Templated Hybrid Reusable Computational Analytics Workflow Management with Cloudmesh, Applied to the Deep Learning MLCommons Cloudmask Application

1st Gregor von Laszewski
Biocomplexity Institute
University of Virginia
Charlottesville, VA 22911, USA
laszewski@gmail.com
0000-0001-9558-179X

2nd J.P. Fleischer
Biocomplexity Institute
University of Virginia
Charlottesville, VA 22911, USA
jacquespfleischer@gmail.com
0000-0002-1102-1910

3rd Geoffrey C. Fox
Biocomplexity Institute
University of Virginia
Charlottesville, VA, USA
gcfexchange@gmail.com
0000-0003-1017-1391

4th Juri Papay
5th Sam Jackson
6th Jeyan Thiyagalingam
Rutherford Appleton Laboratory
Harwell Campus
Didcot, OX11 0QX, UK

***Abstract*—In this paper, we summarize our effort to create and utilize an *integrated* framework to coordinate computational AI analytics tasks with the help of a task and experiment management workflow system. Our design is based on a minimalistic approach while at the same time allowing access to hybrid computational resources offered through the owner's computer, HPC computing centers, cloud resources, and distributed systems in general. Access to this framework includes a GUI for monitoring and managing the workflow, a REST service, a command line interface, as well as a Python interface. It uses a template-based batch management system that, through configuration files, easily allows for the generation of reproducible experiments while creating permutations over selected experiment parameters as typical in deep learning applications. The resulting framework was developed for analytics workflows targeting MLCommons benchmarks of AI applications on hybrid computing resources, as well as an educational tool for teaching scientists and students sophisticated concepts to execute computations on resources ranging from a single computer to many thousands of computers as part of on-premise and cloud infrastructure. We demonstrate the usefulness of the tool while creating FAIR principle-based application accuracy benchmark generation for the MLCommons Science Working Group Cloudmask application. The code is available as an open-source project in GitHub and is based on an easy-to-enhance framework called Cloudmesh. It can be applied to other applications easily.**

***Keywords*—experiment workflow, task workflow, hyperparameter workflow, high-performance computing, batch queue management, workflow web service, cloudmesh.**

I. Introduction

Our long-term involvement as part of the MLCommons Science Working Group [1] has greatly influenced this work. MLCommons is a non-profit organization that brings together over 70 members from industry, academia, and government, to accelerate machine learning innovation for the benefit of all. One of its primary objectives is to develop standardized benchmarks for assessing the performance of machine learning systems and applying them to various applications. These applications encompass a wide range of domains, including healthcare, automotive, image analysis, and natural language processing, among others. MLCommons focuses on benchmarking training [2] and validation algorithms to track progress over time. To achieve this goal, MLCommons explores machine learning efforts in the realms of benchmarking, dataset development for benchmarking purposes, and the establishment of best practices that make effective use of machine learning. MLCommons is organized into several working groups that tackle specific topics, such as benchmarking in relation to training, training on high-performance computing (HPC) resources, and inference performed on data centers, edge devices, mobile devices, and embedded systems. Best practices are investigated in areas such as infrastructure and power management. Moreover, MLCommons operates additional working groups dedicated to investigation various aspects of benchmarking compute resources.

The Science Working Group within MLCommons is primarily focused on advancing the scientific aspects beyond the mere creation of static benchmarks [3]. This is one of the major motivations for our work. While in other working groups the focus is to measure the performance of a system based on a fixed number of static benchmarks, the focus of the science working group is to identify best practices for a number of scientific applications to find the best accuracy through repeatable benchmarks. The work presented here is an outcome of our contributions to the goals set forth by the MLCommons Science Working Group.

