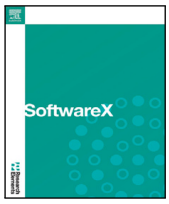
SoftwareX 31 (2025) 102298

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx



Original software publication

yProv4ML: Effortless provenance tracking for machine learning systems

G. Padovani [a,*], V. Anantharaj [b], S. Fiore [a]

[a] University of Trento, Italy
[b] Oak Ridge National Laboratory, USA

ARTICLE INFO

Keywords:
Machine learning
Provenance
yProv4ML
PROV-JSON
Provenance graph

ABSTRACT

The rapid growth in interest in deep learning and foundation models (FMs) in particular, has attracted the attention of a diverse range of researchers thanks to their generalization ability. However, the advent of these techniques has also brought to light the lack of transparency and rigor in the way development is pursued. In particular, the inability to determine the number of epochs and other hyperparameters in advance presents challenges in identifying the best model. To address this challenge, machine learning frameworks such as MLFlow can automate the collection of this type of information. However, these tools capture data using proprietary formats and pose little attention to lineage. This paper proposes yProv4ML, a framework that captures provenance information generated during machine learning processes in PROV-JSON format, with minimal code modification.

Current code version	v1.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-25-00248
Legal Code License	GPLv3
Code versioning system used	git
Software code languages, tools, and services used	Python
Compilation requirements, operating environments & dependencies	Codecarbon, Prov, Pytorch
If available Link to developer documentation/manual	https://hpci-lab.github.io/yProv4ML.github.io/
Support email for questions	gabriele.padovani@unitn.it

1. Motivation and significance

The field of machine learning has experienced a remarkable acceleration in recent years, with new findings being superseded within weeks. While this rapid pace of research undoubtedly offers many advantages, it has also led to a prevalence of works conducted with less respect to accountability and the use of computational resources. Code that is not accompanied by documentation, and results that cannot be reproduced, inevitably lack of a fundamental piece of information which impacts on the overall trustworthiness of the proposed research work [1].

The complexity of the data manipulation process, which frequently involves ad hoc and repeated transformations, further makes matters worse. Several issues may arise when attempting to reproduce these steps in order to trace the creation process. In such instances, the transformation may be lost, resulting in slight differences between the original design process and the finalized one [2]. It is becoming increasingly important to meticulously document the entirety of the design and development process, in order to facilitate comprehensive replication of experiments and prevent the introduction of unverified outcomes.

Additionally, it is often computationally expensive to determine the value of numerous hyperparameters used when training machine learning models. The most common approach is to perform multiple tests to identify the optimal parameter value [3]. With a historical record of experiments, users could look up similar targets, and identify hyperparameter values which could be ideal for their application.

A further issue with hyperparameter tuning arises from repeated attempts at training the optimal model. When the same process is iterated several times, a considerable amount of computing resources are wasted unnecessarily. Given the large size of many machine learning models, this approach quickly becomes unsustainable, particularly when dealing with deep learning architectures comprising billions of

* Corresponding author.
E-mail addresses: gabriele.padovani@unitn.it (G. Padovani), anantharajvg@ornl.gov (V. Anantharaj), sandro.fiore@unitn.it (S. Fiore).

https://doi.org/10.1016/j.softx.2025.102298
Received 14 April 2025; Received in revised form 31 July 2025; Accepted 5 August 2025
Available online 4 September 2025

parameters. Some approaches attempt to solve the parameter optimization problem during the model training phase, in an online manner [4], though this knowledge is not currently easily reusable (transferable) to similar experiments run by other researchers.

