PINNProv: Provenance for Physics-Informed Neural Networks

Lyncoln S. de Oliveira[1,2], Liliane Kunstmann[1], Débora Pina[1], Daniel de Oliveira[2] and Marta Mattoso[1]
[1]Federal University of Rio de Janeiro (COPPE/UFRJ)
[2]Institute of Computing - Universidade Federal Fluminense (IC/UFF)
Email: {oliveiral, lneves, dbpina, marta}@cos.ufrj.br, danielcmo@ic.uff.br

***Abstract*—Machine Learning is being used increasingly in different application areas. Physics-Informed Neural Networks (PINN) stand out, adapting neural networks to predict solutions to Physics phenomena. Incorporating Physics knowledge into the loss function of a neural network, PINNs revolutionize the solutions of partial differential equations. Considering the lack of support for analytics and reproducibility of the trained models, in this paper we propose the capture and use of provenance data, aimed at the analysis of PINN models. We conducted experiments using TensorFlow and DeepXDE, in a high-performance computing environment. Our experiments show the contributions of these provenance queries in different PINN applications.**

***Index Terms*—provenance, physics-informed neural network**

I. Introduction

Deep neural networks (DNNs) have been effectively applied to a variety of problems [1] to assist decision-making or predict the future behavior of new data [2]. One of the recent approaches for DNNs is Physics-Informed Neural Networks (PINNs) which are revolutionizing the approaches to problems governed by partial differential equations (PDEs) in Science and Engineering [3]. The Physics is informed during training by adding new components to the loss function, reflecting, for example, the residue of the PDE and its boundary conditions.

In a recent survey, Cuomo *et al.* [4] present current challenges for PINN implementation like boundary conditions management, PINN architecture design, and optimization aspects. PINNs are computing intensive and require thousands of epochs to converge, using supercomputers, GPUs, and tools like Horovod to manage the parallel execution [5]. PINNs can be designed and trained using classic DNN libraries such as TensorFlow and PyTorch. However, tools like AutoML [6] are incipient for PINNs, which makes PINN model selection quite complex. In addition, these DNN libraries do not automatically register and track the Physics components of the loss function. These values are essential to evaluate the accuracy of the trained model. To analyze hyperparameters with these new PINN loss components, users are required to manually define configurations to capture, register and integrate them into the other metrics [7]. For instance, to track the values of the loss components, the PINN experts have to configure tools such as TensorBoard, using the TensorFlow data summarization routines. This configuration requires programming a script to extract and aggregate information from different sources and persist them in CSV files. Furthermore, in choosing the best PINN model, it is necessary to plan the organization of these files/directories of several model metrics and configurations. Aggregating these data for querying over different PINN models is far from trivial. In addition, users could create their own data representation, generating heterogeneity that leads to extra work to compare trained models.

Supporting humans in the interactive process of creating large-scale DNN models is necessary, according to [7]–[9]. In the case of PINNs, users typically steer the execution, and this support is even more important. One way to help track the PINN's training process is to use a provenance service, to capture the hyperparameters and PINN metrics, persist data with a DBMS, and run queries. Provenance data describes the training execution by relating data, process, parameters, hyperparameters, and metrics following a standard schema. However, most provenance systems are not prepared to manage concurrent provenance calls from distributed and parallel sources in a supercomputer or cloud environment [10]. DNNProv [11] is a library for provenance data services in DNNs, W3C PROV[1] compliant, that allows for extensibility. The provenance data captured by DNNProv during training can support the analysis of data, metadata, and the data derivation path. Based on queries on aggregated data at runtime, it complements graphic tools and integrates trained data from different hyperparameter configurations. DNNProv's source code can be accessed at *https://github.com/dbpina/dnnprov*. However, DNNProv does not represent PINN components or work with specific PINN development frameworks.

Preliminary evaluations of using DNNProv for PINNs are presented in [12] using a grid cluster of CPU-GPU hybrid computing mode and in [13] using GPU in a personal computer. However, programming this extensibility for PINNs is not simple and can lead to heterogeneity in this representation.

We present PINNProv[2] as an extension of DNNProv to facilitate the management of provenance data in PINN scripts. PINNProv helps the user interaction when informing which are the input hyperparameters, loss function components, and output metrics related to the PINN's training. This work aims to address the problem of model evaluation in PINNs using different frameworks and on a larger scale. We explore the generation of provenance data and its use through queries to monitor the evolution of PINN training, complementing

[1]https://www.w3.org/TR/prov-overview/
[2]https://github.com/lyncoln/PINNprov

979-8-3503-8160-3/23/$31.00 ©2023 IEEE
DOI 10.1109/SBAC-PADW60351.2023.00013

typical visualization tools. Our experiments show the capability of PINNProv to represent provenance and PINN metrics in frameworks like TensorFlow, which is popular for training DNNs, and DeepXDE, which is designed specifically for PINNs. We explore solving PDEs in different problems, training PINNs with different architectures, and varying their complexity at the Santos Dumont supercomputer[3].

This paper is structured as follows. Section II presents theoretical concepts about PINNs. Section III discusses the use of provenance in PINNs. Section IV shows PINNProv supporting two experiments with different training frameworks. Section V discusses the existing approaches for provenance data capture in PINNs and ML. Section VI concludes this paper.

II. Physiscs informed neural networks - PINNs

Proposed by Raissi *et al.* [3], PINNs are neural networks trained to solve supervised learning tasks while respecting any given law of Physics described by general nonlinear PDEs. PINNs can effectively solve direct and inverse problems associated with PDEs, as shown in Equation 1.

$$\begin{aligned} \mathcal{F}(\boldsymbol{u}(\boldsymbol{z});\gamma) &= \boldsymbol{f}(\boldsymbol{z}) \quad \boldsymbol{z} \in \Omega \\ \mathcal{B}(\boldsymbol{u}(\boldsymbol{z})) &= \boldsymbol{g}(\boldsymbol{z}) \quad \boldsymbol{z} \in \partial\Omega \end{aligned} \tag{1}$$

The domain is defined by $\Omega \subset \mathbb{R}^d$ with boundary $\partial\Omega$. The vector $\boldsymbol{z}$ informs the space-time coordinates, $\boldsymbol{u}$ represents the unknown solution, and γ is the set of parameters related to Physics. The function $\boldsymbol{f}$ is responsible for identifying the problem input data and $\mathcal{F}$ is the non-linear differential operator. The $\mathcal{B}$ operator indicates the initial or boundary conditions related to the problem and $\boldsymbol{g}$ the boundary function. Equation 1 describes both direct and inverse physical problems. Direct problems aim to find the function $\boldsymbol{u}$ for all $\boldsymbol{z}$ while γ is the set of Physics-specific parameters. As for the inverse problems, γ is also determined from the data [4].

