Efficient Training of Physics-enhanced Neural ODEs via Direct Collocation and Nonlinear Programming

Linus Langenkamp [1] Philip Hannebohm [1] Bernhard Bachmann [1]

[1]Institute for Data Science Solutions, Bielefeld University of Applied Sciences and Arts, Germany,
{first.last}@hsbi.de

Abstract

We propose a novel approach for training Physics-enhanced Neural ODEs (PeN-ODEs) by expressing the training process as a dynamic optimization problem. The full model, including neural components, is discretized using a high-order implicit Runge-Kutta method with flipped Legendre-Gauss-Radau points, resulting in a large-scale nonlinear program (NLP) efficiently solved by state-of-the-art NLP solvers such as Ipopt. This formulation enables simultaneous optimization of network parameters and state trajectories, addressing key limitations of ODE solver-based training in terms of stability, runtime, and accuracy. Extending on a recent direct collocation-based method for Neural ODEs, we generalize to PeN-ODEs, incorporate physical constraints, and present a custom, parallelized, open-source implementation. Benchmarks on a Quarter Vehicle Model and a Van-der-Pol oscillator demonstrate superior accuracy, speed, generalization with smaller networks compared to other training techniques. We also outline a planned integration into OpenModelica to enable accessible training of Neural DAEs.

Keywords: Physics-enhanced Neural ODEs, Dynamic Optimization, Nonlinear Programming, Modelica, Neural ODEs, Universal Differential Equations

1 Introduction

Growing access to real-world data and advances in computational modeling have opened new possibilities for combining measured data with physics-based models. Neural Ordinary Differential Equations (NODEs) (Chen et al. 2018) represent a significant advancement in merging data-driven machine learning with physics-based modeling. By replacing the dynamics of an ODE with a neural network

$$\dot{\boldsymbol{x}}(t) = NN_{\boldsymbol{p}}(\boldsymbol{x}(t), \boldsymbol{u}(t), t), \tag{1}$$

where $\boldsymbol{x}(t)$ are states, $\boldsymbol{u}(t)$ is a fixed input vector, and $\boldsymbol{p}$ are the neural network parameters, NODEs bridge the gap between traditional differential equations and modern deep learning. After obtaining a NODE and given an initial condition $\boldsymbol{x}(t_0) = \boldsymbol{x}_0$, the state trajectory is reconstructed by simulation with an arbitrary ODE solver

$$\boldsymbol{x}(t) := \text{ODESolve}\left(NN_{\boldsymbol{p}}(\boldsymbol{x}(t), \boldsymbol{u}(t), t), \boldsymbol{x}_0\right). \tag{2}$$

However, learning the full dynamics can be unstable, requires a lot of data, and can suffer from poor extrapolation (Kamp, Ultsch, and Brembeck 2023). As a result, hybrid modeling is an emerging field that combines the flexibility of neural networks with known physics and first principle models. Extensions like Universal Differential Equations (UDEs) (Rackauckas et al. 2020) or Physics-enhanced Neural ODEs (PeN-ODEs) (Kamp, Ultsch, and Brembeck 2023; Sorourifar et al. 2023) generalize this paradigm, allowing domain-specific knowledge to be incorporated into the model while still learning observable but unresolved effects. These approaches have demonstrated great success in various fields, including vehicle dynamics (Bruder and Mikelsons 2021; Thummerer, Stoljar, and Mikelsons 2022), chemistry (Thebelt et al. 2022), climate modeling (Ramadhan et al. 2023), and process optimization (Misener and L. Biegler 2023).

Training neural components typically involves simulating (2) for some initial parameters $\boldsymbol{p}$ and then propagating sensitivities of the ODE solver backward in each iteration. Afterward, the parameters are updated via gradient descent. This process is computationally expensive and results in long training times (Lehtimäki, Paunonen, and Linne 2024; Roesch, Rackauckas, and Stumpf 2021; Shapovalova and Tsay 2025), as explicit integrators are low-order and unstable, requiring small step sizes, while stable, implicit integrators involve solving nonlinear systems at each step, thus being computationally demanding.

To address these enormous training times several alternative procedures have been proposed: in (Roesch, Rackauckas, and Stumpf 2021) a collocation technique is introduced, which approximates the right hand side (RHS) of an ODE from data. The NN is then trained on the approximations with standard training frameworks. Further, in (Lehtimäki, Paunonen, and Linne 2024) model order reduction is used to accurately simulate the dynamics in low-dimensional subspaces. Very recent work presented in (Shapovalova and Tsay 2025) introduced global direct collocation with Chebyshev nodes, a method originating from dynamic optimization for optimal control and parameter optimization, for training Neural ODEs. The approach reduces the continuous training problem to a large finite dimensional nonlinear program (NLP) and shows fast and stable convergence, demonstrated on a typical problem, the Van-der-Pol oscillator.

In Modelica-based workflows, the training of Neural ODEs is typically performed externally by exporting a Functional Mock-Up Unit (FMU), subsequently training it in Python or Julia using standard machine learning frameworks and ODE solvers, and finally re-importing the hybrid model. While this approach introduces external dependencies and additional transformation steps, the *NeuralFMU* workflow (Thummerer, Stoljar, and Mikelsons 2022) demonstrates a practical method for integrating hybrid models into real-world applications.

Building on recent advances in direct collocation-based training of Neural ODEs (Shapovalova and Tsay 2025), we significantly extend the approach to PeN-ODEs. We formulate the training process as a dynamic optimization problem and discretize both neural and physical components using a stable, high-order implicit collocation scheme at flipped Legendre-Gauss-Radau (fLGR) points. This results in a large but structured NLP, allowing efficient, simultaneous optimization of states and parameters. Our custom, parallelized implementation leverages second-order information and the open-source NLP solver Ipopt (Wächter and L. T. Biegler 2006). It is designed for a future integration into the open-source modeling and simulation environment OpenModelica (Fritzson, Pop, Abdelhak, et al. 2020), thus providing an accessible training environment independent of external tools.

2 Dynamic Optimization for NODEs

In this section, we introduce a general class of dynamic optimization problems (DOPs) and formulate training for both NODEs and PeN-ODEs as instances of this class. We then discuss the transcription of the continuous problem into a large-scale nonlinear optimization problem (NLP). Finally, necessary considerations and key challenges are presented.

2.1 Generic Problem Formulation

Consider the DOP

$$\min_{\boldsymbol{p}} M(\boldsymbol{x}(t_0),\boldsymbol{x}(t_f),\boldsymbol{p}) + \int_{t_0}^{t_f} L(\boldsymbol{x}(t),\boldsymbol{p},t)\,\mathrm{d}t \tag{3a}$$

s.t.

