The Cost of Teaching Operational ML

Fraida Fund
ffund@nyu.edu
New York University
Brooklyn, New York, USA

Kate Keahey
keahey@mcs.anl.gov
Argonne National Laboratory
Lemont, Illinois, USA

Cody Hammock
hammock@tacc.utexas.edu
Texas Advanced Computing Center
Austin, Texas, USA

Marc Richardson
mtrichardson@uchicago.edu
University of Chicago
Chicago, Illinois, USA

Mark Powers
markpowers@uchicago.edu
University of Chicago
Chicago, Illinois, USA

Michael Sherman
shermanm@uchicago.edu
University of Chicago
Chicago, Illinois, USA

Abstract

Operational machine learning (ML) requires skills beyond model development, including infrastructure provisioning, large-scale training across clusters, model deployment with consideration of operational performance, monitoring, and automation - capabilities grounded in high-performance computing and distributed systems. This paper presents the design and infrastructure requirements of a graduate-level course on ML Systems Engineering and Operations, aimed at equipping students with these skills. Using 186,692 total compute instance hours on the Chameleon Cloud testbed, students built end-to-end ML pipelines incorporating distributed training, reproducible experiment tracking, automated re-training and re-deployment, and continuous monitoring. We analyze compute usage across assignments, compare expected versus actual resource consumption, and estimate that replicating the course on commercial cloud platforms would cost approximately \$250 per student (almost \$50,000 for our course with enrollment of 191 students). All course materials are publicly available for reuse[1].

CCS Concepts

• Social and professional topics → Computing education; **• Computing methodologies → Machine learning**; **• Networks → Cloud computing**.

Keywords

MLOps, machine learning operations, DevOps, education

ACM Reference Format:
Fraida Fund, Kate Keahey, Cody Hammock, Marc Richardson, Mark Powers, and Michael Sherman. 2025. The Cost of Teaching Operational ML. In *Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC Workshops '25), November 16–21, 2025, St Louis, MO, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3731599.3767385

1 Introduction

The rapid expansion of machine learning (ML) across industries has created a growing demand for engineers who can support ML-specific infrastructure and operations. Companies of all sizes - startups, mid-sized firms, and large enterprises - are hiring for roles such as AI infrastructure engineer, MLOps engineer, and ML cluster performance engineer. In parallel, academic research in machine learning systems (MLSys) is expanding to address the full-stack challenges introduced by modern ML development and deployment. These challenges include designing software and hardware systems that support the ML lifecycle, optimizing for constraints beyond accuracy (e.g., cost, latency, privacy, interpretability), and building infrastructure that enables scalable, reliable, and accessible ML applications and services [27]. Universities are beginning to address this demand by introducing courses focused on MLSys, ML operations (MLOps), or software engineering for ML [15, 20, 21, 24, 33]. These courses extend the traditional ML curriculum by incorporating topics central to high performance computing and systems engineering, and train students to reason about broader system-level concerns, such as computational efficiency, long-term maintainability, and sustainability, in the context of ML applications.

Several early experience reports in the academic literature discuss courses on operational machine learning systems [15, 20, 21, 24, 33], highlighting pedagogical approaches, relevant tools, and curricular structure. However, none provide a concrete, in-depth treatment of the substantial *infrastructure* requirements of such a course. This paper addresses that gap by focusing specifically on the infrastructure demands of a graduate-level Machine Learning Systems Engineering and Operations course offered at NYU. The goal of the course is to integrate core machine learning concepts with systems topics such as distributed training, scheduling, monitoring, and versioning, alongside fundamentals of cloud computing, so that students will be capable of provisioning, deploying, maintaining, and using the infrastructure and systems required for operational ML. We provide a detailed analysis of infrastructure usage in our 191-student course, including approximately **109,837 hours** of compute instance time for guided lab assignments and **76,855 hours** for open-ended project work. We estimate that running the course on a commercial cloud would have cost just under \$50,000. Finally, we make all materials available [11] to replicate our course on the public Chameleon Cloud [17] research infrastructure.

[1]Course materials: https://ffund.github.io/ml-sys-ops/

By detailing the compute, network, and storage requirements to teach these topics, this paper contributes to the broader objective of preparing students in compute-intensive fields like machine learning and data science with practical, scalable computing skills aligned with industry and research needs.

The rest of this paper is organized as follows. Section 2 provides an overview of the course, including its intended audience, prerequisites, and core learning objectives. Section 3 details the learning activities in the course and their corresponding infrastructure requirements. Section 4 describes Chameleon Cloud as the primary platform for supporting course activities. Section 5 summarizes the compute, storage, and network resources used by students at each stage, along with estimated costs for running the course on commercial cloud platforms. Finally, Section 6 suggests key takeaways for instructors who are interested in offering a similar course.

2 Course Overview

Machine Learning Systems Engineering and Operations was offered for the first time in Spring 2025 in the Department of Electrical and Computer Engineering at NYU, with a final enrollment of 191 students. This section describes the course's structure, goals, and content, as well as the motivation behind its design.

Learning Objectives. By the end of the course, students should understand the full lifecycle of machine learning systems and have experience designing, building, and maintaining them. Specifically, students should be able to:

- implement scalable training workflows, including training very large models, multi-GPU training, and cluster management,
- manage experiment tracking and versioning,
- deploy performant models, accounting for compute requirements and latency alongside ML metrics such as accuracy,
- evaluate and monitor deployed systems, including both prediction quality and compute resource usage,
- apply DevOps principles such as CI/CD, infrastructure as code, and cloud-native computing, to ML systems, and
- reason about system reliability, maintainability, and risk.

Audience. Most participants were enrolled in the Master of Science program in Electrical Engineering or Computer Engineering. A small number came from other departments such as Computer Science, Data Science, and Mathematical Sciences.

