
# Capturing Provenance to Improve the Model Training of PINNs: first hand-on experiences with Grid5000

Rômulo M. Silva[1], Débora Pina[2], Liliane Kunstmann[2], Daniel de Oliveira[3], Patrick Valduriez[4], Alvaro L. G. A. Coutinho[1], Marta Mattoso[2]

[1]*Dept. of Civil Engineering, Federal University of Rio de Janeiro*
*Av. Athos da Silveira Ramos, 149, CT, Room B101, Rio de Janeiro, RJ 1941-909, Brazil*
*romulo.silva@coc.ufrj.br, alvaro@coc.ufrj.br*
[2]*Dept. of Computer and Systems Engineering, Federal University of Rio de Janeiro*
*Avenida Horácio Macedo 2030, CT, Room H319, Rio de Janeiro, RJ 21941-914, Brazil*
*dbpina@cos.ufrj.br, lneves@cos.ufrj.br, marta@cos.ufrj.br*
[3]*Institute of Computing, Fluminense Federal University*
*Rua Passo da Pátria, 156, Niterói/RJ, Brazil*
*danielcmo@ic.uff.br*
[4]*Inria, University of Montpellier, CNRS, LIRMM*
*Campus Saint-Priest - Bâtiment 5, 860 Rue de St Priest, 34095 Montpellier Cedex 5 France*
*Patrick.Valduriez@inria.fr*

**Abstract.** The growing popularity of Neural Networks in computational science and engineering raises several challenges in configuring training parameters and validating the models. Machine learning has been used to approximate costly computational problems in computational mechanics, discover equations by coefficient estimation, and build surrogates. Those applications are outside of the common usage of neural networks. They require a different set of techniques generally encompassed by Physics-Informed Neural Networks (PINNs), which appear to be a good alternative for solving forward and inverse problems governed by PDEs in a small data regime, especially when it comes to Uncertainty Quantification. PINNs have been successfully applied for solving problems in fluid dynamics, inference of hydraulic conductivity, velocity inversion, phase separation, and many others. Nevertheless, we still need to investigate its computational aspects, especially its scalability, when running in large-scale systems. Several hyperparameter configurations have to be evaluated to reach a trained model, often requiring fine-tuning hyperparameters, despite the existence of a few setting recommendations. In PINNs, this fine-tuning requires analyzing configurations and how they relate to the loss function evaluation. We propose provenance data capture and analysis techniques to improve the model training of PINNs. We also report our first experiences on running PINNs in Grid5000 using hybrid CPU-GPU computing.

**Keywords:** Physics-informed Neural Networks, Eikonal Equation, Provenance, Hybrid Computing

## 1 Introduction

Hey et al. [1] draw attention to the growing popularity of Neural Networks (NNs) in Computational Science and Engineering (CSE). In this area, machine learning (ML) has been used to approximate costly computational problems, discover equations by coefficient estimation, and for building surrogates [2]. Those are outside the common usage of NNs, requiring a different set of techniques. Moreover, the standard rules for hyperparameter tuning usually do not apply to such problems [3]. An emerging framework for NNs in CSE is the Physics-Informed Neural Networks (PINNs). PINNs are Deep Neural Networks (DNN) that aim to solve forward and inverse problems characterized by Partial Differential Equations (PDEs) by inserting the governing equations of the underlying physics in the loss function.

One of the challenges when working with PINNs is to reduce the training cost. For this scenario, it is essential to verify the evolution of the loss function over the epochs in many different situations. Moreover, it is necessary

to associate the loss metrics to the hyperparameters' configurations, for instance, learning rate, batch size, number of epochs, activation function, and optimizer. Improving the selection of these parameters is essential due to the time it takes to train a PINN, the search space of hyperparameters' combinations, and the difficulty of tuning them, even with auto-tuning tools [4]. Another challenge is to make explicit the metrics that guided the model's choice towards explainability.

Provenance has been added to ML to assist hyperparameter analysis in environments like ModelHub [5], and ModelKB [6]. However, these solutions require that the ML algorithm runs in specific platforms other than those being used in PINNs. ML frameworks like TensorFlow and Keras are prevalent, and they can support PINNs, but they have limited provenance data, which are stored in log files. Logs are difficult to query, limiting their ability for hyperparameter analysis at runtime. In addition, PINNs have specific loss function parameters, important for the fine-tuning, not captured in current solutions. This paper presents DNNProv as an alternative to generating provenance in PINNs. DNNProv captures provenance from NN coded in environments like TensorFlow and Keras with low overhead [7]. Provenance in DNNProv captures the metrics associated with the hyperparameters along the epochs during the training execution. Provenance captured is sent asynchronously to be ingested in a database ready for online queries. Initial experiments with DNNProv using a PINN for the Factored Eikonal Equation in the Grid5000 using hybrid computing (CPU-GPU) provide evidence of the flexibility, the efficiency of data capture, and data analysis.

