The Journal of Open Source Software

PyExperimenter: Easily distribute experiments and track results

Tanja Tornede [1¶], **Alexander Tornede** [1], **Lukas Fehring** [1], **Lukas Gehring** [1], **Helena Graf** [1], **Jonas Hanselle** [1], **Felix Mohr** [2], **and Marcel Wever** [3]

1 Department of Computer Science, Paderborn University, Germany **2** Universidad de La Sabana, Chia, Cundinamarca, Colombia **3** MCML, Institut for Informatics, LMU Munich, Germany ¶ Corresponding author

DOI: 10.21105/joss.05149

Software

- Review
- Repository
- Archive

Editor: Tim Tröndle

Reviewers:

- @ArsamAryandoust
- @schnorr

Submitted: 22 November 2022
Published: 20 April 2023

Summary

PyExperimenter is a tool to facilitate the setup, documentation, execution, and subsequent evaluation of results from an empirical study of algorithms and in particular is designed to reduce the involved manual effort significantly. It is intended to be used by researchers in the field of artificial intelligence, but is not limited to those.

The empirical analysis of algorithms is often accompanied by the execution of algorithms for different inputs and variants of the algorithms, specified via parameters, and the measurement of non-functional properties. Since the individual evaluations are usually independent, the evaluation can be performed in a distributed manner on an HPC system. However, setting up, documenting, and evaluating the results of such a study is often file-based. Usually, this requires extensive manual work to create configuration files for the inputs or to read and aggregate measured results from a report file. In addition, monitoring and restarting individual executions is tedious and time-consuming.

PyExperimenter adresses theses challenges by means of a single well defined configuration file and a central database for managing massively parallel evaluations, as well as collecting and aggregating their results. Thereby, PyExperimenter alleviates the aforementioned overhead and allows experiment executions to be defined and monitored with ease.

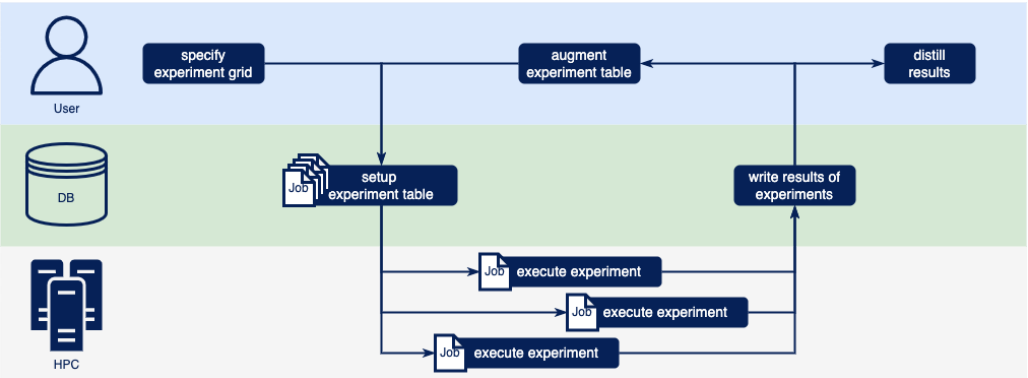

Figure 1: General schema of PyExperimenter.

A general schema of PyExperimenter can be found in Figure 1. PyExperimenter is designed based on the assumption that an experiment is uniquely defined by certain inputs, i.e.,

parameters, and a function computing the results of the experiment based on these parameters. The set of experiments to be executed can be defined through a configuration file listing the domains of each parameter, or manually through code. Those parameters define the experiment grid, based on which `PyExperimenter` sets up the table in the database featuring all experiments with their input parameter values and additional information such as the execution status. Once this table has been created, a `PyExperimenter` instance can be run on any machine, including a distributed system. Each instance automatically pulls open experiments from the database, executes the function provided by the user with the corresponding parameters defining the experiment and writes back the results computed by the function. Errors arising during the execution are logged in the database. In case of failed experiments or if desired otherwise, a subset of the experiments can be reset and restarted easily. After all experiments are done, results can be jointly exported as a Pandas DataFrame (The pandas development team, 2020) for further processing, such as generating a LaTeX table averaging results of randomized computations over different seeds.