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t),\boldsymbol{p},t) \qquad \forall t \in T \tag{3b}$$

$$\boldsymbol{g}^L \le \boldsymbol{g}(\boldsymbol{x}(t),\boldsymbol{p},t) \le \boldsymbol{g}^U \qquad \forall t \in T \tag{3c}$$

$$\boldsymbol{r}^L \le \boldsymbol{r}(\boldsymbol{x}(t_0),\boldsymbol{x}(t_f),\boldsymbol{p}) \le \boldsymbol{r}^U \tag{3d}$$

for a fixed time horizon $T = [t_0, t_f]$ with time variable $t \in T$. The states of the system are given by $\boldsymbol{x} : T \to \mathbb{R}^{d_x}$ and the goal is to find optimal time-invariant parameters $\boldsymbol{p} \in \mathbb{R}^{d_p}$, such that the objective (3a) becomes minimal and the constraints (3b)–(3d) are satisfied. These constraints are divided into the ODE (3b) and *path constraints* (3c), which both must be satisfied at all times on time horizon T, as well as *boundary constraints* (3d), which must only hold at the initial and final time points t_0, t_f. The objective is composed of a *Mayer* term M, that defines a cost at the boundary of T, and a *Lagrange* term L, that penalizes an accumulated cost over the entire time horizon. To ensure compatibility with typical nonlinear optimizers, all model functions must be twice continuously differentiable. This includes neural networks, their activation functions, and error measures. For completeness, the bounds of the constraints are given as $\boldsymbol{g}^L, \boldsymbol{g}^U \in (\mathbb{R} \cup \{-\infty, \infty\})^{d_g}$ and $\boldsymbol{r}^L, \boldsymbol{r}^U \in (\mathbb{R} \cup \{-\infty, \infty\})^{d_r}$.

2.2 Reformulation of PeN-ODE Training

PeN-ODEs embed one or more NNs with parameters $\boldsymbol{p}$ into known, possibly equation-based, dynamics $\dot{\boldsymbol{x}}(t) = \hat{\boldsymbol{\phi}}(\boldsymbol{x},\boldsymbol{u},t)$. The resulting differential equation has the form

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{\phi}(\boldsymbol{x},\boldsymbol{u},t,NN_{\boldsymbol{p}}(\boldsymbol{x},\boldsymbol{u},t)), \tag{4}$$

where $\boldsymbol{u} : T \to \mathbb{R}^{d_u}$ is a fixed input vector and $NN_{\boldsymbol{p}}$ are enhancing NNs. This formalism aims to enhance systems that already express dynamics based on first principles, by further incorporating data-driven observables in form of neural components. Clearly, these components need not be NNs in general, and can be any parameter dependent expression. With additional information about the problem, one could use a polynomial, rational function, sum of radial basis functions or Fourier series.

The subsequent considerations also apply to the training of NODEs, where the goal is to learn the full dynamics without relying on a first principle model. This is evident from the fact that NODEs are a subclass of PeN-ODEs with

$$\dot{\boldsymbol{x}}(t) = NN_{\boldsymbol{p}}(\boldsymbol{x},\boldsymbol{u},t) = \boldsymbol{\phi}(\boldsymbol{x},\boldsymbol{u},t,NN_{\boldsymbol{p}}(\boldsymbol{x},\boldsymbol{u},t)). \tag{5}$$

In this paper, we propose a formulation for training PeN-ODEs as a DOP (3a)–(3d), using known data trajectories $\hat{\boldsymbol{q}}$ and the corresponding predicted quantity $\boldsymbol{q}$. The DOP takes the form

$$\min_{\boldsymbol{p}} \int_{t_0}^{t_f} E\left(\boldsymbol{q}(\boldsymbol{x},\boldsymbol{u},t,NN_{\boldsymbol{p}}(\boldsymbol{x},\boldsymbol{u},t)),\hat{\boldsymbol{q}}(t)\right)\,\mathrm{d}t \tag{6a}$$

s.t.

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{\phi}(\boldsymbol{x},\boldsymbol{u},t,NN_{\boldsymbol{p}}(\boldsymbol{x},\boldsymbol{u},t)) \quad \forall t \in T \tag{6b}$$

for some smooth error measure E, e.g. the squared 2-norm $E(\boldsymbol{q},\hat{\boldsymbol{q}}) = \|\boldsymbol{q} - \hat{\boldsymbol{q}}\|_2^2$. This formulation represents the minimal setup.

By further incorporating the generic constraints (3c) and (3d), it is possible to impose an initial or final condition on the states as well as enforce desired behavior. For example, consider a NN approximation of a force element NN_F, with no force acting in its resting position. Therefore, the NN should have a zero crossing, i.e. $NN_F(0) = 0$. This can be trivially formulated as a constraint, without introducing a penalty term that may distort the optimization as in standard unconstrained approaches like (Kamp,

DOI 10.3384/ecp218445

Ultsch, and Brembeck 2023). Thus, the optimizer can handle the constraint appropriately.

As the loss in (6a) is a continuous-time integral, it enables an accurate and stable approximation using the same discretization employed for the system dynamics. In contrast to MSE loss as in (Shapovalova and Tsay 2025), which effectively corresponds to a first-order approximation of the integral, our formulation benefits from high-order quadrature, potentially preserving the accuracy of the underlying discretization.

2.3 Transcription with Direct Collocation

In the following, the general DOP (3a)–(3d) is reduced to a NLP using orthogonal direct collocation. Direct collocation approaches have proven to be highly efficient in solving DOPs and are implemented in a variety of free and commercial tools, such as PSOPT (Becerra 2010), CasADi (Andersson et al. 2019) or GPOPS-II (Patterson and A. V. Rao 2014), as well as in Modelica-based environments like OpenModelica (Ruge et al. 2014) or JModelica (Magnusson and Åkesson 2015). While OpenModelica only supports optimal control problems, the other frameworks allow for simultaneous optimization of static parameters. Recent work in the field of NODEs (Shapovalova and Tsay 2025) shows that learning the right hand side of small differential equations can be performed stably and efficiently using global collocation with Chebyshev nodes.

In direct collocation the states are approximated by piecewise polynomials, that satisfy the differential equation at so-called *collocation nodes*, usually chosen as roots of certain orthogonal polynomials. If the problem is smooth and with increasing number of collocation nodes, these methods achieve *spectral*, i.e. exponential, convergence to the exact solution. In this paper, the collocation nodes are chosen as the *flipped Legendre-Gauss-Radau points (fLGR)* rescaled from $[-1,1]$ to $[0,1]$. This rescaling is performed, so that the corresponding collocation method is equivalent to the Radau IIA Runge-Kutta method. Radau IIA has excellent properties, since it is A-, B- and L-stable and achieves order $2m-1$ for m stages or collocation nodes. These nodes c_j for $j = 1,\dots,m$ are given as the m roots of the polynomial $(1-t)P^{(1,0)}_{m-1}(2t-1)$, where $P^{(1,0)}_{m-1}$ is the $(m-1)$-th Jacobi polynomial with $\alpha = 1$ and $\beta = 0$. A detailed explanation of the method's construction based on quadrature rules is given in (Langenkamp 2024).