The remainder of this paper is structured as follows. Section 2 details the PINN scheme for solving an inverse problem governed by the Factored Eikonal Equation, Section 3 presents provenance in PINNs. Section 4 details the case study, and the experimental evaluation. Section 5 concludes.

## 2 PINNs for the Factored Eikonal Equation

### 2.1 PINNs in a Nutshell

Consider the problem presented in Eqs. (1)-(3) whose solution is given by $\mathbf{u}(\mathbf{x})$ with $\mathbf{x} \in \mathbb{R}^N$ and $\mathbf{u} \in \mathbb{R}^S$ with its physical parameters $\boldsymbol{\lambda}$ [8],

$$f\left(\mathbf{x}; \mathbf{u}; \frac{\partial \mathbf{u}}{\partial x_1}, ..., \frac{\partial \mathbf{u}}{\partial x_N}; \frac{\partial \mathbf{u}^2}{\partial x_1 x_1}, ..., \frac{\partial \mathbf{u}^2}{\partial x_1 x_N}; \boldsymbol{\lambda}\right) = 0, \forall \mathbf{x} \in \Omega, \Omega \subset \mathbb{R}^N, \tag{1}$$

$$\mathbf{u}(\mathbf{x}) = g(\mathbf{x}), \forall \mathbf{x} \in \Gamma_g, \Gamma_g \subset \mathbb{R}^{N-1}, \tag{2}$$

$$h\left(\mathbf{x}; \frac{\partial \mathbf{u}}{\partial x_1}, ..., \frac{\partial \mathbf{u}}{\partial x_N}; \boldsymbol{\lambda}\right) = 0, \forall \mathbf{x} \in \Gamma_h, \Gamma_h \subset \mathbb{R}^{N-1} \tag{3}$$

in which Eq. (1) is the governing PDE, Eq. (2) represents the *Dirichlet* boundary conditions, while Eq. (3) encapsulates both, *Neumann* and *Robin* boundary conditions. Furthermore, $\Omega$ denotes the internal domain, $\Gamma_h$ and $\Gamma_g$ denote the natural and essential boundaries, respectively.

In general, PDEs are difficult to solve and require approximation to be solved numerically. One can try to approximate the differential operator, but it is also possible to approximate the function space of the solution $\mathbf{u}$. The solution can be approximated using bases composed of monomial functions, piece-wise linear functions, or even by the space of the Neural Networks $\mathcal{H}_{NN}$.

If we choose to approximate the solution $\mathbf{u}(\mathbf{x})$ by $\hat{\mathbf{u}}(\mathbf{x}; \boldsymbol{\theta}) \in \mathcal{H}_{NN}$ in Eqs. (1)-(3) we obtain

$$f\left(\mathbf{x}; \frac{\partial \hat{\mathbf{u}}(\mathbf{x}; \boldsymbol{\theta})}{\partial x_1}, ..., \frac{\partial \hat{\mathbf{u}}(\mathbf{x}; \boldsymbol{\theta})}{\partial x_N}; \frac{\partial \hat{\mathbf{u}}^2(\mathbf{x}; \boldsymbol{\theta})}{\partial x_1 x_1}, ..., \frac{\partial \hat{\mathbf{u}}^2(\mathbf{x}; \boldsymbol{\theta})}{\partial x_1 x_N}; \boldsymbol{\lambda}\right) = \mathcal{R}(\mathbf{x}; \boldsymbol{\theta}; \boldsymbol{\lambda}) \tag{4}$$

where $\mathcal{R}(\mathbf{x}; \boldsymbol{\theta}; \boldsymbol{\lambda})$ is called the residual of the PDE and $\boldsymbol{\theta}$ are the parameters of the neural network.

