Reproducibility and benchmarking of machine learning algorithms

Jonathan Klinginsmith
Department of Computer Science
Indiana University
Bloomington, IN, USA
jklingin@indiana.edu

Geoffrey C. Fox
Department of Intelligent Systems Engineering
Indiana University
Bloomington, IN, USA
gcf@indiana.edu

***Abstract*—Experimental reproducibility is a topic of increasing focus across scientific domains. The ability for researchers to recreate the work of their peers has become imperative in the scientific discovery process. Now as the scientific process further leverages both technology and data, the need to reproduce computational experiments is imperative.**

The broad field of eScience emerged because of data- and technology-driven experimentation. Computational analyses are a foundational component of eScience. Moreover, many disciplines of eScience are leveraging machine learning to make scientific inferences from trained datasets. Thus, there is a need to replicate machine learning algorithms as well.

In this work, we leverage software containers and a container orchestration platform as foundational tools for reproducing and benchmarking computational experiments. We present our findings through the use of an industry supported machine learning benchmark suite.

***Index Terms*—reproducibility, benchmarking, eScience, machine learning**

I. Introduction and Motivation

Experimental reproducibility is a topic of increasing focus across scientific domains. The ability for researchers to recreate the work of their peers has become imperative in the scientific discovery process. Now as the scientific process further leverages both technology and data, the need to reproduce computational experiments is imperative.

The broad field of eScience emerged because of data- and technology-driven experimentation. Computational analyses are a foundational component of eScience. Moreover, many disciplines of eScience are leveraging machine learning to make scientific inferences from trained datasets. Thus, there is a need to replicate machine learning algorithms as well. As a researcher reading a scientific paper on a new algorithm within a particular eScience domain, it can be challenging to replicate the authors' computational experiments. To fully reproduce the computational experiments in the paper, one must have the same versions of software installed and configured, have the original data, and parameters used within the original experiment.

In this work, we leverage software containers and a container orchestration platform as foundational tools for reproducing and benchmarking computational experiments. We present our findings through the use of an industry supported machine learning benchmark suite.

In many cases, having access to all these items is not possible [1]. Even if the original data are not available, it should be reasonable to expect experimental setup to be reproducible. Specifically, if the infrastructure setup and the software installation and configuration can be performed in a reproducible manner then scientists are much more enabled at replicating or extending the experiment in question.

II. eScience Experiments

The Oxford English Dictionary defines the *scientific method* as "a method of procedure that has characterized natural science since the 17th century, consisting in systematic observation, measurement, and experiment, and the formulation, testing, and modification of hypotheses" [2]. Within the computational science research community, Stodden [3] states "the digitization of science combined with the Internet create a new transparency in scientific knowledge, potentially moving scientific progress from building with black boxes, to one where the boxes themselves remain wholly transparent."

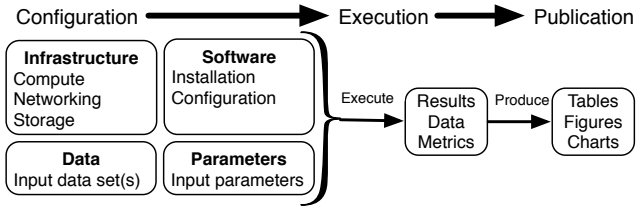

Fig. 1. Experimental progression

Figure 1 models the progression of a computational science or eScience experiment. The three phases: *configuration*, *execution*, and *publication* represent logical constructs where experimental activities performed and replicated. During the *configuration* phase, software must be installed and configured and when necessary infrastructure must be provisioned. Within the *execution* phase, the actual experiment is performed. Data and metrics are generated from experiment for use in the *publication* stage. In the *publication* stage, data tables, figures, and charts are produced for information sharing and presentation of experimental results.

III. Benchmarking

In eScience, a computational benchmark brings conformity to a problem, an experiment, or an analyses such that it serves as a means for performing evaluations, comparisons, and measurements. The concept of a benchmark is not new. Two such benchmarks, TPC and SPEC, have have existed for many years. The Transaction Processing Performance Council (TPC™) "is a non-profit corporation focused on developing data-centric benchmark standards and disseminating objective, verifiable performance data to the industry." [4] The Standard Performance Evaluation Corporation (SPEC) "is a non-profit corporation formed to establish, maintain and endorse standardized benchmarks and tools to evaluate performance and energy efficiency for the newest generation of computing systems." [5].

Benchmarks are actively being developed in the areas of artificial intelligence and machine learning. For example, there is the formation of a TPC Artificial Intelligence Working Group. Additionally, MLPerf [6] is a more recent benchmarking addition to the machine learning community. Both the SPEC benchmark for general purpose computing as well as the TPC benchmark for database systems were motivations for the MLPerf benchmark [6].

In [7], the concept of *reproducibility* is disussed as an important characteristic of good benchmarks and metrics. Because of the necessity to execute a benchmark numerous times when performing comparisons, it is crucial the benchmark is reproducible. Reproducibility is examined in further detail in the following section.

