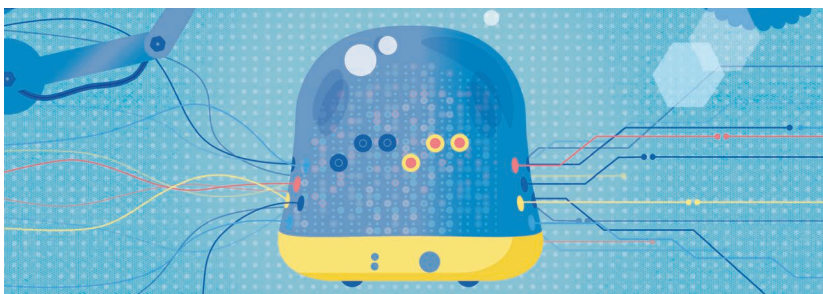


Establishing Machine Learning Operations for Continual Learning in Computing Clusters

A Framework for Monitoring and Optimizing Cluster Behavior

Diana McSpadden¹, Mark Jones¹, Ahmed Hossam Mohammed, Bryan Hess¹, and Malachi Schram, Thomas Jefferson National Accelerator Facility



// We focus on building a machine learning (ML) operations continual learning capability to support an ML research and development project at Jefferson Lab. We describe a composable ML workflow, a custom CGroupV2 exporter, and the implementation of Prometheus, MLFlow, and Grafana. //

THOMAS JEFFERSON NATIONAL Accelerator Facility (Jefferson Lab) is a U.S. Department of Energy Office of Science National Laboratory. Scientists worldwide utilize the lab's particle accelerator to probe the basic building blocks of matter and rely on its high-performance computing resources to process large amounts of collected data.

Jefferson Lab has funded a research and development project to gain insight on the jobs performed on the high-performance computing cluster. Jefferson Lab's computing cluster primarily processes nuclear physics experimental data, including particle event reconstruction and analysis. Multicore jobs are the norm, and for highly parallelizable tasks, multinode message passing interface (MPI) jobs are supported. The overall workload profile changes continually due to experiment-specific software from a base of users that overlaps and changes routinely.

The R&D project was motivated to improve computing operations and develop machine learning operations (MLOps) to support multiple scientific applications and domains. Informally, we call our project and resulting system the *Digital Data Center Twin (DIDACT)* where an ML-based model would facilitate diagnostic and optimization tools. The success criterion for the DIDACT project is to demonstrate the ability to correctly diagnose sources of unexpected cluster behavior, such as memory fragmentation or input/output (I/O)

overcommitment for a particular compute node. Additionally, an embedding that captures the salient features of diverse architectures and evolving jobs may be able to predict an incoming job's impact on the cluster, or be used to understand the power-to-complete-time Pareto front.

Central to this ongoing project is an MLOps capability that supports the continual learning (CL) goals, particularly monitoring and comparing the models as the behavior of the computing cluster evolves (see “[DIDACT'S R&D Approach to](#)

[Continual Learning](#)”). Consequently, a parallel thrust of the project focuses on establishing MLOps capabilities. This collaboration between the Data Science and Computing Operations teams presents an opportunity to develop a capability to improve efficiency and operational cost for computing facilities. This is particularly important, as computing facilities have grown in complexity (e.g., more heterogeneous hardware and workflow configurations). We provide details of our MLOps implementation, which includes modular and composable code, model version



DIDACT'S R&D APPROACH TO CONTINUAL LEARNING

Continual learning (CL) refers to a model's ability to utilize data that arrives incrementally.^{S1} A challenge in CL is integrating new information and learning new tasks without forgetting previously acquired information, a phenomenon known as *catastrophic forgetting*.^{S2} Artificial intelligence-augmented compute center decision control must adapt to the constant evolution of workloads on the cluster, and CL may help address these concerns.

Google's machine learning operations (MLOps) maturity framework^{S3} outlines three levels of maturity: level-0, which involves manual processes; level-1, with partial automation; and level-2, which demonstrates complete automation across various stages, including unit testing, continuous code integration, automated model training, continuous model delivery, and monitoring. In this context, we demonstrate level-1 maturity, focusing on the automation of the CL pipeline, model delivery, model comparison, and monitoring.

