




Curifactory: A research experiment manager

Nathan Martindale ^{1¶}, Scott L. Stewart ^{*1}, Jason Hite ^{†1}, and Mark B. Adams ^{‡1}

¹ Oak Ridge National Laboratory ¶ Corresponding author

DOI: [10.21105/joss.05793](https://doi.org/10.21105/joss.05793)

Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: Kelly Rowland 

Reviewers:

- [@abhishektiware](#)
- [@deniederhut](#)

Submitted: 24 July 2023

Published: 24 October 2023

License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](#)).

Summary

Curifactory is a command line tool and framework for organizing Python experiment code, configuration parameters, and results. It is an opinionated and lightweight approach to workflow management infrastructure and is primarily intended to support researchers conducting experiments on one machine. This software was developed to support the reproducibility of results for several data science projects in the Nuclear Nonproliferation Division at Oak Ridge National Laboratory.

Curifactory is intended to be a general framework and is not specific to machine learning or data science. It can aid in any field in which experiments are primarily computation-based studies and can be implemented in Python (e.g., high-energy physics, astronomy, computational chemistry). The design emphasizes the automated caching of intermediate data analysis artifacts to speed up development involving computationally intensive tasks. It also allows for data provenance and experiment reproduction. Individual experiment runs are tracked through logs and their output reports, and entire copies of a run with all cached data and metadata can be exported for others to run using Curifactory on another machine. Curifactory experiments can either be integrated into a project from the beginning or can be written on top of an existing codebase without needing significant modification. A few important views of the Curifactory library can be seen in [Figure 1](#).

*co-first author

†co-first author

‡co-first author

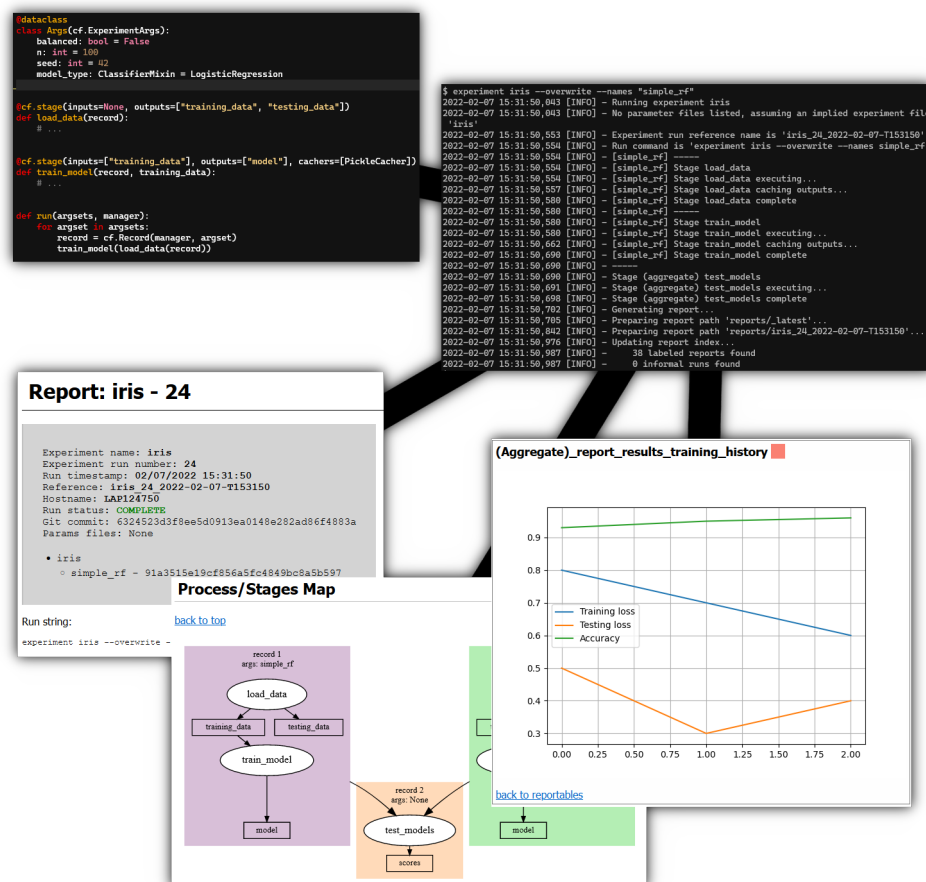


Figure 1: Collection of views from the Curifactory library, including a code example of an experiment, the command line interface, and output report visualization.

