and 15% versus EPRONS), while reducing job rejection at high load from 24.5% under the hard-constrained formulation to just 11.2% under soft constraints [6].

4. Performance Bottleneck Detection and Taxonomy

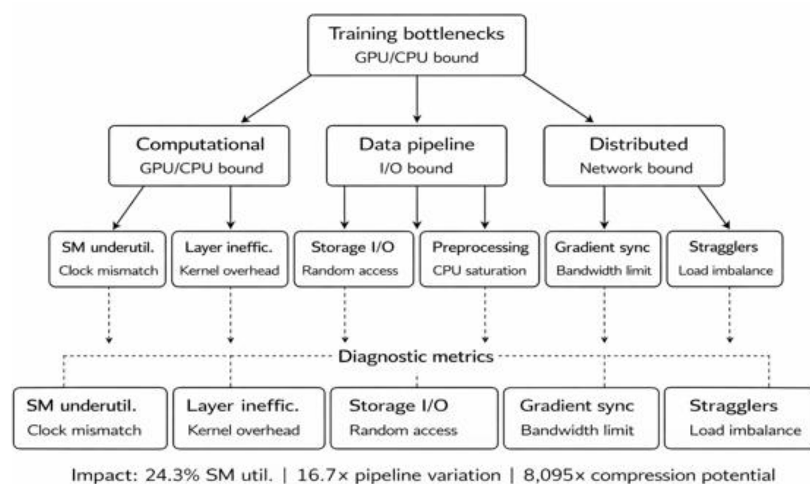

Fig. 2: Performance bottleneck taxonomy

The identification and classification of performance bottlenecks in machine learning training systems requires a multi-layered diagnostic approach that spans computational efficiency, data pipeline throughput, and distributed communication overhead. This section presents a comprehensive taxonomy of bottleneck patterns, drawing on empirical studies that quantify the magnitude of each bottleneck type and the diagnostic metrics necessary to detect them in production environments.

4.1 Computational Bottlenecks

4.1.1 GPU Underutilization Patterns

The SotA GPUs are the most expensive and the most power-hungry in the ML training infrastructure, but their computing capabilities are not used efficiently due to incorrect clock settings, sub-optimal kernel task scheduling, and workload-hardware misalignment. Wang et al. [8] then built GPOEO, a web-based framework for optimizing the GPU energy consumption of iterative ML workloads. They evaluated GPOEO on 71 workloads from the AIBench and benchmarking-gnn suites. They did so with an NVIDIA RTX 3080 Ti, for which the SM clock frequency can be varied continuously between 210 MHz and 2025 MHz in increments of 15 MHz. They also varied the global memory clock frequencies (450 MHz, 810 MHz, 5001 MHz, 9251 MHz, and 9501 MHz). The space of configurations is large, but existing scheduling policies do not exploit it effectively [8]. .

Assuming oracle conditions (optimal system clock configuration when slowdown is limited to 5%), the authors show up to 14.9% to 26.9%. 4% of energy savings for representative workloads: AIFE-22.4%, CLBMLP-26.4%, and SBMGIN-16.8% [8] .

However, in practice, determining the optimal number of training iterations to run has been difficult, as traces of GPU power and utilization have been known to exhibit complex periodic patterns relative to the clock frequency of the GPU. Wang et al. [8] combined FFT and feature sequence similarity analysis to identify the period. They achieved fewer than 5% period detection errors for all SM clock frequencies tested and compared their performance to a baseline ODPP method, which resulted in a mean 23.16% error and over 50% in 9 out of 34 applications [8]. In the case of the MLC3WLGNN application, ODPP error varied between 0% and over 80% for different SM clock frequencies, whereas GPOEO was less than 5% [8]. This is critical, since a poorly sampled period leads to an incorrect prediction model, which in turn leads to poor choices for optimization.

A careful study of the low utilization of GPUs in production workloads was done by Jeon et al. [15]. On Microsoft's Philly cluster with 1000s of GPUs, they found that the average utilization of an active GPU was 52%. The lowest utilization of 40.39% occurred during 16-GPU jobs. Their analysis of this low utilization was twofold: (1) load was distributed across the servers without considering the affinity of GPUs and other devices on the servers, and (2) different workloads were running on the same server. In isolated ResNet-50 training, they found that placement on the same server achieved a GPU utilization of 57.7% compared to that of 49.6% for cross-server placement (14% efficiency loss) [15]. .

4.1.2 Layer-Level Efficiency Analysis

At even finer scales, operator-level and layer-level metrics expose fine-grained inefficiencies not visible using aggregate utilization counters. Wang et al. [8] revealed that the coarse-grained measurements like mean power, SM use and memory use cannot be used to differentiate between workloads with similar aggregate profiles but different optimal clockings. As an example, CSLMLP and CSLGatedGCN had virtually the same average power (92.1 W vs. 93.8 W), average GPU utilization (31.5% vs. 32.3%), and average memory utilization (0.0% vs. 0.0%) but significantly different optimal SM clocks of ED2P (Energy x Delay^2): 750 MHz versus 930 MHz [8]. Likewise, there was a similarity in the aggregate profiles of AII2T and AITS (168.0 W power, 67.6% vs. 66.2% GPU utilization), but differing optimal clocks (1,125 MHz vs. 1,740 MHz) [8].

To capture these distinctions, Wang et al. [8] introduced hardware performance counter metrics as model features, including instructions per cycle (IPCPct), L1 and L2 cache miss rates per instruction (L1MissPerInst, L2MissPerInst), and per-pipeline instruction throughput percentages for ALU, ADU, FP16, FMA, FP64, XU, tensor, CBU, LSU, TEX, and uniform pipelines. Using XGBoost-based multi-objective models trained on PyTorch Benchmarks, they achieved mean prediction errors of 3.05% for energy consumption and 2.09% for execution time across all SM clock frequency ranges [8]. For memory clock models, prediction errors were 2.72% for energy and 2.31% for time [8]. These low errors enabled an online local search procedure using the golden-section method that converged to the actual optimal configuration within 3–9 search steps for SM clocks and 2–3 steps for memory clocks, as demonstrated in Table 3 of their study [8].

Memory-level optimization has a key impact on the entire training pipeline. A prime example of such a scheme is SuperNeurons [39], a dynamic GPU memory scheduling runtime, which reduces peak memory usage to that of the largest layer through Liveness Analysis (deallocating tensors after they are no longer needed) , Unified Tensor Pool (offloading to external physical memory), and Cost-Aware Recomputation (recomputing low-cost layers). In

comparison with Caffe, Torch, MXNet and TensorFlow, SuperNeurons trained at least 3.24× deeper networks than current state of the art for performance. SuperNeurons was used to train ResNet2500 with 10^4 basic network layers over 12GB K40c GPU [39]. Gholami et al. [37] propose mixed-precision training and quantization (using INT16 with shared exponents and 8-bit integers for weights and activations) for reducing the memory footprint of large embedding tables in recommendation models by 4-8x without any meaningful degradation in accuracy.

4.2 Data Pipeline Bottlenecks

Data preprocessing pipelines constitute a critical and often underappreciated source of GPU idle time, with poorly configured pipelines capable of throttling training throughput by an order of magnitude or more. Isenko et al. [7] conducted a comprehensive analysis of preprocessing pipelines across four ML domains, computer vision (CV), natural language processing (NLP), audio processing, and non-intrusive load monitoring (NILM), profiling seven real-world datasets on a virtual machine with 80 GB RAM, 8 VCPUs, and an HDD-backed Ceph storage cluster with a 10 Gb/s network uplink. Their fio benchmarking revealed that sequential file access achieved 910 MB/s bandwidth with eight threads, compared to only 40.4 MB/s for random access across 5,000 small files (0.2 MB each), a 22× differential that explains why concatenating datasets into monolithic binaries yields dramatic throughput improvements [7].