In PINNs, $\boldsymbol{u}(\boldsymbol{z})$ is estimated using a DNN with a set of parameters $\boldsymbol{\theta}$ so that $\hat{\boldsymbol{u}}_\theta(\boldsymbol{z}) \approx \boldsymbol{u}(\boldsymbol{z})$. PINNs approximate PDE solutions by training a DNN that aims to minimize a loss function that incorporates physical knowledge of the problem, such as terms that reflect the boundary conditions, domain, and PDE residence at selected points in the domain.

Inspired by classical DNNs [14], in Figure 1 we propose a specification for the PINN life cycle. The PINNs life cycle also involves model configuration, data preparation, split sets into training, validation and test sets, start training, testing within an interval, and making steering (tune) actions during training. Similarly, the user also adjusts the model and repeats the whole process until the results are satisfactory.

Our adaptation of the classic life cycle for PINNs starts with modeling the problem, which is governed by a function $f(x)$ representing a PDE, and determining the way Physics is going to be constrained into the NN. We represent the process of "Inform Physics", by including the activation functions that best serve the problem, defining boundary and initial conditions, the methods to generate collocation points, loss function components, optimizer, etc. Since the problems that PINNs aim to predict do not have large amounts of data, the user might use methods to generate and regularize data through mathematical simulations. In the regularization process, both the inputs and the regularization methods impact the model. In the data preparation process, PINNs may require using real data (data from the problem environment) or/and regularizing data using the governing PDE and/or boundary and initial conditions. The raw data format is often binary or domain-specific, so there is a data preparation process, where the data can go through a series of transformations to a format that fits and serves the model training better. This data preparation might impact the model results, so provenance helps to keep track of the whole process to allow posterior analysis and reproducibility.

[3]https://sdumont.lncc.br/

III. PINNProv: Provenance for PINNs

Provenance in DNN life cycle is an important asset for reproducibility, interpretability and contributes to the quality and reliability of the model [15]–[20]. Provenance associates the data (entities) with the algorithms/programs (activities) that transform these data, as well as the agents (humans, teams) associated with the entities and activities. These relationships allow for tracing the model back to its data preparation activities [21]. However, PINNs have more complex artifacts, like the loss function coefficients, which need to be represented and tracked.To the best of our knowledge, there is a lack of provenance support for PINNs.

PINNs work with multi-coefficient loss functions, generation methods for the collocation points and data, and other metrics that are not trivial to represent. Depending on the complexity of the PDEs, PINNs can incorporate two NNs to find solutions. In the absence of specific approaches to collect and analyze provenance in PINNs, users have to save data into files or adapt frameworks/libraries to collect it automatically, but those are time-consuming tasks that require manual effort, the data saved can be limited by the framework that has its own data representation which increases the complexity for data aggregation and analysis.

In addition to adding quality and reproducibility to the trained models, provenance traces and metadata can help to monitor the training, if provenance is made available during the training. This monitoring, enhanced by querying aggregated data, helps the user to assess intermediate results and gain insight to make adaptations at runtime [7]. Provenance represents complex relationships of the life cycle artifacts, complemented with model metrics and epochs, the chosen hyperparameter values, environment configuration, components of the loss function that guide the Physics, their weights, *etc.*

A. Design of PINNProv

PINNProv aims to bolster provenance support for the PINNs' life cycle. PINNProv extends DNNProv's default metrics provenance to capture PINN metrics. PINNProv keeps the W3C PROV-compliance, being able to share, persist, and analyze captured provenance data. Using PINNProv, the

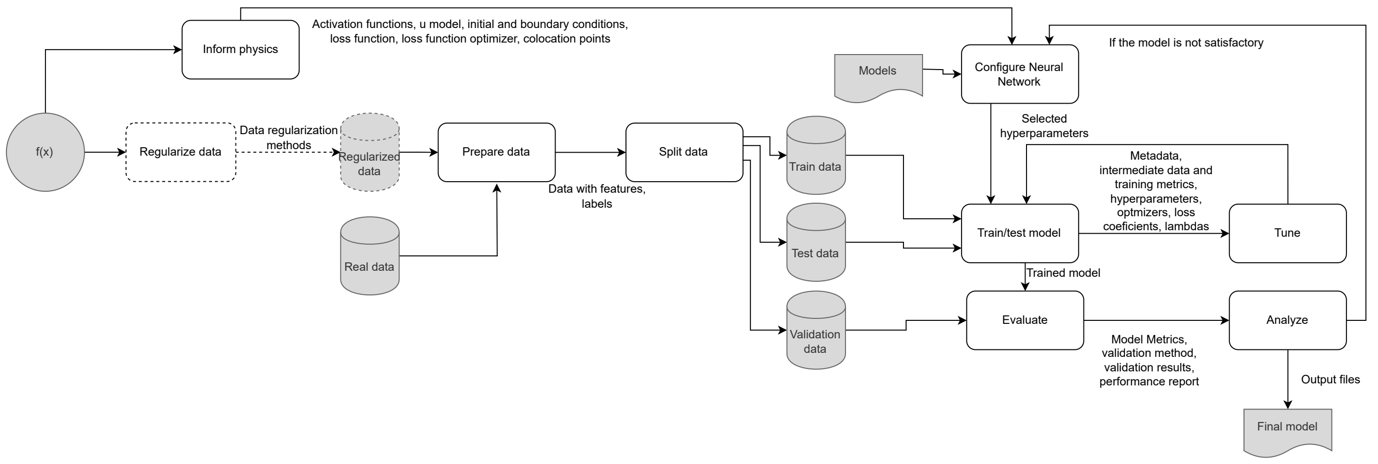

Inform physics
Activation functions, u model, initial and boundary conditions, loss function, loss function optimizer, colocation points
If the model is not satisfactory
Models
Configure Neural Network
f(x)
Regularize data
Data regularization methods
Regularized data
Prepare data
Split data
Train data
Selected hyperparameters
Metadata, intermediate data and training metrics, hyperparameters, optmizers, loss coeficients, lambdas
Data with features, labels
Real data
Test data
Train/test model
Tune
Trained model
Validation data
Evaluate
Analyze
Model Metrics, validation method, validation results, performance report
Output files
Final model

Fig. 1: Physics Informed Neural Networks life cycle. The grey shapes are static objects that might be outputs or inputs of activities, and the white rectangles are activities (i.e. data transformation). The text on the edges is the possible output from an activity that may be input for the next activity. The dotted components indicate optional activities/outputs.

user only needs to pass through pre-defined lists which are the hyperparameters and evaluation metrics to be captured during the PINN training, particularly the loss of the boundary conditions, residuals of the PDE, and other parameters relevant to fine-tuning PINNs. In addition to these typical PINN data, PINNProv can capture other PINN data in its PROV schema. PINNProv services can be invoked from PINN training frameworks using callback functions.