In this context, provenance — *the record of the origins, history, and transformations of data, models, and decisions*— [5] plays a crucial role. While similar approaches have been developed, both in the context of climate science and HPC domain [6], with the aim of tracking the entire workflow lineage [7] in scientific pipelines [8], this work focuses on the specific ML task, allowing for more fine-grained lineage collection. In the context of artificial intelligence, several works aimed at provenance and lineage collection are present, ranging from data-centric ones [8–10], to more model-focused ones [11]. However, many of these tools, have not found widespread adoption due to difficult integration processes [12], or due to high overhead in the collection process [13]. MLFlow's [14] lineage features, and in particular the Model Registry, enables the user to track model versions throughout the entire development lifecycle, fostering easier collaboration and reproducibility. It however lacks lineage tracking when utilizing models coming from other sources, and requires users to define an ad-hoc tag containing information about the origin of the model. This can result in fragmentation and gaps in the way lineage is reported in the model registry, as there is no standard adoption (such as W3C PROV in yProv4ML) able to address interoperability. yProv4ML offers an accessible and intuitive approach to storing information about the dataset, hyperparameters, and energy efficiency metrics. Its functionalities can be accessed in a manner analogous to MLFlow, facilitating a seamless transition. Furthermore, all content that the user elects to track will be stored in PROV-JSON [15], which has become the established standard for recording provenance artifacts. Although this format is verbose and not optimized for querying, it was chosen because PROV-JSON is human readable by default, W3C PROV compliant, as well as interoperable with several tools. A comparison of all the tools and libraries is also provided in Table A.2. The main aspects taken into consideration are whether the software is freely available, open source or paid; the aim of the tools, as well as what are the most important features.

It should be noted that the purpose of this library is not to replace ML tracking tools such as MLFlow, but rather to enrich the software ecosystem in which these modules can coexist thereby improving the software development process and researchers' understanding. To this end, a plug-in for MLFlow is currently being developed to ensure a smooth integration between the two libraries.

2. Software description

We have developed the yProv4ML library which provides access to MLFlow-analogous logging utilities offering a template-based interface for the collection and storage of provenance data, and gathers three primary categories of information: artifacts, parameters, and metrics. The first category identifies any file or output used in subsequent phases of the workflow. In the context of machine learning processes, these predominantly encompass model versions, checkpoints, and source code. In contrast, parameters represent one-time logged values utilized during the training phase. Examples of such values include the learning rate, the size and width of the model, and other hyperparameters. The final category contains information that is updated during the training process. This includes metrics such as loss values as well as statistics related to the execution of the program, such as energy efficiency, power consumption, and GPU usage.

Once data has been recorded and stored during a single run, the library enables the results of consecutive related executions to be compared. This facilitates a more comprehensive understanding of the influence of hyper-parameters and model configurations, while simultaneously maintaining an accurate record of any changes made to the entire script.

2.1. Software architecture

yProv4ML uses an ad hoc data model that efficiently encapsulates all data collected during program execution within a reduced memory footprint. The data model used in the library is shown in Fig. 1, where orange blocks indicate the core software modules, while green and blue ones display a set of products created using the yProv4ML library. The connections between modules and outputs show which software portion is responsible for the collection and writing of each respective data point.

The main modules which make up the core components are four:

- **Main module**: which wraps all functionality of the library and allows for context declaration and shutdown. It also exposes several directives to allow the user to log information and customize the yProv4ML experience;
- **Energy module**: it contains all utility functions to save energy-related metrics, such as emissions, power consumption and others;
- **System module**: contains directives to save information related to the system, such as memory or GPU usage;
- **Time module**: it contains helper functions to manipulate and save information.

In addition to the modules mentioned before, the library can also construct a provenance graph, containing all the data saved during the execution of the program. This graph encompasses all data logged by the user, as well as a set of automatically saved information, such as, among others, environment variables and required libraries. Moreover, the generated graph is fully interoperable with other libraries of the yProv framework [16], such as yProv4WFs. In case the user elects to utilize both libraries, the resulting output will be a multi-level provenance graph, comprising the workflow information at a higher level and the ML task information at a lower more detailed level.

