Towards A Modular End-To-End Machine Learning Benchmarking Framework

Robert Bayer
IT University of Copenhagen
Denmark
roba@itu.dk

Ties Robroek
IT University of Copenhagen
Denmark
roba@itu.dk

Pınar Tözün
IT University of Copenhagen
Denmark
pito@itu.dk

Abstract

Machine learning (ML) benchmarks are crucial for evaluating the performance, efficiency, and scalability of ML systems, especially as the adoption of complex ML pipelines, such as retrieval-augmented generation (RAG), continues to grow. These pipelines introduce intricate execution graphs that require more advanced benchmarking approaches. Additionally, collocating workloads can improve resource efficiency but may introduce contention challenges that must be carefully managed. Detailed insights into resource utilization are necessary for effective collocation and optimized edge deployments. However, existing benchmarking frameworks often fail to capture these critical aspects.

We introduce a modular end-to-end ML benchmarking framework designed to address these gaps. Our framework emphasizes modularity and reusability by enabling reusable pipeline stages, facilitating flexible benchmarking across diverse ML workflows. It supports complex workloads and measures their end-to-end performance. The workloads can be collocated, with the framework providing insights into resource utilization and contention between the concurrent workloads.

CCS Concepts: • **Computing methodologies** → **Machine learning**; **Concurrent computing methodologies**; • **General and reference** → **Evaluation**.

Keywords: Benchmarking, Deep Learning, Edge Computing

ACM Reference Format:
Robert Bayer, Ties Robroek, and Pınar Tözün. 2025. Towards A Modular End-To-End Machine Learning Benchmarking Framework. In *3rd International Workshop on Testing Distributed Internet of Things Systems (TDIS '25), March 30-April 3, 2025, Rotterdam, Netherlands.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3719159.3721223

TDIS '25, Rotterdam, Netherlands

ACM ISBN 979-8-4007-1526-6/2025/03
https://doi.org/10.1145/3719159.3721223

1 Introduction

The rapid advancements in machine learning have led to an explosion in the diversity of algorithms, models, and system architectures. Evaluating these systems effectively has become critical, necessitating benchmarking tools that provide meaningful insights into their performance, efficiency, and scalability. State-of-the-art benchmarks, however, miss complex use cases and broader essential metrics.

Complex pipelines. The fast evolution of AI has brought us machine learning pipelines that are increasingly complex and rarely provide a straight-line-like execution. They often involve interactions with other systems, dependencies, or loops, prominently showcased by retrieval-augmented generation (RAG). These systems are often composed of simpler repeated operations, such as multiple LLM generation calls, introducing opportunities for code reuse.

Collocation. Many machine learning (ML) systems could benefit significantly from collocation ([21]) by running multiple tasks concurrently to maximize efficiency. This is also relevant at the edge as AI increasingly operates on end-user hardware, which must handle diverse and concurrent workloads. However, no existing benchmarking tools measure the interactions or contention between collocated tasks, leaving a significant gap in evaluating performance under realistic operating conditions.

Alternative Metrics. In addition to these concerns, current benchmarks often disregard resource and power utilization. Even when two devices deliver similar performance levels, one may operate at significantly lower utilization, providing opportunities for collocation. Power efficiency and resource usage are particularly important for edge devices and energy-sensitive applications, where performance per watt is often a critical metric.

In this paper, we address these challenges with our vision of a new framework for benchmarking machine learning systems. This benchmark will follow the following principles:

- Modularity and Reusability: We adopt a modular design to address the inefficiency caused by independently developed use cases. Pipelines are constructed from reusable stages, allowing researchers to focus on a specific part of a system.

- Complex Pipelines: We allow for complex execution graphs and dependencies between stages. Furthermore, the framework supports specifying priorities between inputs, combining inputs from multiple stages, or specifying depths of queues between the stages.
- Comprehensive Metrics for Resource Utilization: The framework integrates with radT [20], supporting collection of software and hardware metrics for both the full pipeline and each of the stages, allowing users to identify resource needs and bottlenecks.
- Support for Concurrent Workloads: The framework provides tools for running collocated workloads using the variety of collocation mechanisms available, offering insights into how systems perform under realistic operating conditions and potential ways of mitigating contention between them.