First, we divide the time horizon $[t_0, t_f]$ into $n+1$ intervals $[t_i, t_{i+1}]$ for $i = 0,\dots,n$ with length $\Delta t_i := t_{i+1} - t_i$. In each interval $[t_i, t_{i+1}]$ the collocation nodes $t_{ij} := t_i + c_j \Delta t_i$ for $j = 1,\dots,m_i$ as well as the first grid point $t_{i0} := t_i + c_j \Delta t_i$ with $c_0 = 0$ are added. Since the last node $c_{m_i} = 1$ is contained in any Radau IIA scheme, the last grid point of interval $i-1$ exactly matches the first grid point of interval i, i.e. $t_{i-1,m_{i-1}} = t_{i0}$. Furthermore, the states are approximated as $\boldsymbol{x}(t_{ij}) \approx \boldsymbol{x}_{ij}$ and for each interval i replaced by a Lagrange interpolating polynomial $\boldsymbol{x}_i(t) = \sum_{j=0}^{m_i} \boldsymbol{x}_{ij}\, l_j(t)$ of degree m_i, where

$$l_j(t) := \prod_{\substack{k=0 \\ k\neq j}}^{m_i} \frac{t - t_{ik}}{t_{ij} - t_{ik}} \quad \forall j = 0,\dots,m_i \tag{7}$$

are the Lagrange basis polynomials. Note that the parameters $\boldsymbol{p}$ are time-invariant and thus, need not be discretized. Each $\boldsymbol{x}_i$ must satisfy the differential equation (3b) at the collocation nodes t_{ij} and also match the initial condition $\boldsymbol{x}_{i0}$, which is given from the previous interval $i-1$. By differentiating we get the collocated dynamics

$$\mathbf{0} = \begin{bmatrix} D^{(1)}_{10} I & \dots & D^{(1)}_{1m_i} I \\ \vdots & \ddots & \vdots \\ D^{(1)}_{m_i 0} I & \dots & D^{(1)}_{m_i m_i} I \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_{i0} \\ \vdots \\ \boldsymbol{x}_{im_i} \end{bmatrix} - \Delta t_i \begin{bmatrix} \boldsymbol{f}_{i1} \\ \vdots \\ \boldsymbol{f}_{im_i} \end{bmatrix} \tag{8}$$

with identity matrix $I \in \mathbb{R}^{d_x \times d_x}$, entries of the first differentiation matrix $D^{(1)}_{jk} := \frac{\mathrm{d}\tilde{l}_k}{\mathrm{d}\tau}(c_j)$, where

$$\tilde{l}_k(\tau) := \prod_{\substack{r=0 \\ r\neq k}}^{m_i} \frac{\tau - c_r}{c_k - c_r} \; \forall k = 0,\dots,m_i, \tag{9}$$

and the RHS of the ODE $\boldsymbol{f}_{ij} := \boldsymbol{f}(\boldsymbol{x}_{ij}, \boldsymbol{p}, t_{ij})$. The numerical values of $D^{(1)}_{jk}$ can be calculated very efficiently with formulas provided in (Schneider and Werner 1986).

Approximating L is analogous to discretizing the differential equation. This is done by replacing the integral with a Radau quadrature rule of the form

$$\int_{t_0}^{t_f} L(\boldsymbol{x}(t), \boldsymbol{p}, t)\,\mathrm{d}t \approx \sum_{i=0}^{n} \Delta t_i \sum_{j=1}^{m_i} b_j L(\boldsymbol{x}_{ij}, \boldsymbol{p}, t_{ij}), \tag{10}$$

where the quadrature weights are given by

$$b_j = \int_0^1 \prod_{\substack{k=1 \\ k\neq j}}^{m_i} \frac{\tau - c_k}{c_j - c_k}\,\mathrm{d}\tau \quad \forall j = 1,\dots,m_i. \tag{11}$$

M and the boundary constraints (3d) are approximated by replacing the values on the boundary with their discretized equivalents, i.e. $M(\boldsymbol{x}(t_0), \boldsymbol{x}(t_f), \boldsymbol{p}) \approx M(\boldsymbol{x}_{00}, \boldsymbol{x}_{nm_n}, \boldsymbol{p})$ and $\boldsymbol{r}(\boldsymbol{x}(t_0), \boldsymbol{x}(t_f), \boldsymbol{p}) \approx \boldsymbol{r}(\boldsymbol{x}_{00}, \boldsymbol{x}_{nm_n}, \boldsymbol{p})$, while the path constraints (3c) are evaluated at all nodes, i.e. $\boldsymbol{g}(\boldsymbol{x}(t_{ij}), \boldsymbol{p}, t_{ij}) \approx \boldsymbol{g}(\boldsymbol{x}_{ij}, \boldsymbol{p}, t_{ij})$.

2.4 Training with Nonlinear Programming

By flattening the collocated dynamics (8), we obtain the discretized DOP (3a)–(3d) of the form

$$\min_{\boldsymbol{x}_{ij}, \boldsymbol{p}} M(\boldsymbol{x}_{00}, \boldsymbol{x}_{nm_n}, \boldsymbol{p}) + \sum_{i=0}^{n} \Delta t_i \sum_{j=1}^{m_i} b_j L(\boldsymbol{x}_{ij}, \boldsymbol{p}, t_{ij}) \tag{12a}$$

s.t.

$$\mathbf{0} = \sum_{k=0}^{m_i} D_{jk}^{(1)} \boldsymbol{x}_{ik} - \Delta t_i \boldsymbol{f}(\boldsymbol{x}_{ij}, \boldsymbol{p}, t_{ij}) \quad \forall i, \forall j \geq 1 \quad (12b)$$

$$\boldsymbol{g}^L \leq \boldsymbol{g}(\boldsymbol{x}_{ij}, \boldsymbol{p}, t_{ij}) \quad \leq \boldsymbol{g}^U \quad \forall i, \forall j \geq 1 \quad (12c)$$

$$\boldsymbol{r}^L \leq \boldsymbol{r}(\boldsymbol{x}_{00}, \boldsymbol{x}_{nm_n}, \boldsymbol{p}) \leq \boldsymbol{r}^U \quad (12d)$$

This large-scale NLP (12a)–(12d) can be implemented and solved efficiently in nonlinear optimizers such as Ipopt (Wächter and L. T. Biegler 2006), SNOPT (P. E. Gill et al. 2007; Philip E. Gill, Murray, and Saunders 2005) or KNITRO (Byrd, Nocedal, and Waltz 2006). These NLP solvers exploit the sparsity of the problem as well as the first and second order derivatives of the constraint vector and objective function to converge quickly to a suitable local optimum.

The open-source interior-point method Ipopt requires the already mentioned first derivatives and, in addition, the Hessian of the augmented Lagrangian at every iteration. Since the derivatives only need to be evaluated at the collocation nodes, there is no need to propagate them as in traditional ODE solver-based training. Furthermore, training with ODE solvers usually limits itself to first order derivatives and therefore, does not utilize higher order information as in the proposed approach.

The resulting so-called *primal-dual* system is then solved using a linear solver for symmetric indefinite systems, such as the open-source solver MUMPS (Amestoy et al. 2001) or a proprietary solver from the HSL suite (HSL 2013), after which an optimization step is performed. This step updates all variables $\boldsymbol{x}_{ij}, \boldsymbol{p}$ simultaneously, allowing for direct observation and adjustment of intermediate values. In contrast, ODE solver training only captures the final result after integrating the dynamics over time, without the ability to directly influence intermediate states during the optimization process. Because this linear system must be solved in every iteration anyway, high order, stable, implicit Runge-Kutta collocation methods, e.g. Radau IIA, can be embedded with only limited overhead. As a result, the NLP formulation overcomes key limitations of explicit ODE solvers in terms of order, stability, and allowable step size. Moreover, since the solver performs primal and dual updates, the solution does not need to remain feasible during the optimization, in contrast to ODE solver approaches where the dynamics (3b) are enforced at all times through forward simulation. This results in both advantages and disadvantages: On the one hand, it enables more flexible and aggressive updates, potentially accelerating convergence. On the other hand, it may lead to intermediate solutions that temporarily violate physical consistency or produce invalid function evaluations, which require careful handling.

For a comprehensive overview of nonlinear programming, interior-point methods, and their application to collocation-based dynamic optimization, we refer interested readers to (L. T. Biegler 2010) and the Ipopt implementation paper (Wächter and L. T. Biegler 2006).