2.2. Software functionalities

The primary objective of yProv4ML is to facilitate the extraction of provenance information from pytorch scripts in an intuitive manner. Although a secondary codebase has been translated to TensorFlow, this one has limited features and still requires further work. To achieve the aforementioned objective, the directives accessible through this library were designed to draw upon established tools such as MLFlow, thereby enabling a seamless integration between the two. The execution of any experiment starts with a call to the `start_run` function and concludes with a call to `end_run`. The following sections provide a detailed explanation of these directives.

```
prov4ml.start_run(
    prov_user_namespace: str,
    experiment_name: Optional[str] = None,
    provenance_save_dir: Optional[str] = None,
    collect_all_processes: Optional[bool] = False,
    save_after_n_logs: Optional[int] = 100,
    rank : Optional[int] = None,
)
# full implementation of the codebase
prov4ml.end_run(
    create_graph: Optional[bool] = False,
    create_svg: Optional[bool] = False,
)
```

As a design choice, yProv4ML adopts a library-instrumented approach, as opposed to a decorator-based one, which is easy-to-use and suitable for function-level instrumentation, but tends to be less flexible when applied to highly complex workflows or pipelines.

This section will present an overview of the main functions utilized for the logging of metrics and parameters within the Prov-JSON file. A subset of these functions is provided below as reference.

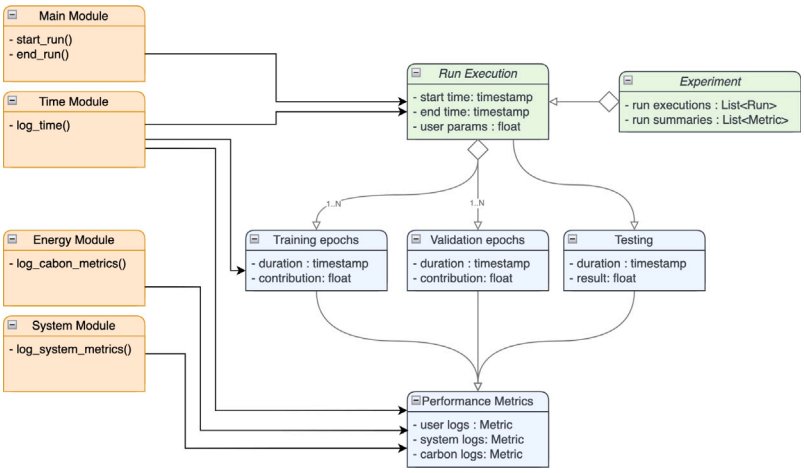


Fig. 1. Data Model used as foundation for yProv4ML, orange blocks identify code modules, blue ones identify contexts, and green ones are related to experiments and individual runs.

```
prov4ml.log_model(model_label, model)
```

The `log_model` directive is used to record information about the architecture configuration that is trained during the experiment. The majority of the metrics saved during this phase pertain to the model's memory footprint, including the number of parameters, the memory allocation used by the network, and the gradients. The complete composition of the model can also be logged in the provenance file, which includes all layers structure, data types, and input and output size.

```
prov4ml.log_param(key, value)
```

The `log_param` function enables the logging of a single parameter key–value pair, which can be stored in either program memory or temporary files. Consequently, any given value can be logged only once with that specific key.

```
prov4ml.log_metric(key, value, context, step)
```

In contrast, the `log_metric` function is used when a sequence of parameters must be stored. This is typically utilized for logging losses or other analogous metrics that fluctuate over time.

As previously stated, the context parameter is utilized to assign a specific level of importance to the metric, thereby facilitating both grouping and the construction of a hierarchical data structure. The available contexts are training, evaluation, and validation; however, additional contexts may be declared and used in a customized manner. The step parameter, on the other hand, indicates the timestamp at which the metric value will be saved, thus enabling the tracking of its behavior during the learning process.

```
prov4ml.log_system_metrics(context, step)
```

The `log_system_metrics` directive is used to record data pertaining to the memory and resources utilized by the entire program. The statistics encompass total memory usage, disk usage, GPU memory usage, and GPU usage.

```
prov4ml.log_carbon_metrics(context, step)
```

Similarly to the previous function, `log_carbon_metrics` is used to save information about system metrics from the point of view of energy efficiency. These metrics include emissions, CPU and GPU power consumed, as well as other memory statistics.

```
prov4ml.log_artifact(artifact_path, context, step,
    timestamp)
```