B. DeepXDE and PINNProv Integration

DeepXDE [22] is a Python library for scientific machine learning and PINNs. It aims to serve both as an educational tool and as a research tool for problem-solving in Computer Science and Engineering. Its implementation is simple, which justifies its popularity among users developing PINNs. DeepXDE is currently used with five training libraries in its *back-end*: TensorFlow, PyTorch, JAX, and PaddlePaddle.

DeepXDE does not have a native solution for persisting provenance data during the training of its models. PINNProv can fill this gap without changing DeepXDE code. To do so, one must simply create a class object inheriting the *deepxde.callbacks.Callback* class. In this module, PINNProv can use DeepXDE methods that are started during the key steps of the PINN model training. The following code is an example of how this instrumentation is done. For example, the code statement to be executed at the beginning of model training (*on_train_begin*); at the end of training (*on_train_end*); and at the end of each epoch (*on_epoch_end*). These steps identify the places where the provenance data needs to be defined and captured by PINNProv, as they evolve along the training. When initializing the *callback* class, given by the *__init__* method, PINNProv starts and sets provenance data to be captured like the hyperparameters, weights for the PINN components of the loss function, and output metrics like the values for each component of the loss function during the training.

```
class PINNProv_calls(deepxde.callbacks.Callback):
    def __init__(self):
        super().__init__()
        self.epoch = 0
        self.dataflow_tag = "deepxde"
        self.exec_tag = self.dataflow_tag + datetime.now()
        df = Dataflow(self.dataflow_tag,
        ['STR_OPTIMIZER_NAME', 'NUM_LEARNING_RATE',
        'NUM_EPOCHS', 'NUM_BATCH_SIZE', 'STR_LAYERS',
        'NUM_WEIGHT_LR', 'NUM_WEIGHT_LB', 'NUM_WEIGHT_LD'],
        ['NUM_epoch', 'NUM_time_elapsed', 'NUM_LOSS',
        'NUM_LR_train', 'NUM_LB_train', 'NUM_LD_train',
        'NUM_Q_train_error', 'NUM_U_train_error'])
        df.save()
    def on_train_begin(self):
        (...)
        self.t1 = Task(1, self.dataflow_tag,
        self.exec_tag, "TrainingModel")
        self.tf1_input = DataSet("iTrainingModel",
        [Element([opt_name, learning_rate, epoch, batch,
        layers_list, weight_lr, weight_lb, weight_ld])])
        self.t1.add_dataset(self.tf1_input)
        self.t1.begin()
    def on_train_end(self):
        self.t1.end()
    def on_epoch_end(self):
       if(self.epoch % 10 == 0):
          (...)
          tf1_output = DataSet("oTrainingModel",
          [Element([epoch, elapsed, loss_value, lr_value,
          lb_value, ld_value, err_q_train, err_u_train])])
          self.t1.add_dataset(tf1_output)
          self.t1.save()
       self.epoch += 1
```

At the DeepXDE method *on_train_begin*, PINNProv persists the values referring to the provenance input entity named *iTrainingModel*. At the *on_epoch_end* method, PINNProv persists the values regarding the training output metrics in the provenance output entity named *oTrainingModel*. PINNs are generally trained for an extensive number of epochs, the user can customize the interval that they want to persist the output metrics, *e.g.* at every 10 epochs. Finally, in *on_train_end*, PINNProv finalizes the provenance capture related to the training activity.

With the *callback* type object created, PINNProv is then ready to be used with DeepXDE for any PINN specification and in any of its five development *back-ends*. The user just has to include PINNProv in the callback parameter of the DeepXDE model train method: model.train(iterations, callbacks=[PINNProv_calls()],batch_size). Similarly, PINNProv can be integrated into other frameworks that accept *callbacks*.

Alternatively, the user can build the object of type *callback* by persisting the data of each model using, for example, the Pandas library. However, the organization of the files would be up to the user and would not have a pre-defined standard representation. The user would also have to create a new *script* to associate the files that persist the data related to the input hyperparameters and output metrics of each model configuration for comparative analyses. Aggregating data from these files is not simple, and following relationships is even harder without IDs, joins and a query language.

IV. Experimental Evaluation

We used the Santos Dumont supercomputer (SDumont), which has approximately 5.1 Petaflop/s of processing capacity, using a hybrid configuration of computational nodes that incorporate different parallel processing architectures. SDumont has a Lustre parallel file system, with a raw storage capacity of about 1.7 PB. In our experiments, we used one partition named GDL, which is specialized for artificial intelligence tasks. GDL has two processors Intel Xeon Skylake Gold 6148 2,4GHz with 20 cores, eight GPUs NVIDIA Tesla V100-16GB with NVLink, and 384GB of RAM.

A. PINN Eikonal

PINNProv was used with a PINN that solves the Factored Eikonal Equation (FEE) [12], which describes phenomena like wave propagation for acoustic and elastic media [23]. Figure 2 shows that two neural networks were used, one to estimate the transit time (TT), related to the direct problem, and the other to estimate the wave propagation velocity (Vel), related to the inverse problem. These networks are indirectly connected for the calculation of the loss function.