Relationship to other courses. The only formal prerequisite for the course was Introduction to Machine Learning, in order to make it broadly accessible. However, the course complements other offerings at NYU, including advanced ML courses such as Deep Learning and Introduction to High Performance Machine Learning, as well as systems-focused courses like Internet Architecture and Protocols and Data Center and Cloud Computing. While prior or concurrent enrollment in these courses is beneficial, it is not required. This course offers students a big-picture perspective and hands-on experience in integrating concepts across these areas to build production-grade ML systems, without duplicating the content of those other courses offered within the department.

Lectures. Students attended a weekly in-person lecture that introduced each unit and provided context for the problems and solutions in that week's lab assignment. Some weeks also included external readings, case studies, or video materials to supplement.

Lab Assignments. Each lecture was followed by a hands-on lab assignment. Students were advised to allocate 4–5 hours per week for this work. Detailed step-by-step instructions were provided for each assignment. Labs were graded primarily for completion: students submitted screenshots demonstrating that they had successfully deployed the relevant systems following the instructions, accounting for 60% of the final course grade.

Projects. The remaining 40% of the course grade was based on final projects submitted at the end of the semester. Students worked in groups of three or four to design and implement a medium- to large-scale machine learning system. Projects were expected to apply the techniques introduced throughout the course to address practical challenges in ML systems engineering and operations.

Certain components of the project - such as the problem formulation, value proposition, data selection, and system integration - were shared responsibilities among all group members. Other aspects were considered individually. Each student was expected to take ownership of a specific subsystem, such as training, serving, monitoring, data pipeline development, or continuous integration and deployment. The specific expectations for each role were aligned with topics covered in the course, with requirements enumerated and mapped to the corresponding instructional units.

Timeline. The course spanned 14 weeks, with most instructional content delivered in the first 10, followed by project work.

Human support infrastructure. Each week included one office hour with the instructor and separate office hours with two course assistants. An online Q&A forum complemented these sessions, with over 700 discussion threads and more than 3,000 unique posts - questions, answers, and comments - by the end of the semester.

3 Content and Infrastructure Requirements

This section describes the core components of the course, organized around weekly lectures and lab assignments and culminating in a large-scale project. For each unit, we summarize the lecture topic, describe the corresponding lab activity, and outline the compute, network, and storage requirements necessary to support it. We also report the expected duration of each lab assignment. These estimates were derived by timing a course assistant as they completed the assignment for the first time, then adding a $\approx 50\%$ buffer.

3.1 Unit 1: Introduction to ML Systems

Lecture Content. The first lecture distinguished between prototype ML models and production ML systems, emphasizing challenges related to scale, data quality, and business alignment. We introduced the concept of technical debt in ML systems [32], and the observation that common failure modes in large-scale ML infrastructure are often just the common failure modes of large-scale distributed systems [25]. In contrast to introductory ML courses, where the model is the central object of engineering, in operational systems the central object is the ML *pipeline*, setting the foundation for the system-level topics addressed throughout the semester.

Lab Activity. The first lab focused on onboarding students to the infrastructure used throughout, Chameleon Cloud [17]. Students created accounts and completed a guided tutorial introducing them to key features of the platform, including provisioning a virtual machine (VM) instance and accessing it via SSH.

Infrastructure Requirements. This lab requires a single minimal VM instance, and an IP address for SSH access. The estimated time required to complete the assignment was 1-2 hours.

3.2 Unit 2: Cloud Computing

Lecture Content. This lecture focused on the cloud infrastructure that modern machine learning systems run on. It introduced key building blocks of a cloud environment, including compute, network, and storage services; shared services like authentication and image management; and interfaces such as GUIs, CLIs, and SDKs. Students learned how responsibility across the stack - hardware, runtime, application logic - shifts between user and provider in service models like IaaS, PaaS, and SaaS. The lecture concluded with a discussion of virtualization and how tools like containers and orchestration frameworks enable scalable service management.

Lab Activity. In this lab, students explored core elements of cloud infrastructure using Chameleon, which is an OpenStack-based research cloud. They first practiced using the OpenStack GUI to provision VM instances, networks, ports, and floating IPs. This "ClickOps" experience motivated the next part, in which they used the OpenStack CLI to perform the same tasks more efficiently. With compute infrastructure in place, students deployed a simple ML application in a Docker container. Finally, they installed Kubernetes using Kubespray and deployed their containerized application using replicas, load balancing, and horizontal scaling.

This deployment introduced the premise that would run throughout the course: students take on the role of ML engineers at a fictional startup, GourmetGram, developing a food-focused photo-sharing platform. In this lab, they deploy a food classification model that assigns tags to photos uploaded to GourmetGram.

Infrastructure Requirements. This lab used three virtual machines, each with 2 vCPUs and 4 GB of RAM. Network requirements included a provider-configured external network and a user-provisioned internal network for inter-VM communication. One publicly routable IP address was provisioned to enable SSH access into the cluster and test the user interface of the service. The estimated time required for the assignment was 5 hours.

3.3 Unit 3: DevOps for ML Systems

Lecture Content. This lecture reviewed core DevOps principles, including continuous integration and delivery (CI/CD), version control, and infrastructure as code. The session also introduced cloud-native computing as a convergence of DevOps and cloud infrastructure. However, while DevOps provides robust tools for managing code and infrastructure, it is less equipped to manage two critical components of ML systems: models and data. This gap motivates the need for MLOps, which extends DevOps principles to the full ML lifecycle. We introduced a practical MLOps framework and lifecycle, along with key organizational capabilities needed to support ML system deployment at scale [30].

Lab Activity. Building on the previous lab, where students deployed a basic ML service using manual steps, in this lab students introduced automation and lifecycle management practices in the GourmetGram context. To reduce manual overhead and improve reproducibility, students used Infrastructure-as-Code (IaC) and Configuration-as-Code (CaC) tools - Terraform to provision infrastructure and Ansible to install Kubernetes and supporting tools. They then used Argo CD to declaratively manage the deployment of GourmetGram's platform components, and to deploy GourmetGram's staging, canary, and production services.