The `log_artifact` function is used for the purpose of saving data in file format, in particular for information utilized or generated during the program execution. In addition, the directive also allows to specify the context, timestamp and the step parameter, to inform the reader of and potential modifications to these documents.

```
prov4ml.save_model_version(model,model_name, context,
    step, timestamp)
```

The `save_model_version` directive is utilized for the purpose of saving a checkpoint of the model in PyTorch format, accompanied by the documentation of pertinent information regarding the file in question, which is then saved as an artifact.

```
prov4ml.log_current_execution_time(time_label,
    context, step)
```

At last, the `log_current_execution_time` function may be used to assign a timestamp to a designated label. This function offers a straightforward interface for recording the time required to complete specific operations, and the resulting data can be plotted in a manner analogous to any other metric.

2.3. Sample code snippets analysis

The following code snippet illustrates a potential application of the yProv4ML library with PyTorch, and can be additionally integrated with PyTorch Lightning loggers. The `prov4 ml.start_run` function is invoked to initialize the context and to maintain a record of the total execution time.

The user can then elect to record artifacts, metrics and parameters coming from the training execution, and all will be stored inside the final PROV-JSON file.

Ultimately, the context is terminated and the PROV-JSON file is stored in the designated directory via the invocation of the `prov4 ml.end_run` function. Subsequent to the file's saving, the user is allowed to transform the JSON file into an SVG image, while the identical information in DOT format is automatically saved.

```
# start the run in the same way as with mlflow
prov4ml.start_run(
    prov_user_namespace="www.example.org", # namespace of
        the user in final prov handler
```

```
    experiment_name="experiment_name", # name of the
        experiment directory
    provenance_save_dir="prov", # name of the global
        directory (containing multiple experiments)
    save_after_n_logs=100, # cache size parameter
)

mnist_model = MNISTModel()
tform = transforms.Compose([RandomRotation(10),
    ToTensor()])
# log the dataset transformation as one-time parameter
prov4ml.log_param("dataset transformation", tform)

train_ds = MNIST(PATH_DATASETS, transform=tform)
train_loader = DataLoader(train_ds,
    batch_size=BATCH_SIZE)
# log the dataset statistics as one-time parameter
prov4ml.log_dataset(train_loader, "train_dataset")

for epoch in tqdm(range(EPOCHS)):
    mnist_model.train()
    for i, (x, y) in enumerate(train_loader):
        optim.zero_grad()
        y_hat = mnist_model(x)
        loss = loss_fn(y_hat, y)
        loss.backward()
        optim.step()

        # log system and carbon metrics (once per epoch),
            as well as the execution time
        prov4ml.log_metric("MSE_train", loss.item(),
            context=prov4ml.Context.TRAINING,
            step=epoch)
        prov4ml.log_carbon_metrics(prov4ml.Context.
        TRAINING, step=epoch)
        prov4ml.log_system_metrics(prov4ml.Context.
        TRAINING, step=epoch)
    # save incremental model versions as artifacts. the
        model version is automatically added
    prov4ml.save_model_version(mnist_model,
        "mnist_model_version",
        prov4ml.Context.TRAINING, epoch)
# save the final model output
prov4ml.log_model(mnist_model, "mnist_model_final")
# save the provenance graph, as well as svg and dot
    visualization
prov4ml.end_run(create_graph=True, create_svg=True)
```

Once the function `prov4 ml.end_run` is invoked, each process generates its own provenance file containing only the data pertinent to its execution. Furthermore, all PROV-JSON files can be linked together using an additional PROV collection, which serves as a summary of the distributed execution.

3. Illustrative examples

In this use case the machine learning application is a simple MNIST [17] classification task created with the code snippet shown above. It makes use of a very small network and a single epoch of training to achieve acceptable performance levels, but different experiments could be recorded, and larger provenance graphs could be constructed. While for brevity some of the initial code in this document was shortened, the full example is available in the library's GitHub repository.[1]

Fig. 2 shows the full provenance graph from the use case, using yProv4ML. In addition, an interactive view of the same provenance graph is available at the University of Trento yProvExplorer,[2] while the JSON version can be downloaded from the yProv service,[3][4] hosted at the same institution.