As part of this activity, we are collaborating with a number of researchers, including aspiring researchers from various academic institutions. However, we are confronted with multiple challenges: (a) Many domain scientists are not sufficiently familiar with using high-performance computing (HPC) or

Funded by NSF, DOE, NIST, and University of Virginia. In part funded by the NSF CyberTraining: CIC: CyberTraining for Students and Technologies from Generation Z with the award numbers 1829704 and 2200409 and NIST 60NANB21D151T.

other resources to conduct reproducible experiments reliably. (b) When dealing with different compute resources, special system configurations need to be developed and adapted for each resource, and these configurations may even vary from user to user on the various systems. (c) Creating a reproducible experiment is often out of scope as the focus for beginning students that may not have the time to run and understand the application during a semester activity. Together these aspects introduce a high entry barrier that is time-consuming to overcome. It also provides an unneccesarry hurdle of reusing MLCommons benchmarks by such users.

What we strive to achieve is to make the situation simpler for the domain scientists and students while working at the same time towards the implementation of the FAIR principle [4] for the targeted MLCommons Science applications addressing the previously mentioned issues. As part of this, we focus on the implementation of hybrid task parallelism to address reproducibility on multiple target machines, leveraging parallel computing concepts. Additionally, we apply experiment management parallelism that automates the process of running permutations over various hyperparameters. Together these two concepts provide a powerful easy-to-use framework that we designed and implemented. Although we are applying our approach to other applications including some not from MLCommons, we focus here on the MLCommons Cloudmask [3] application that tries to identify the cloud cover from satellite images.

The paper is structured as follows. First, we discuss the important requirement aspects that influenced the design and implementation of our work in Section II. We then provide an overview of related research in Section III. Next, we present the design of the workflow framework in Section IV. To demonstrate the practical use of the framework, we showcase its capabilities to specify task-based workflows in Section IV-B, followed by a detailed discussion on how to interface with the workflows. Additionally, we briefly outline other notable Cloudmesh features in Section V. Furthermore, we illustrate how Cloudmesh can effectively coordinate hyperparameter-based workflows for the Cloudmask application in Section VI. Finally, we present our conclusions in Section VII.

II. Requirements

We list briefly the requirements that influence our design while focusing on reusability and design requirements. A more in-depth discussion can be found in [5].

Reusability Requirements. One important requirement is to address the reusability of the benchmark across different platforms under similar conditions. This can be achieved with benchmark templates that can easily be adapted and then their execution replicated on other platforms. However, in scientific applications, we not only target one application on the systems, and one specific hyperparameter configuration showcasing the scalability of the system, but we are interested in finding out which application and which hyperparameters for that application are best to achieve the best accuracy. Hence, it is important to improve the accuracy while identifying guidelines for resource consumption to achieve the best accuracy promoting the FAIR principle [4].

Design Requirements. For the architecture design of the framework, we have the following requirements: (a) Simplicity, so that the framework can be easily reused, (b) Specificity, so the workflow can be specified precisely (c) Generality so that the workflow can be generally integrated as part of different frameworks or languages, (d) State and Performance Monitoribility, so we can monitor long-running workflows, (e) Hybrid Resource integration, allowing the integration of on-premise and off-premises HPC or cloud computing resources (f) Recoverable, so that state management can be recovered upon failure (g) Zero-install management install so that no backend services need to be available on the compute resources and existing services are used.

III. Related Research

Many different workflow systems have been developed in the past. It is out of the scope of this document to present a complete overview of such different systems as presented in [5]. As part of MLCommons MLCube [6] has been developed, which has a small number of overlapping features. It is configuring its workflows through YAML specification for organizing, running, and sharing machine learning experiments. However, it is not focusing on an integrated approach that encompasses task-based execution and the desire to discover new accuracy values as part of the benchmark. It is focused on using compute resources while executing a number of static benchmarks which is advantageous only if computational time benchmarks are considered. Furthermore, the current system has some limitations such that its Singularity implementation is not yet fully supported. We have collaborated with that team to communicate and propose changes to make MLCube more usable by the community. Overall our system provides a significant enhancement. However, once MLCube becomes more usable for our target systems it could also be integrated into our effort.

IV. Design

To fulfill our requirements, we have developed a sophisticated but easy to use framework for controlling several aspects of our workflow needs: (a) *System Integration*: the placement of the workflow onto a number of different compute resources including HPC, cloud, and local computing while also providing adaptations to various batch queues, (b) *Compute Coordination*: the coordination of task-based parallelism to execute the workflow on the various resources, and (c) *Experiment Executor*[1]: the coordination of hyperparameter sweeps used in the application that we call experiment coordinator. The architecture of the framework is depicted in Figure 1.