The Digital Data Center Twin (DIDACT) project implements CL on a configurable cycle, set to 24 h. The specific time interval for CL is determined by production requirements. Factors like retraining costs, the rate of change of the variables and multivariate distributions, and practical limitations for the MLOps implementation influence

the selection of the interval or triggered retraining. This approach, designed for normal cluster operation unless anomalies are flagged by Operations, assumes downstream tasks benefit from daily retraining unless paused manually or due to detected anomalies.

In the initial phase, DIDACT's 24-h cycle lacks a trigger based on model performance. This passive CL approach anticipates gradual compute cluster evolution and allows us to study the impact of adding the new data to the model over time during this R&D phase.

References

- S1. G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter, “Continual learning, lifelong learning, catastrophic forgetting, developmental systems, memory consolidation,” *Neural Netw.*, vol. 113, pp. 54–71, May 2019, doi: [10.1016/j.neunet.2019.01.012](#).
- S2. R. M. French, “Catastrophic forgetting in connectionist networks,” *Trends Cogn. Sci.*, vol. 3, no. 4, pp. 128–135, 1999, doi: [10.1016/S1364-6613\(99\)01294-2](#).
- S3. “MLOps: Continuous delivery and automation pipelines in machine learning.” Google Cloud. Accessed: Feb. 16, 2024. [Online]. Available: <https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning>

tracking, a model repository, model serving, and monitoring.

The MLOps Components of the DIDACT Project

This section provides an overview of the platforms, Node and CGroupV2 Exporters and composable ML workflow code we utilize within our on-premises MLOps framework for analyzing production cluster behavior (detailed in Table 1). (Additional details on the selection and implementation of the third-party, open source tools used can be found at in the supplementary downloadable material available at <https://doi.org/10.1109/MS.2024.3424256>, provided by the authors.)

Modeling a Computing Cluster

We propose ML approaches to acquire a representation of single and multithreaded batch jobs and diverse compute requests across six distinct hardware configurations currently used in the production and R&D sandbox clusters. Typically, the

production cluster operates within expected parameters, with rare occurrences of anomalies necessitating staff intervention. However, as the computing cluster scales, the potential impact of issues, such as misconfigurations, or resource contention grows exponentially, and will affect an increasing number of compute nodes. Reporting potential anomalies in the early stages is essential, as they could easily spread and deteriorate the status of the entire cluster.

Our goal is to capture the system dynamics during active jobs by encoding the CPU and memory metrics into a compressed embedding. Within the DIDACT project, the model development arc focuses on exploring different model architectures and compares their ability to capture the system behavior over time. The hypothesis is that single-threaded jobs may not require graph learning between nodes, while multithreaded jobs may require such an architecture. ML methods employed to learn the system include an autoencoder, a variational

autoencoder, and an autoencoder utilizing graph neural network (GNN) layers. Previously, Lu et al.¹ have examined an attention-based GNN using the CPU and memory features as nodes for anomaly detection on the Jefferson Lab production cluster.

A representative encoder could then be used for anomaly detection or by an agent to control the cluster in a fully automated setup. The process of training multiple models and comparing them is done on a daily basis to determine the new champion model that will replace the previously hosted model. Monitoring the changing performance of all candidate models after daily updates is critical. Our MLOps implementation ensures that all previous models are archived and available for comparison and roll-back.

Composable Code

Our workflow has three major pipelines: the off-line development pipeline, the CL pipeline, and the real-time inference pipeline. We emphasize the need for writing modules as a composable framework for easy reuse across the pipelines. Keeping track of all configurations, parameters, and metrics, is necessary to generate reproducible results and for comparison. All source code is written in Python and managed in GitHub.

Although the development and CL pipelines have a similar structure, there are subtle differences. The development pipeline tests several ML architectures and performs hyperparameter optimization (HPO). After a set of specific ML models is agreed upon, they undergo daily competition in the CL pipeline where HPO does not occur. Each model is initialized with the previous day's parameters and then trained using the newly saved data. The

Table 1. DIDACT essential functions.

Capability	Tool	Deployment
Software suites		
Database	Prometheus	Third-party
Model repository	MLFlow	Third-party
Model monitoring	MLFlow	Third-party
Cluster monitoring	Grafana	Third-party
Automation	SystemD	Third-party
Executables		
Generic metric exporter	Node Exporter	Third-party
CPU-job metric exporter	CGroupV2-Exporter	Custom-built
Software framework	JLab Composable Code	Custom-built

daily champion model is selected by comparing reconstruction error metrics for features selected in the deployment module's configuration settings, using an orthogonal validation dataset that is separated from the daily training data.