2.5 Challenges and Practical Aspects

We identify four main challenges in training PeN-ODEs using the proposed direct collocation and nonlinear programming approach. These challenges are closely related to those encountered in conventional PeN-ODE or general NN training.

2.5.1 Grid Selection

The choice of time grid $\{t_0, \ldots, t_{n+1}\}$ and the number of collocation nodes per interval m_i are crucial for both the accuracy and efficiency of the training process. In practice, the grid can either be chosen equidistant or tailored to the specific problem. While equidistant grids are straightforward to implement and often sufficient for well-behaved systems, non-equidistant grids may reduce computational costs while capturing the dynamics more efficiently. Placing more intervals with low degree collocation polynomials in regions of rapid state change can improve approximation quality without unnecessarily increasing the problem size. Similarly, in well-behaved regions, it is feasible to perform larger steps with more collocation nodes. Because the collocation scheme and grid are embedded into the NLP, these must be given *a-priori*. This leaves room for future developments of adaptive mesh refinement methods with effective mesh size reduction, which have already shown great success for optimal control problems (Zhao and Shang 2018; Liu, Hager, and A. Rao 2015).

2.5.2 Initial Guesses

Due to the size and possible nonlinearity of the resulting NLP, the choice of initial guesses has a strong influence on convergence behavior. Unlike classical NN training, where poor initialization primarily affects convergence speed, the constrained nature of the transcribed dynamic optimization problem can lead to poor local optima or even solver failure. It is therefore of high importance to perform informed initializations for the states $\boldsymbol{x}_{ij}$ and, if possible, for the NN parameters $\boldsymbol{p}$.

One practical approach to obtain the required parameter guesses is to first train the network on a small, representative subset of the full dataset using constant initial values for both the states and parameters. The optimized parameters resulting from this reduced problem then serve as informed initial guesses for the full training problem. Consequently, the states are obtained by simulation, i.e. $\boldsymbol{x}(t) := \text{ODESolve}\left(\boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{u}, t, NN_{\boldsymbol{p}}(\boldsymbol{x}, \boldsymbol{u}, t), \boldsymbol{x}_0)\right)$ and $\boldsymbol{x}_{ij} := \boldsymbol{x}(t_{ij})$ for a given initial condition $\boldsymbol{x}(t_0) = \boldsymbol{x}_0$. By construction, the collocated dynamic constraints (12b) are satisfied, leading to improved convergence and stability in the full NLP.

Clearly, this strategy does not work in general. However, in simple cases where the model can be decomposed and the NN's input-output behavior is observable, e.g. if model components should be replaced by a neural surrogate, the NN can be pre-trained using standard gradient descent. This yields reasonable initial guesses

DOI 10.3384/ecp218445

for the parameters and states by simulation, which can then be integrated into the constrained optimization problem. Another pre-training strategy could be the *collocation technique* proposed in (Roesch, Rackauckas, and Stumpf 2021). Still, developing general, effective strategies to obtain reasonable initial guesses is one of the key challenges and limiting factors we identify for the general application of this approach.

2.5.3 Batch-wise Training

Standard ML frameworks employ batch learning to efficiently split up data. This is not as straightforward when training with the approach described here. One might assume that the entire dataset must be included in a single discretized DOP. However, recent work (Shapovalova and Tsay 2025) demonstrates that batch-wise training is possible and promises significant potential. The Alternating Direction Method of Multipliers (ADMM) (Boyd et al. 2011) allows decomposing the optimization problem into smaller subproblems that can be trained independently, while enforcing consensus between them. This allows for memory-efficient training and opens up the possibility of handling larger models or learning from multiple data trajectories simultaneously.

2.5.4 Training of Larger Networks

While computing Hessians of NNs is generally expensive, it is tractable for small networks. To reduce computational effort for larger networks, it might be reasonable to use partial Quasi-Newton approximations such as SR1, BFGS or DFP (L. T. Biegler 2010) to approximate the dense parts of the augmented Lagrangian Hessian H, e.g. the blocks $H_{\boldsymbol{x}_{ij},\boldsymbol{p}}$ and $H_{\boldsymbol{pp}}$, or solely $H_{\boldsymbol{pp}}$. These blocks, which contain derivatives with respect to the NN parameters, are computationally expensive, while the block $H_{\boldsymbol{x}_{ij},\boldsymbol{x}_{ij}}$, which contains second derivatives with respect to the collocated states, is extremely sparse and comparably cheap. A Quasi-Newton approximation of the block $H_{\boldsymbol{x}_{ij},\boldsymbol{x}_{ij}}$ is therefore disadvantageous. The sparsity can be exploited by computing this block analytically and using it directly in the Quasi-Newton update. An implementation of this partial update using the SR1 Quasi-Newton method is straightforward. Instead of one expensive symmetric rank-one update for the entire Hessian H, one cheap symmetric rank-one update for $H_{\boldsymbol{pp}}$ and one general rank-one update for $H_{\boldsymbol{x}_{ij},\boldsymbol{p}}$ are needed.

This procedure significantly reduces the cost of the Hessian, while still providing fairly detailed derivative information. SR1 is particularly advantageous here, as it can represent indefinite Hessians, which is favorable when dealing with highly nonlinear functions.

Since our current examples perform well with small NNs, we do not explore larger networks in this paper. However, we anticipate that such Quasi-Newton strategies will be necessary in future work with larger networks. Very recent work (Lueg et al. 2025) independently expresses similar ideas, highlighting the potential of the approach.

3 Implementation

The generic DOP (3a)–(3d) and its corresponding NLP formulation (12a)–(12d) are implemented in the custom open-source framework GDOPT (Langenkamp 2024), which is publicly available.[1] For neural network training the code has been extended, including predefined parametric blocks such as neural networks, support for data trajectories and parallelized optimizations. This extended, experimental version is also publicly available.[2]

3.1 GDOPT

GDOPT (General Dynamic Optimizer) consists of two main components: an expressive Python-based package *gdopt* and an efficient C++ library *libgdopt*. The Python interface provides an user-friendly modeling environment and performs symbolic differentiation and code generation. Symbolic expressions are optimized using common subexpression elimination via SymEngine[3], and the resulting expressions together with first and second derivatives are translated into efficient C++ callback functions for runtime evaluation. In the present implementation, all expressions are flattened, resulting in large code, especially for the Hessian. Note that keeping NNs vectorized and employing symbolic differentiation rules with predefined NN functions offers significant advantages.

The library *libgdopt* implements a generalized version of the NLP (12a)–(12d) using Radau IIA collocation schemes, while also supporting optimal control problems. It is interfaced with Ipopt to solve the resulting nonlinear programs. Both symbolic Jacobian and Hessian rely on exact sparsity patterns discovered in the Python interface. Additional functionality includes support for nominal values, initial guesses, runtime parameters, mesh refinements, plotting utilities, and special functions. A detailed overview of features and modeling is provided in the GDOPT User's Guide[4].

Nevertheless, GDOPT lacks important capabilities that established modeling languages and tools offer, such as object-oriented, component-based modeling and support for differential-algebraic equation (DAE) systems. It is possible to model DAEs by introducing control variables for algebraic variables. However, this approach increases the workload and size of the NLP and may lead to instabilities.

3.2 Parallel Callback Evaluations

Since in every optimization step, the function evaluations as well as first and second derivatives of all NLP components (12a)–(12d) must be provided to Ipopt, an efficient callback evaluation is crucial to accelerate the training.