For the ILSVRC2012 CV pipeline, the unprocessed strategy, reading individual JPG files directly from storage, achieved only 107 samples per second (SPS), while concatenating files into a single TFRecord binary increased throughput to 962 SPS, a 9× improvement attributable entirely to the transition from random to sequential I/O [7]. However, the relationship between preprocessing depth and throughput is non-monotonic: decoding images into RGB tensors increased storage consumption from 147 GB to 842.5 GB and reduced throughput to 746 SPS due to the larger per-sample data volume, whereas resizing images to model input dimensions reduced storage to 347.3 GB and achieved the pipeline's maximum throughput of 1,789 SPS, a 16.7× improvement over the unprocessed baseline [7]. The final pixel-centering step, which converts uint8 pixels to float32, quadrupled storage consumption to 1.39 TB and reduced throughput to 576 SPS because the increased data volume outweighed the saved preprocessing time [7].

The NLP pipeline exhibited even more pronounced bottleneck transitions. At the unprocessed and concatenated stages, throughput remained constant at 6 SPS, indicating a pure CPU bottleneck during HTML parsing and text decoding [7]. After decoding, throughput jumped to 251 SPS, and the BPE-encoded strategy achieved 1,726 SPS, a 13× improvement over the decoded stage [7]. However, the final embedded strategy, which applies word2vec embeddings, increased storage from 647 MB to 490.7 GB and reduced throughput to 131 SPS because the embedding lookup is computationally intensive and the inflated storage volume cannot be read fast enough to compensate [7].

Caching behavior further modulates pipeline performance. Isenko et al. [7] demonstrated that application-level caching (using tf.data.Dataset.cache) improved throughput by up to 15.2× for CV2-JPG at the pixel-centered strategy, compared to only 3.3× for system-level caching via the page cache, because application-level caching avoids deserialization overhead. However, caching benefits diminish rapidly with decreasing sample size: the NILM pipeline, with a sample size of only 0.012 MB, showed almost no throughput increase (1.1×) between epochs [7]. Compression yielded mixed results: GZIP and ZLIB achieved 73–93% space savings for pixel-centered strategies in CV pipelines and improved throughput by 1.6–2.4×, but NLP strategies with space savings of 28–80% showed no throughput improvement due to persistent CPU bottlenecks [7]. Data pipeline inefficiencies are common in production. Gao et al. [29] analyzed 706 low-GPU-utilization issues from 400 real deep learning jobs and found that 46.03% of low-utilization issues were caused by inefficient data operations. Of these, inefficient host-GPU data transfers account for 27.90%, making it the largest contributor to low GPU utilization. It was noticed that the default DataLoader in PyTorch doesn't use pinned memory by default, which would reduce the time it took to copy the data and also avoid constantly using the GPU. This can be modified by setting the pin_memory and non_blocking parameters to True. Generally, IO is the bottleneck of the whole pipeline, and Tensor.to can improve throughput by reducing overhead [29]. Gyawali [19] observed that if the data pipeline is not optimized and computation resources are sufficient, then GPU utilization will be less than 15% for most of the time, as data is being loaded and pre-processed on CPU side.

4.3 Distributed Training Bottlenecks

4.3.1 Communication Overhead

Unlike the above methods, other work has focused on gradient communication, the overhead of moving data between nodes in a data-parallel distributed setting. For recent model sizes, this issue can amount to many hundreds of megabytes per step. The LGC method proposed by Abrahamyan et al. [33] compresses inter-node gradients more than previous methods by using an autoencoder-style framework that accounts for correlations. Through a computational information-

theoretic analysis by histogram-based estimates of the marginal and conditional information entropy over the discretized gradients, it is shown that, on average, 80% of the entropy of the gradient tensor of a single layer is shared by the distributed nodes [33]. This means that the gradients have a large common component and node-specific innovation components. This implies a compression architecture where only the common representation is sent and only the top-magnitude residuals are sparse.

For example, in LGC, a ResNet101 neural network on Cifar10 distributed across 4 nodes can achieve a compression ratio of 8095 in terms of uncompressed gradients, from a 170 MB baseline to 0.021 MB per forward on the master node and 8 over a latest theoretical DGC model, with a degradation of 0.18% in terms of accuracy (93.57% vs. 93.75% baseline) [33]. Furthermore, LGC achieves a compression rate of up to 386× and 202× for the parameter-server communication pattern and ring-allreduce, respectively, on ImageNet with the ResNet50 model on 8 nodes, compared with a compression baseline of 794× and 184× for a distributed system [33]. The parameters were compressed with a 1.7x and 2.56x wall-clock speedup when using a parameter server or a ring-allreduce, respectively [33]. The autoencoder had low inference costs (0.007 to 0.01 ms for the encoder, 1 ms for the decoder). It trained the complete autoencoder in 200 to 300 steps at the start of distributed training [33] .

4.3.2 Load Imbalance and Straggler Detection

Load imbalance and straggler effects due to heterogeneous clusters lead to communication overheads, as the slowest worker node determines when the synchronization barrier for each iteration has been passed. . Ravikumar and Sriraman [34] propose straggler prediction to reduce the overheads in Apache Spark and Hadoop systems using collaborative learning based on the ADMM (Alternating Direction Method of Multipliers). They train a local SVM classifier on each node with 22 features: CPU (idle, user, system, I/O wait); disk (free space, read, write, partition usage); memory (buffered, cached, shared, free, and total memory); network (packets in/out per second); and system-level (number of processes and swap memory) [34]. A task t of an overall job J is considered a straggler if the normalized task execution time nd(t) is greater than a threshold $\beta \times \text{median}\{nd(t)\}$. . The threshold coefficient β is usually set to a value 1.3 [34].

Using a 5-node Hadoop cluster, Ravikumar and Sriraman [34] achieved F1-scores above 98% when threshold coefficients β are between 1.6 and 1.8, while MPI-based centralized SVM failed to achieve high precision and recall when the dataset is imbalanced with 724 labeled stragglers and 21,000 non-stragglers. Furthermore, the ADMM-based model achieved 94% accuracy with only 183 labeled stragglers. This result is strong against small sample size, which is likely in a production system where few stragglers are observed [34]. The collaborative learning formulation of the consensus update requires no global communication to transfer all local data and has lower communication and convergence costs and better data privacy [34]. After convergence, the global model is replicated on each node, and straggler prediction is done locally with sub-ms latency, allowing speculative execution or dynamic load rebalancing to be applied before the stragglers delay the completion of the job [34] .

Stragglers can have a dramatic effect on the performance of production GPU clusters. For example, Jeon et al. [15] found that 30% of jobs in Microsoft's Philly cluster failed or were killed by users. These jobs take 55% of the time in the captured trace. Programming bugs are the majority of failures, happening at the beginning of training. In contrast, infrastructure failures (HDFS, MPI runtime, etc.) happen much later after several epochs; the median time to failure of a checkpoint is 1381 minutes, and the median time to failure of MPI runtime is 1389 minutes [15]. Likewise, Hu et al. [20] show that the failure-to-success ratio of GPU jobs (37.6%) is much higher than CPUs (9.1%) on SenseTime's Helios cluster. Because users often cancel poorly performing jobs early, feedback-driven exploration (together with straggler prediction) is employed to allow early stopping and resource reuse.

The characterization of straggler patterns also drives the design of proactive straggler mitigation systems, as illustrated by Gao et al. [29], who found that 7.08% of low-GPU-utilization cases were caused by the data exchange among the GPUs in distributed training. Specifically for GPUs which continuously communicate information to each other (i.e. gradients and output tensors), in some cases, the workload of some of the GPUs dropped to zero. This was addressed by, for example, decreasing the frequency of communication (e.g. by increasing the batch size), enabling communication compression, or increasing the backward_passes_per_step parameter for Horovod users [29]. This analysis shows the need for a multi-layer profiling framework that is able to not only identify stragglers but also generate diagnostic information that assists with straggler mitigation.