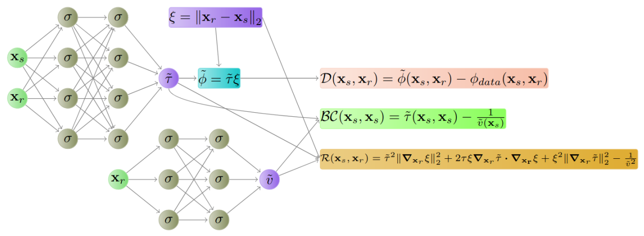

$\xi = \|\mathbf{x}_r - \mathbf{x}_s\|_2$
$\mathbf{x}_s$
$\mathbf{x}_r$
σ
$\tilde{\tau}$
$\tilde{\phi} = \tilde{\tau}\xi$
$\mathcal{D}(\mathbf{x}_s, \mathbf{x}_r) = \tilde{\phi}(\mathbf{x}_s, \mathbf{x}_r) - \phi_{data}(\mathbf{x}_s, \mathbf{x}_r)$
$\mathcal{BC}(\mathbf{x}_s, \mathbf{x}_s) = \tilde{\tau}(\mathbf{x}_s, \mathbf{x}_s) - \frac{1}{\tilde{v}(\mathbf{x}_s)}$
$\mathcal{R}(\mathbf{x}_s, \mathbf{x}_r) = \tilde{\tau}^2\|\nabla_{\mathbf{x}_r}\xi\|_2^2 + 2\tau\xi\nabla_{\mathbf{x}_r}\tilde{\tau} \cdot \nabla_{\mathbf{x_r}}\xi + \xi^2\|\nabla_{\mathbf{x}_r}\tilde{\tau}\|_2^2 - \frac{1}{\tilde{v}^2}$
$\tilde{v}$

Fig. 2: PINN Eikonal scheme, where $\phi(\mathbf{x}_s, \mathbf{x}_r)$ represents the transit times, and $\upsilon(\mathbf{x}_r)$, the propagation speed of the wave in the acoustic medium. They are approximated by two different neural networks, their approximations are denoted by $\tilde{\phi}(\mathbf{x}_s, \mathbf{x}_r)$ and $\tilde{\upsilon}(\ \mathbf{x}_r)$ respectively. Those approximations are then fed to the loss components related to the data assimilation, boundary and initial conditions, and the PDE residual [12].

The input dataset refers to seismic and ground-penetrating radar data. The metric is R^2, which is known as the coefficient of determination. The weights w of the loss function are defined as in Equation 2. The loss function $\mathcal{L}$ has three components related to $\mathcal{L}_{\mathcal{D}}$ the data assimilation, $\mathcal{L}_{\mathcal{BC}}$ as boundary and initial conditions, and $\mathcal{L}_{\mathcal{R}}$ as the PDE residual.

$$\mathcal{L}(\boldsymbol{\theta}; \mathcal{T}) = \mathcal{L}_{\mathcal{R}}(\boldsymbol{\theta}; \mathcal{T}_{\mathcal{R}}) + \mathcal{L}_{\mathcal{BC}}(\boldsymbol{\theta}; \mathcal{T}_{\mathcal{BC}}) + 25 \cdot \mathcal{L}_{\mathcal{D}}(\boldsymbol{\theta}; \mathcal{T}_{\mathcal{D}}) \quad (2)$$

The user assigned values for the hyperparameters, based on past experiments. The hyperparameters batch size = 3000, number of epochs = 400000, initial learning rate = 0.001, and final learning rate = 0.000018 were used with the exponential decay function, with decay = 0.99.

B. Provenance queries

PINNProv assigns an identifier (ID) for the configurations of hyperparameters that are being used to train the PINN. It captures the provenance of the training output information for each configuration, persisting the data at the database at an interval of 100 epochs. The experiment was divided into two rounds that evaluated five and four configurations, respectively. In the first round, the user explores different optimizers and activation functions for each neural network, whereas, in the second round, the user varies the number of neurons and intermediate layers. The idea is that at the end of each round, the user fine-tunes the hyperparameters to improve the performance of the model evaluation metric R^2 and the loss function. During each round, the user queries the provenance database to assess the performance of each configuration. They can aggregate data that complement visual analyses and yet are not trivial to be examined in typical graphical tools like TensorBoard. The following queries show a few examples.

Q1: What are the configurations for the trained PINNs?
Q2: What are the largest values of R^2, and what is the epoch and time taken to obtain them?
Q3: What are the lowest values of the loss function and its components, what is the epoch and time it took?

Q1 shows some of the hyperparameters that are being used in the two networks to train the PINN for each configuration. Q2 and Q3 assist the user in identifying the highest R^2 and lowest loss values, respectively, obtained during the training, as well as the time taken to reach these values. These queries can help to analyze the trade-off between training time and performance of the configurations, so they can drop or adjust the configurations for the next round, or even abort the execution since the queries can be submitted during the training.

C. Monitoring PINN Eikonal with provenance

By analyzing the first round results presented in Tables I, II, III, along with Figures 3 and 4, the user can gain a few insights. For instance, Q2 highlights that configuration 3 took almost as much time as configuration 5 but produced a much lower R^2. This may indicate that Adam with Sigmoid could be discarded in future rounds. Q3 shows details of the PINN loss, which is the main metric for inverse problems.

TABLE I: Q1 results with the configurations for the training

ID	Optimizer	Activation TT	Activation Vel	Neurons TT	Neurons Vel	Intermediate Layers TT	Intermediate Layers Vel
1	Adam	tanh	tanh	20	32	8	4
2	Adam	relu	relu	20	32	8	4
3	Adam	sigmoid	sigmoid	20	32	8	4
4	Adam	relu	tanh	20	32	8	4
5	RMSProp	tanh	tanh	20	32	8	4

TABLE II: Q2 results ordered by the largest values of R^2 with its epoch and time taken to obtain them

ID	Max R^2	Epoch	Time (m)
5	0.470	398600	100.912
1	0.377	88000	22.293
3	0.105	397200	100.756
2	0.093	86300	20.165
4	0.077	3400	0.790

TABLE III: Q3 results ordered by the lowest values of the loss function and its components, with its epoch and time taken

ID	$\mathcal{L}$	$\mathcal{L}_{\mathcal{R}}$	$\mathcal{L}_{\mathcal{BC}}$	$\mathcal{L}_{\mathcal{D}}$	Epoch	Time (m)
5	$4.503 \cdot 10^{-5}$	$3.727 \cdot 10^{-5}$	$4.066 \cdot 10^{-10}$	$3.102 \cdot 10^{-7}$	399900	101.241
1	$1.283 \cdot 10^{-4}$	$3.692 \cdot 10^{-5}$	$3.771 \cdot 10^{-8}$	$3.654 \cdot 10^{-6}$	398100	100.852
4	$2.229 \cdot 10^{-4}$	$1.252 \cdot 10^{-4}$	$7.298 \cdot 10^{-9}$	$3.910 \cdot 10^{-6}$	359500	83.584
2	$3.290 \cdot 10^{-4}$	$1.677 \cdot 10^{-4}$	$1.527 \cdot 10^{-9}$	$6.449 \cdot 10^{-6}$	364000	85.055
3	$3.636 \cdot 10^{-3}$	$5.861 \cdot 10^{-4}$	$2.212 \cdot 10^{-8}$	$1.220 \cdot 10^{-4}$	343700	87.185