In the rest of the paper, we discuss related work and our motivation in Section 2, followed by our design in Section 3, and conclude with Section 4.

2 Background and Motivation

Machine Learning Benchmarking. Standardized benchmarking is essential for evaluating machine learning systems' performance, enabling fair comparisons across diverse hardware and software configurations. Several benchmark suites have been developed to assess various aspects of machine learning system performance.

The MLPerf suite [4, 10, 18, 19] has established itself as the most prominent benchmarking suite that provides distinct benchmarks tailored to various hardware scales, ensuring relevance across a broad range of deployment scenarios. However, this approach is not without drawbacks. Developing benchmarks with focused use cases for specific hardware ranges creates gaps, which are not covered by any specific benchmark, leading to underutilized devices under test, as resource requirements of larger benchmarks exceed the amount available on these devices. For example, small edge devices like Google Coral [9] often fall into gaps between these benchmarks, leaving their performance inadequately measured and their full potential untapped. A significant limitation of existing tools is the lack of modularity. Use cases are often developed independently, which hinders their adaptability to new scenarios and slows the evolution of benchmarking methodologies. Researchers frequently find themselves constrained by predefined use cases or forced to create their own training and inference scripts, which can duplicate efforts and provide inadequate comparison points for systems.

EEMBC MLMark [25] addresses the gap between the different scales of MLPerf benchmarks, evaluating inference on non-phone small edge devices. The benchmark targets vision tasks but has not been updated since 2020.

TPCxAI [7], in contrast, provides an end-to-end evaluation of classical machine learning workloads, including loading and preprocessing of data, which many benchmarks omit. Similar to prior TPC benchmarks, TPCxAI supports scaling the workload, enabling use on different scales of hardware. However, even the smallest scaling factor remains too large for small edge devices [6]. Besides measuring the end-to-end performance, the benchmark decomposes the execution of use cases into separate stages, allowing for independent tracking of their throughput and latency and showing possible contention when running concurrent workloads. Our benchmarking framework borrows these ideas and applies them to deep-learning-specific workloads. We build upon these ideas by leveraging the decomposition of the pipelines into stages not only for the attribution of metrics but also to aid code reuse. This feature is especially useful with the introduction of compound AI systems [13–15], which use simpler modules such as LLM inference and reuse them multiple times to compose their complex underlying pipelines. These stages are further parametrizable, which allows researchers working on specific optimizations to perform comparative studies of their system against the state of the art on a wide variety of use cases, with clearly defined boundaries of optimizations. Besides stage-specific latency and throughput, our framework gives a better overview of the impact the specific stages have on resource utilization and contention.

Hardware Monitoring. While large-scale deep learning is ever-increasing in popularity, there are plenty of use cases where resources are underutilized [11]. GPUs are the prime accelerators for deep learning training and maximizing their utilization is key to efficient use of the hardware. A single training pipeline may not be sufficient to use the accelerator fully, as, for example, the model trained or the dataset may not be large enough to utilize all resources [5, 12, 21, 22, 26]. This underutilization poses a key problem for both small setups, where practitioners want to get the most out of their setup, and on the data center scale, where such inefficiencies build up.

The machine learning ecosystem includes a range of systems for tracking ML workloads and metric collection, including MLOps platforms like MLFlow [31], WandB [1], PolyAxon [17], and Neptune [23], and live monitoring tools such as TensorBoard [2]. These systems share a background focused on collecting model performance metrics for finding the best model configuration, with just barebones support for hardware monitoring. Considering the impact hardware has on machine learning pipelines, systems need to be future-proofed by providing hardware monitoring support as a mainstay feature, not as an afterthought.