The framework is based on a layered architecture so that it can be improved and expanded on at each layer targeting developers and end users. For the end user, we provide a

[1]The repository and name is at this time called cloudmesh-sbatch, but will be renamed to cloudmesh-ee to reflect the better name of Experiment Executor

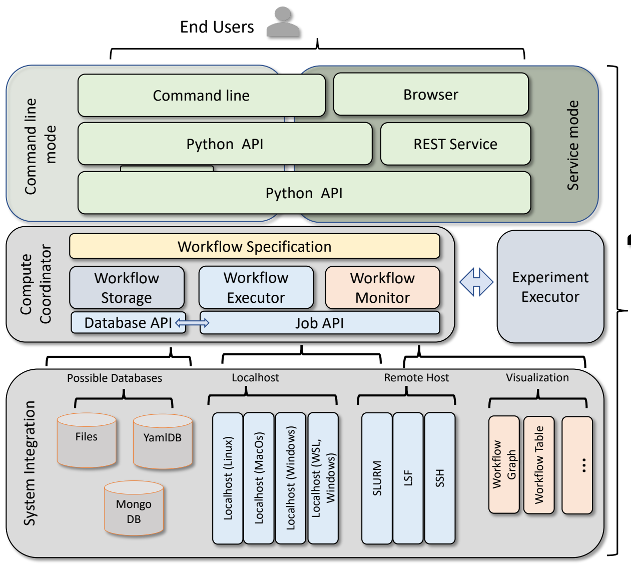


Fig. 1. Architecture of the Framework

command line and an intuitive browser interface. To support developers, we have designed a Python API that is also used to implement a REST Services-based on OpenAPI.

A. Overview of the Task-based Workflow with the Compute Coordinator

The workflow specification plays an important role in not only defining a workflow but also in simplifying status updates that are updating an instantiation of a workflow. As we have completely separated the status of the workflow from obtaining status updates, the system allows it to be shut down while the underlying jobs as part of the system integration can still be executed. Once the system is started again it self synchronizes its status from the system integration services. To summarize, the client is stateless and fetches the state of the submitted jobs on demand. It will return the latest state found on the job execution services.

The workflows are defined with human-readable YAML and can be stored in various formats on databases such as Cloudmesh YamlDB [7], which is file-based and provides the easiest integration. Hence, it is easy to create a variety of monitoring components (for example, as part of a Web browser) to display the Workflow and its status as a graph, table, or even as a continuous log file.

One of the most important parts of the framework is how we manage jobs and monitor their status through a status specification. For this, we have introduced an abstract job class that is integrated into a workflow class. The job class can define jobs, start them, and cancel them, to name only the most important management methods. However, each job defines a status file in which the actual progress of the job is recorded. This status file is managed on the compute resource where the job is run and is queried on demand to return the status to the client. This way, the status of all jobs can be monitored. As our goal is not to run jobs that execute in milliseconds but rather in the second range, such status reporting and propagation is well-suited for us as our tasks are typically long-running. Our status progress update specification is universally applicable fia integration into notifications through files (including stdout and stderr) and thus can be issued by bash scripts, SLURM scripts, Python programs, Jupyter notebooks, or any other computing language. The workflow status updates are implicitly augmented with timestamps, the compute resource, where it is resulting from, and additional messages to be sent to the monitoring component. Obviously, the workflow allows the specification of dependencies between tasks.

The framework is managed through a number of open-source repositories in GitHub [8], [8] and uses Python as the implementation language. The code is compatible with Windows, macOS, and Linux.