Statement of Need

The recent advances in artificial intelligence have uncovered a need for experiment tracking functionality, leading to the emergence of several tools addressing this issue. Prominent representatives include Weights and Biases (Biewald, 2020), MLFlow (Zaharia et al., 2018), TensorBoard (Abadi et al., 2015), neptune.ai (neptune.ai, 2022), Comet.ML (Comet ML Inc., 2021), Aim (Arakelyan et al., 2022), Data Version Control (Kuprieiev et al., 2022), Sacred (Greff et al., 2017), and Guild.AI (Smith, 2019). These tools largely assume that users define the configuration of an experiment together with the experiment run itself. In case of the evaluation of different hyperparameter configurations, this process is suboptimal, since it requires to communicate the hyperparameters through scripts. This task can become cumbersome to manage as the number of configuration options and desired combinations grows and becomes more complex. Weights and Biases (Biewald, 2020), Polyaxon (Mourafiq, 2018), and Comet.ML (Comet ML Inc., 2021) allow so-called sweeps, i.e., hyperparameter optimization, albeit in a limited way. For a sweep, usually hyperparameters that should be optimized are specified along with the desired search domains, and an optimizer can be selected from a pre-defined list to carry out the optimization. However, the implementation of this functionality usually imposes several restrictions on the way the sweep can be carried out.

In contrast, `PyExperimenter` follows an inverted workflow. Instead of experiment runners registering experiments to a tracking entity such as a tracking server or database, the experiments are predefined and runners are pulling open experiments from a database. Similarly, ClearML (ClearML, 2019) and Polyaxon (Mourafiq, 2018) support a more generic workflow where experiments are first enqueued in a central orchestration server and agents can then pull tasks from the queue to execute them. However, both are much more heavyweight than `PyExperimenter` regarding the implementation of both the agents and backend-features. Moreover, they are neither completely free nor completely open-source.

In addition to the inverted workflow, a core property of `PyExperimenter` is that the user has direct access to the experiment database, which is usually not the case for alternative tools. This allows users to view, analyze and modify both the experiment inputs and results directly in the database, although not having to deal with the setup of the database itself. Sticking to available database technology further does not force the user to learn new query languages just to be able to retrieve files from a database. Furthermore, `PyExperimenter` offers some convenience functionality like logging errors and the possibility to reset experiments with a specific status such as experiments that failed.

`PyExperimenter` was designed to be used by researchers in the field of artificial intelligence, but is not limited to those. The general structure of the project allows using `PyExperimenter` for many kinds of experiments as long as they can be defined in terms of input parameters and a correspondingly parameterized function.

Acknowledgements

This work was partially supported by the German Federal Ministry for Economic Affairs and Climate Action (FLEMING project no. 03E16012F) and the German Research Foundation (DFG) within the Collaborative Research Center "On-The-Fly Computing" (SFB 901/3 project no. 160364472).

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., … Zheng, X. (2015). *TensorFlow: Large-scale machine learning on heterogeneous systems*. https://www.tensorflow.org/

Arakelyan, G., Torosyan, A., Alkamyan, A., mihran113, Hovhannisyan, V., Karapetyan, R., Gev, Karo, Aprikyan, R., Arthur, Hambardzumyan, K., Hambardzumyan, H., Manukyan, M., aramaim, Wu, J., Galstyan, K., Jiao, J., Tatevv, Yu, X., … Ulhaq, M. (2022). *Aim* (Version v3.9.3). Zenodo. https://doi.org/10.5281/zenodo.6536395

Biewald, L. (2020). *Experiment tracking with weights and biases*. https://www.wandb.com/

ClearML. (2019). *ClearML - your entire MLOps stack in one open-source tool*. https://clear.ml/

Comet ML Inc. (2021). *Comet.ML*. https://www.comet.com/

Greff, K., Klein, A., Chovanec, M., Hutter, F., & Schmidhuber, J. (2017). The sacred infrastructure for computational research. *Proceedings of the 16th Python in Science Conference*, *28*, 49–56. https://doi.org/10.25080/shinma-7f4c6e7-008

Kuprieiev, R., skshetry, Petrov, D., Redzyński, P., Rowlands, P., Costa-Luis, C. da, Schepanovski, A., Shcheklein, I., Gao, Taskaya, B., Orpinel, J., Iglesia Castro, D. de la, Santos, F., Lamy, R., Sharma, A., daniele, Berenbaum, D., Zhanibek, Hodovic, D., … Mangal, S. (2022). *DVC: Data version control - git for data & models* (Version 2.33.0). Zenodo. https://doi.org/10.5281/zenodo.7264816

Mourafiq, M. (2018). *Polyaxon: Cloud native machine learning platform*. Web page. https://github.com/polyaxon/polyaxon

neptune.ai. (2022). *Neptune: Experiment tracking and model registry*. https://neptune.ai

Smith, G. (2019). *GuildAI: Simple reproducibility in machine learning*. SysML Conference. https://guild.ai/

The pandas development team. (2020). *Pandas* (latest). Zenodo. https://doi.org/10.5281/zenodo.3509134

Zaharia, M., Chen, A., Davidson, A., Ghodsi, A., Hong, S. A., Konwinski, A., Murching, S., Nykodym, T., Ogilvie, P., Parkhe, M., & others. (2018). Accelerating the machine learning lifecycle with MLflow. *IEEE Data Eng. Bull.*, *41*(4), 39–45.