Once a champion model is identified, it is used in the real-time inference pipeline. This pipeline retrieves real-time data using a Prometheus parser. The data are then prepared and provided to the champion model. These predictions are stored on the Prometheus server and visualized for system administrators through Grafana. Figure 1 summarizes the structure of the three pipelines described previously.

Key components of the three pipelines include:

- *Prometheus parser*: A daily service fetches hardware metrics collected in Prometheus from the compute nodes and saves them to disk. This preserves two types of datasets. First, a buffer of datasets is collected for daily training that may be later used to alleviate the problem of catastrophic forgetting in a CL setting. Second, specific datasets are collected from the production cluster and the sandbox

clusters for studies, such as a more detailed analysis of a past anomalous event. This module feeds real-time data to the champion model in the real-time inference pipeline.

- *Disk parser*: This module is used by the development and CL pipelines to load the tabular data saved through the Prometheus parser with the correct datatype.
- *Preparation*: We have three modules that serve three functionalities. First, a module aligns the shifted timestamps among the different time series coming from the compute nodes in the cluster. Second, a module splits the training, validation, and the optional test datasets in time for the development and CL pipelines. This is done by dividing the loaded data into time chunks and randomly assigning each chunk to one of the datasets. Chunking prevents cross-dataset leakage, and the random assignment ensures that all datasets have the same underlying distribution. Third, we convert the data into a format that is usable by the model. Since the data can grow indefinitely, memory-efficient data preparation is important to avoid repetitive reads from the disk during the training. This is done by keeping track of the index of each of the strided sequences in the sorted time series rather than generating unnecessary repetitions.
- *Model*: Although the ML model architecture differs from one experiment to another, this module standardizes the training process, including logging parameters, metrics, and checkpoints.
- *Analysis*: This module evaluates the trained model to produce different metrics and figures, such as regression plots for the different CPU and memory variables.
- *Comparison*: Unlike the analysis module, which is experiment-specific, this module generates plots that allow the comparison of experiment results.
- *Deployment*: The deployment module supports retrieving models by experiment and model version, and MLFlow "model alias." Additionally, the module supports deploying a new model version, comparing models to determine the current model

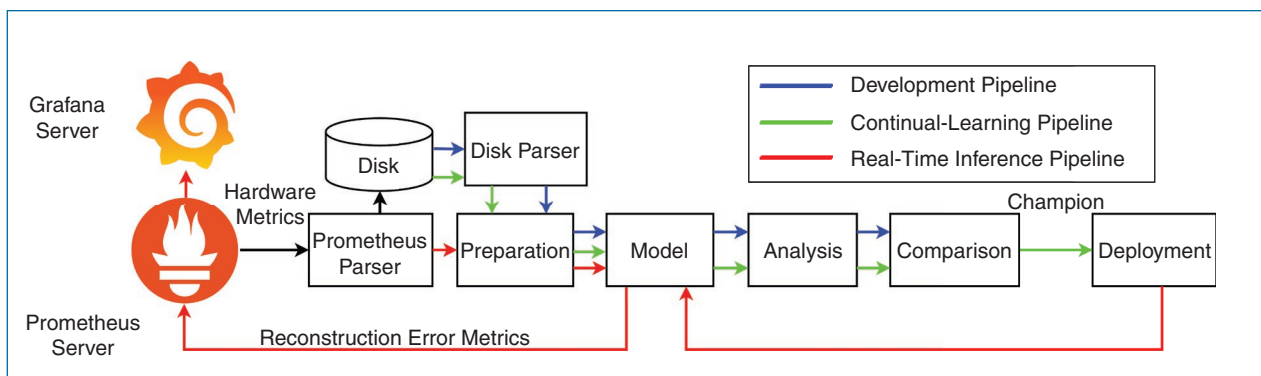


FIGURE 1. The DIDACT pipeline Python code modules, and server and database endpoints. Blue, green, and red arrows represent the development pipeline, the continual-learning pipeline, and the real-time inference pipeline, respectively.

champion, promoting the new model to the current champion model, and archiving previous model versions.