For solving Eq. (1) using $\hat{\mathbf{u}}(\mathbf{x}; \boldsymbol{\theta})$ as an approximation for its solution $\mathbf{u}(\mathbf{x})$, one should minimize the residual (Eq. (4)) at a set of *residual points* $\mathcal{T}_\mathcal{R} = \{\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x}_\mathcal{R}\}$ with $\mathcal{T}_\mathcal{R} \subset \Omega$ and also satisfy the boundary conditions with a acceptable accuracy at a set of points $\mathcal{T}_\mathcal{B} \subset \Gamma$. To measure how good is the approximation of the residual of the PDE we write,

$$\mathcal{L}_\mathcal{R}(\boldsymbol{\theta}; \mathcal{T}_\mathcal{R}) = MSE\{\mathcal{R}(\mathbf{x}; \boldsymbol{\theta}; \boldsymbol{\lambda}), \mathbf{0}\}, \tag{5}$$

and for measuring the quality of the approximation of the boundary conditions

$$\mathcal{L}_\mathcal{BC}(\boldsymbol{\theta}; \mathcal{T}_\mathcal{BC}) = MSE\left\{h\left(\mathbf{x}; \frac{\partial \mathbf{u}}{\partial x_1}, ..., \frac{\partial \mathbf{u}}{\partial x_N}; \boldsymbol{\lambda}\right), \mathbf{0}\right\} + MSE\{\hat{\mathbf{u}}(\mathbf{x}; \boldsymbol{\theta}), g(\mathbf{x})\}, \tag{6}$$

where $MSE\{\mathbf{y}_{true}(\mathbf{x}), \mathbf{y}_{pred}(\mathbf{x})\}$ denotes the Mean Squared Error between the vectors $\mathbf{y}_{true}(\mathbf{x})$ and $\mathbf{y}_{pred}(\mathbf{x})$.

In addition, if one want to solve an inverse problem it may be required to incorporate the data through

$$\mathcal{L}_{\mathcal{D}}(\boldsymbol{\theta}; \mathcal{T}_{\mathcal{D}}) = MSE\{\hat{\mathbf{u}}_{pred}(\mathbf{x}), \hat{\mathbf{u}}_{data}(\mathbf{x})\}, \tag{7}$$

where $\mathcal{T}_{\mathcal{D}}$ is the set of points from which we can take measurements from our system, $\hat{\mathbf{u}}_{data}$ denotes the measurements and $\hat{\mathbf{u}}_{pred}$ denotes the predictions at $\mathbf{x} \in \mathcal{T}_{\mathcal{D}}$.

Notice that, Equation (6) involves not only the information about the boundary conditions, but also the initial conditions for time-dependent problems since the components of $\mathbf{x}$ denotes not only the spatial dimensions but also time. With these measures in hand, we can now write the final loss function for *informing* the physical laws to a neural network during its training phase,

$$\mathcal{L}(\boldsymbol{\theta}; \mathcal{T}) = w_{\mathcal{R}}\mathcal{L}_{\mathcal{R}}(\boldsymbol{\theta}; \mathcal{T}_{\mathcal{R}}) + w_{\mathcal{BC}}\mathcal{L}_{\mathcal{BC}}(\boldsymbol{\theta}; \mathcal{T}_{\mathcal{BC}}) + w_{\mathcal{D}}\mathcal{L}_{\mathcal{D}}(\boldsymbol{\theta}; \mathcal{T}_{\mathcal{D}}) \tag{8}$$

where $w_{\mathcal{R}}$, $w_{\mathcal{BC}}$ and $w_{\mathcal{D}}$ are the weights of the loss function, which play an essential role in the minimization of the loss function.

Finally, for the solution of inverse problems, one may want to estimate the parameters $\boldsymbol{\lambda}$ as a set of physical constants or even in the form of a function $\boldsymbol{\lambda}(\mathbf{x}; \boldsymbol{\theta}_{\boldsymbol{\lambda}})$, which is easily achievable and implemented thanks to computing libraries such as TensorFlow and PyTorch.

