IJRAR.ORG E-ISSN: 2348-1269, P-ISSN: 2349-5138

INTERNATIONAL JOURNAL OF RESEARCH AND ANALYTICAL REVIEWS (IJRAR) | IJRAR.ORG

An International Open Access, Peer-reviewed, Refereed Journal

# Designing Profiling Tools for Scalable Distributed AI Workloads

**[1]Ankush Jitendrakumar Tyagi**

[1]University of Texas at Arlington, Texas, USA

***Abstract:*** Managing distributed workloads has become even more challenging as artificial intelligence (AI) gets scaled up in data centers, cloud infrastructure, and the edge network. Such workloads can frequently traverse heterogeneous computing landscapes, include large-scale datasets and multiple simultaneous operations, which makes it progressively harder to discern performance bottlenecks, resource inefficiencies, and operation balances. To overcome these challenges, it will be necessary to develop state-of-the-art profiling tools that are purpose-defined on distributed AI workloads and also scalable. In contrast to being before performance analysis tools, recent profiling systems need to encompass an entire tool set of functionalities, such as real-time monitoring capability of the intra-node transmissions, memory consumption, use of hardware resources, and a task scheduler. Insights into the distribution of AI system metrics (e.g., GPU kernel utilization, model inference latency, and data pipeline throughput) collection and analysis are vital to optimizing distributed AI systems. Further, lightweight instrumentation and distributed logging can be used to aid observability without adding excessive operational overhead, further enabling continuous system improvement. Scalability Profiling must support a tool architecture that can be easily interoperable with common AI frameworks, in addition to key orchestration platforms, including, but not limited to, TensorFlow, PyTorch, Kubernetes, and Apache Spark. Successful profiling packages should further possess ease of visualization dashboards, auto detection of bottlenecks, and intelligent recos systems to aid in quick diagnostics and identify problems. Improving the AI development lifecycle with these tools will allow system architects and developers to markedly decrease latency, improve model performance, and optimize resource and model deployment. This yields a high-performance, adaptive, distributed AI environment with the ability to dynamically react to changing workload. Thus, this study critically explores the architectural principles and integration strategies necessary for building scalable profiling tools tailored to distributed AI workloads.

***IndexTerms*** - Distributed AI Workloads, Performance Profiling Tools, Scalable Systems Monitoring, AI Infrastructure Optimization, Resource Utilization in Machine Learning

## I. INTRODUCTION TO DISTRIBUTED AI WORKLOADS AND SCALABILITY CHALLENGES

Managing distributed workloads has become even more challenging as artificial intelligence (AI) gets scaled up in data centers, cloud infrastructure, and the edge network. Such workloads can frequently traverse heterogeneous computing landscapes, include large-scale datasets and multiple simultaneous operations, which makes it progressively harder to discern performance bottlenecks, resource inefficiencies, and operation balances [1-4]. To overcome these challenges, it will be necessary to develop state-of-the-art profiling tools that are purpose-defined on distributed AI workloads and also scalable. In contrast to being before performance analysis tools, recent profiling systems need to encompass an entire tool set of functionalities, such as real-time monitoring capability of the intra-node transmissions, memory consumption, use of hardware resources, and a task scheduler [2-6]. Insights into the distribution of AI system metrics (e.g., GPU kernel utilization, model inference latency, and data pipeline throughput) collection and analysis are vital to optimizing distributed AI systems. Further, lightweight instrumentation and distributed logging can be used to aid observability without adding excessive operational overhead, further enabling continuous system improvement.

Scalability Profiling must support a tool architecture that can be easily interoperable with common AI frameworks, in addition to key orchestration platforms, including, but not limited to, TensorFlow, PyTorch, Kubernetes, and Apache Spark. Successful profiling packages should further possess ease of visualization dashboards, auto detection of bottlenecks, and intelligent recos systems to aid in quick diagnostics and identify problems. Improving the AI development lifecycle with these tools will allow system architects and developers to markedly decrease latency, improve model performance, and optimize resource and model deployment. This yields a high-performance, adaptive, distributed AI environment with the ability to dynamically react to changing workload [2-7]. The paper aims to illuminate the essential performance metrics, data collection techniques, and emerging trends that enable real-time optimization across diverse AI infrastructures.