Then, students built a simplified ML pipeline using Argo Workflows, triggered manually with dummy steps to simulate the model lifecycle, including model registration and promotion.

Infrastructure Requirements. This assignment has exactly the same compute requirements as the previous one: students provisioned three virtual machines, each with 2 vCPUs and 4 GB of RAM, an internal network for inter-VM communication, and a publicly routable IP address to enable SSH access into the cluster and to test the user interface of the service. However, we note that to achieve the learning objective, it was important that students provision infrastructure using "standard" IaC tools - Terraform in this case - so the infrastructure was required to support this. The estimated hands-on time required to complete the assignment was 5 hours, but students were advised to block out some time in the middle while Kubernetes is installed (unattended), so from an infrastructure perspective, the expected duration was 7-8 hours.

3.4 Unit 4: Model Training at Scale

Lecture Content. Given the importance of large language models at the current time and the challenges associated with training them, this lecture introduced practical strategies for training machine learning models with billions of parameters, focusing on techniques that enable models to scale beyond the memory limitations of a single GPU (gradient accumulation, reduced and mixed-precision arithmetic, and parameter-efficient fine-tuning methods such as LoRA [14] and QLoRA [7]). The lecture also covered paradigms for distributed training across GPUs: distributed data parallelism [22], fully sharded data parallelism [37], and model parallelism. The ring all-reduce communication pattern [26], which was first introduced in an HPC context and then later applied to efficient gradient aggregation for distributed training [12], was covered in detail.

Lab Activity. Students fine-tuned a 13B LLM using PyTorch Lightning, first on a single GPU to explore memory optimizations, then across 4 GPUs using distributed training techniques.

Infrastructure Requirements. The "Single GPU" portion of the lab required access to a GPU with NVIDIA CUDA compute capability 8.0 or higher to support `bfloat16` reduced precision training, as well as substantial GPU memory (e.g., an A100 80GB GPU). The "Multiple GPU" portion required access to a node with at least four such GPUs. For both parts, students also provisioned a public IP for SSH access and access to a Jupyter environment deployed on the instance. Each section was designed to take approximately two hours of hands-on time, including setup time.

3.5 Unit 5: Model Training Infrastructure

Lecture Content. This lecture continued the focus on model training, with an emphasis on infrastructure and platforms for training. Using OPT-175B [36] and Alibaba [34] as case studies, we identified key requirements: the ability to store detailed records for every run, monitor ML-specific metrics, track infrastructure-level metrics, and swap hardware while jobs are running. We then introduced job scheduling and placement concepts from HPC, e.g., backfilling, gang scheduling, and fair sharing, specifically for ML training jobs.

Lab Activity. Students deployed an MLFlow tracking server [6, 35], including all necessary services (backend store, artifact store, UI) using Docker containers. They configured a training script to log experiment metadata, system metrics, hyperparameters, ML metrics, and models to MLFlow, and used its UI to identify training bottlenecks, compare experiment results, and inspect model artifacts. Then, students deployed a Ray [23] training cluster. They learned how to define resource requirements for training jobs, modify a training script to integrate Ray Train for distributed execution and fault tolerance, and use Ray Tune for hyperparameter search.

Infrastructure Requirements. The first part, on experiment tracking, required a single GPU instance, with sufficient vCPU and RAM to supply data to the training process. The second part, involving distributed training across a multi-GPU cluster, required two GPUs. Students also provisioned a public IP for SSH access, access to a Jupyter environment deployed on the instance, and access to the MLFlow and Ray web-based UIs. Each part was estimated to require approximately 3 hours of infrastructure time.

3.6 Unit 6: Model Serving

Lecture Content. This lecture introduced model serving, focusing on tradeoffs between latency, throughput, cost, and accuracy in both cloud and edge environments. Students examined model-level optimizations (compact architectures, graph compilation, operator fusion, quantization, pruning, distillation) and system-level strategies (concurrent execution, dynamic batching) for inference.

Lab Activity. Continuing the GourmetGram use case introduced in earlier units, students were tasked with preparing multiple model serving configurations that balance cost, latency, disk space and throughput under tight performance budgets.

First, students applied model-level optimizations using ONNX Runtime including graph optimizations, INT8 quantization, and use of hardware-specific execution providers, on server-grade hardware. Next, students benchmarked the models in a low-resource environment typical of mobile/edge use cases. Finally, students explored system-level optimizations using NVIDIA Triton Inference Server, including concurrency, dynamic batching, and scaling across multiple GPUs or multiple model instances.

Infrastructure Requirements. For the three parts of the lab, student required a 3-hour block on a GPU-enabled instance with recent CUDA support (e.g., A100, A30); then a 2-hour session on a low-resource edge device; and finally, a 3-hour block on an instance with 2 GPUs. For all parts, students also provisioned a public IP for SSH access, access to a Jupyter environment deployed on the instance, and access to the web-based UI for the service.

3.7 Unit 7: Monitoring and Evaluation

Lecture Content. This lecture focused on the evaluation and monitoring stages of the ML system lifecycle, as discussed in [3]. Students learned about different offline evaluation modalities beyond general ML metrics (e.g., loss, accuracy), including domain-specific metrics (e.g., BLEU, ROUGE), operational metrics (e.g., inference latency, throughput, retraining cost), evaluation across key population slices, assessments for fairness and bias, and behavioral testing [28]. The lecture also covered online evaluation methods such as shadow testing, canary deployments, and A/B testing, along with the challenges of prediction monitoring in production - particularly the difficulty of detecting performance degradation due to data drift when ground truth labels are not readily available.

Lab Activity. This lab focused on evaluating and monitoring ML systems across their lifecycle stages, organized in three parts. For offline evaluation, students evaluated models using domain-specific metrics and explainability tools for sanity checks, applied template-based unit tests to ensure behavioral robustness, evaluated performance on key data slices and known failure modes, and assembled a unified test suite. In the online evaluation phase, students implemented live monitoring of operational metrics (e.g., latency, throughput) and model-specific metrics (e.g., output distribution), along with drift detection. Finally, students explored strategies for collecting supervision signals in production settings, using both "real users" and dedicated human annotators.