Fig. 3 shows two metrics (training loss and CPU power consumption) collected at runtime in the use case using yProv4ML. These metrics are logged with a timestamp so that the user is able to decide how to collect, process and compare the recorded data. The figure on the left shows spikes in CPU usage in relation to the loss achieved by the model at the current epoch. In both cases, data is collected after each sample passes through the model. In the figure on the right, however, CPU power consumption is recorded for each sample, as well as the total energy consumed by the training, which is calculated in real time during the run.

An additional benchmark was run to calculate the overhead of using yProv4ML. Fig. 4 shows on the left a set of runs with an increasingly large amount of epochs in three different configurations. The first one is produced by running the vanilla experiments, without any logging activity or provenance tracking, with only the model weights saved incrementally at each epoch. The second set of runs saves data using the default yProv4ML configuration, which includes inputs and outputs, model weights, and a minimal set of metrics. The final configuration collects all system metrics in addition to the default set of provenance information. Fig. 4(a) displays the full execution time taken by each configuration, with the latter being the heaviest one. On the other hand, Fig. 4(b) shows the RAM usage percentage for the same three configurations of experiments, in the context of a fifteen epoch training. In this case, the caching system helps both the latter configuration, displaying minimal differences when running with and without provenance collection.

4. Impact

The usefulness of yProv4ML is evident when it comes to monitoring energy consumption processes. A preliminary testing phase can reveal which component of the program exerts the greatest influence, thereby facilitating a reduction in overall consumption during the subsequent training. The ability to log several versions of the same experiment is also crucial for understanding which hyper-parameters work better with the current execution, and avoiding repeating the same mistakes over several runs.

Due to the word limit, a few use cases will be briefly discussed in this section, but more comprehensive information and context on these workflows can be found in the referenced papers.

For instance, yProv4ML was exploited to assess the outcome of a series of ML benchmarking experiments ran on Frontier at Oak Ridge National Laboratory (ORNL) [18]. In this context, the library was assessed on both single-node and multi-node experiments, considering in the latter case, up to 512 GPUs in parallel. The benchmarks were designed to understand the quantity of computation, data, and parameters required for optimal training of a series of foundation models. The capability to readily save a substantial number of provenance artifacts has proven to be of great importance.

Similarly, yProv4ML was used to gather data on a machine learning process designed to predict the occurrence and behavior of tropical cyclones [19,20]. In this instance, the library was used to log information about the graph neural network which was being developed.

The library is undergoing additionally testing in various domains (i.e., Climate, Earth Observation, High Energy Physics, Cybersecurity) and in collaboration with several teams worldwide to disseminate,

[1] https://github.com/HPCI-Lab/yProvML/tree/main/swx

[2] https://explorer.yprov.disi.unitn.it/?file=http%3A%2F%2Fyprov.disi.unitn.it%3A3000%2Fapi%2Fv0%2Fdocuments%2Fsoftwarex