## II. FUNDAMENTALS OF PERFORMANCE PROFILING IN AI SYSTEMS

Performance profiling is a term used to describe the act of dynamically quantifying the actions of a system to learn about performance characteristics, uncover inefficiencies, and inform optimisation. Profiling in AI systems is especially hard because of computation-intensive operations (e.g., matrix products, convolutions), asynchronous I/O, as well as the existence of multiple layers of data spaces [5]. Fundamentally, profiling tools measure quantitative statistics about system functions like the utilization of the CPU/GPU, memory utilization, caches, thread scheduling, and I/O throughput. These measures are later examined to identify the hotspots, stalls, and underutilization of the resources, as well as contending points [6]. As an illustration, one classic bottleneck in machine learning train/inference pipelines is latency due to poor memory access patterns or sub-optimal batching: this latency can severely impact the performance of deep learning models [7]. AI systems profiling should be expanded beyond conventional runtime profiling and support AI characteristics like layer-wise operation breakdown, tensor data motion tracking, kernel execution traces, and automatic differentiation overheads. NVIDIA Nsight, Intel VTune, and PyTorch Profiler are tools that can be used to gain insight into these dynamics; however, most are optimized to work on standalone systems as opposed to distributed deployments [8]. In addition, distributed performance profiling has to integrate spatio-temporal analysis, where the interdependence of computation and data transfer is analyzed in the context of more than one node. This necessitates support of distributed traces, event correlation, and cross-layer visibility, usually through a hybrid of agents, APIs, and shared instrumentation stacks [9]. Figure 1 highlights how profiling quantifies system actions to optimize AI workloads. It emphasizes profiling dimensions such as hardware, software, and synchronization, leading to actionable insights for performance improvement.

When profiling is used as part of the AI development lifecycle, the search time to debug an issue can be drastically reduced, and optimization of the computation graph is also possible. The realization of such goals at scale does however, requires architectural considerations, which are discussed in the next section.

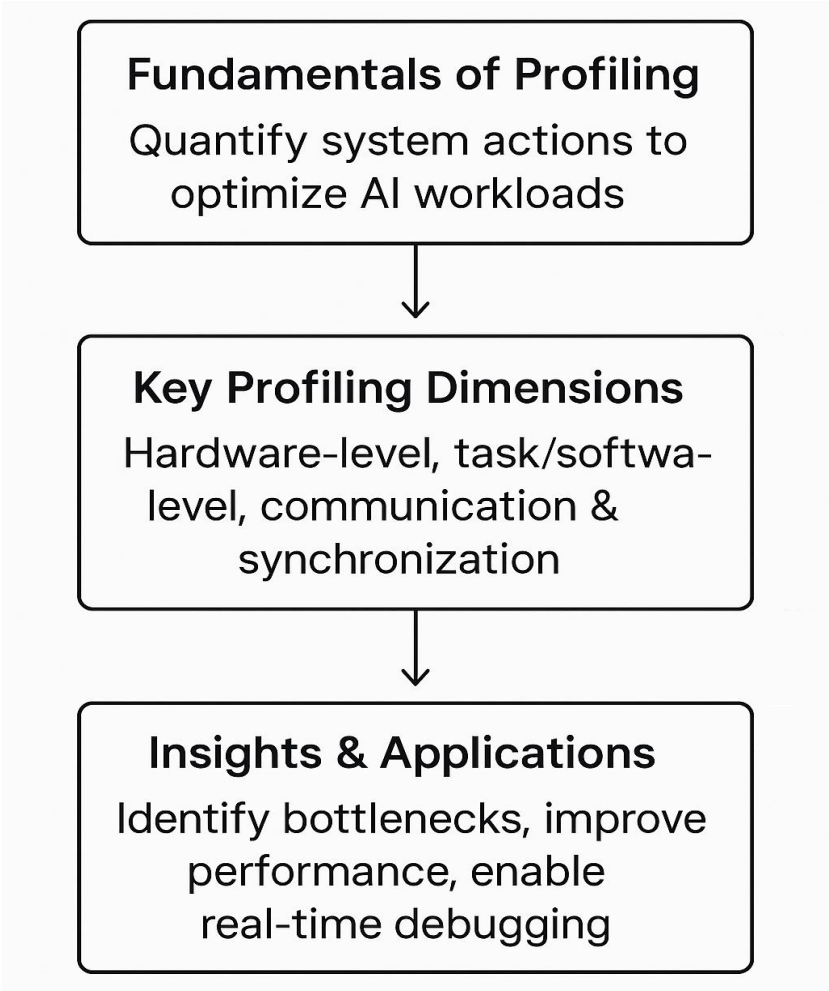


Figure 1. Fundamentals of Performance Profiling in AI Systems.