### 2.2 Factored Eikonal Equation

The Eikonal equation is a first-order nonlinear equation relevant in many fields. It describes phenomena like wave propagation for acoustic and elastic media, as well as electromagnetic media [9]. Therefore, the Eikonal equation plays an important role in problems like optics, shortest path problems, image segmentation, and seismic and medical imaging. The Eikonal equation is given by,

$$\|\boldsymbol{\nabla}\phi(\mathbf{x})\|_2 = \frac{1}{v(\mathbf{x})}, \forall \mathbf{x} \in \Omega. \tag{9}$$

where $\Omega \subset \mathbb{R}^{n_{sd}}$, is the domain, $n_{sd} = 1, 2, 3$ is the number of space dimensions, $\mathbf{x} = \{x, y, z\}$ is the position vector, and $\|.\|$ stands for the $L^2$-norm. Solving this equation for $\phi(\mathbf{x})$ for a given velocity field, $v(\mathbf{x})$, and proper boundary conditions become feasible employing numerical methods as, for example, the Fast Marching Method [10], and the Fast Sweeping Method [11]. However, when point singularities are present, as, in seismic imaging, the Factored Eikonal Equation (FEE for short) is preferable, as pointed out in [12]. In this case, the Eikonal equation solution can be factored as $\phi(\mathbf{x_s}, \mathbf{x_r}) = \tau(\mathbf{x_s}, \mathbf{x_r})\xi(\mathbf{x_s}, \mathbf{x_r})$, where $\mathbf{x_s}$ is the source position, $\mathbf{x_r}$ is the receiver position, which can be any point that lies in the domain $\Omega$. For the particular scenario where a point source is applied as boundary condition, $\xi(\mathbf{x_s}, \mathbf{x_r}) = \|\mathbf{x_s} - \mathbf{x_r}\|_2$ represents the solution of equation (9) for the case where a point source is applied in a domain such as $v(\mathbf{x}) = v(\mathbf{x_r}) = 1$. Substituting the factorization of $\phi$ in (9), the resulting equation,

$$\tau^2\|\boldsymbol{\nabla}_{\mathbf{x_r}}\xi\|_2^2 + 2\tau\xi\boldsymbol{\nabla}_{\mathbf{x_r}}\tau \cdot \boldsymbol{\nabla}_{\mathbf{x_r}}\xi + \xi^2\|\boldsymbol{\nabla}_{\mathbf{x_r}}\tau\|_2^2 = \frac{1}{v(\mathbf{x_r})^2} \tag{10}$$

is the FEE. Here, $\boldsymbol{\nabla}_{\mathbf{x_r}}$ is the gradient operator with respect to the receiver position. This formulation is advantageous in situations where $\phi(\mathbf{x})$ has point source singularities, which can be well captured by $\xi(\mathbf{x_s}, \mathbf{x_r})$, while $\tau(\mathbf{x_s}, \mathbf{x_r})$ acts as a smooth correction at the neighborhood of the point source singularities [12]. Equation (10) is only solved for $\tau$ which has as boundary condition $\tau(\mathbf{x_s}, \mathbf{x_r} = \mathbf{x_s}) = \frac{1}{v(\mathbf{x_s})}$.

Given the source position $\mathbf{x}_s$ and the properties of the domain $v(\mathbf{x})$ it is possible to solve Eq. (10) for the whole domain. This is called the *forward problem*. A way more challenging task is to find the properties of the domain $v(\mathbf{x})$, given a set of scattered measurements of the travel times $\phi(\mathbf{x}_s, \mathbf{x}_r)$, which is the equivalent of solving the *inverse problem*. Figure 1 shows a simple scheme for using PINNs to solve an inverse problem governed by the FEE.