[3] http://yprov.disi.unitn.it:3000/api/v0/documents/softwarex

[4] https://hpci-lab.github.io/yProv4ML.github.io/prov_graph.html

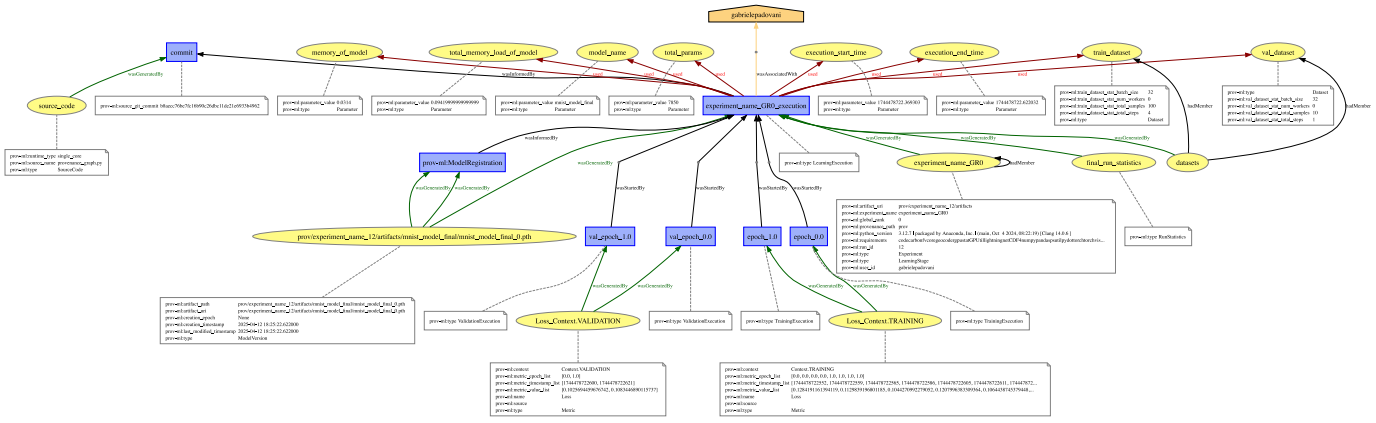


Fig. 2. Provenance graph of the ML training process specified in Section 3 (Illustrative examples). The figure is available in svg format at https://github.com/HPCI-Lab/yProvML/blob/main/swx/provgraph_experiment_name_GR0.svg.

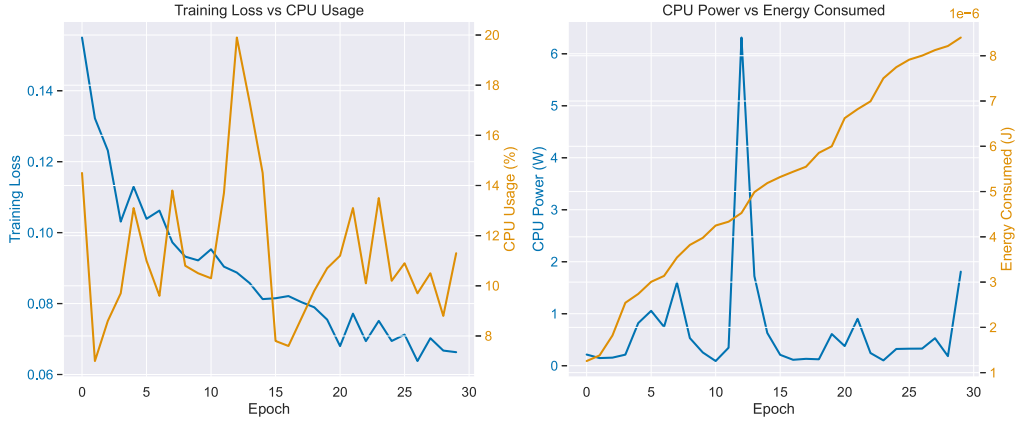
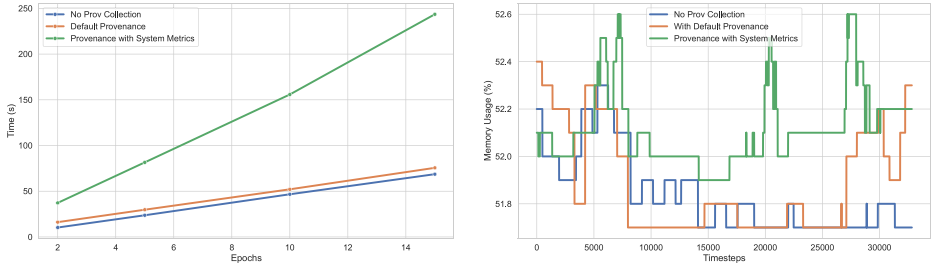


Fig. 3. Left: the training loss is recorded for every epoch using yProv4ML directives and compared to the CPU usage during training. Right: CPU power consumption is recorded for every batch and compared with the total energy consumed during the same training process.