## III. ARCHITECTURAL DESIGN PRINCIPLES FOR SCALABLE PROFILING TOOLS

Profiling tools used to manage distributed AI workloads must be thoughtful of their architecture to strike the right mix of efficiency, scalability, and system overhead. Profiling tools should have modular and scalable structures that are responsive to a variety of runtime environments and hardware configurations because, in many cases of AI workloads, parallelism and/or complex scheduling are involved [10]. Figure 2 shows a three-layer architecture: data collection and storage at the foundation, a profiling core handling modularity, instrumentation, and distributed monitoring, and a visualization layer that delivers analytics and AI-driven recommendations. This layered design ensures scalability, interoperability, and actionable optimization in distributed AI systems.

### 3.1 Modularity and Extensibility

Variance of modules is a basic rule to let individual components of the profiler, i.e., logging agents, metric collectors, visualization layers, and decision engines to be developed and maintained independently. Such flexibility means this modularity can relieve the flexibility of the AI workloads and infrastructure. Consider, as an example, a data center may require full-scale telemetry of memory and interconnect traffic, yet an edge node may only require monitoring of CPU utilization. An example of such modularity is illustrated by profilers such as OpenTelemetry and Perfetto that can be customized or extended in terms of APIs and SDKs to fit specific needs, without modifying the whole framework [11]. The extensibility is also essential. Profiling tools should enable plug-and-play compatibility with a range of machine-learning frameworks (e.g., TensorFlow, PyTorch), libraries (e.g., CUDA, MKL-DNN), and accelerators (e.g., NVIDIA GPUs, Google TPUs, AMD ROCm). This is normally accomplished by the use of abstraction layers and plugin interfaces that insulate the hardware-specific functions from core profiler functions [12].

### 3.2 Lightweight Instrumentation

Low-overhead Instrumentation is required by scalable profiling. Profiling of distributed AI cannot interfere with or degrade performance in the same way as traditional debugging tools. This is preferred to exhaustive logging in favor of sampling-based profiling. Linux perf and eBPF (Extended Berkeley Packet Filter) are examples of such tools that can monitor events at the level of a single instruction without the overhead of a full system trace, which enables them to scale to high-throughput inference settings [13]. Also, the lightweight run-time hooks can be placed inside the training and inference loop to monitor the number of GPU kernel invocations, the size of allocated tensors, and the latency of batches processed. This allows constant tracking during production without the need for a work delay.

### 3.3 Distributed Monitoring and Synchronization

Distributed profiling capabilities have to synchronize among a large number of nodes, containers, or virtual machines, potentially in real time. This demands a coherent synchronization strategy to provide timestamp alignment, event correlation, and cross-node traceability. à Politely, gRPC, Message Passing Interface (MPI), and Apache Kafka are some of the most common technologies deployed to facilitate communication and STREAMING of telemetry data [14]. Additionally, there are normally profiling tools that contain trace aggregation servers or central collectors, which collate and analyse logs on all the compute nodes. Such an architecture compares to the Prometheus gathering of metrics using exporters via the exporter scraping architecture in a Kubernetes cluster. Metadata such as timestamps is a crucial part of being able to diagnose cross-node problems, such as load distribution being skewed between the nodes and lags in communication [15].

### 3.4 Visualization and Interpretability

An often-overlooked architectural feature is the inclusion of interactive dashboards and visual analysis tools. Visualizations assist developers and system engineers in interpreting the performance data collected. Tools like TensorBoard, Grafana, and FlameGraphs offer intuitive representations of CPU and GPU usage, kernel timelines, and memory bandwidth, allowing stakeholders to spot anomalies at a glance [16]. The profiler must also support exportable logs, diagnostic reports, and recommendation systems that highlight optimization opportunities. Advanced profilers may employ machine learning models themselves to predict future bottlenecks or suggest tuning parameters based on prior performance patterns [17].