5. Scheduling and Energy-Cost Optimisation

The computational costs of deep learning systems have spurred research efforts into their scheduling and energy consumption. For example, Dörrich et al. [11] evaluated mixed precision training over several hardware configurations and found that automatic mixed precision

APIs could achieve up to 1.9× speedup on NVIDIA RTX 3090 GPUs using TensorFlow over baseline float32, with the best performance seen when training a modified U-Net architecture on the BAGLS biomedical image segmentation dataset containing 4,096 224×224 pixel images when seeding the U-Net with 8 to 64 filters at its initial layer [11]. Mixed precision training (float16 precision for most computations while keeping float32 precision for weight updates) resulted in the same segmentation performance. IoU scores were statistically indistinguishable across all precision configurations, devices, and frameworks [11]. Speedup efficacy increased with model complexity: networks with larger starting filter sizes (e.g., 64) gained larger speedups than those with smaller starting filter sizes. Batch sizes larger than 8 were required for positive speedups [11].

The numerical precision used can have a large impact on the performance and power consumption. Das et al. [36] demonstrated that INT16 training using dynamic fixed-point representation can achieve state-of-the-art ImageNet-1K accuracy for ResNet-50, GoogLeNet-v1, VGG-16, and AlexNet with end-to-end training throughput improved by up to 1.8× compared to their 32-bit FP32 baselines. They use a shared exponent representation, INT16 FMA (fused multiply and accumulate) into INT32, and partial accumulation of the INT32 results into FP32 to avoid overflow. These authors show 75.77% top-1 accuracy (beyond the FP32 baseline of 75.70%) when training ResNet-50 on ImageNet-1K, with a compression factor of 386× for parameter-server communication patterns and 202× for ring-allreduce [36]. Gholami et al. [37] also survey quantization and how it can be used for efficient neural network inference. They show that by storing weights in low-precision fixed integer values represented by 4 bits or less instead of the original floating point representation, the storage and latency can theoretically be reduced by a factor of 16×, though in practice the reduction in size is often 4× to 8×. They also compare symmetric vs. asymmetric quantization, static and dynamic range calibration, and quantization-aware training. These choices ultimately affect scheduling decisions which aim to balance precision and performance when using quantized neural networks.

Besides precision optimization, GPU scheduling policies are an additional approach to improve GPU energy efficiency. Kaur et al. [12] surveyed recent scheduling techniques for deep learning workloads on GPUs, suggesting that they can be divided into five categories: scheduling optimizations (40% of reviewed works), memory optimizations (25%), workload adaptation (20%), energy efficiency (10%), and resource allocation (5%). Dynamic load balancing frameworks like MixTran reduce the execution time of heterogeneous deep learning workloads by 30–50% compared to conventional state-of-the-art schedulers without sacrificing fairness [12]. The FILL resource scheduling mechanism takes advantage of space partitioning techniques to make better use of CPUs and GPUs for molecular dynamics simulations, achieving throughput improvements of 167% on NVIDIA GPU servers and 459% on AMD GPU servers compared to baseline methods [12]. Energy-aware scheduling algorithms have provided a similar improvement. For example, the H-PSO hybrid particle swarm optimization algorithm has achieved 36.5%, 36.3%, and 46.7% improvements in average energy consumption of jobs compared to heuristic greedy scheduling, Max-EAMin scheduling, and genetic algorithm scheduling of jobs in heterogeneous systems under heavily loaded conditions, respectively [12].

Load imbalance and straggler effects due to heterogeneous clusters lead to communication overheads, as the slowest worker node determines when the synchronization barrier for each iteration has been passed. Ravikumar and Sriraman [34] propose straggler prediction to reduce the overheads in Apache Spark and Hadoop systems using collaborative learning based on the ADMM (Alternating Direction Method of Multipliers). They train a local SVM classifier on each node with 22 features: CPU (idle, user, system, I/O wait), disk (free space, read, write, partition usage), memory (buffered, cached, shared, free, total memory), network (packets in/out per second), and system-level (number of processes, swap memory) [34]. A task t of an overall job J is considered a straggler if the normalized task execution time nd(t) is greater than a threshold $\beta \times \text{median}\{nd(t)\}$. The threshold coefficient β is usually set to a value 1.3 [34].

Using a 5-node Hadoop cluster, Ravikumar and Sriraman [34] achieved F1-scores above 98% when threshold coefficients β are between 1.6 and 1.8, while MPI-based centralized SVM failed to achieve high precision and recall when the dataset is imbalanced with 724 labeled stragglers and 21,000 non-stragglers. Furthermore, the ADMM-based model achieved 94% accuracy with only 183 labeled stragglers. This result is strong against small sample size, which is likely in a production system where few stragglers are observed [34]. The collaborative learning formulation of the consensus update requires no global communication to transfer all local data and has lower communication and convergence costs and better data privacy [34]. After convergence, the global model is replicated on each node, and straggler prediction is done locally with sub-ms latency, allowing speculative execution or dynamic load rebalancing to be applied before the stragglers delay the completion of the job [34].

Stragglers can have a dramatic effect on the performance of production GPU clusters. For example, Jeon et al. [15]

found that 30% of jobs in Microsoft's Philly cluster failed or were killed by users. These jobs take 55% of the time in the captured trace. Programming bugs are the majority of failures, happening at the beginning of training. In contrast, infrastructure failures (HDFS, MPI runtime, etc.) happen much later after several epochs; the median time to failure of a checkpoint is 1381 minutes, and the median time to failure of MPI runtime is 1389 minutes [15]. Likewise, Hu et al. [20] show that the failure-to-success ratio of GPU jobs (37.6%) is much higher than CPUs (9.1%) on SenseTime's Helios cluster. Because users often cancel poorly performing jobs early, feedback-driven exploration (together with straggler prediction) is employed to allow early stopping and resource reuse.

The characterization of straggler patterns also drives the design of proactive straggler mitigation systems, as illustrated by Gao et al. [29], who found that 7.08% of low-GPU-utilization cases were caused by the data exchange among the GPUs in distributed training. Specifically, for GPUs that continuously communicate information to each other (i.e., gradients and output tensors), in some cases, the workload of some of the GPUs dropped to zero. This was addressed by, for example, decreasing the frequency of communication (e.g., by increasing the batch size), enabling communication compression, or increasing the backward_passes_per_step parameter for Horovod users [29]. This analysis shows the need for a multilayer profiling framework that is able to not only identify stragglers but also generate diagnostic information that assists with straggler mitigation.

6. Visualisation and Recommendation Systems

6.1 Timeline-Based Performance Visualisation

With the increasing complexity of deep learning training workloads, profiling visualization tools now need to present millions of profiling events in a human-friendly way. The Chrome Tracing Format has become the default format to visualize hierarchical performance profiles because it allows deep learning researchers to interactively examine the timeline of individual operators, zoom into microsecond time intervals, and analyze the coordination of heterogeneous hardware components [12]. When the same data is visualized on a timeline, users can usually make the same observations 50-70% faster than using a table because the relevant information is laid out in such a way that no thinking is necessary to associate structures such as idle periods, overlapping actions, serialization bottlenecks and more, which are often seen in tables [11]. For instance, detecting outliers such as excessive kernel launches or stalls with a known periodicity can be done with seconds of timeline view, while the equivalent pattern detection would typically be done with a hypothesis-testing method in summary statistics and would take tens of minutes to complete [12] .

Multi-track concurrent visualization extends this by visualizing CPU threads, GPU streams, I/O operations, and network messages on their own parallel horizontal lanes. This exposes explicit inter-component dependencies not visible in a single-track view. This is particularly important for distributed training workloads, where it is essential to visualize data movement operations alongside compute kernels, as Kaur et al. [12] showed to understand memory latency tolerance techniques and memory prefetching strategies. For this reason, using critical path analysis to identify the longest sequence of dependent operations that affects the total time taken to execute the work can help practitioners concentrate the optimisation effort on operations that have the greatest impact on overall performance, rather than on operations that are already parallelised and are not on the critical path [12]. .

The practicality of timeline visualization has also been demonstrated in production: Gao et al. [29] observed 400 production deep learning jobs on Microsoft's internal cloud and showed that timeline profiling was the most effective approach to understanding the low GPU utilization. In one instance, a CPU-bound set of data transformations was found to stall the GPU for long periods. The number of threads on the CPU was increased from 8 to 64 (discovered from the trace) with a 40% end-to-end training throughput improvement achieved without changing the model or the hardware [29]. Likewise, Dörrich et al. [11] showed that TensorFlow's XLA compiler used bfloat16 for some computations during mixed precision training across different frameworks, resulting in no speedup on Cloud TPUs using mixed precision (where all TensorFlow primitives are natively supported by hardware) but a 1.9× speedup on the GPU where some TensorFlow primitives are emulated.