(a) Time (y axis) taken for running a full trainings for 2, 5, 10, 15 epochs (x axis)

(b) RAM usage percentage (y axis) when running a full trainings for 15 epochs (x axis)

Fig. 4. Statistics taken in completing a MNIST digit classifier training when collecting default provenance (orange), provenance with system metrics (green), and no provenance (blue).

exploit and extend the work. Besides the open source license (GPLv3) and the current documentation, the authors plan to (i) provide in the future training material (i.e., videos/demos) and notebooks as well as (ii) increase the link with external software belonging to the "same scientific ecosystem", which can complement the yProv4ML support, such as MLFlow and RO-Crate [21].

5. Conclusions and future work

In this paper, we propose yProv4ML, a library that serializes provenance information collected during a machine learning process in JSON format, adhering to the W3C PROV family of standards. yProv4ML provides users with a transparent and user-friendly tool capable of

Table A.2
Comparison of the main ML and workflow provenance tools.

Tool	Type	Features	Open source	Focus
MLflow	Experiment Tracking	Logs metrics, parameters, models	Yes	Tracks runs, parameters, artifacts, models
MLflow2prov	Provenance Exporter	Convert MLflow to W3CPROV	Yes	Interoperable, W3C-compliant provenance
WandB	Online Experiment Tracking	Dashboards, artifact versioning	Paid	Experiment logs, metrics, artifacts
DVC	Data Versioning	Git-like versioning	Yes	Prov. data, model, pipeline
ProvLake	Workflow Prov. Manager	Captures provenance across tools	Yes	W3C-PROV-based, cross-tool provenance
OpenLineage	Metadata Lineage Collection	Event tracking, pipelines and metadata	Yes	Real-time data and job lineage
DLProv	DL Workflow Analyses	Suite-agnostic W3C PROV-compliant	Yes	Data-centric support for DL workflows
Braid-DB	Provenance Collection	Low Overhead tracking	Yes	Model training, inputs, learning
ModelKB	DL Model Tracking	Automatically saves metadata	Yes	Integrated with TF and Keras
yProv4ML	Provenance Tracking	Captures data and ML Prov.	Yes	W3C-PROV-based, cross-tool provenance

translating the collected information into the PROV-JSON standard format. Although we believe this tool to be a substantial step towards intuitive and lightweight ML provenance collection, additional fundamental features will be addressed in future development. In particular, we will focus on making the system and carbon metrics collection methods more lightweight, as well as on reducing the library overhead in general.

Furthermore, future development will target the reproducibility aspect of machine learning, enabling experiments to be easily reproduced starting from a single PROV-JSON file.

CRediT authorship contribution statement

G. Padovani: Writing – original draft, Visualization, Validation, Software, Methodology. **V. Anantharaj:** Writing – review & editing, Validation. **S. Fiore:** Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was partially funded under the National Recovery and Resilience Plan (NRRP), Mission 4 Component 2 Investment 1.4 - Call for tender No. 1031 of 17/06/2022 of Italian Ministry for University and Research funded by the European Union – NextGenerationEU (proj. nr. CN_00000013) and the EU InterTwin project (Grant Agreement 101058386). Moreover this research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

Appendix. Provenance tools comparison

See Table A.2.

References

[1] Semmelrock H, Kopeinik S, Theiler D, Ross-Hellauer T, Kowald D. Reproducibility in machine learning-driven research. 2023, arXiv preprint arXiv:2307.10320.

[2] Alahmari SS, Goldgof DB, Mouton PR, Hall LO. Challenges for the repeatability of deep learning models. IEEE Access 2020;8:211860–8.

[3] Isdahl R, Gundersen OE. Out-of-the-box reproducibility: A survey of machine learning platforms. In: 2019 15th International conference on eScience. IEEE; 2019.

[4] Balaprakash P, Salim M, Uram TD, Vishwanath V, Wild SM. Deephyper: Asynchronous hyperparameter search for deep neural networks. In: 2018 IEEE 25th international conference on high performance computing. IEEE; 2018, p. 42–51.

[5] Cheney J, Chong S, Foster N, Seltzer M, Vansummeren S. Provenance: a future history. In: Proceedings of the 24th ACM SIGPLAN conference companion on object oriented programming systems languages and applications. 2009, p. 957–64.