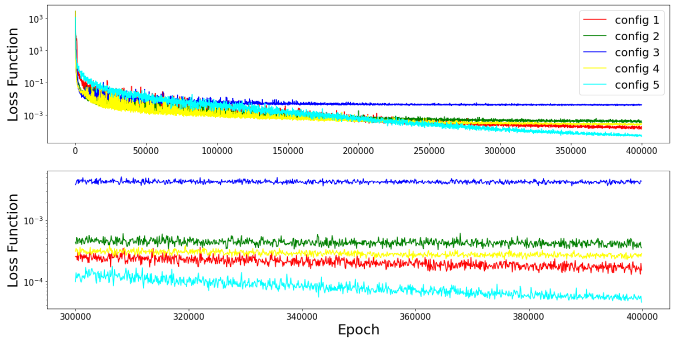


Fig. 4: Loss function value for all epochs (Upper); Focus on the last 100,000 epochs (Lower). Regarding the first round.

In the second round, the user maintains the hyperparameters related to configurations 1 and 5, but changes the topology of the TT and Vel networks, doubling the number of intermediate layers and adjusting the neurons for each one of them.

When analyzing Tables IV and V, with Figures 5 and 6, the user can assess that, overall, changing the number of intermediate layers and neurons did not improve the results of the previous round. Although configuration 8 presented an R^2 of 0.380, which is greater than the R^2 of configuration 1, configuration 8 took a considerably longer time ($\approx$ 74 minutes). On the other hand, the loss function achieved lower values (Table VI). Based on these analyses, one can also notice that as the complexity of the PINN increases, the loss function tends to be smaller. However, this reduction does not follow the improvement in the predictive capacity of the model given by the metric R^2, indicating the need for new configurations.

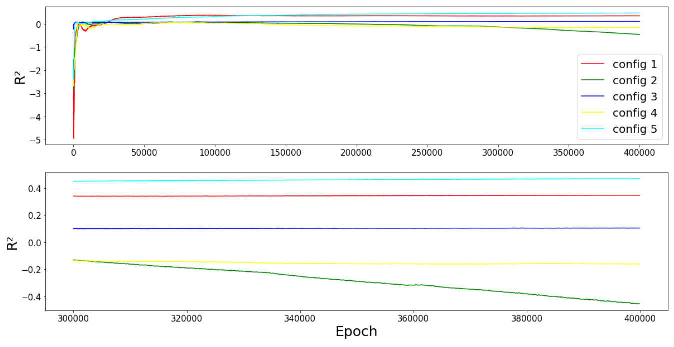


Fig. 3: Metric value R^2 for all epochs (upper); Focus on the last 100,000 epochs (lower). Regarding the first round.

TABLE IV: Q1 results with configurations of the second round

ID	Optimizer	Activation TT	Activation Vel	Neurons TT	Neurons Vel	Intermediate Layers TT	Intermediate Layers Vel
6	Adam	tanh	tanh	40	64	16	8
7	RMSProp	tanh	tanh	40	64	16	8
8	Adam	tanh	tanh	20	32	16	8
9	RMSProp	tanh	tanh	20	32	16	8

TABLE V: Q2 results ordered by the largest values of R^2, with its epoch and time in the second round

ID	Max R^2	Epoch	Time (m)
8	0.380	399800	174.046
6	0.219	8200	3.664
7	0.204	37400	17.011
9	0.194	124100	54.914

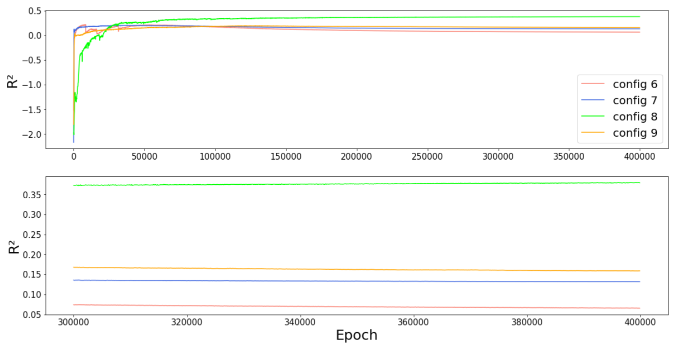


Fig. 5: Metric value R^2 for all epochs (upper); Focus on the last 100,000 epochs (lower). Regarding the second round.

TABLE VI: Q3 results ordered by the lowest values of the loss with its components, its epoch and time in the second round

ID	$\mathcal{L}$	$\mathcal{L}_{\mathcal{R}}$	$\mathcal{L}_{\mathcal{BC}}$	$\mathcal{L}_{\mathcal{D}}$	Epoch	Time (m)
6	$2.919 \cdot 10^{-6}$	$2.866 \cdot 10^{-6}$	$1.182 \cdot 10^{-10}$	$2.130 \cdot 10^{-9}$	399700	78.599
7	$6.568 \cdot 10^{-6}$	$5.213 \cdot 10^{-6}$	$3.841 \cdot 10^{-10}$	$5.419 \cdot 10^{-8}$	395100	79.705
8	$7.503 \cdot 10^{-6}$	$6.897 \cdot 10^{-6}$	$4.287 \cdot 10^{-9}$	$2.408 \cdot 10^{-8}$	392100	70.694
9	$3.415 \cdot 10^{-5}$	$2.719 \cdot 10^{-5}$	$4.834 \cdot 10^{-10}$	$2.783 \cdot 10^{-7}$	397200	75.761

The user can continue to fine-tune the hyperparameters to search for better PINN models. The provenance database can be used to query data combined from the two rounds of PINN configurations. By submitting query Q3 to configurations 1 to 9, the user identifies that the best model was configuration 5 with the lowest loss value for all its components, despite the