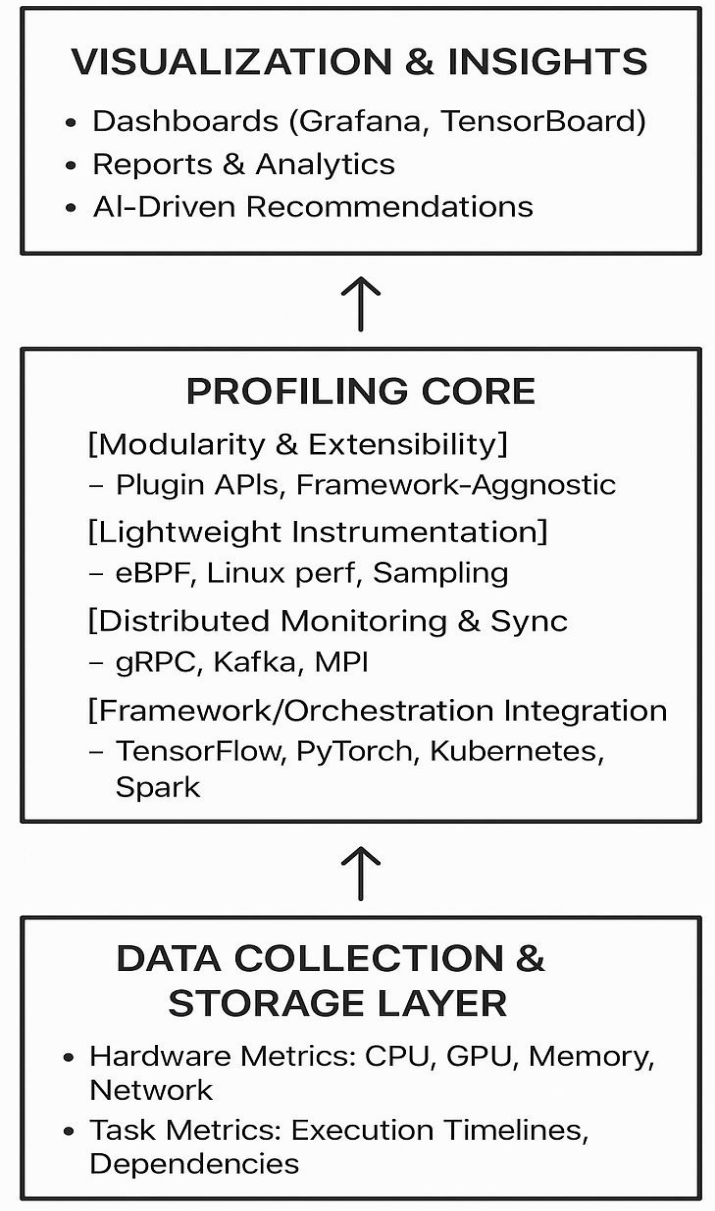


Figure 2. Architectural Design Principles for Scalable Profiling Tools.

## IV. KEY METRICS AND DATA COLLECTION TECHNIQUES FOR DISTRIBUTED ENVIRONMENTS

A core responsibility of any profiling tool is the accurate, real-time collection and interpretation of performance metrics that are relevant to AI workloads. Distributed environments introduce new complexities in metric gathering due to hardware heterogeneity, network variability, and task fragmentation [18].

### 4.1 Hardware Utilization Metrics

Key hardware metrics include CPU utilization, GPU core and memory utilization, disk I/O throughput, network latency, and cache performance. Modern GPUs, for instance, provide rich telemetry through tools like NVIDIA's NVML (NVIDIA Management Library), which can be queried to monitor temperature, memory usage, kernel execution time, and PCIe bandwidth utilization [19]. Profiling tools must offer native support or drivers to collect such metrics across platforms.

### 4.2 Memory and Storage Metrics

In ML workflows, inefficient memory allocation, especially tensor duplication or fragmentation, can result in catastrophic slowdowns. Profilers must track dynamic memory allocations, garbage collection, and tensor life cycles. Framework-specific hooks, or TensorFlow's memory profiler API, can be integrated for these purposes [20]. In distributed settings, shared storage access and disk I/O contention can become bottlenecks. Metrics like I/O wait time, read/write latency, and disk queue length are vital to understanding how well the system handles data ingress and egress.

### 4.3 Communication and Synchronization Metrics

Communication overhead in distributed training, especially during gradient synchronization in data-parallel training, can be a performance bottleneck. Profiling tools should capture latency, bandwidth, message size, and frequency of communication across interconnects such as InfiniBand, NVLink, or Ethernet. Tools like Horovod Timeline, NCCL Profiler, and MPI Trace are essential for analyzing these aspects [21]. Profilers may also need to measure barrier synchronization time, straggler impact, and clock skew, especially in loosely coupled systems.