Statement of Need

Experiment organization and results management are often overlooked aspects of conducting research with any significant amount of software code. Project folders can become a mess of temporary runnable scripts with inconsistent outputs and conflicting or outdated data files. This is often a result of rapid code prototyping during the research process, combined with the difficulty of managing and re-using intermediate experimental outputs (i.e., computationally expensive data transforms). This can make it difficult to track experiment results over time and greatly complicate the task of reproducing a set of results. Managing large collections of possible parameter sets for experiment code can also be challenging, sometimes requiring scripts to generate configuration files that can be difficult to update if parameter changes occur later in the research lifecycle.

Various solutions have been implemented to address some aspects of research experiment management, and in prior work, the authors explored some of these solutions and detailed a set of attributes to consider for comparing such systems: orchestration, parameterization, caching, provenance, portability, reporting, and scalability (Martindale et al., 2022). We briefly explore three of those here, Kedro (Linux Foundation AI & Data, 2019) and MLFlow (LLC, 2018), as well as Pachyderm (Pachyderm, 2014) to highlight what distinguishes Curifactory and the types of problems it is suited for. We additionally compare Luigi (Spotify, 2012) as an example of a more general task pipeline manager.

First, Curifactory is a more generalized approach to experiments. Tools such as Pachyderm and MLFlow are specifically designed for machine learning models, whereas Curifactory makes no assumptions about the objective of an experiment or data analysis task. Pachyderm in particular is more complicated to set up because it requires a Kubernetes cluster ([Foundation, 2014](#)). In practice, this means that Pachyderm is better suited for use by a medium or large team in a production setting, rather than a team with one to a few researchers. MLFlow differs in its abilities to configure parameters, relying on raw string key-value pairs. Curifactory, in contrast, directly handles configuration in Python files, allowing greater flexibility and dynamic parameter set construction. MLFlow also has no explicit means for orchestration, relying solely on the script in which it is run. On the other end of the spectrum is Luigi, a more general tool than Curifactory designed to help run batch job pipelines in a highly scalable way. While many of the same features are present, the ability to parameterize jobs, orchestrating job runs, and of course scalability, the emphasis of the tool is not on experiments. Luigi is likely well suited to certain tasks within an experiment, such as large amounts of relatively static data processing, but specifically the lack of detailed provenance per-output and integrated output reporting make it less effective at managing overall experiment workflows by default.

Curifactory is very similar to Kedro in terms of the abstractions it provides for orchestration. Kedro's "nodes" are functionally similar to Curifactory "stages," but Kedro's "pipelines" design stresses modularity and reusability such that sections of code could be used easily across multiple projects. This is less of an emphasis in Curifactory's approach, which focuses on the organization of multiple experiments within a single project. Kedro is more effectively designed to support one primary experiment per project with pipelines that are easily shared between projects. The two tools also take different approaches for caching and reporting. Curifactory caches and reports information based on each specific set of parameters run through a select experiment. This makes re-entrancy through caching easier for a given experiment and parameter set. Kedro, on the other hand, does not explicitly version cached outputs by default. Finally, the two projects differ in their preferred file approach for parameterization. Curifactory uses Python configuration files for parameterizing experiments while Kedro relies on a YAML-based system by default.

Workflow

The development team created Curifactory based on several high-level concepts discussed in this section. The first of these concepts is a "stage", which functionally represents one step of an experimental process such as a data transformation. The stage is a function that should be used to make calls into the main research codebase and configure these calls based on user-specified experiment parameters.

This leads to the next core concept of Curifactory, which are "parameters". Parameters represent the algorithm/analysis hyperparameters and experiment settings necessary to configure how stages run. Rather than relying on typical configuration file formats (e.g., YAML, JSON) Curifactory assumes that these configuration values are stored in instances of Python dataclasses that are generated in "parameter files". Keeping this type of information in code rather than in a more typical static format has several advantages, including:

1. A parameter can store any Python object.
2. Parameter sets can be modular/compositional.
3. Parameter sets can inherit from another parameter set.
4. Parameter sets can be dynamically generated. For example, this allows researchers to generate an experiment-wide parameter grid search, potentially generating hundreds of parameter sets from a single file.

Finally, an "experiment" is a Python function that takes in a collection of parameter sets and routes them through one or more sequences of stages. Experiments have a manager that is available throughout a run which helps automatically maintain the state of data passing

through the various stages. These interactions are shown in [Figure 2](#). Experiments represent the different overall analysis tasks that a researcher may want to test. For example, a researcher might have an experiment that evaluates which type of data-driven model converges best for a scenario of interest.