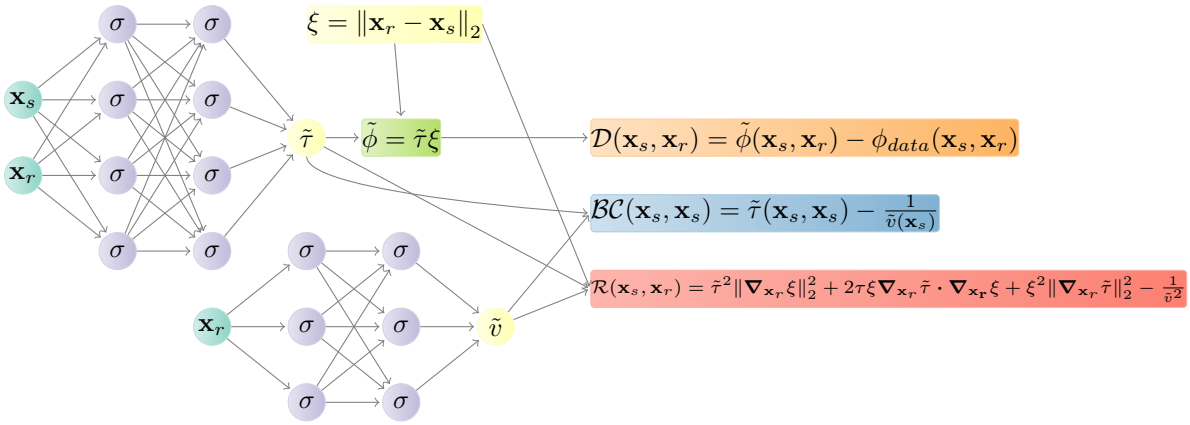


Figure 1. PINN scheme for solving the inverse FEE. The $\phi(\mathbf{x}_s, \mathbf{x}_r)$ and $v(\mathbf{x}_r)$ are approximated by two different neural networks and their approximations are denoted by $\tilde{\phi}(\mathbf{x}_s, \mathbf{x}_r)$ and $\tilde{v}(\mathbf{x}_r)$. Those approximations are then fed to the loss components related to the data assimilation, boundary and initial conditions, and the PDE residual.

## 3 Provenance for PINNs

According to the W3C PROV reommendation [13], "provenance is information about entities, activities, and people involved in producing a piece of data or thing, which can be used to form assessments about its quality, reliability, or trustworthiness". Provenance represents the data derivation path, associating the data (entities) with the algorithms/programs (activities) that transform these data, in addition to registering the agents (humans, teams) associated with the entities and activities. As in any NN, the PINN data derivation path includes the activities: (i) *Training* and (ii) *Adaptation*. Provenance data is automatically captured during the PINN execution and stored in a database that associates input data to activities, their parameters, and outputs. The input of the *Training* activity is the training dataset and the hyperparameters are, for example: the name of the optimizer, the learning rate, number of epochs, batch size and activation functions, and the output is a set of metrics that helps in the evaluation of results obtained during training, *e.g.*, the loss function and its components, the elapsed time, and the date and time of the end of the execution of each epoch. The inputs of *Adaptation* are the dataset generated by *Training* and parameters for the adaptation. After an adaptation activity, its output can be a new learning rate and epoch values, which are also in the database with the date and time when the adaptation occurred, and the adaptation identification.

Automatic provenance capture during the training of a PINN improves the quality and reliability of the model. As the training takes a long time, it is worth having access to the provenance of what has already been executed to decide on adapting hyperparameters. The evaluation of the several hyperparameters requires us to be aware of the relationship between metadata, *e.g.*, the chosen hyperparameter values, performance data, environment configuration, model metrics, *etc.* Unlike in other NN provenance approaches, in the case of PINNs, specific loss function component values need to be tracked. This evaluation, through provenance data queries during the training activity, can support fine-tuning decisions, complementing auto-tuning solutions [4].

DNNProv is a W3C PROV compliant software library that allows for online hyperparameters' analysis through provenance data [7]. DNNProv captures typical ML hyperparameters with the flexibility to include other data. In this paper, DNNProv captures parameters relevant to fine-tune PINNs, like those in the loss function. The DNNProv services are used in ML-based workflows with different DL frameworks (*e.g.*, Tensorflow, Theano) while being able to share and analyze captured provenance data using the same W3C PROV representation.

## 4 Experiments

This section presents how provenance data captured with DNNProv help improving the training of PINNs to solve an inverse FEE problem, the crosshole traveltime tomography. Fig. 2 shows that the PINN solution, using the FEE as the forward solver, is closer to the ground truth than the PyGIMli[14] solution. However, when querying the provenance database during the training, we notice room to improve the PINNs' performance in several different training scenarios. In all experiments, the FEE PINN runs on GPUs, while DNNProv captures provenance data from the same GPU and sends them asynchronously to a database running on a CPU managed by the columnar relational database system MonetDB[1]. All the experiments use a CPU-GPU hybrid computing mode