B. Specifying Task-based Workflows

The workflow definition for Cloudmesh is rather simple and intuitive. An example is provided in Figure 2 that specifies a task graph with the tasks ($start \rightarrow fetch-data \rightarrow compute \rightarrow analyze \rightarrow end$) representing a typical minimalistic use case for Deep Learning (DL). In this example the workflow executes three scripts ([fetch-data,compute,analyze].sh). Dependencies can easily be specified in human-readable format while using the node name. The nodes contain easy-to-manage information such as the name of the node, a label that is used to print the node's progress, and can contain templated variables such as any value defined as part of a particular node, or specially formatted time stamps. To demonstrate the easy use our label contains the *name* and *progress* of the workflow which is rendered by the graph or table monitoring components. One can also use variables accessible from Python including operating system or batch system variables to name only a few. Selected examples of values usable in the nodes are listed in Tables I and II.

```
workflow:
  nodes:
    start:
       name: start
    fetch-data:
       name: fetch-data
       user: gregor
       host: localhost
       status: ready
       label: '{name}\nprogress={progress}'
       script: fetch-data.sh
    compute:
       name: compute
       user: gregor
       host: localhost
       status: ready
       label: '{name}\nprogress={progress}'
       script: compute.sh
    analyze:
      name: analyze
      user: gregor
      host: localhost
      status: ready
      label: '{name}\nprogress={progress}'
      script: analyze.sh
    end:
       name: end
  dependencies:
    - start,fetch-data,compute,analyze,end
```

Fig. 2. Workflow YAML Configuration file.

Other features can also be passed along and may depend on the supporting underlying visualization framework. in our case, we have provided the ability to also adjust, the node style, color, font, and other to us available display parameters.

How easy it is to integrate the task framework into a Python script using our Cloudmesh StopWatch class and progress

TABLE I
NODE ATTRIBUTES.

Attribute	Description
name	A unique name of the job, must be the same as defined in the : line
user	The username for the host
host	The hostname
kind	The kind of the job, which can be local, ssh, wsl, or slurm
status	The status of the job in integer value between 0 and 100
label	A custom-designed label
script	The script name to be executed

TABLE II
LIST OF POSSIBLE LABELS FOR NODES ON THE GRAPH.

Name	Description
progress	progress of job from 0-100
now	current time
now.`%Y%m%d,%H--%M--%S`	now in particular format (this can be used for other times as well)
created	time when workflow was created
t0.`%Y%m%d,%H--%M--%S`	workflow start time
t1.`%Y%m%d,%H--%M--%S`	workflow end time
dt0.`%Y%m%d,%H--%M--%S`	elapsed time since workflow began
dt1.`%Y%m%d,%H--%M--%S`	total time of workflow once complete
tstart.`%Y%m%d,%H--%M--%S`	job start time
tend.`%Y%m%d,%H--%M--%S`	job end time
modified.`%Y%m%d,%H--%M--%S`	job modified time
os.	operating system environment variable (like os.HOME)
cm.	Cloudmesh variable that is read from `cms set`

method is shown in Figure 3. Figure 4 shows the table and graph view of a possible cloudmask workflow. During execution, the information in the views will be automatically updated through server-sent events. Additional details of these APIs go beyond the purpose of this document and are summarized in [5].

```python
from cloudmesh.common.StopWatch import progress
filename = './cloudmesh-log-file.log'
Stopwatch.start("total")
progress(progress=1, filename=filename)
# ... execute your analysis here ...
progress(progress=100, filename=filename)
Stopwatch.end("total")
Stopwatch.benchmark() # prints the benchmark results
```

Fig. 3. Python API progress monitoring.

C. *Experiment Execution*

It is important to note that the system not only includes a task workflow system but also allows the execution of iterations over experiment parameters as used in identifying optimal parameter sets maximizing for example the accuracy of a scientific application. The component to coordinate this is called *Experiment Executor* (EE). In traditional machine learning workflows, hyperparameter tuning and configuration are key elements in assessing and optimizing the performance of models. However, scaling hyperparameters for highly parallel execution with heterogeneous hardware is complex. The EE can be used to generate many parameter settings including the utilization of the type of GPU or the system hardware on which the experiment is executed. In Figure 4 we showcase the execution of an experiment that combines the Cloudmesh components of CC and EE to find the best accuracy across a number of different compute resources and GPUs. The EE jobs can utilize various queuing systems such as SLURM, and LSF, but also sequential and parallel ssh jobs. The scheduling of the jobs to be executed could be customized. For our simple example, we provide a simple Cartesian product for all experiment parameters provided. The advantage of this system is through the templated configuration files steering the application, quick adaptations can be made to identify more suitable hyperparameter settings. Furthermore, the use of the system implicitly promotes the reaction of a unique directory for each experiment in which the executed program and its output are stored. This fosters reproducible experiments following the FAIR principles As the structure of the experiment is including the hyperparameter settings, results from different computing systems can be merged into the overall results. Thus the EE supports the creation of coordinated results while allowing the generation of cooperating, selective, and distributed result generation. A simple experiment configuration file is showcased next, where we execute a scientific application called cloudmask for epochs 1, 30, and 60, on the GPUs v100 and a100 repeating it 5 times.