[1] https://github.com/linuslangenkamp/GDOPT
[2] https://github.com/linuslangenkamp/GDOPT_DEV
[3] https://github.com/symengine/symengine
[4] https://github.com/linuslangenkamp/GDOPT/blob/master/usersguide/usersguide.pdf

Note that these callbacks themselves consist of the continuous functions evaluated at all collocation nodes. For simplicity we write $\boldsymbol{z}_{ij} := [\boldsymbol{x}_{ij}, \boldsymbol{p}]^T$ for the variables at a given collocation node and, in addition,

$$\boldsymbol{\psi}(\boldsymbol{x}, \boldsymbol{p}, t) := [L(\cdot), \boldsymbol{f}(\cdot), \boldsymbol{g}(\cdot)]^T \tag{13}$$

for the vector of functions that are evaluated at all nodes. Clearly, the required callbacks

$$\boldsymbol{\psi}\big|_{z_{ij}} \quad \nabla\boldsymbol{\psi}\big|_{z_{ij}} \quad \nabla^2\boldsymbol{\psi}\big|_{z_{ij}} \quad \forall i\, \forall j \geq 1$$

are independent with respect to the given collocation nodes t_{ij} and thus, allow for a straightforward parallelization. In the extension of GDOPT, we use OpenMP (Chandra et al. 2001) to parallelize the callback evaluations required by Ipopt. Depending on the specific callback, i.e. objective function, constraint violation, gradient, Jacobian, or Hessian of the augmented Lagrangian, a separate `omp parallel for` loop is used to evaluate the corresponding components at all collocation nodes. This design results in a significant reduction in computation time, especially for the comparably expensive dense derivatives of neural components.

4 Performance

In order to test the proposed training method and parallel implementation, two example problems are considered. The first example is the Quarter Vehicle Model (QVM) from (Kamp, Ultsch, and Brembeck 2023), where an equation-based model is enhanced with small neural components, such that physical behavior is represented more accurately. The second example is a standard NODE, where the dynamics of a Van-der-Pol oscillator (Roesch, Rackauckas, and Stumpf 2021) are learned purely from data. In both cases, the experimental setup closely follows the configurations used in the respective paper. Training is performed on a laptop running Ubuntu 24.04.2 with Intel Core i7-12800H (20 threads), 32 GB RAM and using GCC v13.3.0 with flags `-O3 -ffast-math` for compilation, while MUMPS (Amestoy et al. 2001) is used to solve linear systems arising from Ipopt (Wächter and L. T. Biegler 2006). All dependencies are free to use and open-source.

4.1 Quarter Vehicle Model

We follow the presentation in (Kamp, Ultsch, and Brembeck 2023) for an overview of the model and the data generation process. The Quarter Vehicle Model (QVM) captures the vertical dynamics of a road vehicle by modeling one wheel and the corresponding quarter of the vehicle body. It consists of two masses connected by spring-damper elements representing the suspension and tire dynamics. The linear base model is described by the differential equations $\dot{z}_r = u$, $\dot{z}_b = v_b$, $\dot{z}_w = v_w$, and

$$\dot{v}_b := a_b = m_b^{-1}\left(c_s \Delta z_s + d_s \Delta v_s\right) \tag{14a}$$

$$\dot{v}_w := a_w = m_w^{-1}\left(c_t \Delta z_t + d_t \Delta v_t - c_s \Delta z_s - d_s \Delta v_s\right) \tag{14b}$$

where $\Delta z_s = z_w - z_b$, $\Delta v_s = v_w - v_b$, $\Delta z_t = z_r - z_w$, $\Delta v_t = u - v_w$. In addition, m_w is the mass of the wheel, m_b is mass of the quarter body, and c_s and d_s are the coefficients of the linear spring-damper pair between these masses, modeling the suspension. Furthermore, c_t and d_t define an additional linear spring-damper pair between the tire and ground. The state vector $\boldsymbol{x} = [z_b, z_w, v_b, v_w, z_r]^T$ contains the positions of the body z_b and wheel z_w, their velocities v_b and v_w, and the road height z_r. The differential road height $\dot{z}_r = u$ is given as an input and the observable outputs $y = [a_w, a_b]$ measure the wheel and body accelerations. A corresponding Modelica model of the linear QVM is depicted in Figure 1. (Kamp, Ultsch, and Brembeck 2023)

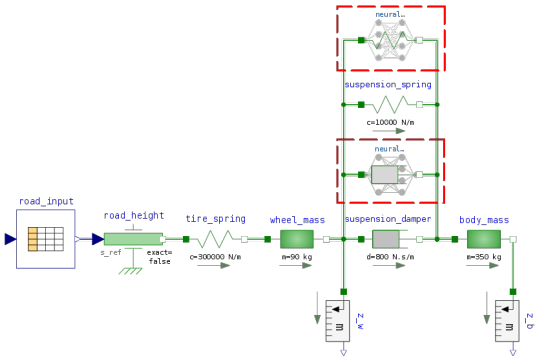


Figure 1. Modelica Models of the Linear (without boxes) and Neural (with boxes) QVM. Modified from T. Kamp.

4.1.1 Data Generation

The linear model is extended by introducing two additional nonlinear forces between both masses, a translational friction force F_{fr} and a progressive spring characteristic F_{pr}. These nonlinear components are introduced only for data generation, creating a more complicated, nonlinear model whose behavior deviates from the known base dynamics. Therefore, the differential equations (14a) and (14b) become

$$\dot{v}_b = m_b^{-1}\left(c_s \Delta z_s + d_s \Delta v_s + F_{pr}(\Delta z_s) + F_{fr}(\Delta v_s)\right) \tag{15a}$$

$$\dot{v}_w = m_w^{-1}\left(c_t \Delta z_t + d_t \Delta v_t - c_s \Delta z_s - d_s \Delta v_s - F_{pr}(\Delta z_s) - F_{fr}(\Delta v_s)\right) \tag{15b}$$

To generate data, a simulation of the nonlinear model for an imitation of a realistic, rough road (ISO8608, Type D (Múčka, Peter 2018)) is performed. The observables a_b and a_w as well as the states are disturbed by random Gaussian noise and sampled with 1000 Hz for a 42 s trajectory as in (Kamp, Ultsch, and Brembeck 2023).

4.1.2 Training Setup

The nonlinear components $F_{fr}(\Delta v_s)$ and $F_{pr}(\Delta z_s)$ introduced for data generation are replaced by two neural network surrogates $F_{fr}^{NN}(\Delta v_s)$ and $F_{pr}^{NN}(\Delta z_s)$, illustrated in Figure 1. The goal is to find suitable replacements that minimize the mismatch between the simulated, disturbed

DOI 10.3384/ecp218445

outputs $\hat{a}_b$ and $\hat{a}_w$ of the nonlinear model and the observed data during the optimization. As discussed before, we write this objective as an integral over the entire time horizon, i.e.

$$\min \int_{t_0}^{t_f} \left(\frac{\hat{a}_b - a_b}{\sigma_{\hat{a}_b}} \right)^2 + \left(\frac{\hat{a}_w - a_w}{\sigma_{\hat{a}_w}} \right)^2 \mathrm{d}t, \tag{16}$$

where $\sigma_{\hat{a}_b}$ and $\sigma_{\hat{a}_w}$ are corresponding standard deviations of the data, ensuring that both accelerations contribute equally to the objective. Furthermore, it is known that both force elements have a zero crossing and therefore, the additional constraints

$$F_{fr}^{NN}(0) = 0 \quad \text{and} \quad F_{pr}^{NN}(0) = 0 \tag{17}$$

are simply added to the optimization problem.