[1] https://www.monetdb.org/

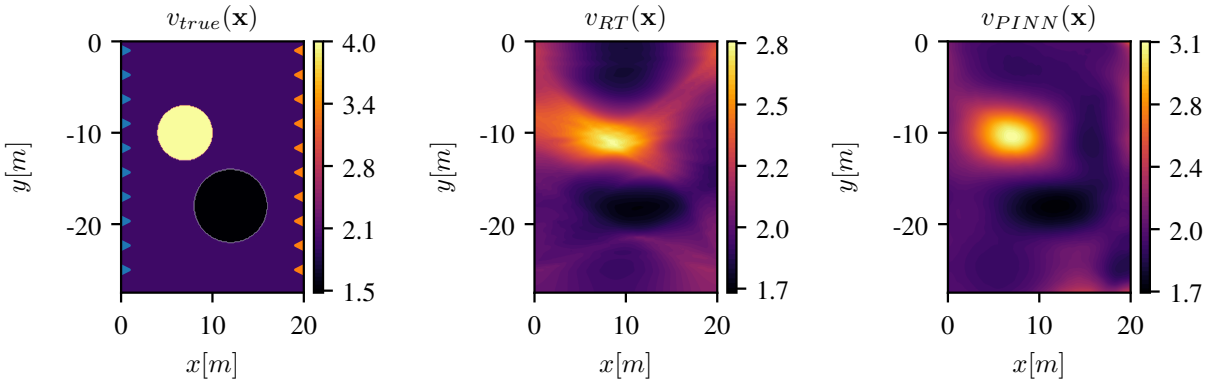


Figure 2. Crosshole traveltime tomography problem setup. **(left)**: The ground truth velocity model corresponds to a background velocity model with $v_{true}(\mathbf{x}) = 2.0[km/s]$ with two inclusions with $v_{true}(\mathbf{x}) = 4.0[km/s]$ (top-left) and $v_{true}(\mathbf{x}) = 1.5[km/s]$ (bottom-right). **(middle)**: PyGIMLi[14] solution, $R^2 = 0.32$. **(right)**: PINN solution (present work), $R^2 = 0.52$.

in the Inria's Grid5000 computer system, a large-scale and flexible testbed for experiment-driven research, with a focus on parallel and distributed computing, including Cloud, HPC, and Big Data and AI [15].

Several training runs were performed with the FEE PINN, with variations in the activation function (ReLU, Tanh, Sine, Sigmoid) and optimizer (Adam, RMSProp). We use provenance to monitor metrics by epoch during the PINN training and steer the training at runtime, inspecting how the evaluation metrics are evolving, including the loss, $\mathcal{L}$, and its components, $\mathcal{L}_{\mathcal{BC}}$, $\mathcal{L}_{\mathcal{R}}$, and $\mathcal{L}_{\mathcal{D}}$. Thus, we can change parameters and start training again if, for instance, the loss value is not meeting a chosen criteria. In this case, these adaptations are also registered with provenance data (time and date when it happened, and parameters used, such as decay rate and decay steps) since they are essential for *a posteriori* data analysis. We may also want to discover if the training with adaptation at runtime is effective and learn for the next PINN configurations.

During the online data analysis for current training (optimizer Adam and activation function Tanh), we submit a query such as "What are the elapsed time and loss for training each epoch?", which the result is presented in Table 1. With this query, we can investigate, for example, if any epoch is taking longer than usual. In addition, with the loss value attribute selected along with the epoch identification, we can verify whether this value is improving over the epochs. Moreover, this query helps decision-making, such as tuning the optimizer or the learning rate.

Table 1 shows the current training epochs, time, and parameters. Based on Table 1, we decide on a further evaluation, considering training with two hyperparameter combinations for the PINN, executing in parallel, one with the optimizer Adam and the other with the optimizer RMSProp, both with the activation function Sigmoid. Then, we submit another query to compare the two optimizers: "What is the elapsed time and training loss in the latest epoch for each combination?", with the results shown in Table 2. From this query, we observe that Adam's optimizer loss presents the best results. Therefore, we decide to stop the training with RMSProp. Consequently, the relationship between the provenance data and the hyperparameter's configuration can help us fine-tune the hyperparameters' values during the search for the most satisfactory configuration.

Additionally, when we train a model, it is often helpful to lower the learning rate as the training progresses. Although decreasing the learning rate is a known technique, the registered values help trace the consequences of changes when analyzing different models. These data are also important when evaluating the impact of different parameters for decreasing the learning rate. We chose to use an exponential decay function and only one value for the decay rate and the decay steps. However, different values for these parameters can be tested. As these adaptations are saved, we can submit a query like "Retrieve the hyperparameters configuration and the lowest learning rate that was reached during the training.". The results of this query are shown in Table 3. In addition to tuning, the provenance database contributes to the reproducibility and explainability of the model [5, 6].