### 4.4 Software and Task-Level Metrics

Beyond hardware, profiling must encompass task granularity, such as the time spent in data preprocessing, model loading, layer-wise execution, and inference serving. This level of detail helps diagnose pipeline inefficiencies. Metrics like operation duration, dependency wait time, and CPU-GPU transfer delays are important indicators. Framework-integrated tools like TensorFlow Profiler and PyTorch Profiler can provide timeline-based views of the model's execution graph, highlighting slow operators and underutilized resources [22].

## V. INTEGRATION WITH FRAMEWORKS AND ORCHESTRATION SYSTEMS

Profiling tools designed for distributed AI workloads must integrate seamlessly with the broader ML software ecosystem, including deep learning frameworks, data orchestration tools, and deployment infrastructures. Such integration ensures that profiling can occur continuously and contextually embedded within the development, training, and inference lifecycle without requiring manual instrumentation or pipeline disruption [23].

### 5.1 Integration with ML Frameworks

Profiling tools should also integrate directly with mainstream ML frameworks such as TensorFlow, PyTorch, JAX, and MXNet to get task-specific measurements and layer-level information. Such frameworks already introduce hooks and APIs to the internal profiling. As an example, TensorFlow Profiler is integrated with TensorBoard, which enables visualization of per-operation latency, memory footprint, and device placement. On the same note, PyTorch Profiler also allows scheduling of execution phases and supports integration with Kineto, a trace visualizer that shows operator timelines, CPU/GPU usage, and backward/forward pass behavior [24]. It is also highly desirable that the profiler itself be framework-agnostic in nature, enabling it to monitor performance data regardless of whether multiple frameworks, or particularly custom training loops, are used within the same pipeline. It can be done, e.g., through the use of standardised tracing protocols (e.g., OpenTracing, OpenTelemetry), which allow cross-framework instrumentation and consistent data collection [25].

### 5.2 Integration with Orchestration and Scheduling Systems

Most modern orchestration systems, such as Kubernetes, Apache Spark, Ray, and Slurm, can be used to deploy modern AI workflows. These are used to run and manage resource distribution, pod scheduling, job queuing, and fault recovery over distributed systems. Combining profiling tools with some of these orchestrators allows run-time awareness and adaptive profiling where metrics could be dynamically adjusted to best suit the system condition. As an example, metrics in Kubernetes can be exported as Prometheus endpoints, scraped by a monitoring stack, and displayed using Grafana dashboards. Profilers allow attaching metadata of the pod name, namespaces, and job identifiers to the metrics and offer some contextual traceability [26]. In addition, it is possible to add profiling agents to containers when jobs are deployed using Kubernetes operators or Daemon Sets (automatically and non-intrusively). Profiling tools may integrate with the Spark UI, event log system, or structured streaming metrics API to profile distributed training or data processing tasks in Apache Spark. It can be integrated with Ray time profiler and potentially help profile dynamic task graphs and nested actors in reinforcement learning loads or hyperparameter tuning jobs [27].

### 5.3 Profiling Across Hybrid and Edge Environments

With an increasing many AI workloads being deployed into hybrid edge computing systems spanning centralized cloud clusters and decentralized edge nodes, profiling tools need to provide support in the hybrid space. That involves the need to support heterogeneous runtime environments, variable connectivity, and different data collection strategies. Profilers should be lightweight so as to run on edge devices and have limited resources, but they should transparently be able to export metrics to central dashboards when the network connectivity permits. Such an approach can include deploying Lightstep or Jaeger to perform distributed tracing in a setting with periodic edge nodes that sync their logs or metrics to a centralized store. Federated profiling is the other method, and the edge profilers can summarize local performance details and only transmit aggregated results to prevent network choking or the disclosure of sensitive information [28].

## VI. FUTURE TRENDS: INTELLIGENT ANALYTICS AND AUTONOMOUS OPTIMIZATION

The next evolution of profiling tools lies in intelligent, AI-driven analytics that not only detect performance issues but also recommend or autonomously apply solutions. This shift transforms profiling from a reactive debugging tool to a proactive optimization engine.

Emerging tools leverage machine learning to detect anomalies in performance data, predict resource exhaustion, and recommend corrective actions. For instance, time-series models can forecast GPU memory spikes based on training batch size trends, while unsupervised learning techniques can cluster system traces to identify abnormal execution paths [29]. Profilers may also learn from historical profiling sessions, using them to build heuristics that automatically tune parameters such as data loader concurrency, kernel fusion thresholds, or batch sizes depending on hardware characteristics and model types.