While GPU manufacturers such as Nvidia provide tools for monitoring their hardware, there are challenges in both the collection and interpretation of these metrics [8, 20, 21, 28–30]. We defer our metric collection to radT [20], a tool that

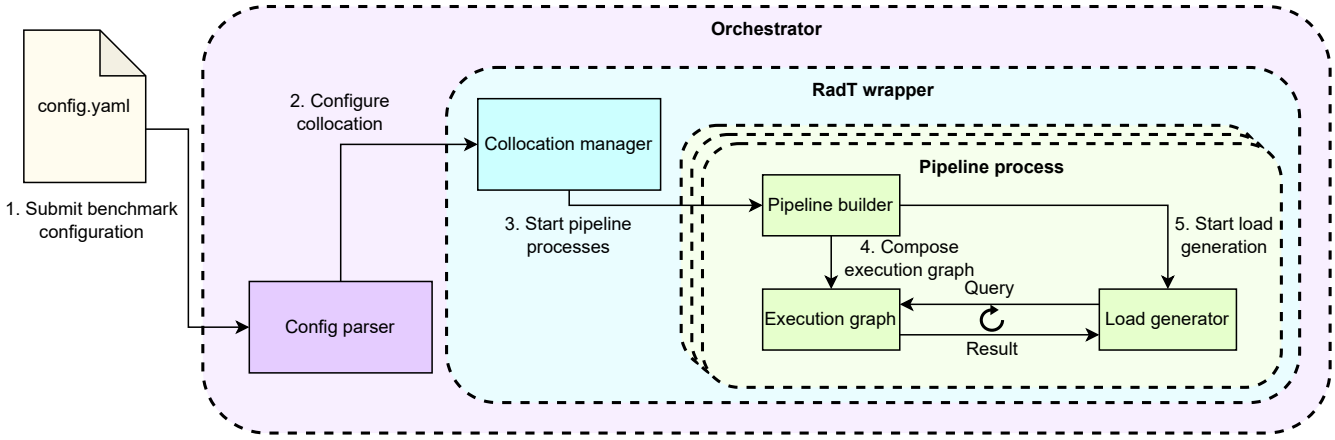

Figure 1. System architecture of the benchmarking framework, showing the interactions between its components.

expands on MLFlow [31] by providing rigorous support for software and hardware metric collection. This allows us to collect, process, and visualize all artifacts and data from pipeline runs in a single place. In particular, radT has dedicated support for metric collection under workload collocation, which further aligns with our requirements.

3 Modular ML Benchmarking Framework

Overview of End-to-End ML Training. Figure 1 visualizes the design of the benchmarking framework and the interactions between its components. The framework consists of the orchestrator, responsible for orchestrating the concurrent pipeline runs based on user-defined benchmark configurations. After submission (1.) and subsequent parsing of the configurations, the orchestrator starts a new process for each of the pipelines using the radT wrapper by providing a collocation configuration (2.). radT starts (3.) and registers the pipelines in MLFlow, initializes the data collection, and sets up the workload collocation mechanisms when applicable. Each of the pipelines is then responsible for building its own execution graph (4.) and load generator before starting the load generation (5.) and processing of the generated queries.

Modular Architecture. The benchmark runs are configured through YAML files, which modify the execution of the benchmark and all of its components. At the root of a benchmark run is the benchmark itself with all of the concurrently running machine learning pipelines, as shown in an excerpt of an example configuration file in Figure 2. For each of these pipelines, users configure the load generation behaviour and the structure of the execution graph. The load generator is responsible for accepting an input dataset of the pipeline and generating workload by sampling from the dataset at specific time intervals. Users can choose between synchronous or asynchronous load generation, with planned support for replay of traces. The execution graphs of the pipelines are

```yaml
name: "Mixed benchmark with training"
pipelines:
  - name: EfficientNetv2 S Imagenette inference
    inputs: [0]
    outputs: [2]
    loadgen:
      type: poisson
      max_queries: 2000
      timeout: 20000
      queue_depth: 100
      config:
        rate: 15 # average #requests/sec
    stages:
      - name: Imagenette dataset
        id: 0
        outputs: [1]
        type: dataset
        module_name: torchvision_dataset
        config:
          dataset_name: Imagenette # must be title-case
          split: [val]
          #preload: True
          batch_size: 1
          ...
```

Figure 2. Head of a .yaml file that defines a pipeline. Our modular approach allows for rapid construction and reuse.

built from available stages. In order to accommodate the use of complex ML systems such as RAG, the execution graphs support forks and cycles. To achieve this, each of the stages runs in a separate thread and accepts a variable number of inputs. Incoming requests are stored in queues, which can be polled separately based on priority or combined to merge incoming requests.