MLFlow

MLFlow (<https://mlflow.org/>) enables efficient model development and management. We leverage MLFlow's Tracking component to log all pipeline configurations as parameters. During training, we log quantitative analysis results as metrics. Relevant artifacts, such as figures, model inputs and outputs, and the serialized model are logged during the analysis of the development and CL pipelines. Additionally, MLFlow's model registry facilitates the systematic management of versions of our models. This feature is instrumental in ensuring the reproducibility of our results and providing insights into the evolution of our models over time. For example, selecting a single champion model is made possible by comparing

metrics saved to MLFlow during the CL pipeline as we select the model that wins most often. The MLFlow implementation is described in the supplementary downloadable material (available at <https://doi.org/10.1109/MS.2024.3424256>) provided by the authors.

Prometheus

Prometheus (<https://prometheus.io/>) is a monitoring tool kit commonly used in data center operations. It ingests time series numeric data that can be queried using a flexible query language. Its modular approach to networking, security, and storage make it a reliable tool for data sharing in an MLOps framework. A sample of the data collected from the production cluster is available from McSpadden et al.²; it includes five distinct hardware configurations, eight CPU, and 11 disk metrics, totaling more than one million records (>180 GB of raw data).

Exporters

Node Exporter (https://github.com/prometheus/node_exporter) is a standard host metrics endpoint that provides an interface to kernel metrics in/proc (i.e., CPU, memory, disk, and networking metrics, and additional data, such as filesystem usage and performance metrics). The Node Exporter is the primary source for instrumenting the load on a Linux system. It lacks certain advanced features, such as mapping SLURM jobs to pinned resource allocations on worker nodes. To address this limitation, we developed the CGroupv2-Exporter. This exporter harvests information about CGroups provisioned by SLURM and SystemD enabling the mapping of Node Exporter metrics to individual SLURM jobs.

Grafana

Grafana (<https://grafana.com/>) is a visualization tool that can be used by multiple backend services, including Prometheus. To benefit this project, a Prometheus exporter that collates the model training results and general runtime health of the training node was placed in production. The data are used to generate a dashboard in Grafana to track the models' health over time. It queries data for visualization, typically from time series sources. Grafana includes interactive filtering features on dashboards, for example, by compute node. An example dashboard is shown in Figure 2.

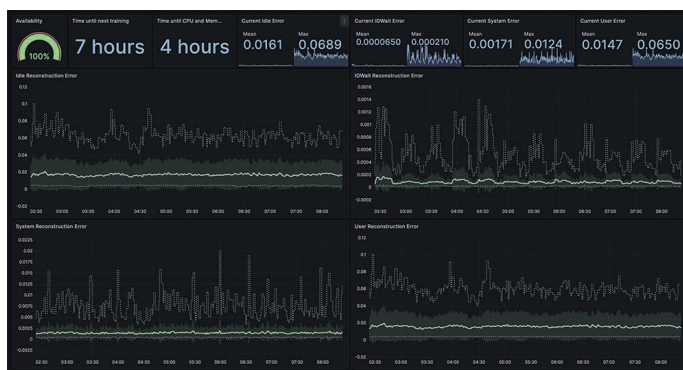


FIGURE 2. The DIDACT Grafana dashboard is used to visualize the model's reconstruction error in real time. The top of the dashboard is general operating information, such as the uptime of the model on the system, time until the next data capture, and model training; the rest relates to the reconstruction error. For our figures, we use the solid line to represent the mean of all errors, the filled area to represent the standard deviation of the errors, and the dotted line to represent both the minimal and maximal errors. For this dashboard, the statistics are for the CPUs on the selected compute node.

Implementing MLOps: Putting the Pieces Together

The third-party, open source tools utilized here are commonly used in similar environments and workflows. For example, Litzinger et al.³

describe the use of Prometheus and Grafana for high-performance computing monitoring. Luo et al.⁴ implemented Prometheus, MLFlow, Grafana, and additionally KubeFlow (<https://www.kubeflow.org/>), KServe (<https://kserve.github.io/website/latest/>), Evidently (<https://www.evidentlyai.com/>), and Iter8 (<https://iter8.tools/>) to prototype a level-2 maturity MLOps pipeline demonstrating continuous integration and continuous delivery using two toy datasets.

We implemented an MLOps process, shown in Figure 3, using a Prometheus database, MLFlow for model tracking and model deployment, and Grafana for online monitoring. Automation is enabled using SystemD services and sockets, and managed pipelines are defined as *SystemD Services*. Prometheus exporters are written to accept sockets handed to the services by SystemD. For daily data archiving and the CL pipeline, timers track the time left before the service is run for the exporter and provide runtime logs in case an error occurs. SystemD sockets provide a mechanism to start the web server, handle a collection of requests from the scraper or other hosts, and allow the service to shut down gracefully when there are no longer any additional requests.