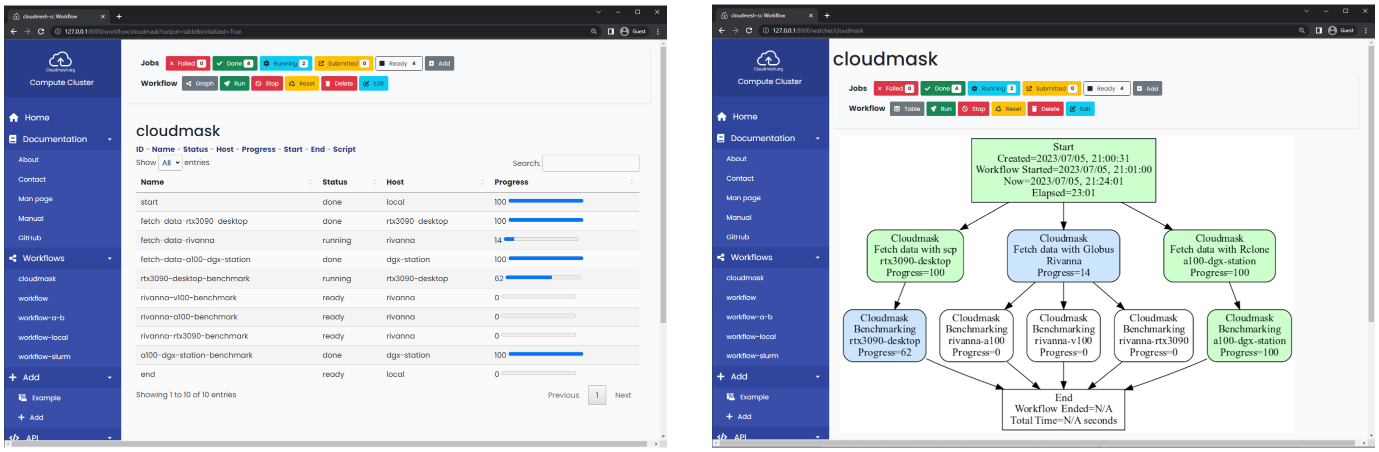
```yaml
application:
    name: cloudmask
data: "/scratch/{os.USER}/{application.name}"
experiment:
    epoch: "1,30,60"
    gpu: "a100,v100"
    repeat: "1,2,3,4,5"
```

Dependent on the queuing system a custom templated batch script utilizing these variables is created and used during the experiment execution. As it is templated and user-independent, it can then also be executed via different user accounts, either separately, or jobs being split up across different user accounts. An additional benefit from our templated executions is that an energy monitor can be integrated creating energy traces throughout the calculation. This helps as we can also compare and predict the energy consumption for long-running jobs which is of considerable importance due to the overall resources consumed by many scientific deep learning applications. Furthermore, we developed the best practice to predict for deep learning applications the computational and storage needs base on a few hyperparameter combinations so that planing can take place to acquire sufficient resources for long running templated experiments.

While practically working with the system, we observed that students (as part of research experiences) not using our experiment executor spend a significant amount (weeks-months) of a semester on setting up a benchmark and replicating only a fraction of the functionality provided by the EE. However, we tested the system out while other students used EE and we observed that the applications for which a template and configuration file has been designed reduced the on-ramp time to less than a day.