IV. Reproducibility

The ability for researchers to reproduce the work of their peers has become imperative in the scientific discovery process. Researchers from a variety of scientific fields have called for the experimental data and code be made available such that published results can be conveniently reproduced [8]. For scientific workflows, reproducibility is considered one of the requirements [9].

There is a spectrum when it comes to considering reproducibility. The bare minimum researchers should provide are the software and data used in the experiments and data presented [10]; however, these two items together do not truly provide reproducibility. To fully reproduce a computational science experiment, several items from the original experiment must be considered. Initially, one must have the software configured in the same manner as the original experiment. Access to the original experimental data may not always be possible, but its usage also aides in reproducibility. It further necessary to provide information on how to execute the experiment, such as how to run the software and what appropriate parameters are. The topic of reproducibility has appeared in many software and technical disciplines.

Two programming languages, Python and R, are leveraged heavily in machine learning. Both languages have introduced tools and systems to aid in reproducibility. For the R programming language, the use of the integrated development environment RStudio and the knitr R package were discussed in [11]. Reproducibility and portability in the R programming language are two of the key movitations for Packrat [12], a system for package dependency management. Python's package installer tool, pip [13] provides a requirements file convention that allows for package dependencies and versions to be specified.

Software module systems such as evironment modules [14] and lmod [15] were created by administrators of high performance computing clusters to manage versions of computational software, programming language compilers, numerical and other system libraries. Scientists using these module systems can load the necessary artifacts, including specific versions to create an environment to run a computational set of tasks.

Computational reproducibility at TACC was discussed here [16]. lmod was discussed in this podcast

The use of cloud computing has been discussed by researchers as a means to aid in the reproducibility of *in silico* experiments [17], [18]. In [18], the author demonstrates how virtual appliances, which are virtual machines with pre-installed and pre-configured software, can be used for reproducible research. The concept of a software configuration snapshot can be applied across a variety of technology stacks. Just as a virtual machine appliance preserves the pre-configured software, containers can also be considered a snapshot of a software environment. Containers are discussed in detail in the next section.

A. Containers

Advances in the Linux operating system kernel provided the underpinnings for containers. Docker [19] is the most populate container ecosystem. Singularity [20] was introduced by the high performance computing community as a container technology that could be executed without administrator access and without a daemon process. Both Docker and Singularity have commands that can be executed to create the container. With Docker, the convention is to create a Dockerfile to execute. Similarly, Singularity has a concept of a definition file which contains the set of instructions to execute to build the container. Tie into reproducibility and specifically tie into MLPerf's goal to run the same experiments across a variety of architectures A common process when building the container is to version it. Both Docker and Singularity provide web-accessible hosting repositories so that the versioned container artifact can be downloaded for future usage. In the next section, artifacts and artifact repositories are discussed in further detail.

B. Artifacts

An *artifact* at its fundamental level can be thought of the packaged item or set of items. Typically the items packaged within the artifact are related to software. For example, an artifact can be a package for a programming language, a package for an operating system, or a software container,

among other possible items. The following properties important when defining an artifact. First, the artifact must be versioned or it must contain a unique identifier. By versioning the artifact, it is possible to capture a point-in-time snapshot of it. Second, the artifact must be accessible. Typically by hosting the artifact from a downloadable location, it can be accessed. A hosting location for a set of artifacts, an *artifact repository*, allows artifacts to uploaded, stored, and distributed. Artifact repositories may also contain capabilities for indexing artifacts, searching for artifacts, among other advanced and specific functionality.

Programming languages such as Python and R, host their software packages in repositories. The R programming language has two main package repositories. CRAN [21] is the primary package repository. Bioconductor [22] hosts bioinformatics packages and reference data. Python's package index, PyPI [23], hosts the language's software packages.

The Linux operating system has several different distributions. Each distribution, such as CentOS and Ubuntu, has create a software package format for installation of operating system libraries. Moreover, the distribution providers host the packages in web-accessible repositories.

Docker Hub [24] and Singulary Hub [25] are the container repositories for Docker and Singularity, respectively.

All of the repositories mentioned below provide capabilites in addition to Additionally, it is possible to create private repositories for all of the artifacts mentioned above. For example, tools such as Artifactory [26] provide a universal repository management platform to either proxy the public repositories or create private repositories.

Data can also be considered an artifact. Providing unique urls, versions, or tags allows researchers the ability to access a uniquely identifiable version of the data set.

Collective Knowledge [27] is a . Collective Knowledge is being used in the MLPerf benchmark to retrieve datasets. Artifact Evaluation - part of Collective Knowledge

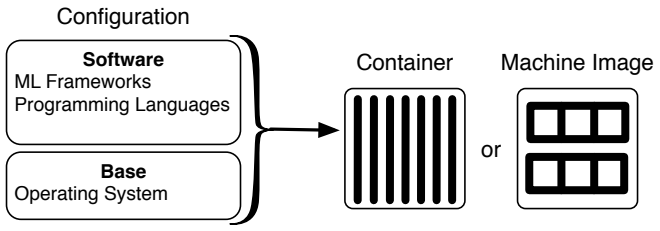

Fig. 2. Building a container or machine image

V. Contributions

We submitted code modifications to MLPerf Inference Github repository to provide a consistent execution methodology to the benchmark.