We employ three different training strategies. At first, both feedforward neural networks are trained directly on the full trajectory using randomly initialized parameters, while the initial state guesses are obtained from a simulation of the linear model (I). Each network has the structure $1 \times 5 \rightarrow 5 \times 5 \rightarrow 5 \times 1$ and therefore, both nets contain just 92 parameters in total. We use the smooth squareplus activation function

$$\text{squareplus}(x) := \frac{x + \sqrt{1 + x^2}}{2} \tag{18}$$

to ensure a twice continuously differentiable NN as required for Ipopt. In the second strategy (II), described in Section 2.5.2, first an acceleration scheme is employed, where the same networks are trained on a short segment one eighth of the entire trajectory. After that, a simulation of the neural QVM with the obtained parameters is performed. The resulting states are used as initial guesses in the subsequent optimization with full data. To show that the surrogates need not be NNs and training can be performed efficiently with other parameter-dependent expressions, the third strategy (III) uses rational functions to model the unknown behavior. For instance, F_{fr} is replaced by

$$F_{fr}^{RC}(\Delta v_s) := \frac{\sum_{k=0}^{N} \omega_k T_k(\Delta v_s)}{\sum_{k=0}^{D} \theta_k T_k(\Delta v_s)}, \tag{19}$$

where T_k is the k-th Chebyshev polynomial, ω_k and θ_k are parameters to be optimized, and N and D are the numerator and denominator degrees. For both rational functions we choose $N = D = 7$, resulting in a total of merely 32 learnable parameters.

Since the QVM contains very fast dynamics due to high-frequency excitations, in all cases the time horizon is divided into a tightly spaced, equidistant grid of 2500 intervals and using a constant 5-step Radau IIA collocation scheme of order 9. This leads to a total of 12500 collocation nodes and more than 2.7×10^6 nonzeros in the Jacobian and roughly 4.73×10^6 nonzeros in the Hessian of the large-scale NLP.

4.1.3 Results

Table 1 presents the training times for all strategies. Each optimization was run for a maximum of 150 NLP iterations and was automatically terminated early if no further significant improvement in objective could be achieved. We want to stress that all trainings, performed on a laptop, are executed in under 7 minutes, compared to 4.5 hours for the fastest optimization in (Kamp, Ultsch, and Brembeck 2023) using ODE solver-based training. Clearly, this is also due to the fact that smaller neural components are used. Furthermore, Figure 2 depicts the objective value with respect to training time, where both the first and second optimizations of (II) are concatenated.

Strategy	Total	Ipopt	Callbacks
(I)	394.43	284.18	110.25
(II) - initial	26.75	15.08	11.67
(II) - final	173.06	124.65	48.41
(II)	199.81	139.73	60.07
(III)	34.96	27.10	7.85

Table 1. Training Times in Seconds

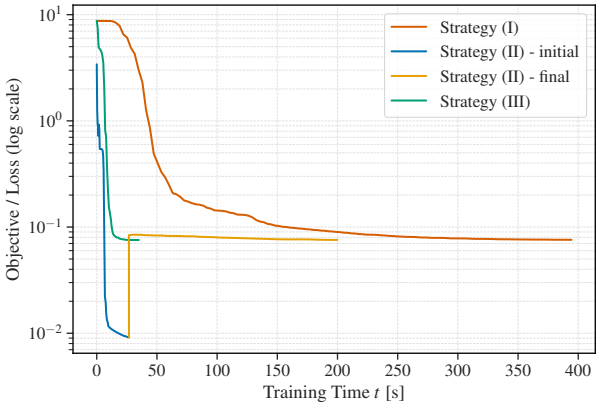


Figure 2. Objective History with Respect to Training Time

Even though a poor initialization has been used and thus the optimization required more time, the naive strategy (I) converged stably to a suitable optimum with a similar objective as strategies (II) and (III). For this example, the acceleration scheme (II) proves effective, since the initial optimization for a shorter data trajectory takes just 26.75 s in total, and the subsequent initial guesses for states and parameters become very good approximations of the real solution. This can be observed in Figure 3 and Figure 4, where the resulting neural surrogates are depicted. By performing the second optimization, (II) effectively halves the time required by strategy (I), i.e. less than 3.5 minutes, and furthermore results in indistinguishable neural components.

Moreover, as seen in Figure 3 and Figure 4, the resulting very small NNs match the reference in almost perfect

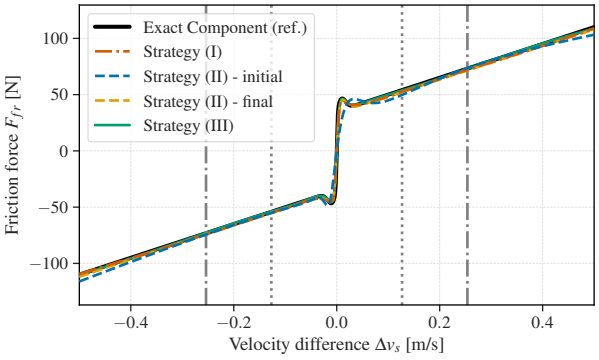


Figure 3. Neural and Reference Damper Characteristics

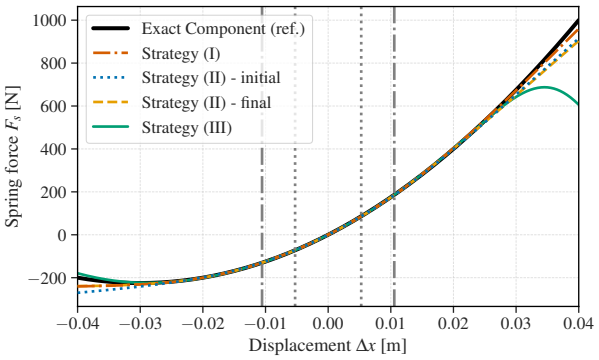


Figure 4. Neural and Reference Spring Characteristics

accordance, correctly representing highly nonlinear parts of the damper. These also generalize in a very natural way, as can be seen from the behavior outside the vertical dotted lines, which represent the first and second standard deviations of the inputs to the nets. Performing simulations on a new, unknown input road shows that the obtained PeNODE from (II) matches the nonlinear reference model perfectly, as illustrated in Figure 5. Note that, our characteristics of the damper and spring serve as even better surrogates than those reported in (Kamp, Ultsch, and Brembeck 2023), despite using significantly smaller networks and requiring considerably shorter training times. While (Kamp, Ultsch, and Brembeck 2023) relied on larger models with longer ODE solver-based training, our approach yields more accurate and better-generalizing results.

By having principal knowledge of the underlying characteristics shown in Figure 3 and Figure 4, it is possible to model observed behavior with minimal parameters. Since the number of Hessian nonzeros grows quadratically with the number of parameters, such knowledge of the procedure can greatly benefit both training time and surrogate quality. Therefore, consider simple rational functions as an educated guess for both missing components. This optimization, with 32 instead of 92 free parameters, is performed in under 35 seconds without any acceleration strategy. Moreover, the obtained surrogates are of mostly equal quality to the NN components from (II), although

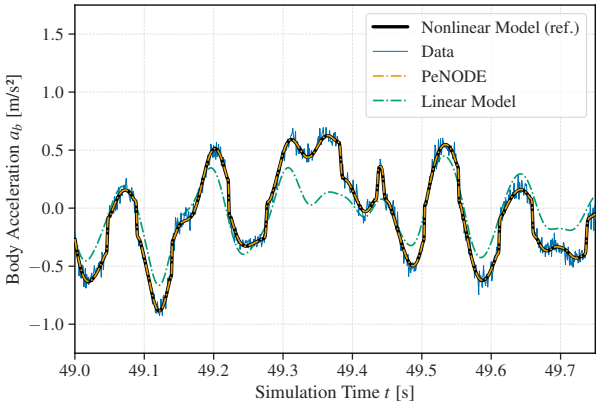


Figure 5. Body Accelerations a_b for Simulations of PeNODE and Standard Models on a Type C Road (Múčka, Peter 2018)

Figure 4 shows that the rational function does not generalize as well. Nevertheless, these results demonstrate that unknown behavior may be expressed with fewer parameters and yield equivalent quality.