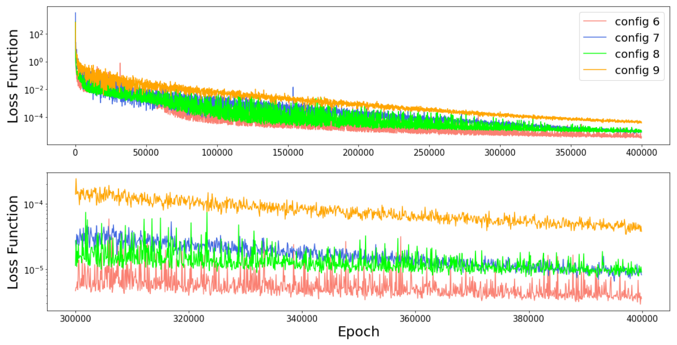
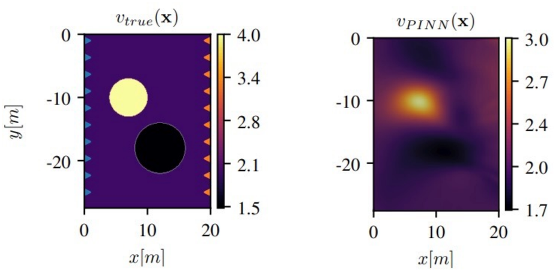


Fig. 6: Loss function value for all epochs (Upper); Focus on the last 100,000 epochs (Lower). Regarding the second round.

metric $R^2 = 0.47$. This result is similar to the value obtained in [12], which was validated with the best numerical method. Figure 7b shows how the prediction made by the PINN model generated with configuration 5 approaches the real solution to the problem (Figure 7a).



(a) Ground truth (b) PINN model

Fig. 7: Comparison of prediction. **(a)** The ground truth velocity model corresponds to a background velocity model with $v_{true}(\mathbf{x}) = 2.0[km/s]$ with two inclusions with $v_{true}(\mathbf{x}) = 4.0[km/s]$ (top-left) and $v_{true}(\mathbf{x}) = 1.5[km/s]$ (bottom-right) [12]. **(b)** Prediction of the best PINN model with $R^2 = 0.47$.

If the user seeks to go deeper in evaluating configuration 5, specific queries like in Table VII can be formulated, for example, to check the behavior of learning rate adaptations at specific epochs. This query seeks to find relations between the values for the loss components and the learning rates. Similarly, the user can keep track of automatic weight adaptations for the components of the loss function. These weight values can be set to be captured using PINNProv so that they are persisted in the provenance database, as is done with the adaptation of the learning rate. Then, the user can also query these weight evolutions.

TABLE VII: Query results for the learning rate adaptation values at epoch 300000 and the next, with their metrics for configuration 5

Epoch	R^2	$\mathcal{L}$	$\mathcal{L}_\mathcal{R}$	$\mathcal{L}_{BC}$	$\mathcal{L}_\mathcal{D}$	Learning Rate
300000	0.451	$9.787 \cdot 10^{-5}$	$5.215 \cdot 10^{-5}$	$3.867 \cdot 10^{-9}$	$1.829 \cdot 10^{-6}$	$7.397 \cdot 10^{-4}$
300100	0.452	$1.372 \cdot 10^{-4}$	$5.300 \cdot 10^{-5}$	$5.461 \cdot 10^{-9}$	$3.369 \cdot 10^{-6}$	$7.390 \cdot 10^{-4}$

If the user did not use the support of PINNProv, they would have to perform the management and collection of data referring to the evaluation metrics and hyperparameters of each model using log files, which would not provide the benefits of persistence of data made from PINNProv. They would also have to define an organization to store these files, which in turn would not respect a predefined standard. In addition, they would have to write another script to associate the different log files referring to each model in order to carry out their analyses.

D. Experiments with PINNProv and DeepXDE

We used PINNProv to capture provenance data using one of the examples provided by DeepXDE, the inverse problem for the Poisson equation with unknown force field[4]. This type of equation is often found in Physics and engineering problems. The equation is in one dimension and is of the form shown in Equation 3 and with the Dirichlet boundary conditions as presented in Equation 4.

$$\frac{d^2u(x)}{dx^2} = q(x), x \in [-1, 1] \tag{3}$$

$$u(-1) = u(1) = 0 \tag{4}$$

In this Poisson example $u(x)$ and $q(x)$ are unknown. To solve the problem, the value of $u(x)$ is set to 100 points. The reference solution is $u(x) = sin(\pi x), q(x) = -\pi^2 sin(\pi x)$. Similarly to the Eikonal PINN, DeepXDE defines two networks for this Poisson example, one to train $u(x)$ and the other to train $q(x)$. The loss function $\mathcal{L}$ was also defined with three components related to $\mathcal{L}_\mathcal{R}$ as the PDE residual, $\mathcal{L}_{BC}$ as boundary and initial conditions, and $\mathcal{L}_\mathcal{D}$ the data assimilation. PINNProv persists provenance data at every 10 epochs. We show results for 50000 epochs. The $L2$ relative error is used as an evaluation metric.

To conduct the comparative analysis of the Poisson PINN models, we submitted queries to the PINNProv database, similar to the ones defined in the Eikonal PINN. Table VIII shows the results of query Q1 with some attributes for the three configurations evaluated. We defined alternative combinations for the three weights of the $\mathcal{L}$ components. The hyperparameters referring to the neural network are the same for the two networks responsible for the estimates of $u(x)$ and $q(x)$.

TABLE VIII: Q1 results with the training configurations

ID	Optimizer	Activation	Neurons	Intermediate Layers	Learning Rate	Loss weights (w_R,w_{BC},w_D)
1	Adam	tanh	20	4	$1 \cdot 10^{-4}$	(10,100,1000)
2	Adam	tanh	20	4	$1 \cdot 10^{-4}$	(15,150,1500)
3	Adam	tanh	20	4	$1 \cdot 10^{-4}$	(20,200,2000)

It is worth noticing that this is a simple PINN training execution, and the elapsed time for each configuration in the 50,000 epochs was a maximum of 90 seconds. Queries Q2 and Q3 provide an analysis of the best $L2$ relative error metric and the lowest value of the loss function for the three model

[4]https://deepxde.readthedocs.io/en/latest/demos/pinn_inverse.html

configurations. Table IX may indicate that configuration 3 has an advantage over the others given the relative $L2$ error. These aggregated max and min query results complement the evolutions shown in graphical figures. In Figure 8 we can see that configurations 1 and 3 converged to the $L2$ relative error early in training, before epoch 5,000, and did not improve significantly after that. The behavior of the loss function during training, as depicted in Figure 9, shows improvements over the epochs, however, this improvement is not accompanied by a substantial evolution in the $L2$ metric. Taking into account the relative error $L2$, we can see that the best-trained model is given by configuration 3. The estimate of this model compared to the ground truth is in Figure 10.