[6] Souza R, Skluzacek TJ, Wilkinson SR, Ziatdinov M, da Silva RF. Towards lightweight data integration using multi-workflow provenance and data observability. In: 2023 IEEE 19th international conference on e-science. Limassol, Cyprus: IEEE; 2023, p. 1–10.

[7] Souza R, Caino-Lores S, Coletti M, Skluzacek TJ, Costan A, Suter F, et al. Workflow provenance in the computing continuum for responsible, trustworthy, and energy-efficient AI. In: 2024 IEEE 20th international conference on e-science. IEEE; 2024, p. 1–7.

[8] Chapman A, Lauro L, Missier P, Torlone R. Supporting better insights of data science pipelines with fine-grained provenance. ACM Trans Database Syst 2024;49(2):1–42.

[9] Barreto Simedo Pacheco L, Rahman M, Rabbi F, Fathollahzadeh P, Abdellatif A, Shihab E, et al. DVC in open source ML-development: The action and the reaction. In: Proceedings of the IEEE/ACM 3rd international conference on AI engineering-software engineering for AI. 2024, p. 75–80.

[10] Wozniak JM, Liu Z, Vescovi R, Chard R, Nicolae B, Foster I. Braid-db: Toward ai-driven science with machine learning provenance. In: Smoky mountains computational sciences and engineering conference. Springer; 2021, p. 247–61.

[11] Gharibi G, Walunj V, Rella S, Lee Y. ModelKB: towards automated management of the modeling lifecycle in deep learning. In 2019 IEEE/ACM 7th international workshop on realizing artificial intelligence synergies in software engineering (RAISE). 2019.

[12] Pina D, Chapman A, Kunstmann L, de Oliveira D, Mattoso M. Dlprov: A data-centric support for deep learning workflow analyses. In: Proceedings of the eighth workshop on data management for end-to-end machine learning. 2024, p. 77–85.

[13] Schlegel M, Sattler K-U. MLflow2PROV: extracting provenance from machine learning experiments. In: Proceedings of the seventh workshop on data management for end-to-end machine learning. 2023, p. 1–4.
[14] Zaharia M, Chen A, Davidson A, Ghodsi A, Hong SA, Konwinski A, et al. Accelerating the machine learning lifecycle with mlflow. IEEE Data Eng Bull 2018;41(4):39–45.
[15] Niu X, Glavic B, Gawlick D, Liu ZH, Krishnaswamy V, Radhakrishnan V. Interoperability for provenance-aware databases using {PROV} and {JSON}. In: 7th USeNIX workshop on the theory and practice of provenance. 2015.
[16] Fiore S, Rampazzo M, Elia D, Sacco L, Antonio F, Nassisi P. A graph data model-based micro-provenance approach for multi-level provenance exploration in end-to-end climate workflows. In: 2023 IEEE international conference on big data (bigData). IEEE; 2023, p. 3332–9.
[17] Deng L. The mnist database of handwritten digit images for machine learning research. IEEE Signal Process Mag 2012;29(6):141–2.
[18] Anantharaj V, Kurihana T, Dash S, Padovani G, Fiore S. Exploring vision transformers on the frontier supercomputer for remote sensing and geoscientific applications. In: IGARSS 2024-2024 IEEE international geoscience and remote sensing symposium. IEEE; 2024, p. 3085–8.
[19] Padovani G, Anantharaj V, Sacco L, Kurihana T, Bunino M, Tsolaki K, et al. A software ecosystem for multi-level provenance management in large-scale scientific workflows for AI applications. In: SC24-w: workshops of the international conference for high performance computing, networking, storage and analysis. IEEE; 2024.
[20] Padovani G, Anantharaj V, Fiore S. Provenance tracking in large-scale machine learning systems. In: 54th international conference on parallel processing. ACM; 2025.
[21] Soiland-Reyes S, Sefton P, Crosas M, Castro LJ, Coppens F, Fernández JM, et al. Packaging research artefacts with RO-crate. Data Sci 2022;5(2):97–138.