Infrastructure Requirements. This lab required a single VM instance with 2 vCPUs and 4 GB of RAM. Students provisioned the instance with a public IP for access to SSH, a Jupyter environment deployed on the instance, web-based UIs for the evaluation and monitoring platforms, and the GourmetGram service. The total expected duration was 6 hours.

3.8 Unit 8: Data Systems

Lecture Content. This lecture introduced the types of data storage systems used in ML pipelines: relational databases, data warehouses, document and columnar databases, data lakes, and data lakehouses. The lecture also covered ETL (extract, transform, load) pipelines for batch data and the broker–producer–consumer model for streaming data. Feature stores are introduced as infrastructure that unifies batch and streaming sources for use in ML training and inference.

Lab Activity. This lab introduced students to persistent storage options for managing training data and system state across the ML lifecycle. Students worked with two types of storage:

- Block Storage: Students provisioned a block storage volume, attached it to a VM, formatted and mounted it, and used it to persist service data across ephemeral compute environments.
- Object Storage: Students provisioned object storage to persist large training datasets. They created object store buckets, loaded data into them, and mounted the object store as a filesystem on compute instances, reducing setup overhead.

Infrastructure Requirements. This lab required a VM instance with 4 vCPUs and 8 GB of RAM, a 2 GB block storage volume, and approximately 1.2 GB of object storage. Students provisioned the instance with a public IP for access to SSH. The estimated time required to complete the assignment was 3 hours.

3.9 Unit 9: Safeguarding ML Systems

Lecture Content. This lecture introduced students to some of the risks posed by ML systems, and strategies to safeguard against them across design, implementation, and production stages. Students examined categories of harm - including bias and fairness issues, privacy violations, harmful content, and overreliance - and discussed how these risks vary with the model type and use case. The lecture introduced mitigation strategies such as red-teaming, filtering, RLHF, onboarding practices, transparency measures, and cognitive forcing functions, as well as their limitations.

Lab Activity. To accommodate project work, there was no lab.

3.10 Unit 10: Commercial Clouds

Lecture Content. The final lecture was a demo of the long-running GourmetGram example, as it might be deployed on Google Cloud Platform (GCP). The intent of this lesson was to show how to transfer skills and ideas from the infrastructure and platforms used in the course, to other contexts. It also included a demo of platform-managed Kubernetes and serverless functions.

Lab Activity. Instructions to replicate the demo were made available as an optional lab activity.

Infrastructure Requirements. The 2-hour demo included a VM instance, a managed Kubernetes cluster, a serverless service, a managed notebook environment with GPU acceleration, and storage for data and artifacts (e.g. container images).

3.11 Project

The final project required students to design and implement a medium-scale machine learning system that integrated techniques from across the course. Projects were completed in groups of 3-4 students. Each group jointly defined the use case, dataset, target variable, and overall system architecture. Individual members took ownership of specific subsystems: model training (Units 4-5), model serving and monitoring (Units 6-7), data storage and pipelines (Unit 8), and, for four-person groups, continuous integration and deployment (Unit 3). (In three-person groups, CI/CD responsibilities were shared.) Infrastructure requirements varied according to the details of the project, with some groups requiring extremely large-scale data processing capabilities or extended time on multi-GPU nodes for training, and others having less intensive requirements.

4 Chameleon Cloud

To support the infrastructure requirements of Units 1-9 and the course projects, we used the Chameleon Cloud testbed [17]. This section describe the capabilities of Chameleon in support of our educational goals, in contrast to other platforms we considered.

Consideration of other platforms. Traditional HPC platforms were not suitable, because the learning objectives of this course emphasize full ML system design, including provisioning and managing infrastructure and software systems from scratch. These goals require full infrastructure control, which is not possible in the batch- or notebook-based environments of a traditional HPC setting.

Commercial clouds offer flexibility, access to large-scale compute resources, and a wide range of managed services, but pose challenges in a large-scale course setting due to cost and billing complexity. Some require a credit card to be attached to every project, in which case students risk charges if they unintentionally exceed the education or free tier offerings of the platform. Even when no credit card is required and education credits are provided, students risk exceeding the credit limit early in the course - potentially leaving them unable to complete later assignments. These uncertainties made commercial platforms less suitable for our instructional goals.

Other research testbeds (e.g., FABRIC [2], CloudLab [9]) provide networking, storage, and compute capabilities (including GPU), and allow students to work without the risk of incurring charges, but they use a specialized interface for access, rather than one that is compatible with mainstream “cloud” tools. Given the course's emphasis on infrastructure provisioning and management, we considered it a significant advantage for students to use standard tools.

Capabilities of Chameleon Cloud. Chameleon is built on the open-source OpenStack platform, which closely mirrors the architecture and tooling of many commercial cloud providers. It supports multiple methods for provisioning resources, including the OpenStack command-line interface, a Python API, a browser-based GUI (Horizon), and Terraform. This diversity of interfaces allowed students to interact with the infrastructure using widely adopted, industry-relevant tools.

Chameleon provides a rich set of compute, network, and storage capabilities to support the hands-on ML systems and infrastructure work we had envisioned. Students could launch on-demand VM instances or reserve bare-metal GPU nodes for training workloads. The platform supports public and private network configuration, block storage for persistent volumes, and object storage for large datasets. Its capabilities are well-used and battle-tested in an education setting, having been used to teach high performance computing [8, 29, 31], cloud computing [4, 13], machine learning [5], and autonomous driving [1, 10], in classroom settings as well as less traditional education and training settings.

Some learning activities specifically require less-resourced devices. Unique among the platforms we considered, Chameleon also includes the CHI@Edge edge testbed [19], which offers access to Raspberry Pi and NVIDIA Jetson devices and supports a “bring your own device” (BYOD) model.