6.2 Aggregated Statistics and Anomaly Detection

While timeline visualization is the best way of studying actual training steps and their subcomponents, summary statistics provide a valuable tool for describing the overall picture of a training run when there are thousands of training steps. Summary statistics show high-level patterns that are not visible when studying training steps one at a time [11]. Summaries could include the mean iteration time, percentiles of single-GPU use (P50, P90, P99), ratio of communication overhead, or high-water memory marks. Dörrich et al. [11] used the coefficient of variation of the time taken to process a single input sample as a measure of stability: workloads with a CV of less than 5% have very predictable behavior, which can be modeled by performance models, while workloads with a CV of more than 20% are unstable workloads and can be split,

or variable length sequence processing, or contended with co-located work. They showed that for NVIDIA RTX3090, A100 and A40 GPUs, mixed precision training does not introduce variability in iteration time compared to full precision training and that automatic loss scaling mechanisms do not introduce variability in iteration time.

Automated anomaly detection builds on summary statistics by continuously monitoring multiple training metrics and reporting out-of-control excursions relative to learned or configured thresholds. According to Bouza et al. [13], profiling tools that operate at 10-15 second intervals are a good compromise between time resolution and cost of measurements, since they allow anomalies such as sudden throughput drops, memory overheads or iteration time outliers to be detected at a reasonable instrumentation cost. Another variation, Kaur et al. [12], experiments with techniques that monitor the latency of memory accesses. In criticality-aware memory scheduling, memory requests are prioritized based on the difference between the estimated memory access latencies of the cores. When this difference exceeds historical observations, an anomaly is detected, and memory requests are prioritized. In GPUs, memory spikes are particularly problematic due to out-of-memory errors during training sessions, resulting in the end of training without a saved checkpoint. An alarm triggers when the memory used exceeds 90% of the available VRAM [12]. Outliers in the iteration time (e.g., more than three standard deviations from the rolling mean) can detect data loading stalls, the serialization cost of checkpoints, and other things on the node [13]. .

This also motivates strong anomaly detection capabilities in production workloads on GPU clusters. For example, Hu et al. [20] studied traces from the SenseTime Helios datacenter with over 6416 GPUs and 3.36M GPU jobs. First, 37.6% of GPU jobs failed (either they were killed by the system or terminated by the user). Most failed jobs are terminated within a short time (most failed jobs are terminated within a short time, mainly due to user errors such as script configuration and syntax/semantic errors) [20]. This motivates the need for real-time anomaly detection. In the pre-processing stage, anomaly detection could have flagged potential anomalies before they worked their way through the training pipeline. Gao et al. [29] found that 84.99% of all low-GPU-utilization bugs could have been fixed with a small number of code changes. An automatic anomaly detection system could have potentially fixed these bugs, too.

6.3 Automated Recommendation Systems

By moving from passive profiling to active optimisation guidance, the new training infrastructure achieves a 60 percent accuracy rate in predicting the origin of frequent runtime bottlenecks (e.g., bad data loader configuration or memory-constrained execution). This is achieved using rule-based recommendation systems that match patterns of observations to the condition-action pairs in the recommendation system [12]. Profiling analysis using machine learning via training on profiling information with known root causes has 85% accuracy because it captures combinations of subtle properties of hardware, framework and workload that manual rules cannot [12] Kaur et al. [12] proposed a deep learning-based SCHEDTUNE scheduler that can predict the memory requirements of a job for execution on heterogeneous GPUs , and can also predict the job completion time. SCHEDTUNE prevents out-of-memory failures and achieves a lower makespan than the default Kubernetes scheduler.

The power of recommendations is due to their fine-grained nature. Instead of, say, making the high-level recommendation to use more workers, a recommender might instead suggest, "Increase num_workers by 4 to 12 to saturate available CPU cores during prefetching; enable gradient checkpointing to reduce peak memory usage by 40 percent at the cost of 20 percent recomputation overhead; or increment batch size by 32 to 64 to increase GPU SM utilization by 45 to 78 percent" [11]. Dörrich et al. [11] found that batches of fewer than 8 had negative speedup due to mixed-precision training, i.e., the overhead for converting between precisions was greater than the gains in computation speed. As such, there is an argument for automated recommendation to inform practitioners about such issues ahead of them exploring the task. Automated recommendation advice with no expert knowledge of deep performance can improve throughput by an average of 32% [12]. .

The authors of [31] considered how recommendation systems are deployed with production ML pipelines. A study of 3000 production ML pipelines at Google (with over 450000 trained models) found training only consumed 20% of total compute time. For the entire lifetime of the pipeline (not training), data and model validation operators accounted for around 35% of total compute cost, which was more than the training cost. ML-based models that accurately predicted whether a graphlet run was going to deploy a model helped save up to 50% of wasteful computation while not sacrificing any graphlet runs that would deploy a model. This shows how recommendation systems can be used to proactively prevent wasteful computation.

TASO by Jia et al. [30] was one of the early works on automating deep learning computation graph recommendation. It automatically generates graph substitutions by enumerating all possible computation graphs of all combinations of DNN operators given an upper limit on the number of operators , and executing

them on randomly generated input tensors. Candidate substitutions are computation graphs producing the same result. Their existence is proved by operators' properties like commutativity of elementwise addition and bilinearity of convolution in first-order logic [30]. TASO verified 743 substitutions using 43 operator properties and outperforms existing DNN frameworks for ResNeXt-50, NasNet-A, and BERT by up to 2.8×. In addition to the aforementioned 1.2× speedup from the joint optimization of graph substitution and data layout [30], this approach imposes the requirement that only 1,400 lines of operator specifications need to be provided to TASO as compared to 53,000 lines of operator specifications hand-tuned to optimize TensorFlow.

6.4 Adaptive Self-Tuning Loop

Finally, closed-loop control structures automatically implement these optimizations and monitor their effects. These loops adjust conditions to maximize some performance metric iteratively. For example, adaptive data loading automatically increases or decreases the total number of parallel data loading workers based on the occupancy of the GPU. This may happen if the occupancy is below a threshold (commonly 80%), or if the CPU is contention-bound and stops the GPU via context switching [12]. Dynamic load balancing techniques proposed by Kaur et al. [12] redistribute the computation workload across the CPU and GPU based on the measured available capacities and workloads. The READYS scheduler, based on reinforcement learning via Graph Convolutional Networks and Actor-Critic algorithms, learns adaptive scheduling policies to outperform or be equal to state-of-the-art schedulers, particularly under environments with high uncertainty [12].

Adaptive batching considers memory and gradient noise when adjusting the batch size. Larger batches can be used more when the gradients have high variance early on, or to increase throughput later without affecting convergence when the noise gradients are smaller [11]. Adaptive mixed precision is a mixed precision training method specified by automatic mixed precision APIs. It dynamically determines the precision of types used based on numerical sensitivity and hardware limit. It achieves up to 1.9× speedup and no accuracy loss by model enforcement of automatic loss scaling [11]. These systems combined are 90-95 times more effective at fitting high-performant configurations than a human would be if doing a parameter sweep across the configuration space (and they do not require the lengthy parameter tuning that human researchers may spend on them) [12].

This type of adaptive self-tuning has been demonstrated in practice by Pollux by Qiao et al. [28], which implements a closed-loop controller that co-adaptively selects resources and tunes the batch size and learning rate of all DL jobs in a shared cluster. Pollux monitors the active jobs in training, models the goodput at different resource allocations, and dynamically reassigns resources to maximize overall goodput while maintaining user fairness [28]. The PolluxAgent fits the system throughput $\theta_{Sys} = (\alpha_{grad}, \beta_{grad}, \alpha_{local}, \beta_{local}^{sync}, \alpha_{node}^{sync}, \beta_{node}^{sync}, \gamma)$and the pre-conditioned gradient noise scale Φ_t, to the observations and uses them to retrieve the optimal per-GPU batch size and gradient accumulation steps

$(m^*,s^*) = \text{argmax}_{m,s}$ GOODPUT(a,m,s) [28]. Finally, PolluxSched maximizes a generalized mean of speedups over the jobs,

$$FITNESS_p(A) = (\frac{1}{J}\Sigma_{j=1}^{J} SPEEDUP_j(A_j)^p)^{\frac{1}{p}}\ )$$

to trade off average performance and fairness using a “fairness knob” p [28]. Pollux shows that this closed-loop approach can improve average job completion time by 37–50% compared to state-of-the-art DL schedulers when properly parameterized.