DNNProv can also be connected to data visualization tools, such as Kibana[2] to create dashboards and other resources to improve the runtime data analysis. Figure 3 shows the training loss of different executions, presenting an overall perspective of each run. From Figure 3, we observe that the configuration with RMSProp and Sigmoid shows the worst results. Continuing the training until the loss decreases to $10^{-3}$, we see that it is a good decision to stop earlier the training for this configuration.

Provenance capturing introduces overhead on the PINN execution. We measure this overhead for all the runs shown in Figure 3. The overhead corresponds to an increase of 4% over the total time, which is considered negligible, given the hyperparameter fine-tuning benefits. Such low overhead is due to CPU-GPU hybrid-computing

[2]https://www.elastic.co/kibana

model, where the provenance management engine runs on the CPU and the PINN on the GPU.

Table 1. Query: "What are the elapsed time and loss for each training epoch?"

| Epoch | Time (s) | $\mathcal{L}$ | $\mathcal{L}_{\mathcal{BC}}$ | $\mathcal{L}_{\mathcal{R}}$ | $\mathcal{L}_{\mathcal{D}}$ |
|---|---|---|---|---|---|
| 0 | 3.157 | 1191.801 | 0.0122093 | 0.5859 | 47.64812 |
| 10000 | 2.194 | 0.016 | 0.0000047 | 0.0015 | 0.00058 |
| 20000 | 2.125 | 0.011 | 0.0000013 | 0.0004 | 0.00043 |
| 30000 | 2.347 | 0.025 | 0.0000018 | 0.0011 | 0.00097 |

Table 2. Query: "What is the elapsed time and training loss in the latest epoch for each combination?"

| Time (s) | $\mathcal{L}$ | $\mathcal{L}_{\mathcal{BC}}$ | $\mathcal{L}_{\mathcal{R}}$ | $\mathcal{L}_{\mathcal{D}}$ | Optimizer | Act Func |
|---|---|---|---|---|---|---|
| 2.193 | 0.438 | 0.0000392 | 0.0009 | 0.017 | RMSProp | Sigmoid |
| 2.169 | 0.013 | 0.0000004 | 0.0008 | 0.0005 | Adam | Sigmoid |

Table 3. Query: "Retrieve the hyperparameters configuration and the lowest learning rate that was reached during the training." *LRate* stands for the learning rate.

| Function | Decay Rate | Decay Steps | Initial *LRate* | Lowest *LRate* |
|---|---|---|---|---|
| Exponential decay | 0.99 | 1000 | 0.001 | 0.00067 |

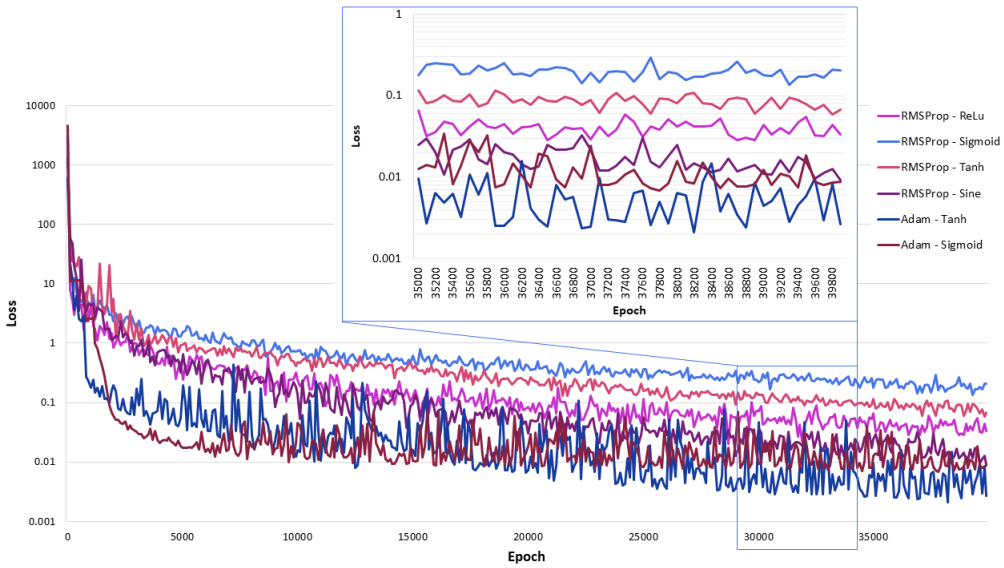


Figure 3. Graphical view of the training loss

## 5 Conclusions

In this paper, we propose using provenance capture to improve online data analysis while training PINNs. By automatically capturing provenance data with DNNProv, it is possible to analyze the chosen hyperparameter values related to the training stages and adjust them during the execution to achieve results with more quality. A

case study has been conducted with an actual PINN application on Grid5000. Our experiments show how using the DNNProv approach for online provenance query data analysis, and monitoring can support decision-making with very low overhead. As future work, we plan to capture provenance with different PINN architectures and provide more data to assist the training better. Also, we intend to explore GPU-based databases to store the captured data.