Including 33 memory and eight CPU features, the dataset size for a single day is almost 3.2 GB for both the production and the sandbox clusters. This size is kept relatively low thanks to the preparation module's index tracking, without which the size would grow by a factor of the length of the model input sequence. Daily candidate models are on the scale of 3.1 million to 3.2 million parameters.

In complex systems, such as a high-performance computing cluster, we will observe nonstationary time series data with multiple modalities. Future work will need to investigate the effect of the nonstationary behavior and compare approaches to address the challenges of data drift.⁵ Strategies, such as

managed by MLOps, which will mature in parallel with the ML research. Enhancing our existing MLOps requires a more robust code deployment process. Furthermore, implementing a “push-button” pause of the CL pipeline in case of an operations-identified production anomaly, along with an automatic restart



Our composable workflow
separates the model from the
workflow, promoting reusability.

automated triggering and replay⁶ will be considered. Future work will investigate active CL approaches, e.g., a dynamic trigger based on high reconstruction error compared to a training or holdout dataset.

Prometheus monitors the cluster at 1-min intervals, which provides a challenge to capture higher-resolution dynamics. While increasing resolution improves data detail, it also increases storage and computational demands. Future work can explore this tradeoff.

Our ultimate goal is to model the dynamics across different levels in a computing cluster. We propose a hierarchical GNN-based anomaly detection model. This model captures relationships between components (e.g., compute nodes, shared storage) at different levels (e.g., CPUs versus nodes), enabling pinpointing issues at specific granularities. Our composable workflow separates the model from the workflow, promoting reusability.

Investigation, implementation, and online monitoring must be

once the anomaly is addressed, will improve the current process.

We emphasize the need for high-quality datasets, and in our case jobs, to validate the model and the MLOps pipeline. Example cluster jobs, representing common production issues, such as poorly leveraged resources, were necessary to validate our MLOps pipelines and to verify the ability of the model to distinguish job types in its latent space. We relied on operations' expertise to develop four synthetic jobs: a “bad” I/O job that poorly leverages operating system and network resources, a normal single-node job, a normal MPI-2-node job, and a normal MPI-16-node job. We ran the four jobs on the sandbox cluster, and then applied principle component analysis on the resulting data to visualize the groupings compared to the model's embeddings. A well-performing model should reproduce similar patterns, while anomalies should be identified by groups outside the normal data.