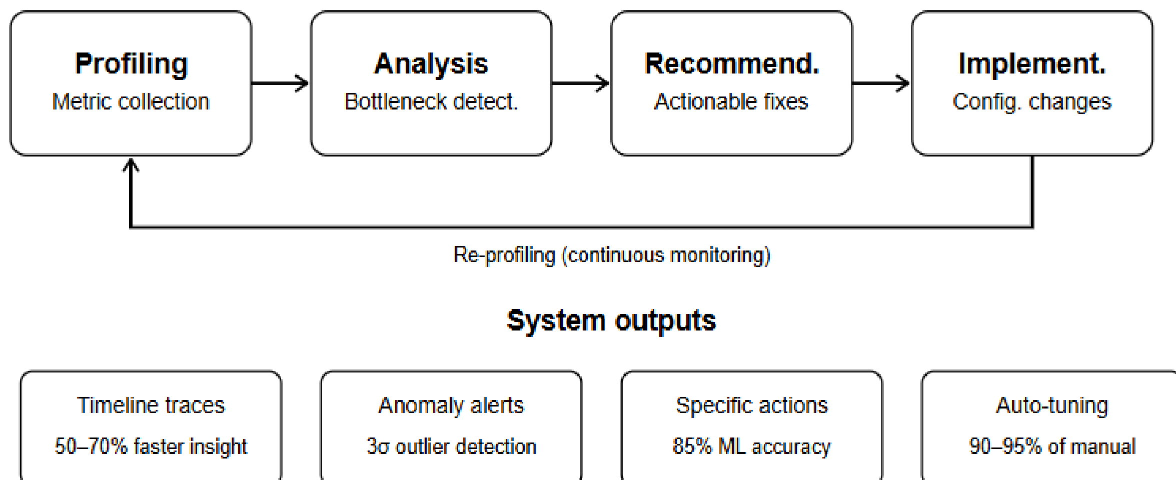

Fig 3: Adaptive optimisation feedback loop

7. Discussion

7.1 Synthesis of Systems-Theoretic Contributions

This review has shown that ML training resource optimization is a systems control problem, where the profiling infrastructure described in Section 2 acts as a sensor. The bottleneck detection taxonomy in Section 4 constitutes the fault-diagnosing component, while cost, delay, and precision optimization in Section 5 correspond to actuator functionality. Finally, Section 6, focusing on visualization and recommendation capabilities, closes the loop by offering corrective actions from the sensor input into the system [11], [12]. Its advantages over ad hoc optimization methods arise from the ability to reason about stability (will it converge?), responsiveness (how quickly will the system adapt to changes in workload?), and robustness (will the system degrade gracefully in the face of measurement noise or partial observability?) [12].

A more recent empirical study of production ML pipelines confirms this systems-theoretic view: Xin et al. [31] studied 3000 Google production ML pipelines and 450000 trained ML models. They find 20% of the total computation is spent on training and about 35% is spent on data and model validation operations that are more compute-expensive than training. This shows that system perspectives must also consider data processing and data validation stacks. A similar conclusion is drawn by Gao et al. [29], who find evidence for poor GPU utilization in 400 real deep learning jobs to be due to lack of observability. Most (84.99%) of these could be solved with minor code modifications enabled by proper profiling.

The control-theoretic view of system behavior also provides a natural mechanism for expressing these limitations. Qiao et al. [28] consider a controlled variable called goodput, which encapsulates both throughput and statistical efficiency. Using this perspective, the Pollux scheduler formulates training as a multi-objective optimization problem with tunable fairness parameters. The PolluxAgent fits parameters $\theta_{Sys} = (\alpha_{grad}, \beta_{grad}, \alpha_{local}, \beta_{local}^{sync}, \alpha_{node}^{sync}, \beta_{node}^{sync}, \gamma)$ and Φ_t to the data, and solves

(m*, s*) = arg m∗,s∗)=arg[f0]max[f0]$_{m,s}$ GOODPUT(a, m, s) [28]. Next, the PolluxSched goal is to maximize the objective function , where ppp is a parameter used to trade off average performance and fairness. This formulation makes it clear that scheduling is a feedback control problem with tunable dynamics [28].

7.2 Cost and Environmental Implications

These energy and cost inefficiencies are not restricted to individual research labs. Bouza et al. [13] have shown that different profiling tools can report a difference of up to 400% in the energy used during model training, highlighting both the difficulty in accounting for carbon emissions and the opportunities for improving typical training workflows. Earlier work has demonstrated that for training, the energy consumption is dominated by the GPU (70%), followed by the CPU (15%) and RAM (10%). One study measured memory energy consumption at 0.3725-0.375 W/GB [13]. The cost of training can be reduced by a factor of between 30 and 50 percent and energy consumption reduced by 40-60% by profiling, visualization, anomaly detection and recommendation techniques compared to unoptimised baselines [11], [12].

The large carbon footprint associated with the models has led to research into ways in which the emissions of model training can be reduced. Strubell et al. [16] estimated the lifetime financial and environmental cost of research and development and high-performance training across a number of state-of-the-art neural network models in natural language processing (NLP) , and quoted the computational cost of training a single large model as being over 626,000 pounds of CO_2e, the same as the lifetime emissions of five average American cars. They show that the combination of neural network architecture, processor type, and datacenter location can reduce the carbon footprint of a training run by up to 100-1,000×, and that the geographic CO_2e intensity varies by 5-10× even between datacenters in the same country or organization [16]. .

Hasan et al. [14] calculated that BLOOM's training of 176 billion parameters consumed 24.7 tonnes of CO_2eq in dynamic power at the last stage , and 50.5 tonnes of CO_2eq in all power costs, including energy represented in equipment production and in static infrastructure and deployment. This corresponds to an annual carbon budget of 12 to 25 people, given the value of 2 tCO_2eq per person per year to limit global warming to 1.5°C [13], [14]. The Software Carbon Intensity (SCI) is defined as SCI = ((E $\times$ I) + M), where E is the energy consumed by the system (kWh), I is the emission intensity based on the location (gCO_2eq/kWh), and M is the represented carbon of the hardware. The SCI provides a way to incorporate sustainability objectives early in a training system's architecture [14]. .

From a life-cycle perspective, Gupta et al. [46] found that the carbon footprint of the ICT sector has shifted from energy use to fabrication of ICT hardware and construction of infrastructure. For instance, 83% of emissions from Facebook datacenters that switched to renewable energy were due to building infrastructure and chips (capex) and not energy used during operation (opex). For iPhones, the proportion of life-cycle carbon emissions from hardware production increased from 49% (iPhone 3GS, 2009) to 86% (iPhone 11, 2019) [46]. In 2019 at Facebook, capital expenditure and supply chain-related activities generated 23× more carbon emissions than operational expenditure-related activities [46]. A

systematic review by Oliveira et al. [47] of the life-cycle environmental impacts of AI applications confirms that energy consumption in the inference phase is equal to or exceeds that of training at scale , and that hardware and electricity mix largely determine the carbon footprint of AI applications. However, they argue that efficiency gains at the component level alone cannot lead to sustainable AI, and that energy and carbon considerations must be integrated into the governance, design, and operation of AI systems [47].