**Acknowledgements.** This work is funded by CNPq, FAPERJ (Center of Excellence in Digital Transformation and Artificial Intelligence of Rio de Janeiro State: Thematic Network in Renewable Energy and Climate Change), and Inria (HPDaSc associated team). We are indebted to Inria by providing us access to Grid5000. D. Pina, L. Kunstmann, and R. M. Silva are supported by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

**Authorship statement.** This section is mandatory and should be positioned immediately before the References section. The text should be exactly as follows: The authors hereby confirm that they are the sole liable persons responsible for the authorship of this work, and that all material that has been herein included as part of the present paper is either the property (and authorship) of the authors, or has the permission of the owners to be included here.

## References

[1] T. Hey, K. Butler, S. Jackson, and J. Thiyagalingam. Machine learning and big scientific data. *Philosophical Transactions of the Royal Society A*, vol. 378, n. 2166, pp. 20190054, 2020.

[2] J. D. Jakeman, M. Perego, and W. M. Severa. Neural networks as surrogates of nonlinear high-dimensional parameter-to-prediction maps. Technical report, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 2018.

[3] R. K. Tripathy and I. Bilionis. Deep uq: Learning deep neural network surrogate models for high dimensional uncertainty quantification. *Journal of computational physics*, vol. 375, pp. 565–588, 2018.

[4] 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. *Proceedings of the ACM on Human-Computer Interaction*, vol. 3, n. CSCW, pp. 1–24, 2019.

[5] H. Miao, A. Li, L. S. Davis, and A. Deshpande. Towards unified data and lifecycle management for deep learning. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pp. 571–582. IEEE, 2017.

[6] G. Gharibi, V. Walunj, S. Rella, and Y. Lee. Modelkb: towards automated management of the modeling lifecycle in deep learning. In *Proceedings of the 7th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering*, pp. 28–34. IEEE Press, 2019.

[7] D. Pina, L. Kunstmann, de D. Oliveira, P. Valduriez, and M. Mattoso. Provenance supporting hyperparameter analysis in deep neural networks. In *Provenance and Annotation of Data and Processes*, pp. 20–38. Springer, 2021.

[8] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis. DeepXDE: A deep learning library for solving differential equations. *SIAM Review*, vol. 63, n. 1, pp. 208–228, 2021.

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

[10] R. F. a. Stanley Osher. *Level Set Methods and Dynamic Implicit Surfaces*. Applied Mathematical Sciences 153. Springer-Verlag New York, 1 edition, 2003.

[11] H. Zhao. A fast sweeping method for eikonal equations. *Mathematics of Computation*, vol. 74, n. 250, pp. 603–628, 2004.

[12] S. Fomel, S. Luo, and H. Zhao. Fast sweeping method for the factored eikonal equation. *Journal of Computational Physics*, vol. 228, n. 17, pp. 6440 – 6455, 2009.

[13] L. Moreau and P. Groth. Provenance: an introduction to prov. *Synthesis Lectures on the Semantic Web: Theory and Technology*, vol. 3, n. 4, pp. 1–129, 2013.

[14] C. Rücker, T. Günther, and F. M. Wagner. pyGIMLi: An open-source library for modelling and inversion in geophysics. *Computers and Geosciences*, vol. 109, pp. 106–123, 2017.

[15] D. Balouek, A. Carpen Amarie, G. Charrier, F. Desprez, E. Jeannot, E. Jeanvoine, A. Lèbre, D. Margery, N. Niclausse, L. Nussbaum, O. Richard, C. Pérez, F. Quesnel, C. Rohr, and L. Sarzyniec. Adding virtualization capabilities to the Grid'5000 testbed. In I. I. Ivanov, van M. Sinderen, F. Leymann, and T. Shan, eds, *Cloud Computing and Services Science*, volume 367 of *Communications in Computer and Information Science*, pp. 3–20. Springer International Publishing, 2013.