We recommend prioritizing the use of existing tools within the

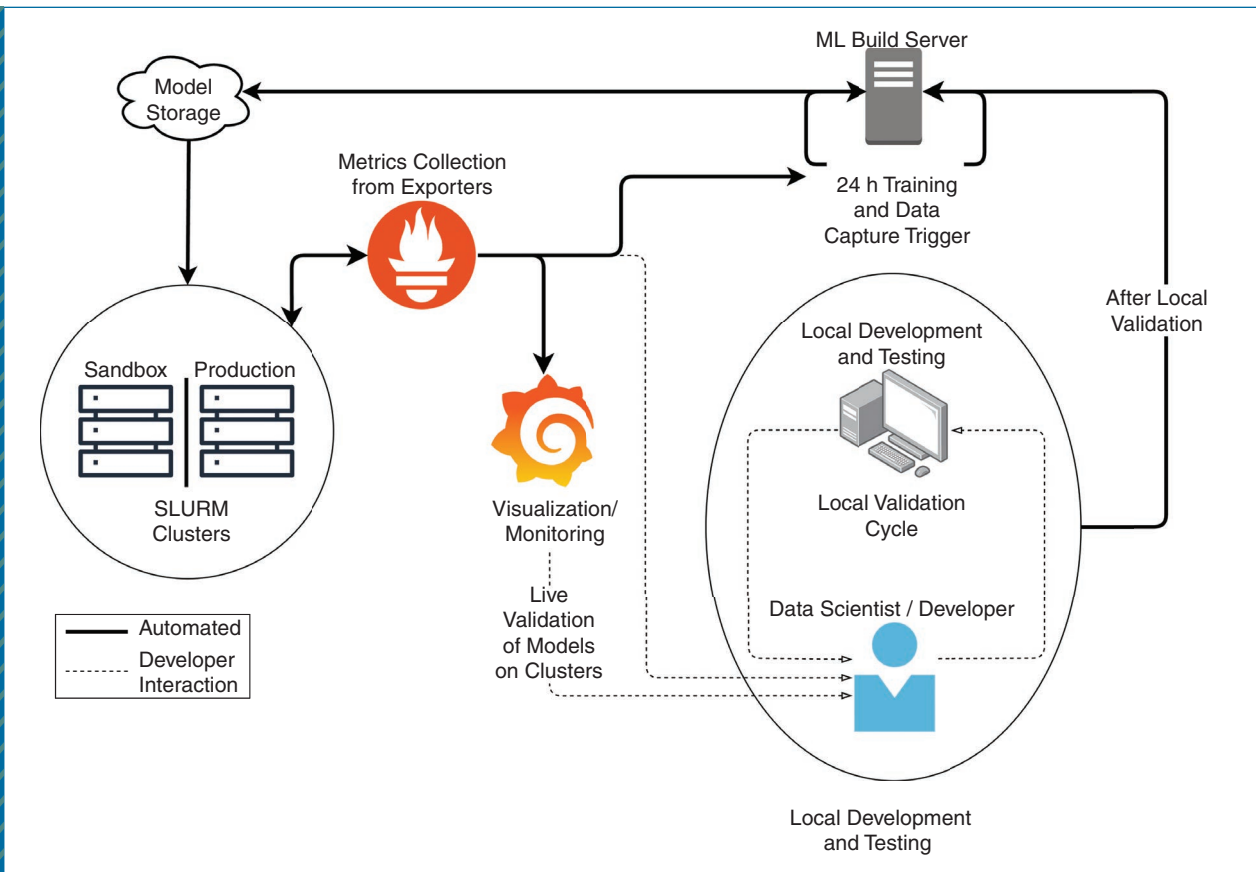


FIGURE 3. The DIDACT MLOps overview. The following MLOps processes are illustrated: data retrieval and preparation, model developments and local validation, automated model retraining, manual/automated model deployment, and model monitoring.

organization’s tech stack when they meet technical requirements. This reduces the learning curve and improves team collaboration through familiarity, and decreases infrastructure overhead. Grafana was already established for monitoring within the lab. Here we extend Grafana for visualizing key metrics associated with model performance. Similarly, as Prometheus is part of the existing monitoring infrastructure used by Ops, it was seamlessly integrated to collect and store metrics specific to ML models and pipelines.

Development, CL, and real-time inference pipelines share similar technical functionalities, leading to a significant challenge: the potential

for accidental inconsistencies arising from code duplication during development. To mitigate this risk and ensure data integrity across pipelines, we implement a modular workflow (illustrated in Figure 1). This workflow promotes code reuse for common processes like data preparation and analysis. By standardizing these steps, we significantly reduce the likelihood of inconsistencies, leading to trustworthy and reproducible results.

We highlight the significance of the collaboration between operations and data science staff, particularly when the data source and the purpose of the ML model involve computing resources. The operation

staffs’ understanding of the data input and their requirements for successful monitoring were fundamental to an effective MLOps implementation. Additionally, as Haertel et al.’s⁷ review of MLOps publications found, the field is new. We emphasize the benefit of sharing the utility of our practical, real-world application. 

Acknowledgment

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Nuclear Physics under contract DE-AC05-06OR23177. The research described in this paper was conducted under the Laboratory Directed Research and Development



DIANA MCSPADDEN is a data scientist at Thomas Jefferson National Accelerator Facility, Newport News, VA 23606 USA. Her research interests include continual learning, machine learning surrogate models for flooding prediction, experimental control and calibration, and generative artificial intelligence for science. McSpadden received her M.S. in data science from the University of Virginia. Contact her at dianam@jlab.org.



BRYAN HESS is the scientific computing operations group lead at Thomas Jefferson National Accelerator Facility, Newport News, VA 23606 USA. His research interests include network transport of scientific data, high performance storage systems, and data center design and operations. Hess received his M.S. in computer science from the College of William and Mary. Contact him at bhess@jlab.org.



MARK JONES is a software infrastructure developer at Thomas Jefferson National Accelerator Facility, Newport News, VA 23606 USA. His research interests include distributed computing, DevOps automation, data analytics and visualization, and software services. Jones received his B.S. in computer science from Old Dominion University. Contact him at maj@jlab.org.