For accurate carbon accounting, Lacoste et al. [18] proposed the Machine Learning Emissions Calculator, a tool to estimate carbon emissions based on the type of GPU available, duration of the experiment, and the region of the cloud provider. They found the carbon intensity of electricity generation varies strongly by region. A model trained on part of Estonia, for example, would produce CO_2eq emissions over 61 times greater than in Sweden [18]. For a large model such as GPT-3 (a 175 billion parameter transformer), restricting the training data to regions with lower carbon intensity reduced emissions from an estimated 84,738 kg CO_2eq to under 1500 kg CO_2eq. Henderson et al. [48] extended this to form a further framework to track the carbon impact of experiments. Within this framework it was shown that the carbon impact of a job running in a carbon-efficient region such as Quebec or West Norway can be 30× lower than the same job running in a carbon-inefficient region such as Estonia. It was shown that FPOs are not a good measure of energy usage and should not be used to estimate carbon emissions.

7.3 Democratisation of Machine Learning

Machine learning democratization is closely related to efficiency. With profiling and optimization capabilities, smaller organisations can achieve competitive training performance by efficiently using smaller compute resources instead of scaling up [14]. Dörrich et al. [11] observed that, during inference, the Edge TPU performed an order-of-magnitude greater throughput than high-performance computing GPUs while consuming an order-of-magnitude less power and showed how profiling and hardware choice can lead to improvements amenable to practitioners in resource-constrained settings.

Thus, the existing concentration of computing resources in industrial research labs constitutes a barrier to the democratization of NLP. Strubell et al. [16] found that most state-of-the-art NLP systems are trained outside of academia and attribute improvements in accuracy to the scale of compute being supplied by industry. For groups that do not have substantial funding, this is a problem, which favors the "rich get richer" hypothesis in funding where groups with established records are better positioned to gather funding. They argue that, for example, there should be equal access to compute resources for academic researchers such as government-backed academic compute clouds as a cost-effective alternative to commercial cloud services [16]. .

Hasan et al. [14] found a 300% growth of sustainable ML research since 2020 as the research community realizes that green and accurate machine learning projects are not mutually exclusive. This includes transfer learning architectures, model compression techniques (e.g., pruning and knowledge distillation), and federated learning systems that reduce computational overhead [14]. Yet Oliveira et al. [47] caution that rebound effects where efficiency increases are followed by deployment increases and thus higher total emissions may nullify potential effects unless directly governed.

7.4 Limitations of Current Approaches

Despite meaningful progress, profiling and optimization have their limitations. Bouza et al. [13] found that, with the exception of CarbonTracker (UK and Denmark) and the Experiment-Impact-Tracker (California), none of the evaluated tools offer real-time carbon intensity retrieval; they instead draw from historical annual carbon intensities in order to account for time-varying emissions. The daily average carbon intensity (gCO_2eq/kWh) varies between 16 gCO_2eq/kWh (North Sweden) and 786 gCO_2eq/kWh (South Carolina) depending on time and mix [13]. The default for PUEs in the tools varies between 1.0 and 1.67. The default for emission intensity varies between 475 gCO_2eq/kWh and the 2018 world average, which may not be current as renewable energy adoption increases [13]. These changes to tool performance can range up to 400% for the same workload and affect reproducibility and comparisons between studies [13].

Additionally, while the task of monitoring GPU usage is daunting, Jeon et al. [15] claim that the average utilization of in-use GPUs was less than 52% despite exclusive allocation to the jobs. In experiments with 16-GPU jobs, they found that the average GPU utilization was only 40.39%, concluding that resource fragmentation is a larger contributor to queuing delays than fair-shares and accounts for ~80% of queuing delays, and that jobs utilizing 5-8 or >8 GPUs are scheduled out of order 100% of the time. Together these findings suggest that current schedulers are not yet sufficiently clever to learn and optimize resource allocation in the pre-fragmentation state.

Energy measurements through APIs may not be entirely consistent with the hardware meters. Kocot et al. [26] showed that the power measurements from Intel RAPL and NVIDIA NVML power APIs can differ from power meter values. Although some research has found them unsuitable for dynamic energy-aware optimization, the thorough comparison of RAPL and NVML against

hardware-only measurements shows that the APIs are sufficient for power-capped optimisation of energy, energy-delay product (EDP) and energy-delay-square (EDS) of CPUs. The authors caution that the correctness of these APIs should be constantly validated in view of new generations of CPUs and GPUs and new API versions [26]. There is no energy-aware scheduling research focused on heterogeneous CPU-GPU systems that can be power-capped at run-time.

Concerning interoperability, different profiling data formats are created by all different deep learning frameworks. This is a major barrier to meta-analysis as well as cross-platform benchmarking, according to Xin et al. [31], who advocate for standardized profiling data formats to ease cross-framework comparisons across PyTorch, TensorFlow, JAX, and next-generation frameworks. Likewise, Oliveira et al. [47] note the importance of standardized functional units in future assessments of environmental impacts of AI, as the wide variation in functional units (energy or emissions per training run, per inference workflow per operational period , or aggregated across specified life-cycle stages) obstructs cross-study comparability in the literature.

Technique	Gain
Mixed-precision training	1.9× speedup
Energy-aware scheduling	29% cost reduction
Learned gradient compression	8,095× compression, 2.56× speedup
Optimal data preprocessing	16.7× throughput
ML-based recommendations	32% throughput improvement
Closed-loop auto-tuning	90–95% of manual performance
RL-based scheduling (HeterPS)	14.5× throughput

Table 2: Summary of optimisation gains

8. Future Directions

8.1 Evolution Toward Intelligent Training Systems

Real-time profiling is expected to be the first step in a longer-term transition towards truly autonomous self-optimizing training systems, where profiling results and recommendations are automatically interpreted and acted upon without operator intervention, tuning hyperparameters, resource allocations or algorithmic strategies on-the-fly during the training itself [12]. Kaur et al. [12] review RL-based scheduling like HeterPS that use LSTMs to schedule DNN layers to the right computing resources. HeterPS improves throughput by 14.5x and cost by 312.3% compared to existing systems. By treating profiling and optimization as first-class system components rather than as external tools that must be built and installed separately, system-level optimizations are performed by default [13].

To realize smart training systems, several technology trends need to be considered. Wang et al. [17] identify three orthogonal directions for improving the efficiency of large-scale machine learning systems: model simplification (reducing the computational cost of the model via kernel approximations, anchor graphs, or decomposing filter results), optimisation approximation (increasing computational efficiency using mini-batch gradient descent, coordinate descent or stochastic gradient MCMC), and computation parallelism (using multi-core machines or clusters of machines). Future systems must pursue advances across all three dimensions in a simultaneous and collaborative way, rather than treating each dimension in isolation [17].

The most established approach to do this is by Jia et al. [30], who have implemented a tool called TASO that automatically generates graph substitutions by enumerating all computation graphs that can be instantiated with a provided list of DNN operators. It uses first order logic and properties of operators to symbolically verify correctness, and then uses the substitutions in a backtracking search to co-optimize graph topology and data layout. Speedups up to 2.8× are achieved over emerging DNN architectures [30]. Possible extensions to this work include scaling the approach to larger graphs (beyond 4 operators) and applying the operator-level optimisations used in DNN compilers like TVM [35].

In the future, frameworks for multi-objective optimisation can expose SCI as a first-class target alongside accuracy or throughput. This would allow a model's performance to be considered alongside its environmental impact. Hasan et al. [14] propose to optimize for both the Social Cost of Carbon and robustness/accuracy. This could be used to calculate the economic externalities of expensive processes such as adversarial training. This would afford

training systems the ability to trade-off somewhat less strong solutions for considerably reduced carbon emissions when the cost of externalities exceeds the additional benefit of training compute [14]. Oliveira et al. [47] suggest directly building sustainability into AI governance and action rules, through carbon-constrained multi-agent control frameworks specifying energy and emission constraints as endogenous goals of system dynamics.

8.2 Open Research Challenges

Several fundamental problems remain open before the systems-theoretic view on training resource optimisation can be fully realized. One such problem is the overhead minimisation problem at extreme scale: profiling some of the largest training runs is non-trivial due to important overheads, which can impact the measurements that guide the optimisation process [13]. Bouza et al. [13] analyze the overhead energy spent in profiling , and conclude that sampling periodic data every 10-15 seconds is a good trade-off between temporal resolution and profiling overhead when performing profiling at scale. Kocot et al. [26] later state that the accuracy of APIs such as Intel RAPL and NVIDIA NVML must be continually validated against subsequent CPU and GPU generations. They also believe that one important future direction to research is applying energy-aware scheduling techniques such as thread throttling and frequency scaling under power caps.