Use Cases. Supporting our design goals of modularity and reusability (Section 2), we include a breadth of use cases that reflects the current deep learning landscape. New pipelines can reuse modules from these implementations.

Image classification has been one of the leading areas in deep learning research. Our implementation of TorchVision [16] allows the users to train 20 model families, based on CNN and transformer architectures, on all 38 classification

datasets available in TorchVision. TorchTune [24] allows users to use LoRa-based fine-tuning methods on 13 LLM model families and datasets available in the library. Furthermore, users can again use the newly trained or pre-trained model for inference.

HuggingFace [27] is a staple in LLM fine-tuning and inference. We include this library as an LLM alternative to TorchTune with the same feature set but a broadened set of models and datasets.

Finally, we include Self-RAG [3] to provide an example of a modern complex machine learning use case. This use case presents a more complex pipeline than the rest, having multiple LLM generation calls, branches, and loops. The Self-RAG use case currently supports knowledge graph and SQL database retrieval, including LLM generation as a fallback.

4 Conclusion

We introduce a benchmarking framework for modern machine learning pipelines that promotes modularity, reuse of pipeline stages, and a high decree in supporting novel use cases. Furthermore, we identify the ever-increasing importance of resource utilization, providing immediate support for wide metric collection and collocation of workloads. Our benchmarking framework is easily adaptable while also supporting a wide range of use cases out-of-the-box. This way, we provide a unique solution that can handle increasingly complex modern pipelines while also minimizing friction in the user experience.

Acknowledgments

The work is funded by the Novo Nordisk Foundation Natural and Technical Sciences program under grant agreement number NNF22OC0079398.