Logistics for classroom use. Chameleon Cloud is primarily a research infrastructure, and our large-scale education use case involved some additional logistical arrangements.

First, we requested a quota increase for the KVM@TACC site so that our project could simultaneously use:

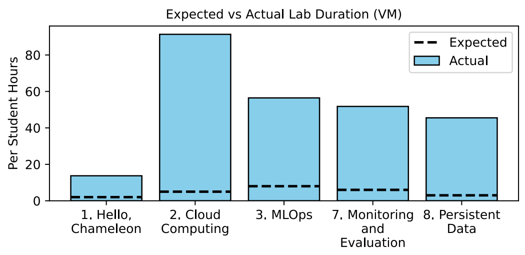
- **Compute**: 600 VM instances, 1200 cores, and 2.5 TB RAM
- **Network**: unlimited private networks and subnets, 200 routers, 300 floating IPs, and 100 security groups
- **Storage**: 200 block storage volumes, 10 TB total block storage

For bare metal sites, the default quotas were sufficient.

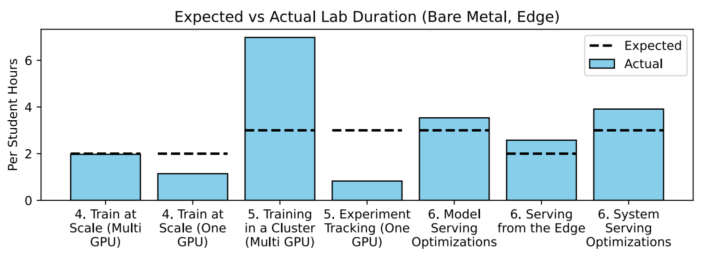
We also made advance arrangements for access to scarce in-demand GPU resources. Course staff reserved [18] specific bare-metal GPU instances for week-long blocks aligned with the course schedule. At the start of each reservation, Chameleon staff temporarily restricted access to these resources to our project. Students could then reserve short (2-hour or 3-hour) time slots on these GPU instances without having to contend with other Chameleon users.

Finally, in preparation for our unit on deploying machine learning models on low-resource devices, we added 7 of our own Raspberry Pi 5 devices with ARM Cortex A76 processor to the CHI@Edge platform, enabling our students to use them via Chameleon.

Use of a commercial cloud. To practice using commercial clouds, we distributed Google Cloud Platform education credits, which allowed students to use GCP without having to associate a credit card. Because this platform was only used for the last assignment, and it was optional, we were not concerned about students exceeding the usage permitted by the educational credit.



(a) VM Instances



(b) Bare Metal and Edge Instances

Figure 1: Comparison of expected vs. actual duration of infrastructure usage per student, per lab assignment. Fig. (a) shows labs conducted on general-purpose VM instances, which do not require an advance reservation and are not terminated automatically; (b) shows bare metal and edge environments, which require an advance reservation. Dashed lines indicate expected duration.

Table 1: Usage and estimated cost overall (and per student) by lab assignment and Chameleon node type or VM flavor.

Assignment	Instance Type	Instance Hours	Floating IP Hours	AWS Cost	GCP Cost
1. Hello, Chameleon	`m1.small`	2,620	2,620	$40 ($0.21)	$57 ($0.3)
2. Cloud Computing	`m1.medium` (x3)	52,332	17,444	$2,264 ($12)	$5,347 ($28)
3. MLOps	`m1.medium` (x3)	32,344	10,781	$1,399 ($7.3)	$3,305 ($17)
4. Train at Scale (Multi GPU)	`gpu_a100_pcie`	167	167	$2,993 ($16)	$2,456 ($13)
	`gpu_v100`	210	210	$3,764 ($20)	$3,088 ($16)
4. Train at Scale (One GPU)	`compute_gigaio`	218	218	$722 ($3.8)	$1,106 ($5.8)
5. Training in a Cluster (Multi GPU)	`compute_liqid_2`	330	330	$1,524 ($8)	$662 ($3.5)
	`gpu_mi100`	1,002	1,002	$4,627 ($24)	$2,009 ($11)
5. Experiment Tracking (One GPU)	`compute_gigaio`	28	28	$41 ($0.21)	$32 ($0.17)
	`compute_liqid`	130	130	$190 ($0.99)	$150 ($0.78)
6. Model Serving Optimizations	`compute_gigaio`	215	215	$191 ($1)	$154 ($0.81)
	`compute_liqid`	460	460	$410 ($2.1)	$329 ($1.7)
6. Serving from the Edge	`raspberrypi5`	492	492	NA	NA
6. System Serving Optimizations	`gpu_p100`	707	707	$3,582 ($19)	$1,417 ($7.4)
7. Monitoring and Evaluation	`m1.medium`	9,889	9,889	$461 ($2.4)	$381 ($2)
8. Persistent Data	`m1.large`	8,693	8,693	$1,490 ($7.8)	$626 ($3.3)
Total		**109,837**	**53,387**	**$23,698 ($124)**	**$21,119 ($111)**

5 Infrastructure usage and cost

During the course, we monitored the actual usage at VM and bare metal sites, and the Chameleon team provided reservation data from bare metal and edge sites. Using the course timeline and the naming conventions specified in the lab instructions, we were able to associate most individual compute instances with specific lab assignments, and therefore gain insight into student's usage of the platform. (The data in this section is based on actual usage of VM and bare metal instances, and reserved usage of edge instances.)

Actual usage per assignment. Fig. 1 shows the actual duration of infrastructure usage for each lab assignment detailed in Section 3, normalized by the number of students in the course. For bare metal and edge instances, which require an advance reservation and which automatically terminate the instance at the end of the reservation, the actual usage closely tracks expected usage. In the case of Unit 4 and Unit 5, students could optionally complete the single-GPU part on the same instance used for the multi-GPU part, thereby saving on instance creation time; this explains their higher-than-expected usage for multi-GPU instances along with lower-than-expected usage for single-GPU instances.