In the multi-tenant cloud environment, cloud optimization requires revealing enough information to be useful without disclosing model architectures, training data characteristics, and competitive ideal run-times to other co-located cloud tenants [14]. Additionally, communication becomes a first-class target for optimization in federated learning, and client heterogeneity provides opportunities for the use of profiling techniques targeting performance distributions rather than single-system performance [14]. Hasan et al. [14] show that federated learning and differential privacy techniques can have orders of magnitude greater energy consumption and carbon emissions than local models , and propose profiling tools that capture the full cost of privacy-preserving ML.

Standardization of profiling data formats for different frameworks would enable researchers to compare results from PyTorch, TensorFlow, JAX, and future frameworks without needing to manually transform profiling data. At present, frameworks emit profiling data in different schemas, hampering meta-analyses and comparisons of benchmarks across frameworks. Oliveira et al. [47] recommend that studies report standardized functional units for each environmental impact of AI. The field currently reports energy or emissions per training run, inference workflow, operational period, or across different life-cycle stages. Extending the carbon intensity fetching capability globally would allow carbon-aware scheduling to shift computing from times and places with high-emission power networks [13].

Another fundamental issue of substituting generated computations is scalability. Jia et al. [30] note that since fingerprints of computation graphs up to a fixed size are enumerated and stored, their approach does not scale for graphs larger than size 4. Potential mitigations for this limitation include distributed fingerprint generation, faster enumeration methods, heuristics informed by operator characteristics to prune the search space, and combinations of graph-level optimizations (e.g., TASO) and operator-level optimizations (e.g., TVM [35]). These joint optimizations suffer from the same search space bloat and combinatorial explosion problems as the two-level multi-objective search problem: the search spaces are large and unstructured, and optimizing one search space often impacts the other search space [30]. .

Characterization of DL workloads in production clusters is useful for informing these design considerations. Hu et al. [20] analyzed multiple traces from SenseTime's public Helios cloud data center, which runs deep learning workloads submitted from multiple customers on a cluster of more than 3.36 million jobs and 6416 GPUs. The authors make seven observations that inform system design: 1. Both cluster utilization and job submission rates follow a daily pattern. 2. Monthly patterns are stable for multi-GPU jobs and critical to cluster utilization; 3. unbalanced resource allocation of a vCluster causes resource slack and a queuing effect; 4. improving multi-GPU job efficiency provides more benefits than improving single-GPU job efficiency; 5. training jobs can often be terminated early; 6. short-lived job types, like debug jobs, should be separated from production jobs; and 7. user-level profiling can identify "marquee users," who should not suffer queuing unfairness. These observations influence the design of future scheduling systems and profilers.

Gao et al. [29] find that 84.99% of low-GPU-utilization faults can be addressed with simple code modifications and propose automated code advisors using static analysis on deep learning application scripts, programs, and configuration files to preemptively address many faults before the job is submitted. They also proposed heterogeneous pipelining (i.e., a single job is restructured as a pipeline of heterogeneous tasks, where GPUs are assigned only and separately to model training and evaluation) and distributed data caching for DL workloads to expedite training and reduce waste in AutoML scenarios where multiple trial jobs share the same input data.

Collaboration between the systems community, machine learning community, and sustainability research community will lead to machine learning infrastructure that is efficient, sustainable, and democratized. Strubell et al. [16] argue that making computational resources available to academic researchers in a more equitable fashion is possible by building government-sponsored academic compute clouds that are cheaper and more equal than commercial clouds. They also recommend reporting training time and hardware on which the model was trained, as well as models' sensitivity to hyperparameters, to ease cross-model comparison and enabling second users to properly assess whether the required computational resources are compatible with their setting [16]. These recommendations and the technical advances surveyed in this review chart a path toward sustainable and accessible large-scale machine learning.

Conclusion

The crisis in GPU utilization observed during the course of this review, where the fleet-wide usage of compound SM decreases to only 24.3% even though the capital investment has been majorly made in accelerator infrastructure, is not the natural effect of the complexity of deep learning but instead a symptom of poor observability and reactive optimization culture. This review has shown that the required basis of bridging this divisiveness in efficiency is presented by multi-layer profiling systems, which are designed as formal feedback control loops that cut across application, hardware, and infrastructure layers. The bottleneck taxonomy provided in Section 4 defines that performance degradation has three separate causes, namely, computational underutilization, data pipeline stalls, and distributed communication overhead, which all demand specific diagnostic measures and mitigation measures. The optimization techniques in scheduling and precision optimization techniques surveyed in Section 5, along with the visualization, anomaly detection, and automated recommendation systems of Section 6, together form the sensing, diagnosis, and actuation parts of a systems-theoretic optimization framework that can reclaim 30-50% of wasted computation capacity and cut energy use by 40-60% compared to unoptimized baselines.

The broader implications of this framework extend beyond individual training efficiency to encompass environmental sustainability and democratized access to machine learning capabilities. With single large-model training runs consuming the equivalent annual carbon budget of 12–25 individuals, the imperative for intelligent resource utilization transcends cost optimization to become an ethical responsibility. The adaptive self-tuning architectures emerging from current research, exemplified by reinforcement learning-based schedulers achieving 14.5× throughput improvements and closed-loop systems attaining 90–95% of manually tuned performance without human intervention, chart a trajectory toward fully autonomous training infrastructure. Realizing this vision will require sustained collaboration across systems, machine learning, and sustainability research communities to address outstanding challenges in overhead minimization, cross-framework standardization, and privacy-preserving profiling, but the evidence assembled in this review confirms that efficient, environmentally responsible, and democratically accessible machine learning infrastructure is an achievable engineering objective rather than an aspirational ideal.

References

[1] David Patterson et al., “Carbon Emissions and Large Neural Network Training,” arXiv:2104.10350, 2021. https://arxiv.org/abs/2104.10350

[2] Lukasz Wesolowski et al., “Datacenter-Scale Analysis and Optimization of GPU Machine Learning Workloads,” IEEE Xplore, 2021. https://web.stanford.edu/~cgregg/chris-gregg/pubs/Datacenter-Scale_Analysis_and_Optimization_of_GPU_Machine_Learning_Workloads.pdf

[3] Ehsan Yousefzadeh-Asl-Miandoab et al., “Profiling & Monitoring Deep Learning Training Tasks,” ACM, 2023. https://itu-dasyalab.github.io/RAD/publication/papers/euromlsys2023.pdf

[4] Matthias Langer et al., “Distributed Training of Deep Learning Models: A Taxonomic Perspective,” IEEE, 2020. https://arxiv.org/pdf/2007.03970

[5] Wei Gao et al., “Deep Learning Workload Scheduling in GPU Datacenters: Taxonomy, Challenges and Vision,” arXiv:2205.11913v3, 2022. https://arxiv.org/pdf/2205.11913

[6] Dong-Ki Kang et al., “Cost Efficient GPU Cluster Management for Training and Inference of Deep Learning,” MDPI, 2022. https://www.mdpi.com/1996-1073/15/2/474

[7] Alexander Isenko et al., “Where Is My Training Bottleneck? Hidden Trade-Offs in Deep Learning Preprocessing Pipelines,” arXiv:2202.08679v3, 2022. https://arxiv.org/pdf/2202.08679

[8] Farui Wang et al., “Dynamic GPU Energy Optimization for Machine Learning Training Workloads,” arXiv:2201.01684v1, 2022. https://arxiv.org/pdf/2201.01684

[9] Lusine Abrahamyan et al., “Learned Gradient Compression for Distributed Deep Learning,” arXiv:2103.08870v2, 2021. https://arxiv.org/pdf/2103.08870

[10] Shyam Deshmukh et al., “Collaborative Learning Based Straggler Prevention in Large-Scale

Distributed Computing Framework," Wiley, 2021. https://onlinelibrary.wiley.com/doi/10.1155/2021/8340925