For example there are several key steps necessary to run the experiments. First, a researcher must build the Docker container. Our contributions made it so every use-case now has a Dockerfile and instructions on how to build the Docker container. Second Lastly, is the execution of the benchmark. We fixed software bugs and corrected documentation Leveraging Docker for reproducibility – Leveraging Docker across all MLPerf inference experiments for reproducibility Consistent execution methodology – build container, download data, and run benchmark experiment Fixed bugs and corrected documentation – Contributed software and documentation updates so that the experiment execution can be executed properly and successfully.

VI. Conclusions & Future Work

We made evident the We detailed core requirements for reproducibility and The commonalitites betweening benchmarking and reproducibility were

References

[1] S. Staff, "Challenges and opportunities," *Science*, vol. 331, no. 6018, pp. 692–693, 2011.

[2] *Oxford English Dictionary*. Oxford University Press, 2019.

[3] V. Stodden, "The scientific method in practice: Reproducibility in the computational sciences," MIT Sloan School of Management, Tech. Rep. 4773-10, 2010.

[4] http://www.tpc.org/, 2019.

[5] https://www.spec.org/, 2019.

[6] "Mlperf," https://www.mlperf.org/, 2019.

[7] C. Bourrasset, F. Boillod-Cerneux, L. Sauge, M. Deldossi, F. Wellenreiter, R. Bordawekar, S. Malaika, J.-A. Broyelle, M. West, and B. Belgodere, "Requirements for an enterprise ai benchmark," in *Performance Evaluation and Benchmarking for the Era of Artificial Intelligence*, R. Nambiar and M. Poess, Eds. Cham: Springer International Publishing, 2019, pp. 71–81.

[8] V. Stodden, I. Mitchell, and R. LeVeque, "Reproducible research for scientific computing: Tools and strategies for changing the culture," *Computing in Science and Engineering*, vol. 14, no. 4, pp. 13–17, 2012.

[9] Y. Gil, E. Deelman, M. Ellisman, T. Fahringer, G. Fox, D. Gannon, C. Goble, M. Livny, L. Moreau, and J. Myers, "Examining the challenges of scientific workflows," *Computer*, vol. 40, no. 12, pp. 24–32, 2007.

[10] R. D. Peng, "Reproducible research in computational science," *Science*, vol. 334, no. 6060, pp. 1226–1227, 2011.

[11] L. Madeyski and B. Kitchenham, "Would wider adoption of reproducible research be beneficial for empirical software engineering research?" *Journal of Intelligent & Fuzzy Systems*, vol. 32, no. 2, pp. 1509–1521, 2017.

[12] "Packrat," https://rstudio.github.io/packrat/, 2019.

[13] "pip - the python package installer," https://pip.pypa.io/en/stable/, 2019.

[14] J. L. Furlani, "Modules: Providing a flexible user environment," in *Proceedings of the Fifth Large Installation Systems Administration Conference (LISA V)*, 1991, pp. 141–152.

[15] M. Geimer, K. Hoste, and R. McLay, "Modern scientific software management using easybuild and lmod," in *2014 First International Workshop on HPC User Support Tools*, Nov 2014, pp. 41–51.

[16] "Science on repeat - computational reproducibility," https://soundcloud.com/usetacc/science-on-repeat-computational-reproducibility, 2019.

[17] J. Dudley and A. Butte, "In silico research in the era of cloud computing," *Nature Biotechnology*, pp. 1181–1185, 2010.

[18] B. Howe, "Virtual appliances, cloud computing, and reproducible research," *Computing in Science and Engineering*, vol. 14, pp. 36–41, 2012.

[19] J. Cito and H. C. Gall, "Using docker containers to improve reproducibility in software engineering research," in *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*, May 2016, pp. 906–907.

[20] G. M. Kurtzer, V. Sochat, and M. W. Bauer, "Singularity: Scientific containers for mobility of compute," *PLOS ONE*, vol. 12, no. 5, pp. 1–20, 05 2017. [Online]. Available: https://doi.org/10.1371/journal.pone.0177459

[21] "The comprehensive r archive network," https://cran.r-project.org/.

[22] "Bioconductor," https://www.bioconductor.org/.

[23] "Pypi – the python package index," https://pypi.org/, 2019.
[24] "Docker hub," https://hub.docker.com/.
[25] "Singularity container registry," https://singularity-hub.org/.
[26] "Artifactory - universal artifact repository manager," https://jfrog.com/artifactory/, 2019.
[27] G. Fursin, A. Lokhmotov, and E. Plowman, "Collective knowledge: towards r&d sustainability," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE'16)*, March 2016.