MALACHI SCHRAM is head of Data Science at Thomas Jefferson National Accelerator Facility, Newport News, VA 23606, USA. His research interests include uncertainty-aware data-driven machine learning, deep reinforcement learning, distributed computing, and anomaly detection. Schram received his Ph.D. in high energy physics from Carleton University, Ottawa, ON, Canada. Contact him at schram@jlab.org.



AHMED HOSSAM MOHAMMED is a data scientist at Thomas Jefferson National Accelerator Facility, Newport News, VA 23606 USA. His research interests include digital signal processing, machine learning, graph neural networks, epileptic spike detection in electroencephalogram signals, charged particle tracking, anomaly detection, and high performance computing. Mohammed received his Ph.D. in electrical and computer engineering from Florida International University. Contact him at ahmedm@jlab.org.

program at Thomas Jefferson National Accelerator Facility for the U.S. Department of Energy. This article has supplementary downloadable material available at <https://doi.org/10.1109/MS.2024.3424256>, provided by the authors.

References

1. Y. Lu et al., “Investigating anomalies in compute clusters: An unsupervised learning approach,” SC23, Denver, CO, USA, Feb. 2024. [Online]. Available: https://sc23.supercomputing.org/proceedings/tech_poster/tech_poster_pages/rpost227.html
2. D. McSpadden, Y. Alanazi, B. Hess, and L. Hild, Oct. 2023, “Dataset for investigating anomalies in compute clusters,” Zenodo, doi: [10.5281/zenodo.10058230](https://doi.org/10.5281/zenodo.10058230).

3. J. Litzinger, R. Hallquist, and J. Tesmer, "HPC monitoring & visualization: Understanding usage of HPC systems," in *Proc. Pract. Experience Adv. Res. Comput. (PEARC)*, Portland, OR, USA, Jul. 2023, pp. 453–456, doi: [10.1145/3569951.3597554](https://doi.org/10.1145/3569951.3597554).
4. Y. Luo, M. Raatikainen, and J. K. Nurminen, "Autonomously adaptive machine learning systems: Experimentation-driven open-source pipeline," in *Proc. 49th Euromicro Conf. Softw. Eng. Adv. Appl. (SEAA)*, Durres, Albania, 2023, pp. 44–52, doi: [10.1109/SEAA60479.2023.00016](https://doi.org/10.1109/SEAA60479.2023.00016).
5. G. Ditzler, M. Roveri, C. Alippi, and R. Polikar, "Learning in nonstationary environments: A survey," *IEEE Comput. Intell. Mag.*, vol. 10, no. 4, pp. 12–25, Nov. 2015, doi: [10.1109/MCI.2015.2471196](https://doi.org/10.1109/MCI.2015.2471196).
6. G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter, "Continual lifelong learning with neural networks: A review," *Neural Netw.*, vol. 113, pp. 54–71, May 2019, doi: [10.1016/j.neunet.2019.01.012](https://doi.org/10.1016/j.neunet.2019.01.012).
7. C. Haertel, D. Staegemann, C. Daase, M. Pohl, A. Nahhas, and K. Turowski, "MLOps in data science projects: A review," in *Proc. IEEE Int. Conf. Big Data (BigData)*, Sorrento, Italy, 2023, pp. 2396–2404, doi: [10.1109/BigData59044.2023.10386139](https://doi.org/10.1109/BigData59044.2023.10386139).
8. G. Bhardwaj, "An open-source MLOps pipeline for particle accelerators [oral presentation]," in *Proc. 4th ICFA Beam Dyn. Mini-Workshop Mach. Learn. Part. Accel.*, Gyeongju, South Korea, 2024.
9. B. T. Klein, C. Tyler, and S. Fields, "DevOps and data: Faster-time-to-knowledge through SageOps, MLOps, and DataOps." Sandia National Laboratories (.gov). Accessed: May 28, 2024. [Online]. Available: <https://www.sandia.gov/research/publications/details/devops-and-data-faster-time-to-knowledge-through-sageops-mlops-and-dataops-2022-05-01/>



IEEE COMPUTER SOCIETY

Call for Papers

It's Author's Choice!
Publish open-access or
traditionally based on your
financial needs.



GET PUBLISHED

www.computer.org/cfp



IEEE
**COMPUTER
SOCIETY**



Digital Object Identifier 10.1109/MS.2024.3493515