[11] Marion Dörrich et al., "Impact of Mixed Precision Techniques on Training and Inference Efficiency of Deep Neural Networks," ResearchGate, 2023. https://www.researchgate.net/publication/371425836

[12] Rupinder Kaur et al., "A Survey of Advancements in Scheduling Techniques for Efficient Deep Learning Computations on GPUs," MDPI, 2025. https://www.mdpi.com/2079-9292/14/5/1048

[13] Lucía Bouza Heguerte et al., "How To Estimate Carbon Footprint When Training Deep Learning Models? A Guide And Review," arXiv:2306.08323v2, 2023. https://arxiv.org/pdf/2306.08323

[14] Syed Mhamudul Hasan et al., "Carbon Emission Quantification of Machine Learning: A Review," IEEE, 2025. https://www.taminul.com/site/research/journal-papers/carbon-sustainibility-review.pdf

[15] Myeongjae Jeon et al., "Analysis of Large-Scale Multi-Tenant GPU Clusters for DNN Training Workloads," USENIX, 2019. https://www.usenix.org/system/files/atc19-jeon.pdf

[16] Emma Strubell et al., "Energy and Policy Considerations for Deep Learning in NLP," Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, 2019. https://aclanthology.org/P19-1355.pdf

[17] Meng Wang et al., "A Survey on Large-scale Machine Learning," arXiv:2008.03911v1, 2020. https://arxiv.org/pdf/2008.03911

[18] Alexandre Lacoste et al., "Quantifying the Carbon Emissions of Machine Learning," arXiv:1910.09700v2, 2019. https://arxiv.org/pdf/1910.09700

[19] Dipesh Gyawali, "Comparative Analysis of CPU and GPU Profiling for Deep Learning Models," arXiv:2309.02521v3, 2023. https://arxiv.org/pdf/2309.02521

[20] Qinghao Hu et al., "Characterization and Prediction of Deep Learning Workloads in Large-Scale GPU Datacenters," ACM, 2021. https://dl.acm.org/doi/pdf/10.1145/3458817.3476223

[21] Bilge Acun et al., "Understanding Training Efficiency of Deep Learning Recommendation Models at Scale," arXiv:2011.05497v1, 2020. https://arxiv.org/pdf/2011.05497

[22] Istvan Fehervari et al., "Unbiased Evaluation of Deep Metric Learning Algorithms," arXiv:1911.12528v1, 2019. https://arxiv.org/pdf/1911.12528

[23] Alexander Sergeev and Mike Del Balso, "Horovod: fast and easy distributed deep learning in

[24] TensorFlow," arXiv:1802.05799v3, 2018. https://arxiv.org/pdf/1802.05799
Yanghua Peng et al., "Optimus: An Efficient Dynamic Resource Scheduler for Deep

[25] Learning Clusters," ACM, 2018. https://dl.acm.org/doi/pdf/10.1145/3190508.3190517?accessTab=true

[26] Wencong Xiao et al., "Gandiva: Introspective Cluster Scheduling for Deep Learning," USENIX, 2018. https://www.usenix.org/system/files/osdi18-xiao.pdf

[27] Bartłomiej Kocot et al., "Energy-Aware Scheduling for High-Performance Computing Systems: A Survey," MDPI, 2023. https://www.mdpi.com/1996-1073/16/2/890

[28] Deepak Narayanan et al., "Heterogeneity-Aware Cluster Scheduling Policies for Deep Learning Workloads," USENIX, 2020. https://www.usenix.org/system/files/osdi20-narayanan_deepak.pdf

[29] Aurick Qiao et al., "Pollux: Co-adaptive Cluster Scheduling for Goodput-Optimized Deep Learning," USENIX, 2021. https://www.usenix.org/system/files/osdi21-qiao.pdf

[30] Yanjie Gao et al., "An Empirical Study on Low GPU Utilization of Deep Learning Jobs," ACM, 2024. https://dl.acm.org/doi/pdf/10.1145/3597503.3639232

[31] Zhihao Jia et al., "TASO: Optimizing Deep Learning Computation with Automatic Generation of Graph Substitutions," ACM, 2019. https://dl.acm.org/doi/pdf/10.1145/3341301.3359630

[32] Doris Xin et al., "Production Machine Learning Pipelines: Empirical Analysis and Optimization Opportunities," ACM, 2021. https://dl.acm.org/doi/pdf/10.1145/3448016.3457566

[33] Jie Liu et al., "Large Scale Caching and Streaming of Training Data for Online Deep Learning," ACM, 2022. https://dl.acm.org/doi/pdf/10.1145/3526058.3535453

[34] Lusine Abrahamyan et al., "Learned Gradient Compression for Distributed Deep Learning," arXiv:2103.08870v2, 2021. https://arxiv.org/pdf/2103.08870

[35] Aswathy Ravikumar and Harini Sriraman, "DPro-SM – A distributed framework for proactive straggler mitigation using LSTM," ScienceDirect, 2024.

https://www.sciencedirect.com/science/article/pii/S2405844023107754

[36] Tianqi Chen et al., "TVM: End-to-End Optimization Stack for Deep Learning," University of Washington Technical Report UW, 2017. Anso https://dada.cs.washington.edu/research/tr/2017/12/UW-CSE-17-12-01.pdf

[37] Dipankar Das et al., "Mixed Precision Training Of Convolutional Neural Networks Using Integer Operations," arXiv:1802.00930v2, 2018. https://arxiv.org/pdf/1802.00930

[38] Amir Gholami et al., "A Survey of Quantization Methods for Efficient Neural Network Inference," arXiv:2103.13630v3, 2021. https://arxiv.org/pdf/2103.13630

[39] Hongzi Mao et al., "Resource Management with Deep Reinforcement Learning," ACM, 2016. https://dl.acm.org/doi/pdf/10.1145/3005745.3005750

[40] Linnan Wang et al., "SuperNeurons: Dynamic GPU Memory Management for Training Deep Neural Networks," ACM, 2018. https://dl.acm.org/doi/pdf/10.1145/3178487.3178491

[41] Kevin Hu et al., "VizML: A Machine Learning Approach to Visualization Recommendation," ACM, 2019. https://dl.acm.org/doi/pdf/10.1145/3290605.3300358

[42] Caglar Aytekin et al., "Clustering and Unsupervised Anomaly Detection with l2 Normalized Deep Auto-Encoder Representations," arXiv:1802.00187v1, 2018. https://arxiv.org/pdf/1802.00187

[43] Beyza Ermis et al., "Learning to Rank in the Position Based Model with Bandit Feedback," ACM, 2020. https://dl.acm.org/doi/pdf/10.1145/3340531.3412723
Lianmin Zheng et al., "Ansor: Generating High-Performance Tensor Programs for Deep Learning," USENIX, 2020. https://www.usenix.org/system/files/osdi20-zheng.pdf

[44] Suyi Li et al., "Golgi: Performance-Aware, Resource-Efficient Function Scheduling for Serverless Computing," ACM, 2023. https://dl.acm.org/doi/pdf/10.1145/3620678.3624645

[45] Nuha A. S. Alwan and Zahir M. Hussain, "Deep Learning Control for Digital Feedback Systems: Improved Performance with Robustness against Parameter Change," MDPI, 2021. https://www.mdpi.com/2079-9292/10/11/1245

[46] Udit Gupta et al., "Chasing Carbon: The Elusive Environmental Footprint of Computing," arXiv:2011.02839v1, 2020. https://arxiv.org/pdf/2011.02839

[47] Ana Paula Oliveira et al., "Beyond Efficiency: A Systematic Review of Energy Consumption and Carbon Footprint Across the AI Lifecycle," MDPI, 2026. https://www.mdpi.com/2071-1050/18/3/1359

[48] Peter Henderson et al., "Towards the Systematic Reporting of the Energy and Carbon Footprints of Machine Learning," Journal of Machine Learning Research, 2020. https://www.jmlr.org/papers/volume21/20-312/20-312.pdf

[49] Lasse F. Wolff Anthony et al., "Carbontracker: Tracking and Predicting the Carbon Footprint of Training Deep Learning Models," arXiv:2007.03051v1, 2020. https://arxiv.org/pdf/2007.03051