V. OTHER CLOUDMESH FEATURES

Cloudmesh has its origin in supporting various cloud computing resources, making it easy to develop and integrate them into the execution workflow to set up and use cloud resources as computational resources. It provides AWS, Azure, Google, and OpenStack cloud interfaces for virtual machine and data file services. Cloudmesh was the first tool that developed as a hybrid cloud API, command line, and command shell framework. As it uses templated virtual machine specifications, it is easy to switch from one cloud to another with a one-line



cloudmask

ID - Name - Status - Host - Progress - Start - End - Script

Show All entries Search:

Name	Status	Host	Progress
start	done	local	100
fetch-data-rtx3090-desktop	done	rtx3090-desktop	100
fetch-data-rivanna	running	rivanna	14
fetch-data-a100-dgx-station	done	dgx-station	100
rtx3090-desktop-benchmark	running	rtx3090-desktop	62
rivanna-v100-benchmark	ready	rivanna	0
rivanna-a100-benchmark	ready	rivanna	0
rivanna-rtx3090-benchmark	ready	rivanna	0
a100-dgx-station-benchmark	done	dgx-station	100
end	ready	local	0

Showing 1 to 10 of 10 entries

Previous 1 Next

Fig. 4. Table and Graph view. Please zoom in for more details.

command. Other services include the automated generation of REST services from Python APIs as documented [9].

For benchmarking, Cloudmesh also automatically exports MLCommons log events but uses the much more sophisticated, intuitive, and easier-to-use Cloudmesh StopWatch, making it possible to export benchmark timers in a human-readable format.

VI. MLCommons Cloudmask Workflow

We are applying the CC and EE workflow system to a number of MLCommons and non MLCommons applications [3], [5]. However, here we only summarize our experience as applied to the MLCommons Science Group Cloudmask application.

Cloudmask is a program that develops a model to classify sections of satellite images as either containing clouds or clear skies by using machine learning [10]. We used CC to coordinate the execution of cloudmask between different systems and EE to execute various parameter settings and iterate over the chosen hyperparameters. This includes an HPC computer at the University of Virginia called Rivanna, as well as two desktop computers (Figure 4). As we are able to coordinate the execution of the workflows across different machines and merge the results we were able to easily create a runtime comparison between the different compute resources (see Figure 5, here shown only for one epoch).

An automated script has been developed that produces the selected outputs that one usually finds with deep learning parameter studies (5-11). The workflow runs approximately 24 hours while producing the results in parallel on multiple compute resources and multiple GPUS.

VII. Conclusion

We have designed and implemented a Hybrid Reusable Computational Analytics Workflow Management with the help of the Cloudmesh component framework. The component added focuses on the management of workflows for computational analytics tasks and jobs. The tasks can be executed on remote resources via ssh and even access queuing systems such as Slurm. In addition, we can integrate the current computer on which the workflow is running. This can include