Looking ahead, profilers will be integrated with autonomous system controllers that can reconfigure the runtime environment in response to profiling feedback. For example, if a profiler detects persistent GPU underutilization, the controller might reduce the number of worker replicas or reassign workloads to better-matched nodes [30]. Such adaptation is already seen in adaptive data parallelism, on-the-fly quantization, and automatic mixed precision (AMP) strategies that optimize memory and compute utilization during training. When informed by real-time profiling feedback, these systems can be extended into generalized auto-scaling and load-balancing mechanisms across distributed clusters.

Another important trend is the standardization of profiling APIs and telemetry formats. Initiatives such as OpenTelemetry, Cloud Native Computing Foundation (CNCF) observability stack, and MLPerf Logging aim to define universal schemas for performance data, making tools more interoperable and reducing integration overhead [31]. Community-led efforts also drive the growth of open-source profiling libraries (e.g., Py-Spy, Flamebearer, and Scalene), which can be embedded into training scripts or container runtimes with minimal effort. Open ecosystems foster innovation and enable cross-platform support, particularly important for democratizing profiling in resource-constrained academic and startup environments.

## VI. CONCLUSION

As an intensely distributed workload arrives, profiling has become a key enabler of performance, scale, and reliability. With the proliferation of artificial intelligence systems into multi-node systems, cloud-edge systems, and heterogeneous hardware accelerators, the problem of determining performance bottlenecks and managing the use of resources has become exponentially more complex. The AI profiling tool retained as part of traditional profiling tools was only applicable to isolated systems and monolithic applications during the early times, and it is not applicable anymore in contemporary AI development. The paper examines the critical need for scalable profiling tools to support the evolving demands of AI systems. It presents a breakdown of modern profiler architectures, highlighting the importance of modularity, extensibility, lightweight instrumentation, and distributed synchronisation. Key metrics requiring measurement, such as hardware-level utilisation, task-level execution timelines, and techniques for capturing these metrics with minimal disruption, are analysed in detail. Additionally, the integration of profiling tools with widely used machine learning frameworks (e.g., TensorFlow, PyTorch) and orchestration platforms (e.g., Kubernetes, Spark) is demonstrated to enable contextual use in production environments, offering real-time spatial and temporal insights for performance optimisation. Advances in profiling into the future can be characterized by the trends of artificially-enhanced analysis, self-optimizing systems, open, and normalizing tooling ecosystems. Such inventions will change profiling into a branch of smart self-optimizing infrastructure. Finally, the integration of sophisticated profiling in the workflow of AI system inception and production will also be essential towards the development of highly creative, elastic, and reliable AI services. Researchers, system architects, and developers who adopt this philosophy can be better placed to use the full capabilities of distributed artificial intelligence to accomplish its real-world potential.

## REFERENCES

[1] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," Communications of the ACM, vol. 51, no. 1, pp. 107–113, 2008.

[2] T. Kraska, A. Talwalkar, J. C. Duchi, R. Griffith, M. J. Franklin, and M. I. Jordan, "MLbase: A Distributed Machine-learning System," in CIDR, vol. 1, pp. 2-1, Jan. 2013.

[3] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in 2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 10), 2010.

[4] M. Satyanarayanan, "The emergence of edge computing," Computer, vol. 50, no. 1, pp. 30–39, 2017.

[5] J. L. Hennessy and D. A. Patterson, Computer Architecture: A Quantitative Approach, Elsevier, 2011.

[6] Q. Duan, "Cloud service performance evaluation: status, challenges, and opportunities–a survey from the system modeling perspective," Digital Communications and Networks, vol. 3, no. 2, pp. 101–111, 2017.

[7] E. Jayanthi and A. L. Vallikannu, "A Review on Workload Characteristics for Multi Core Embedded Architectures using Machine Learning and Deep Learning Techniques," in 2021 Sixth International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET), Mar. 2021, pp. 456–461.

[8] B. Hanindhito, B. Patel, and L. K. John, "Bandwidth characterization of deepspeed on distributed large language model training," in 2024 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), May 2024, pp. 241–256.

[9] F. M. Rohrhofer, S. Posch, C. Gößnitzer, and B. C. Geiger, "Data vs. physics: The apparent Pareto front of physics-informed neural networks," IEEE Access, vol. 11, pp. 86252–86261, 2023.