4.1.4 Parallel Implementation

Finally, it is stressed that the parallel implementation, described in Section 3.2, leads to 5.44 times less time taken in the generated function callbacks. This yields a total training time that is more than halved and clearly shows that the implementation efficiently exploits the independence of collocation nodes.

Method	Total	Ipopt	Callbacks
GDOPT (default)	385.90	122.52	263.38
GDOPT (parallel)	173.06	124.65	48.41

Table 2. Comparison of Parallel and Sequential Optimization Times in Seconds of (II) - final

4.2 Van-der-Pol Oscillator

To illustrate the ability of learning a full NODE, we follow an example from (Roesch, Rackauckas, and Stumpf 2021), where a different kind of collocation method was proposed. This method approximates the RHS of the ODE with data, thus enabling faster, unconstrained training without the need for ODE solvers. Consider the Van-der-Pol (VdP) oscillator

$$\dot{x} = y \tag{20a}$$

$$\dot{y} = \mu y \left(1 - x^2\right) - x \tag{20b}$$

with $\mu = 1$ and initial conditions $x(t_0) = 2$ and $y(t_0) = 0$. Data generation is performed by simulating the dynamics with OpenModelica (Fritzson, Pop, Abdelhak, et al. 2020) on an equidistant grid with 200 intervals and artificially perturbing the observed states by additive Gaussian noise. To test sensitivities of the approach, we use 3 different

DOI
10.3384/ecp218445

levels of noise $\mathcal{N}(0,\sigma)$ to disturb the observable states, i.e. no noise ($\sigma = 0$), low noise ($\sigma = 0.1$) and high noise ($\sigma = 0.5$).

The continuous DOP has the form

$$\min_{\boldsymbol{p}_x,\boldsymbol{p}_y} \int_{t_0}^{t_f} (\hat{x}_\sigma - x)^2 + (\hat{y}_\sigma - y)^2 \,\mathrm{d}t + \lambda \|\boldsymbol{p}\|_2^2 \tag{21a}$$

s.t.

$$\dot{x} = NN^x_{\boldsymbol{p}_x}(x,y) \tag{21b}$$

$$\dot{y} = NN^y_{\boldsymbol{p}_y}(x,y) \tag{21c}$$

$$x(t_0) = 2,\ y(t_0) = 0, \tag{21d}$$

where $\hat{x}_\sigma, \hat{y}_\sigma$ is the disturbed state data, $NN^x_{\boldsymbol{p}_x}(x,y)$, $NN^y_{\boldsymbol{p}_y}(x,y)$ are neural networks of the architecture $2 \times 5 \to 5 \times 5 \to 5 \times 1$ with sigmoid activation function, $\boldsymbol{p} = [\boldsymbol{p}_x, \boldsymbol{p}_y]^T$ are 102 learnable parameters, while $\lambda > 0$ is a regularization factor to enhance stability.

As in (Roesch, Rackauckas, and Stumpf 2021), the training is performed on a 7 second time horizon, thus including a little over one period of the oscillator. Furthermore, 500 intervals and the 5-step Radau IIA method of order 9 are used. The initial guesses for the state variables are trivially chosen as the constant initial condition and the NN parameters are initialized randomly. No acceleration strategy or simulation is used for educated initial guesses. We set a maximum number of Ipopt iterations / epochs of 200 and an optimality tolerance of 10^{-7}.

4.2.1 Results

In almost all cases, the optimization terminates prematurely, since the optimality tolerance is fulfilled and thus, a local optimum was found. The corresponding training times are displayed in Table 3. Note that, because the high noise ($\sigma = 0.5$) leads to larger objective values, the regularization λ is increased in this case. Nevertheless, the optimization with a total of 2500 discrete nodes is very rapid and terminates in several seconds from an extremely poor initial guess, while in some instances runs settle in poor local optima. Clearly, more reasonable initial guesses and sophisticated initialization strategies will further enhance these times and stability.

σ	λ	Total	Ipopt	Callbacks	#Epochs
0	10^{-4}	8.17	5.95	2.22	80
0.1	10^{-4}	7.69	5.60	2.09	76
0.5	10^{-3}	13.49	9.79	3.70	134

Table 3. Training Times (in seconds) and Number of Ipopt Iterations / Epochs of the Van-der-Pol Oscillator NNs

Figure 6 presents the simulation results and training data for various levels of noise. It is evident that for no or low noise, the solution obtained is virtually indistinguishable from the reference, which is quite impressive. Even with high noise ($\sigma = 0.5$), the Neural ODE remains close to the true solution, showing significantly better performance compared to the results reported in (Roesch, Rackauckas, and Stumpf 2021), where NODE and reference do not align for $\sigma \geq 0.2$. While smaller neural networks are employed here, these compact models still demonstrate exemplary performance and fast training, highlighting the effectiveness of the approach under severe noise.

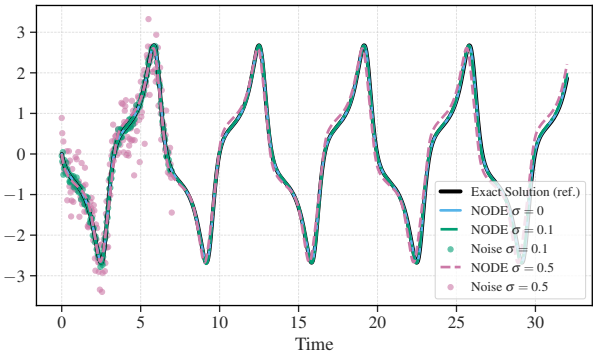


Figure 6. Simulation Results of the Neural and Reference Models as well as the Data for $y(t)$

To verify the robustness of the training procedure, a comprehensive sensitivity analysis, consisting of 100 training runs for each noise level, was conducted. The results, detailed in the Appendix and Figure 10, demonstrate that while all runs converge perfectly for the no-noise case ($\sigma = 0$) and approximately 95% show excellent agreement under low noise ($\sigma = 0.1$), as expected the robustness diminishes with high noise ($\sigma = 0.5$). In these cases, several runs converge to poor local optima or fail to converge, leading to solutions with noticeable period and amplitude mismatches or an outright collapse of the trajectory.