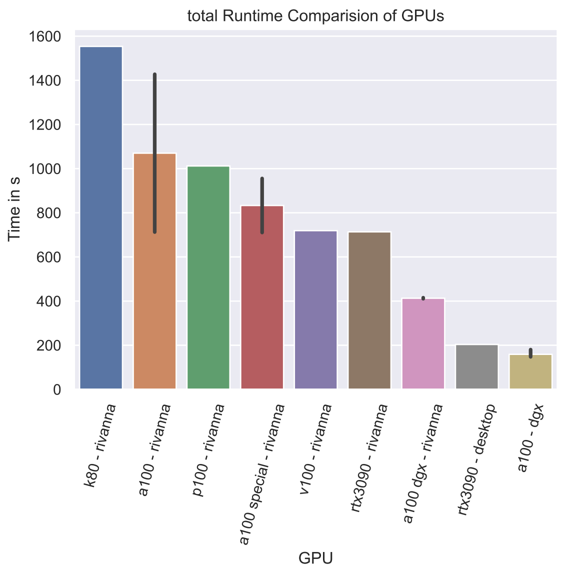


Fig. 5. Runtime comparisons of Cloudmask between graphics cards

operating systems such as Linux, macOS, Windows, and even Windows Subsystem for Linux. Through Cloudmesh, access to a command line and a command shell is provided. A simple API and a REST interface are provided. The framework also has an elementary Web browser interface that allows visualizing the execution of the workflow. It is important to know that the workflow can be started on remote resources and is running completely independently from the client tool once a task is started in contrast to many other workflow systems. This allows a "stateless" model that can synchronize with the remotely started jobs on demand through pull request. Hence, the framework is self-recovering in case of network interruptions or power failure eve on the client side. Due to our experiences with real (and many) infrastructure failures at the authors' locations, the availability of such a workflow-guided system was beneficial. Although very large graphs can be created we envision the utilization for graphs between 1 - 1000 nodes. The targeted task based run from minutes to hours and days and the state synchronization is fast in comparison to the runtime of individual tasks. The developed code is rather small and, in contrast to other systems, is less complex.

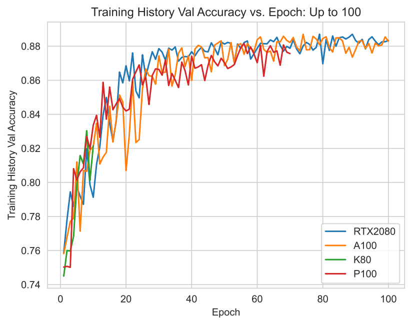


Fig. 6. Training history validation accuracy

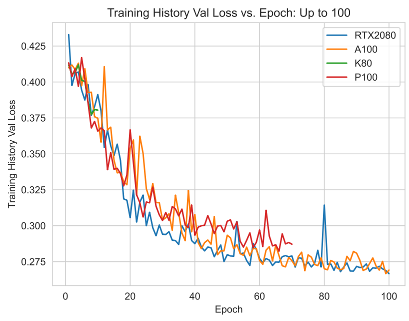


Fig. 7. Training history validation loss

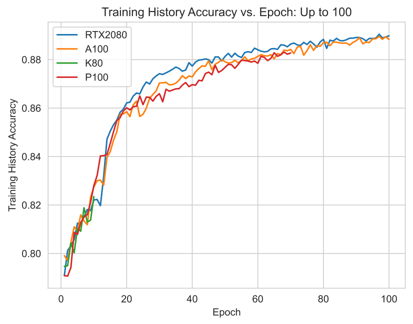


Fig. 8. Training history accuracy

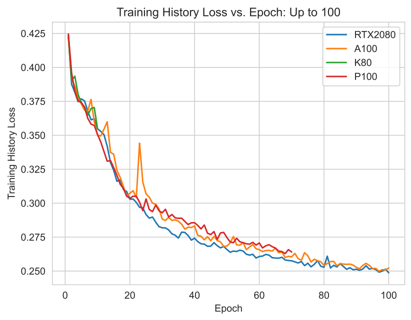


Fig. 9. Training history loss

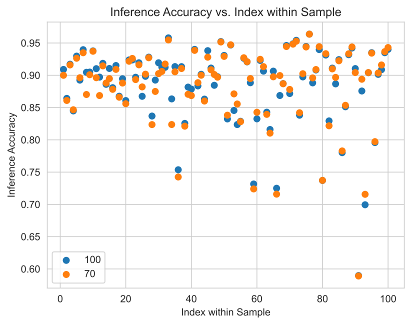


Fig. 10. Inference accuracy

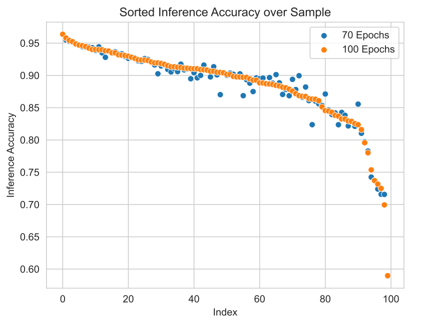


Fig. 11. Inference accuracy over sample as a spectrum.