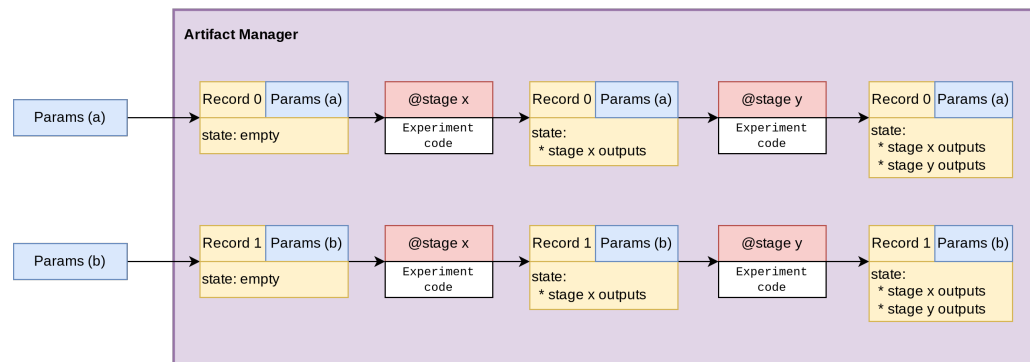


Figure 2: An experiment with multiple sets of parameters proceeding through stages, maintaining the state of data passing through various stages on records managed by the manager.

Researchers can interact with Curifactory on the command line through the experiment command. This interface allows the user to specify an experiment to run and the set(s) of parameters to apply to that experiment. Various flags control Curifactory's functionality, such as whether an experiment Jupyter Notebook summary is automatically generated at the end, whether to ignore some specific parameters, and whether to parallelize experiment execution.

Audience

The target audience for Curifactory is small research teams performing multiple data analysis tasks as part of a research activity. This could include activities like graduate research, business analysis tasks, or scientific experimentation. This open-source software is licensed under a BSD-3 clause license, is registered on [DOE Code](#), and is available on [GitHub](#). The package is also pip installable with `pip install curifactory` with Sphinx ([Komiya et al., 2008](#)) built [documentation](#). Finally, linting for this project is performed using black ([Python Software Foundation, 2019](#)) and flake8 ([Python Code Quality Authority, 2010](#)) as well as other pre-commit hooks with pre-commit ([pre-commit Organization, 2017](#)).

Acknowledgments

The authors would like to acknowledge the US Department of Energy, National Nuclear Security Administration's Office of Defense Nuclear Nonproliferation Research and Development (NA-22) for supporting this work.

This manuscript has been authored by UT-Battelle, LLC, under contract DE-AC05-00OR22725 with the US Department of Energy (DOE). The US government retains and the publisher, by accepting the article for publication, acknowledges that the US government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for US government purposes. DOE will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (<http://energy.gov/downloads/doe-public-access-plan>).

References

- Foundation, T. L. (2014). *Kubernetes: Production-Grade Container Orchestration*. <https://kubernetes.io/>.
- Komiya, T., Brandl, G., & al, et. (2008). *Sphinx: Python Document Generator*. <https://www.sphinx-doc.org/>.
- Linux Foundation AI & Data. (2019). *Kedro - a Python framework for creating reproducible, maintainable and modular data science code*. <https://github.com/kedro-org/kedro>.
- LLC, L. P. (2018). *MLflow: An open source platform for the machine learning lifecycle*. <https://mlflow.org/>.
- Martindale, N., S.L. Stewart, Hite, J., & Adams, M. (2022). Design of a Scientific Data Analysis Support Platform. In Meghann Agarwal, Chris Calloway, Dillon Niederhut, & David Shupe (Eds.), *Proceedings of the 21st Python in Science Conference* (pp. 179–186). <https://doi.org/10.25080/majora-212e5952-01b>
- Pachyderm. (2014). *Pachyderm: The Leader in Data Versioning and Pipelines for MLOps*. <https://www.pachyderm.com/>.
- pre-commit Organization. (2017). *Pre-commit - a framework for managing and maintaining multi-language pre-commit hooks*. <https://pre-commit.com/>.
- Python Code Quality Authority. (2010). *Flake8 - your tool for style guide enforcement*. <https://flake8.pycqa.org/en/latest/>.
- Python Software Foundation. (2019). *Black - The Uncompromising Code Formatter*. <https://github.com/psf/black>.
- Spotify. (2012). *Luigi - a python module that helps you build complex pipelines of batch jobs*. <https://github.com/spotify/luigi>.