TABLE IX: Q2 results ordered by the lowest relative error values $L2$ for the function $u(x)$, with its epoch and time taken

ID	Minimum $L2$ relative error	Epoch	Time (s)
3	$1.98{\cdot}10^{-4}$	49590	83.064
2	$2.89{\cdot}10^{-4}$	49990	83.778
1	$2.97{\cdot}10^{-4}$	49860	88.809

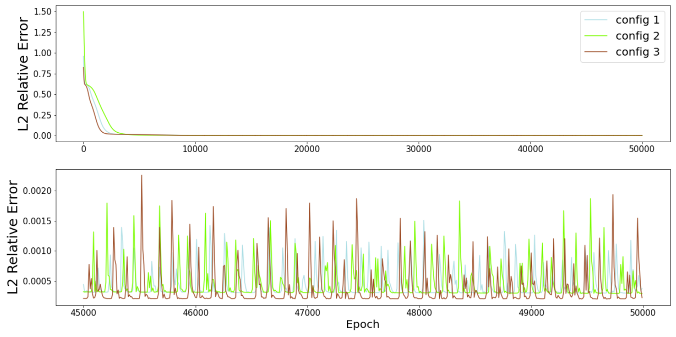


Fig. 8: Relative error value $L2$ for function estimates $u(x)$ for all epochs (upper); Focus on the last 5,000 epochs (lower).

TABLE X: Q3 results ordered by the lowest values of the loss with its components, its epoch and time taken

ID	$\mathcal{L}$	$\mathcal{L}_{\mathcal{R}}$	$\mathcal{L}_{\mathcal{BC}}$	$\mathcal{L}_{\mathcal{D}}$	Epoch	Time(s)
1	$5.384{\cdot}10^{-5}$	$9.980{\cdot}10^{-7}$	$1.975{\cdot}10^{9}$	$4.366{\cdot}10^{-8}$	49860	88.809
3	$6.359{\cdot}10^{-5}$	$1.017{\cdot}10^{-6}$	$2.037{\cdot}10^{8}$	$1.959{\cdot}10^{-8}$	49830	83.466
2	$6.751{\cdot}10^{-5}$	$1.666{\cdot}10^{-7}$	$1.166{\cdot}10^{8}$	$4.218{\cdot}10^{-8}$	49990	83.778

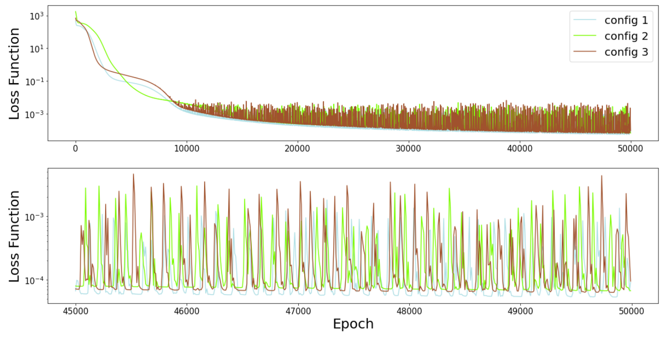


Fig. 9: Loss function value for all epochs (Upper); Focus on the last 5,000 epochs (Lower).

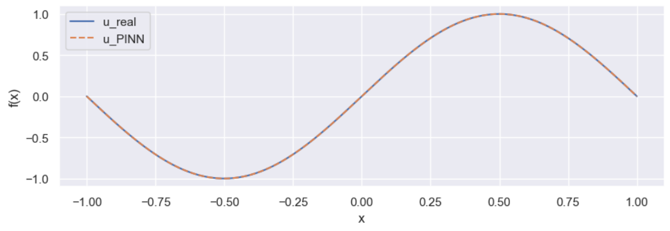


Fig. 10: Comparison between the ground truth given by the real function $u(x)$ and our best PINN.

Despite being a simple example, it shows the advantages of persisting provenance data with PINNProv in DeepXDE. Once the callback object is defined, PINNProv can be used to capture provenance in other PINN trainings for solving different problems with DeepXDE, and all databases follow the same representation, unlike when using Pandas or CSV.

V. Related Work

Despite the provenance support of many ML frameworks, a recent survey [24] highlights the need to use standard representations and support for provenance queries, which most frameworks do not provide. We performed a literature search to find provenance solutions developed to capture provenance data specifically for PINNs but found no results. There are provenance systems that aim at scientific ML workflows, which support ML provenance capture in HPC and address scientific data representation [11], [20], [25] , but still lack tracking the loss function components evolution, *etc.*

Braid-DB [25] aims at capturing metadata and provenance records for reproducibility, evolution, and explainability of scientific ML. Braid-DB is part of an exascale project aiming at continuum executions. Braid-DB represents provenance relationships between activities and entities but does mention the use of W3C PROV or how its provenance model could be extended for PINNs.

Provenance systems like PROV-IO [26], ProvLake [20], [27], [28] and DNNProv [11] have shown their abilities in providing W3C PROV-compliant provenance data for scientific ML workflows in HPC environments. ProvLake and DNNProv specialize their PROV-DM schema to represent typical ML artifacts, helping the specification of provenance capture in ML scripts. This specialization schema can be further explored by the user for PINNs. However, we consider it to be a lot of burden and complexity for the PINN user who would have to go deep into PROV-DM concepts and relationships to model, then instrument, and query PINN data.

VI. Conclusion

The provenance data of a deep learning experiment contributes to model evaluation, reproducibility, and explainability. There are challenges in provenance data capture in distributed and HPC environments and in defining what data should be captured for analytical queries. Having a W3C PROV-compliant representation provides a uniform way of querying relationships between datasets, training activities, parameters, and agents. It also helps to share and interpret PROV documents associated with models. Training PINN models is more complex due to the representation of the Physics and complexity of the neural networks, which add more challenges to provenance capture. PINNProv contributes by providing a set of services to be invoked in PINN scripts. The basic data to be captured are predefined to familiarize the user with provenance queries. Then, more data can extend the provenance database. Different frameworks for PINN scripts can provide provenance using PINNProv in *callback* class objects and other specializations. Experiments with PINNProv, in a supercomputer, show how the PINN provenance database complements graphic chart evaluations. Database queries show aggregated values ordered by the lowest errors and highest metrics from separate executions of different rounds of iterations of model configurations. These databases can be used in future configurations of new PINNs or adjustments. When a trained model is selected for production, it can carry its own representation of provenance in PROV documents to add quality and trust to the model put into production.

Acknowledgments

This work was partially funded by FAPERJ, CNPq, and the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

References

[1] P. Kidger and T. Lyons, "Universal approximation with deep narrow networks," in *Conf. on learning theory*, pp. 2306–2327, PMLR, 2020.