The workflow specification and integration of monitoring is easy. Hence, it is suitable for educational aspects as it is used for master's and undergraduate-level research projects to apply the workflow, but also to enhance it. The project has and is practically utilized while generating benchmarks for the MLCommons Science Working Group showcasing real-world applicability beyond a student research project. Most importantly, the work to execute such benchmarks has been reduced from weeks to a day. The framework was first used to unexpectedly discover and report IO latency's while comparing traditional HPC systems with very fast desktops using NVMe's, resulting in recommendations improving the IO storage of the HPC system. Thus the framework not only allows fast on-ramp time for users, but has been used to communicate system related performance issues that since have been improved. Future activities include adaptation to more applications.

Acknowledgment

Work was in part funded by (a) NIST 60NANB21D151T (a) NSF CyberTraining: CIC: CyberTraining for Students and Technologies from Generation Z with the award numbers 1829704 and 2200409 and NIST 60NANB21D151T (b) Department of Energy under the grant Award No. DE-SC0023452 (c) the Biocomplexity Institute and Initiative at the University of Virginia. We like to thank the reviewers for their valuable feedback.

References

[1] "Machine learning innovation to benefit everyone," *Web page*, Apr. 2023, https://mlcommons.org/ [Accessed April 13, 2023]. [Online]. Available: https://mlcommons.org/

[2] P. Mattson, C. Cheng, C. Coleman, G. Diamos, P. Micikevicius, D. Patterson, H. Tang, G.-Y. Wei, P. Bailis, V. Bittorf, D. Brooks, D. Chen, D. Dutta, U. Gupta, K. Hazelwood, A. Hock, X. Huang, A. Ike, B. Jia, D. Kang, D. Kanter, N. Kumar, J. Liao, G. Ma, D. Narayanan, T. Oguntebi, G. Pekhimenko, L. Pentecost, V. J. Reddi, T. Robie, T. S. John, T. Tabaru, C.-J. Wu, L. Xu, M. Yamazaki, C. Young, and M. Zaharia, "MLPerf Training Benchmark," *arXiv*, 2019. [Online]. Available: https://arxiv.org/abs/1910.01500

[3] J. Thiyagalingam, G. von Laszewski, J. Yin, M. Emani, J. Papay, G. Barrett, P. Luszczek, A. Tsaris, C. Kirkpatrick, F. Wang, T. Gibbs, V. Vishwanath, M. Shankar, G. Fox, and T. Hey, "AI Benchmarking for Science: Efforts from the MLCommons Science Working Group," in *High Performance Computing. ISC High Performance 2022 International Workshops*, H. Anzt, A. Bienz, P. Luszczek, and M. Baboulin, Eds. Cham: Springer International Publishing, 2022, pp. 47–64.

[4] "Fair principles," Web Page, Jul. 2023. [Online]. Available: https://www.go-fair.org/fair-principles/

[5] G. von Laszewski, J. P. Fleischer, and G. C. Fox, "Hybrid reusable computational analytics workflow management with cloudmesh," arXiv, Tech. Rep., Oct. 2022. [Online]. Available: https://arxiv.org/abs/2210.16941

[6] "Mlcube," Web Page, Jul. 2023. [Online]. Available: https://mlcommons.org/en/mlcube/

[7] G. von Laszewski, "yamldb," Jul. 2023, [Online; accessed 24. Jul. 2023]. [Online]. Available: https://github.com/cloudmesh/yamldb

[8] "Cloudmesh Experiment Executor Repository," Aug. 2023, [Online; accessed 12. Aug. 2023], formerly also know as cloudmesh-sbatch https://github.com/cloudmesh/cloudmesh-ee. [Online]. Available: https://github.com/cloudmesh/cloudmesh-ee

[9] G. v. Laszewski, A. Orlowski, R. H. Otten, R. Markowitz, S. Gandhi, A. Chai, G. C. Fox, and W. L. Chang, "Using cloudmesh gas for speedy generation of hybrid multi-cloud auto generated ai services," in *2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)*, 2021, pp. 144–155.

[10] J. Papay, "CloudMask benchmark notes," GitHub, Oct. 2022, [Online; accessed 24. Oct. 2022]. [Online]. Available: https://github.com/laszewsk/mlcommons/tree/main/benchmarks/cloudmask/#readme