VM instances, however, often persisted beyond expected durations - sometimes intentionally (to avoid repeating lengthy setup), other times due to neglect. This raises a potential concern if a similar course was conducted in a commercial cloud environment, where non-terminated instances could lead to unexpectedly high costs. (Since the initial offering of this course, Chameleon has introduced advance reservation for VM instances as well, with automatic termination at the end of the reservation.)

Overall, the portion of the course with hands-on lab assignments used a total of **109,837** hours of compute instance time. Table 1 further enumerates the usage per assignment.

Estimated cost of lab assignments on commercial clouds. To quantify the infrastructure cost of this course, we translated the resources consumed on the Chameleon testbed into their equivalent costs on commercial cloud platforms, specifically Amazon Web Services (AWS) and Google Cloud Platform (GCP). For this analysis, an "equivalent" resource was defined as the most cost-effective cloud instance that met the specific needs of each assignment.

The cost model is built on several key assumptions. All prices are derived from the on-demand, per-hour rates listed in the official

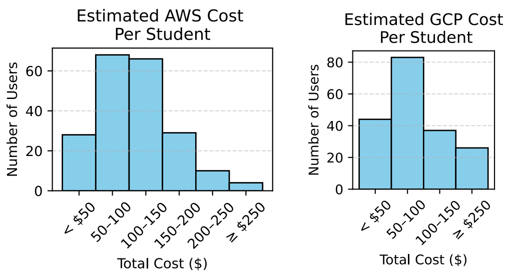


Figure 2: Distribution of estimated cost per student to execute lab assignments on commercial clouds.

public pricing calculators for AWS and GCP as of July 2025 for a single region (`us-central1` for GCP and `us-east-1` for AWS). These figures likely represent a conservative estimate, as short-duration usage can sometimes incur higher effective hourly rates. Furthermore, while students used bare metal nodes on Chameleon for some coursework, the course assignments do not strictly require them. Our analysis therefore assumes standard virtualized instances, as the performance monitoring exercises in the curriculum are not sensitive enough to be significantly impacted by potential "noisy neighbors." Finally, the total cost also includes charges for networking services (floating IPs). (For lab assignments, we do not include storage costs, which are negligible; these will be significant for project work, however.) It is also worth noting that commercial cloud pricing is volatile. Providers frequently adjust prices based on hardware availability, regional demand, and broader market forces. This analysis is a snapshot based on July 2025 pricing.

As illustrated in Table 1, the average cost per student for the guided lab assignments is $124 on AWS, or $111 on GCP. (The "Serving from the Edge" experiment is excluded from this calculation, since no commercial clouds offer Raspberry Pi devices.)

However, this average cost hides a "long tail" of students whose usage would be much more costly, with the "most expensive" student's usage for lab assignments estimated at $665 on AWS, or $590 on GCP. Compared to the "expected" cost for lab assignments based on the durations listed in Section 3 ($79.80 on AWS, $58.85 on GCP), 75% of students would have exceeded this cost on AWS, and 73% would have exceeded this cost on GCP. The distribution of estimated cost per student is illustrated in Fig. 2.

Actual usage and estimated cost for open-ended projects. The rest of the infrastructure usage we logged is attributed to student projects. Over the course of approximately one and a half months of project development, students used:

- **70,259** hours of VM instance time without GPU acceleration,
- **5,446** hours of compute time on instances with GPUs,
- **975** hours of compute time on bare metal instances without a GPU, most often for large-scale data processing pipelines,
- **175** hours on low-resource edge devices,
- **9 TB** of block storage volumes,
- and **1,541 GB** of object storage.

Fig. 3 further breaks down this usage by instance type, for the 70,259 hours of VM instances and 5,446 hours of GPU instances.

The cost estimate for this usage is less precise, since each project has its own requirements and so we cannot exactly identify lowest-equivalent-cost commercial cloud instances. Based on conservative assumptions, we estimate a cost of **$25,889** on AWS (approximately $136 per student) and a similar cost of **$26,218** (approximately $137 per student) on GCP for these open-ended projects.

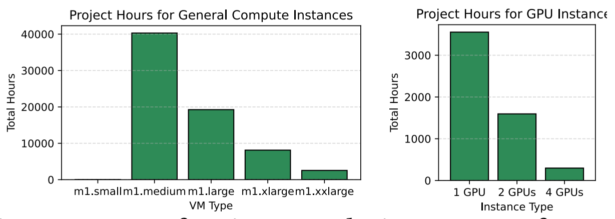


Figure 3: Hours of project usage by instance type for non-GPU and GPU instances.

6 Takeaways

Our key takeaways from this experience are as follows:

Teaching operational ML involves very different infrastructure capabilities than teaching ML. Traditional ML education focuses on one-off model development - training and evaluating a single model on static data - which aligns well with environments like Jupyter notebooks or batch-processing systems on traditional HPC platforms. In an operationalized system, however, models must be continuously retrained and redeployed in response to data drift, quality degradation, or new business requirements. Teaching this workflow requires exposing students to: provisioning infrastructure, automating pipelines, managing data systems, deploying and monitoring services, and implementing feedback loops. These capabilities demand full infrastructure control across compute, network, and storage layers, which notebook-based or batch-only environments are not designed to support.

Teaching operational ML is expensive (on commercial infrastructure!). The hands-on nature of operational ML coursework demands substantial compute resources, especially for GPU-accelerated training, multi-node clusters, and long-running experiments. On commercial platforms, a single instance left running unintentionally can accumulate substantial charges, and our usage strongly suggests a high likelihood of this occurring in an environment where instances are not terminated automatically. The estimated cost per student to run our course on a commercial cloud is in the $250 range (including lab assignments and open-ended project work), but there is a long tail of high-usage students with substantially higher costs, that makes commercial clouds risky and potentially cost-prohibitive for an education setting.