[2] M. Learning, "Tom mitchell," *Publisher: McGraw Hill*, 1997.

[3] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational physics*, vol. 378, pp. 686–707, 2019.

[4] S. Cuomo, V. S. Di Cola, F. Giampaolo, G. Rozza, M. Raissi, and F. Piccialli, "Scientific machine learning through physics–informed neural networks: Where we are and what's next," *Journal of Scientific Computing*, vol. 92, no. 3, p. 88, 2022.

[5] P. Stiller, F. Bethke, M. Böhme, R. Pausch, S. Torge, A. Debus, J. Vorberger, M. Bussmann, and N. Hoffmann, "Large-scale neural solvers for partial differential equations," in *SMC 2020*, pp. 20–34, Springer, 2020.

[6] S. K. Karmaker, M. M. Hassan, M. J. Smith, L. Xu, C. Zhai, and K. Veeramachaneni, "Automl to date and beyond: Challenges and opportunities," *ACM CSUR*, vol. 54, no. 8, pp. 1–36, 2021.

[7] D. Wang, J. D. Weisz, M. Muller, P. Ram, W. Geyer, C. Dugan, Y. Tausczik, H. Samulowitz, and A. Gray, "Human-ai collaboration in data science: Exploring data scientists' perceptions of automated ai," vol. 3, pp. 1–24, 2019.

[8] A. Kumar, S. Nakandala, Y. Zhang, S. Li, A. Gemawat, and K. Nagrecha, "Cerebro: A layered data platform for scalable deep learning," in *CIDR '21*, 2021.

[9] M. Vartak, H. Subramanyam, W.-E. Lee, S. Viswanathan, S. Husnoo, S. Madden, and M. Zaharia, "Modeldb: a system for machine learning model management," in *HILDA*, pp. 1–3, 2016.

[10] J. M. Wozniak, R. Chard, K. Chard, B. Nicolae, and I. Foster, "Xd/ml pipelines: Challenges in automated experimental science data processing," in *ASCR Workshop on the Management and Storage of Scientific Data*, 2022.

[11] D. Pina, L. Kunstmann, D. de Oliveira, P. Valduriez, and M. Mattoso, "Provenance supporting hyperparameter analysis in deep neural networks," in *IPAW*, pp. 20–38, Springer, 2021.

[12] R. Silva, D. Pina, L. Kunstmann, D. de Oliveira, P. Valduriez, A. Coutinho, and M. Mattoso, "Capturing provenance to improve the model training of pinns: first hand-on experiences with grid5000," in *42nd CILAMCE*, pp. 1–7, 2021.

[13] L. S. de Oliveira, R. M. Silva, L. Kunstmann, D. Pina, D. de Oliveira, A. L. Coutinho, and M. Mattoso, "Dados de proveniência para redes neurais guiadas pela física: o caso da equação eikonal," in *XXXVII SBBD*, pp. 373–378, SBC, 2022.

[14] H. Miao, A. Li, L. S. Davis, and A. Deshpande, "Towards unified data and lifecycle management for deep learning," in *IEEE ICDE*, pp. 571–582, IEEE, 2017.

[15] M. Schlegel and K.-U. Sattler, "Management of machine learning lifecycle artifacts: A survey," *ACM SIGMOD Record*, vol. 51, no. 4, pp. 18–35, 2023.

[16] M. Vartak and S. Madden, "Modeldb: Opportunities and challenges in managing machine learning models.," *IEEE Data Eng. Bull.*, vol. 41, no. 4, pp. 16–25, 2018.

[17] S. Schelter, J.-H. Boese, J. Kirschnick, T. Klein, and S. Seufert, "Automatically tracking metadata and provenance of machine learning experiments," 2017.

[18] M. Zaharia *et al.*, "Accelerating the machine learning lifecycle with mlflow.," *IEEE Data Eng. Bull.*, vol. 41, no. 4, pp. 39–45, 2018.

[19] G. Gharibi, V. Walunj, R. Nekadi, R. Marri, and Y. Lee, "Automated end-to-end management of the modeling lifecycle in deep learning," *Empirical Software Engineering*, vol. 26, pp. 1–33, 2021.

[20] R. Souza, L. Azevedo, V. Lourenço, E. Soares, R. Thiago, R. Brandão, D. Civitarese, E. Brazil, M. Moreno, P. Valduriez, M. Mattoso, R. Cerqueira, and M. A. Netto, "Workflow provenance in the lifecycle of scientific machine learning," *CCPE*, vol. 34, no. 14, p. e6544, 2022.

[21] D. Pina, A. Chapman, D. De Oliveira, and M. Mattoso, "Deep learning provenance data integration: a practical approach," in *Companion Proceedings of the ACM Web Conference 2023*, pp. 1542–1550, 2023.

[22] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis, "DeepXDE: A deep learning library for solving differential equations," *SIAM Review*, vol. 63, no. 1, pp. 208–228, 2021.

[23] L. Debnath, *First-Order Nonlinear Equations and Their Applications*, pp. 227–256. Boston: Birkhäuser Boston, 2012.

[24] A. Kale, T. Nguyen, F. C. Harris Jr, C. Li, J. Zhang, and X. Ma, "Provenance documentation to enable explainable and trustworthy ai: A literature review," *Data Intelligence*, pp. 1–41, 2022.

[25] J. M. Wozniak, Z. Liu, R. Vescovi, R. Chard, B. Nicolae, and I. Foster, "Braid-db: Toward ai-driven science with machine learning provenance," in *SMC 2021*, pp. 247–261, Springer, 2022.

[26] R. Han, S. Byna, H. Tang, B. Dong, and M. Zheng, "Prov-io: An i/o-centric provenance framework for scientific data on hpc systems," in *HPDC*, pp. 213–226, 2022.

[27] R. Souza, L. Azevedo, V. Lourenço, E. Soares, R. Thiago, R. Brandão, D. Civitarese, E. Brazil, M. Moreno, P. Valduriez, and M. Mattoso, "Provenance data in the machine learning lifecycle in computational science and engineering," in *IEEE/ACM WORKS*, pp. 1–10, 2019.

[28] R. Souza, L. Azevedo, R. Thiago, E. Soares, M. Nery, M. A. Netto, E. Vital, R. Cerqueira, P. Valduriez, and M. Mattoso, "Efficient runtime capture of multiworkflow data using provenance," in *IEEE eScience*, pp. 359–368, 2019.