[10] S. Sanchez et al., "Design and implementation of a scalable HPC monitoring system," in 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), May 2016, pp. 1721–1725.

[11] M. Tuhacek and P. Svoboda, "Quality of project documentation," in IOP Conference Series: Materials Science and Engineering, vol. 471, no. 5, p. 052012, Feb. 2019.

[12] P. Vadde, M. H. S. Mohammed, O. Mayekar, and N. R. Shanigaram, "Investigating Computational Bottlenecks in MRI Pre-Processing Pipelines," in 2025 International Conference on Intelligent Control, Computing and Communications (IC3), Feb. 2025, pp. 353–359.

[13] M. A. Vieira et al., "Fast packet processing with eBPF and XDP: Concepts, code, challenges, and applications," ACM Computing Surveys (CSUR), vol. 53, no. 1, pp. 1–36, 2020.

[14] S. Meng and L. Liu, "Enhanced monitoring-as-a-service for effective cloud management," IEEE Transactions on Computers, vol. 62, no. 9, pp. 1705–1720, 2012.

[15] B. Burns, J. Beda, K. Hightower, and L. Evenson, Kubernetes: Up and Running: Dive into the Future of Infrastructure, O'Reilly Media, Inc., 2022.

[16] J. K. R. Matsoui, M. Fernandes, L. Wanner, and S. Rigo, "Integração de sensores de baixo-nível com TensorBoard," in Escola Regional de Alto Desempenho de São Paulo (ERAD-SP), May 2021, pp. 5–8.

[17] J. Mu et al., "A history-based auto-tuning framework for fast and high-performance DNN design on GPU," in 2020 57th ACM/IEEE Design Automation Conference (DAC), Jul. 2020, pp. 1–6.

[18] M. Langer, Z. He, W. Rahayu, and Y. Xue, "Distributed training of deep learning models: A taxonomic perspective," IEEE Transactions on Parallel and Distributed Systems, vol. 31, no. 12, pp. 2802–2818, 2020.

[19] P. Von Behren, "NVML: implementing persistent memory applications," in Proceedings of the 13th USENIX Conference on File and Storage Technologies (FAST), 2015.

[20] E. Stevens, L. Antiga, and T. Viehmann, Deep Learning with PyTorch: Build, Train, and Tune Neural Networks Using Python Tools, Manning, 2020.

[21] A. L. V. Solórzano, A Practical Evaluation of Parallel and Distributed Deep Learning Frameworks, 2022.

[22] H. Dai et al., "Reveal training performance mystery between TensorFlow and PyTorch in the single GPU environment," Science China Information Sciences, vol. 65, no. 1, p. 112103, 2022.

[23] M. Abadi et al., "TensorFlow: a system for large-scale machine learning," in 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), 2016, pp. 265–283.

[24] M. Liang et al., "Fine-grained trace-driven performance modeling and simulation for large-scale ML training," in Machine Learning for Computer Architecture and Systems 2024, 2024.

[25] D. Gomez Blanco, "Protocol and Collector," in Practical OpenTelemetry: Adopting Open Observability Standards Across Your Organization, Berkeley, CA: Apress, 2023, pp. 157–178.

[26] J. Turnbull, Monitoring with Prometheus, Turnbull Press, 2018.

[27] P. Moritz et al., "Ray: A distributed framework for emerging AI applications," in 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18), 2018, pp. 561–577.

[28] C. Georgiou, N. Nicolaou, and A. Trigeorgi, "Tracing the latencies of Ares: A DSM case study," in Proceedings of the 2024 Workshop on Advanced Tools, Programming Languages, and Platforms for Implementing and Evaluating Algorithms for Distributed Systems, Jun. 2024, pp. 1–6.

[29] Z. Ye et al., "Deep learning workload scheduling in GPU datacenters: A survey," ACM Computing Surveys, vol. 56, no. 6, pp. 1–38, 2024.

[30] V. J. Reddi et al., "MLPerf inference benchmark," in 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA), May 2020, pp. 446–459.

[31] D. Ellison, C. Van Buren, X. Jiang, and A. Dholakia, "Benchmarking Large Language Models," in Performance Evaluation and Benchmarking: 15th TPC Technology Conference, TPCTC 2023, Vancouver, BC, Canada, August 28–September 1, 2023, Revised Selected Papers, vol. 14247, Springer Nature, Sep. 2024, p. 77.