Course materials for operational ML on public research infrastructure can be replicated, reused, and adapted. Finally, we note that by designing lab assignments around open-source tools, and using public research testbeds like Chameleon Cloud, we created materials that are easily reused. All of the lab materials developed for this course are available under an open-source license [11], and can be executed directly from the Chameleon Trovi [16] artifact sharing platform.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant Nos. 2230079, 2230077, and 2431425. We thank Chameleon Cloud for providing computing resources for the class and reservation data, and we thank Google Cloud for Education for providing education credits for GCP.

References

[1] Richard Anderson. 2022. *Driving Autonomous Cars From Edge to Cloud with CHI@Edge.* https://chameleoncloud.org/blog/2022/12/19/driving-autonomous-cars-from-edge-to-cloud-with-chiedge/

[2] Ilya Baldin, Anita Nikolich, James Griffioen, Indermohan Inder S Monga, Kuang-Ching Wang, Tom Lehman, and Paul Ruth. 2020. Fabric: A national-scale programmable experimental network infrastructure. *IEEE Internet Computing* 23, 6 (2020), 38–47.

[3] E. Breck, S. Cai, E. Nielsen, M. Salib, and D. Sculley. 2017. The ML test score: A rubric for ML production readiness and technical debt reduction. In *2017 IEEE International Conference on Big Data (Big Data)*. IEEE, Boston, MA, USA, 1123–1132. doi:10.1109/BigData.2017.8258038

[4] Massimo Canonico. 2023. *Presentation for mini-symposium on education using Chameleon.* Video available in program listing at: https://chameleoncloud.org/chameleon-cloud-users-meeting/user-meeting-2023/.

[5] Priyanka Bose Chandra Shekhar Pandey. 2023. *Reproduce, Rerun, Repeat: The Fun Way to Learn Machine Learning!* https://www.chameleoncloud.org/blog/2023/03/28/reproduce-rerun-repeat-the-fun-way-to-learn-machine-learning/

[6] Andrew Chen, Andy Chow, Aaron Davidson, Arjun DCunha, Ali Ghodsi, Sue Ann Hong, Andy Konwinski, Clemens Mewald, Siddharth Murching, Tomas Nykodym, Paul Ogilvie, Mani Parkhe, Avesh Singh, Fen Xie, Matei Zaharia, Richard Zang, Juntai Zheng, and Corey Zumar. 2020. Developments in MLflow: A System to Accelerate the Machine Learning Lifecycle. In *Proceedings of the 3rd International Workshop on Data Management for End-to-End Machine Learning (DEEM).*

[7] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLORA: efficient finetuning of quantized LLMs. In *Proceedings of the 37th International Conference on Neural Information Processing Systems.*

[8] Jose Manuel Monsalve Diaz. 2023. *Using Chameleon for HPC Education: an OpenMP Tutorial.* https://www.chameleoncloud.org/blog/2023/05/30/jose-monsalve-education-user-blog-post-interview/

[9] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. 2019. The Design and Operation of CloudLab. In *Proceedings of the USENIX Annual Technical Conference (Usenix ATC '19').*

[10] Alicia Esquivel Morel, William Fowler, Kate Keahey, Kyle Zheng, Michael Sherman, and Richard Anderson. 2023. AutoLearn: Learning in the Edge to Cloud Continuum. In *Proceedings of the SC '23 Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis* (Denver, CO, USA) *(SC-W '23).* Association for Computing Machinery, New York, NY, USA, 350–356. doi:10.1145/3624062.3624101

[11] Fraida Fund. 2025. Machine Learning Systems Engineering and Operations. https://ffund.github.io/ml-sys-ops/. Accessed: 2025-08-01.

[12] Andrew Gibiansky. 2017. Bringing HPC Techniques to Deep Learning. https://andrew.gibiansky.com/blog/machine-learning/baidu-allreduce/ Originally published on Baidu Research technical blog.

[13] Aniruddha Gokhale. 2023. *Educating with Chameleon at Vanderbilt.* https://www.chameleoncloud.org/blog/2023/07/17/educating-with-chameleon-at-vanderbilt/

[14] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-Rank Adaptation of Large Language Models. In *International Conference on Learning Representations (ICLR).*

[15] Christian Kästner and Eunsuk Kang. 2020. Teaching software engineering for AI-enabled systems. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering Education and Training.* 45–48.

[16] Kate Keahey, Jason Anderson, Matt Powers, and Alex Cooper. 2023. Three Pillars of Practical Reproducibility. In *Proceedings of the Reproducible Workflows, Data Management, and Security (ReWorDS 23) at eScience'23.* doi:10.1109/e-Science58273.2023.10254846

[17] Kate Keahey, Jason Anderson, Zhuo Zhen, Pierre Riteau, Paul Ruth, Dan Stanzione, Mert Cevik, Jacob Colleran, Haryadi S. Gunawi, Cody Hammock, Joe Mambretti, Alexander Barnes, François Halbach, Alex Rocha, and Joe Stubbs. 2020. Lessons Learned from the Chameleon Testbed. In *Proceedings of the 2020 USENIX Annual Technical Conference (USENIX ATC '20).*

[18] Kate Keahey, Pierre Riteau, Jason Anderson, and Zhuo Zhen. 2019. Managing Allocatable Resources. In *Proceedings of the IEEE 2019 International Conference on Cloud Computing (CLOUD 2019).*

[19] Kate Keahey, Michael Sherman, Jason Anderson, and Mark Powers. 2025. CHI@Edge: Supporting Experimentation in the Edge to Cloud Continuum. In *Practice and Experience in Advanced Research Computing 2025: The Power of Collaboration (PEARC '25).* doi:10.1145/3708035.3736014

[20] Filippo Lanubile, Silverio Martínez-Fernández, and Luigi Quaranta. 2023. Teaching MLOps in Higher Education through Project-Based Learning. In *2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET).* 95–100.