References

[1] 2020. Weights and Biases. https://wandb.ai/site
[2] Martın Abadi. 2016. TensorFlow: A system for large-scale machine learning. (2016), 21.
[3] Akari Asai, et al. 2023. Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection.
[4] Colby Banbury, et al. 2021. MLPerf Tiny Benchmark. arXiv:2106.07597 [cs]
[5] Sebastian Baunsgaard, et al. 2020. Training for Speech Recognition on Coprocessors. *arXiv:2003.12366* (March 2020).
[6] Robert Bayer, et al. 2023. TPCx-AI on NVIDIA Jetsons. In *Performance Evaluation and Benchmarking*. Vol. 13860. Springer Nature Switzerland, Cham, 49–66.
[7] Christoph Brücke, et al. 2023. TPCx-AI - An Industry Standard Benchmark for Artificial Intelligence and Machine Learning Systems. *Proceedings of the VLDB Endowment* 16, 12 (Aug. 2023), 3649–3661.
[8] Paul Elvinger, et al. 2025. Measuring GPU utilization one level deeper.
[9] IEEE Spectrum. 2019. The Coral Dev Board Takes Google's AI to the Edge. *IEEE Spectrum* (2019). https://spectrum.ieee.org/the-coral-dev-board-takes-googles-ai-to-the-edge
[10] Vijay Janapa Reddi, et al. 2022. Mlperf Mobile Inference Benchmark: An Industry-Standard Open-Source Machine Learning Benchmark for on-Device Ai. *Proceedings of MLSys* 4 (2022), 352–369.
[11] Myeongjae Jeon, et al. 2019. Analysis of Large-Scale Multi-Tenant GPU Clusters for DNN Training Workloads. In *Proceedings of the 2019 USENIX Annual Technical Conference, USENIX ATC 2019, Renton, WA, USA, July 10-12, 2019*. USENIX Association, 947–960.
[12] Alexandros Koliousis, et al. 2019. Crossbow: scaling deep learning with small batch sizes on multi-GPU servers. *Proceedings of the VLDB Endowment* 12, 11 (July 2019), 1399–1412.
[13] Patrick Lewis, et al. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Systems*, Vol. 33. Curran Associates, Inc., 9459–9474.
[14] Xinzhe Li. 2025. A Review of Prominent Paradigms for LLM-Based Agents: Tool Use, Planning (Including RAG), and Feedback Learning. In *Proceedings of the 31st International Conference on Computational Linguistics*. Association for Computational Linguistics, Abu Dhabi, UAE, 9760–9779.
[15] Samuel Madden, et al. 2024. Databases Unbound: Querying All of the World's Bytes with AI. *Proc. VLDB Endow.* 17, 12 (Aug. 2024), 4546–4554.
[16] Sébastien Marcel. 2010. Torchvision the machine-vision package of torch. In *Proceedings of the 18th ACM international conference on Multimedia*. 1485–1488.
[17] Mourad Mourafiq. [n. d.]. Polyaxon: Cloud native machine learning platform. https://github.com/polyaxon/polyaxon
[18] Vijay Janapa Reddi, et al. 2020. MLPerf Inference Benchmark. *arXiv:1911.02549 [cs, stat]* (May 2020). arXiv:1911.02549 [cs, stat]
[19] Vijay Janapa Reddi, et al. 2020. MLPerf Mobile Inference Benchmark: Why Mobile AI Benchmarking Is Hard and What to Do about It. *arXiv:2012.02328* (2020). arXiv:2012.02328
[20] Ties Robroek, et al. 2023. Data Management and Visualization for Benchmarking Deep Learning Training Systems. In *Proceedings of the Seventh Workshop on Data Management for End-to-End Machine Learning*. ACM, Seattle WA USA, 1–5.
[21] Ties Robroek, et al. 2024. An Analysis of Collocation on GPUs for Deep Learning Training. In *Proceedings of the 4th Workshop on Machine Learning and Systems*. ACM, Athens Greece, 81–90.
[22] Foteini Strati, et al. 2024. Orion: Interference-aware, Fine-grained GPU Sharing for ML Applications. In *Proceedings of the Nineteenth European Conference on Computer Systems, EuroSys 2024, Athens, Greece, April 22-25, 2024*. ACM, 1075–1092.
[23] Neptune Team. 2019. *neptune.ai*. Technical Report. https://neptune.ai/
[24] torchtune maintainers. 2024. *torchtune: PyTorch's finetuning library*. https//github.com/pytorch/torchtune
[25] Peter Torelli. 2021. Measuring Inference Performance of Machine-Learning Frameworks on Edge-Class Devices with the Mlmark Benchmark. *Techincal Report. Available online: https://www.eembc.org/techlit/articles/MLMARK-WHITEPAPERFINAL-1.pdf* (2021).
[26] Shang Wang, et al. 2021. Horizontally Fused Training Array: An Effective Hardware Utilization Squeezer for Training Novel Deep Learning Models. *arXiv:2102.02344 [cs]* (March 2021).
[27] Thomas Wolf, et al. 2020. HuggingFace's transformers: State-of-the-art natural language processing.
[28] Zeyu Yang, et al. 2024. Part-time Power Measurements: nvidia-smi's Lack of Attention.
[29] Gingfung Yeung, et al. 2020. Towards GPU utilization prediction for cloud deep learning. In *12th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 20)*.
[30] Ehsan Yousefzadeh-Asl-Miandoab, et al. 2023. Profiling and Monitoring Deep Learning Training Tasks. In *Proceedings of the 3rd Workshop on Machine Learning and Systems*. ACM, Rome Italy, 18–25.
[31] Matei Zaharia, et al. 2018. Accelerating the machine learning lifecycle with MLflow. *IEEE Data Eng. Bull.* 41, 4 (2018), 39–45.

Received 7th February 2025