[21] Filippo Lanubile, Silverio Martínez-Fernández, and Luigi Quaranta. 2024. Training Future Machine Learning Engineers: A Project-Based Course on MLOps. *IEEE Software* 41, 2 (2024), 60–67. doi:10.1109/MS.2023.3310768

[22] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and Soumith Chintala. 2020. PyTorch Distributed: Experiences on Accelerating Data Parallel Training. *Proc. VLDB Endow.* 13, 12 (2020), 3005–3018. doi:10.14778/3415478.3415530

[23] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, and Ion Stoica. 2018. Ray: A Distributed Framework for Emerging AI Applications. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI '18).* 561–577.

[24] Fabio Palomba, Giuseppe Voria, Antonio Parziale, Vincenzo Pentangelo, Antonio Della Porta, Valerio De Martino, Giuseppe Recupito, and Gennaro Giordano. 2025. Teaching Software Engineering for Artificial Intelligence: An Experience Report. In *51st Euromicro Conference Series on Software Engineering and Advanced Applications (SEAA).* Euromicro, Salerno, Italy. Experience Report.

[25] Daniel Papasian and Todd Underwood. 2020. How ML Breaks: A Decade of Outages for One Large ML Pipeline. In *Proceedings of the USENIX Conference on Operational Machine Learning (OpML).*

[26] Pitch Patarasuk and Xin Yuan. 2009. Bandwidth Optimal All-Reduce Algorithms for Clusters of Workstations. *J. Parallel and Distrib. Comput.* 69, 2 (2009), 117–124. doi:10.1016/j.jpdc.2008.09.002

[27] Alexander Ratner, Dan Alistarh, Gustavo Alonso, David G. Andersen, Peter Bailis, Sarah Bird, Nicholas Carlini, Bryan Catanzaro, Jennifer Chayes, Eric Chung, Bill Dally, Jeff Dean, Inderjit S. Dhillon, Alexandros Dimakis, Pradeep Dubey, Charles Elkan, Grigori Fursin, Gregory R. Ganger, Lise Getoor, Phillip B. Gibbons, Garth A. Gibson, Joseph E. Gonzalez, Justin Gottschlich, Song Han, Kim Hazelwood, Furong Huang, Martin Jaggi, Kevin Jamieson, Michael I. Jordan, Gauri Joshi, Rania Khalaf, Jason Knight, Jakub Konečný, Tim Kraska, Arun Kumar, Anastasios Kyrillidis, Aparna Lakshmiratan, Jing Li, Samuel Madden, H. Brendan McMahan, Erik Meijer, Ioannis Mitliagkas, Rajat Monga, Derek Murray, Kunle Olukotun, Dimitris Papailiopoulos, Gennady Pekhimenko, Theodoros Rekatsinas, Afshin Rostamizadeh, Christopher Ré, Christopher De Sa, Hanie Sedghi, Siddhartha Sen, Virginia Smith, Alex Smola, Dawn Song, Evan Sparks, Ion Stoica, Vivienne Sze, Madeleine Udell, Joaquin Vanschoren, Shivaram Venkataraman, Rashmi Vinayak, Markus Weimer, Andrew Gordon Wilson, Eric Xing, Matei Zaharia, Ce Zhang, and Ameet Talwalkar. 2019. MLSys: The New Frontier of Machine Learning Systems. arXiv:1904.03257 [cs.LG] https://arxiv.org/abs/1904.03257

[28] Marco Tulio Ribeiro, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. 2020. Beyond Accuracy: Behavioral Testing of NLP Models with CheckList. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics.*

[29] John Rieffel. 2023. *Presentation for mini-symposium on education using Chameleon.* Video available in program listing at: https://chameleoncloud.org/chameleon-cloud-users-meeting/user-meeting-2023/.

[30] Khalid Salama, Jarek Kazmierczak, and Donna Schut. 2021. *Practitioners Guide to MLOps: A Framework for Continuous Delivery and Automation of Machine Learning.* Technical Report. Google Cloud. https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning

[31] Darshan Sarojini and Aroua Gharbi. 2022. IndySCC: A New Student Cluster Competition That Broadens Participation. In *Practice and Experience in Advanced Research Computing 2022: Revolutionary: Computing, Connections, You* (Boston, MA, USA) *(PEARC '22).* Article 72, 4 pages. doi:10.1145/3491418.3535153

[32] D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. 2015. Hidden Technical Debt in Machine Learning Systems. In *Advances in Neural Information Processing Systems 28 (NeurIPS).* MIT Press, 2503–2511.

[33] Andrey Sozykin, Evgeniy Kuklin, and Irina Iumanova. 2022. Teaching Advanced AI Development Techniques with a New Master's Program in Artificial Intelligence Engineering. In *Supercomputing.* Springer, 562–573.

[34] Qizhen Weng, Wencong Xiao, Yinghao Yu, Wei Wang, Cheng Wang, Jian He, Yong Li, Liping Zhang, Wei Lin, and Yu Ding. 2022. MLaaS in the Wild: Workload Analysis and Scheduling in Large-Scale Heterogeneous GPU Clusters. In *Proceedings of the 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI '22).*

[35] Matei Zaharia, Andrew Chen, Aaron Davidson, Ali Ghodsi, Sue Ann Hong, Andy Konwinski, Siddharth Murching, Tomas Nykodym, Paul Ogilvie, Mani Parkhe, Fen Xie, and Corey Zumar. 2018. Accelerating the Machine Learning Lifecycle with MLflow. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* (2018).

[36] Susan Zhang. 2023. Trials of developing OPT-175B. Stanford MLSys Seminar, Episode 77, YouTube. https://www.youtube.com/watch?v=p9IxoSkvZ-M

[37] Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, Alban Desmaison, Can Balioglu, Pritam Damania, Bernard Nguyen, Geeta Chauhan, Yuchen Hao, Ajit Mathews, and Shen Li. 2023. PyTorch FSDP: Experiences on Scaling Fully Sharded Data Parallel. *Proc. VLDB Endow.* 16, 12 (2023), 3848–3860. arXiv:2306.16768 doi:10.14778/3611540.3611569