To further illustrate the obtained NODEs, we compare the learned and true vector fields of the ODE

$$\begin{bmatrix} x \\ y \end{bmatrix} \mapsto \begin{bmatrix} NN^x_{\boldsymbol{p}_x}(x,y) \\ NN^y_{\boldsymbol{p}_y}(x,y) \end{bmatrix}, \quad \begin{bmatrix} x \\ y \end{bmatrix} \mapsto \begin{bmatrix} y \\ \mu y\left(1-x^2\right) - x \end{bmatrix}. \tag{22}$$

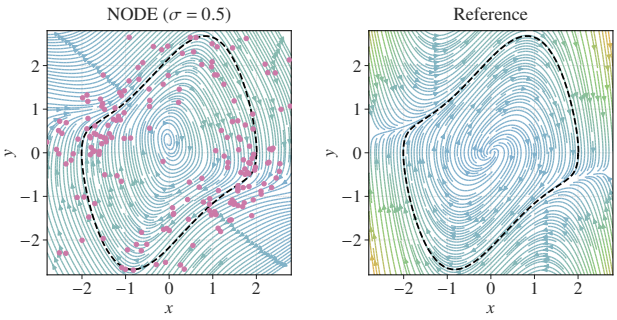


Figure 7. Neural ($\sigma = 0.5$) and Reference Vector Fields of the ODE and the Exact VdP Trajectory (dashed)

In Figure 7, we show the high-noise NODE, its training data, and the true vector field. Despite the heavily

scattered observable states, the NODE still manages to recover a vector field that aligns well with the true dynamics in the vicinity of the solution trajectory.

Further comparison is given in Figure 8, where the scalar fields visualize the 2-norm error between the neural and reference vector fields. The low-noise NODE ($\sigma = 0.1$) yields way smaller error values along the trajectory, but even the high-noise model produces a fairly accurate vector field in regions close to available training data. As expected, generalization outside this domain remains limited, but within the training region, the results demonstrate strong consistency.

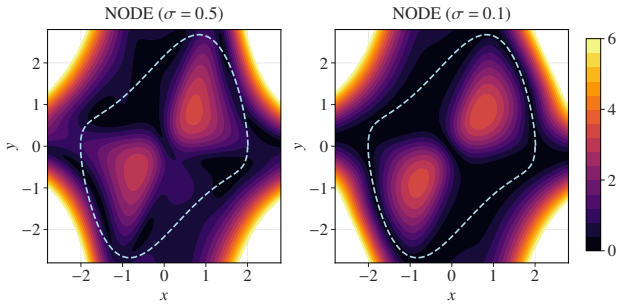


Figure 8. Scalar Fields Representing the 2-Norm Error Between the Neural ($\sigma = 0.5$, $\sigma = 0.1$) and Reference Vector Fields (Values > 6 are white)

In addition to the local collocation approach with 500 intervals, we also reproduce the results in (Shapovalova and Tsay 2025) using a global, *spectral* collocation method. We employ a single interval with 70 fLGR nodes, corresponding to Radau IIA of order 139. This high-order global formulation yields an approximation of equal quality to the figures above, even under high noise ($\sigma = 0.5$). Furthermore, the optimization terminates in outstandingly fast time, i.e. under 1.2 seconds. This demonstrates the remarkable efficiency and accuracy of spectral methods for such smooth problems.

5 Future Work

Although OpenModelica currently includes an optimization runtime implementing direct collocation (Ruge et al. 2014), it remains limited to basic features: it supports only 1- or 3-step Radau IIA collocation, does not allow parameter optimization, lacks analytic Hessians, and does not perform parallel callbacks. To overcome these limitations, work is underway to embed an extended version of *libgdopt* into OpenModelica. This integration combines recent developments from GDOPT with the existing strengths of OpenModelica, particularly its native ability to handle DAEs. The extended framework enables expressive Modelica-based modeling, native support of neural components via the NeuralNetwork Modelica library and incorporates recent advancements in mesh refinement for optimal control problems (Langenkamp 2024).

5.1 NeuralNetwork Modelica Library

The NeuralNetwork library, originally developed in (Codecà and Casella 2006), is an open-source Modelica library[5] modeling ML architectures with pure Modelica. Neural components can be constructed by connecting dense feedforward layers of arbitrary size with layers for PCA, standardizing, or scaling. These blocks contain all equations and parameters as pure Modelica code, which makes seamless integration of neural components into existing Modelica models straightforward. Together with this library, OpenModelica will enable modeling and training of PeN-ODEs within a single development environment.

5.2 Workflow

The workflow presented in Figure 9 illustrates the intended process for native Modelica-based modeling and training of PeN-ODEs. While some components of the workflow are operational, the full integration is still under active development. Users model physical systems and NN components with free parameters directly in Modelica and provide corresponding data. Both the model

[5] https://github.com/AMIT-HSBI/NeuralNetwork

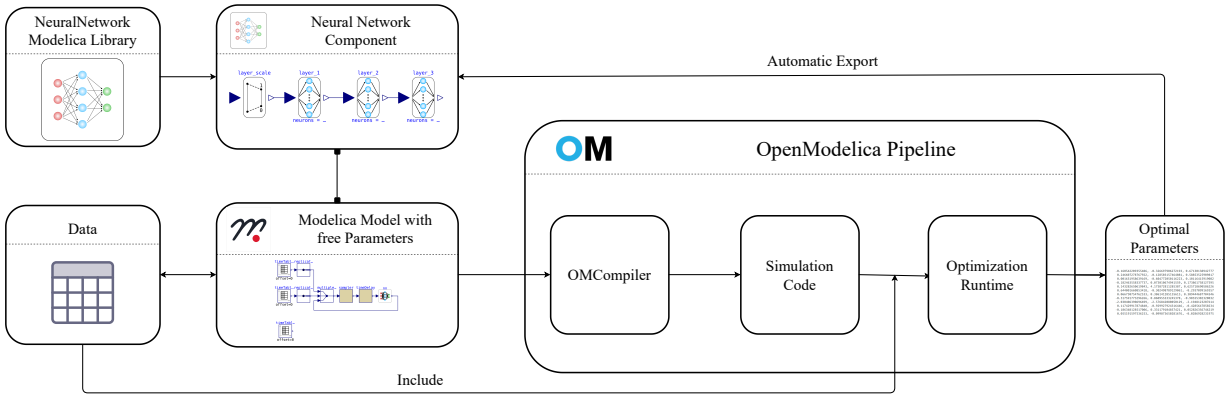


Figure 9. OpenModelica Workflow for PeN-ODE Training (under development)

DOI 10.3384/ecp218445

and data are processed by the standard OpenModelica pipeline, while the OpenModelica Compiler (OMC) generates fast C code. The new backend of OMC introduces efficient array-size-independent symbolic manipulation (Abdelhak, Casella, and Bachmann 2023). Ongoing work further targets resizability of components after compilation (Abdelhak and Bachmann 2025), offering significant benefits for array-based components such as NNs.

Furthermore, current work focuses on leveraging the generated simulation code to enable fast callback functions for a new optimization runtime. After training, the optimal parameters are directly inserted into a NN block, enabling immediate simulations by swapping connectors.

This workflow removes the need for export and import steps for neural components and models, e.g. in the form of a Functional Mock-up Unit (FMU). It also eliminates reliance on external training routines in Julia or Python and avoids external C functions in the model, since the neural components are pure Modelica blocks. This integrated workflow unifies modeling, training, and simulation within a single toolchain, enabling free and accessible optimizations.

5.3 Neural DAEs

One of the main advantages of the integration into OpenModelica is the native ability to handle DAEs. Modelica compilers such as OpenModelica systematically apply index reduction and block-lower triangular (BLT) transformations to restructure DAEs into semi-explicit ODE form with index 1 (Ruge et al. 2014; Åkesson et al. 2012). As the simulation code already resolves algebraic variables during evaluations, no additional handling, e.g. inclusion of algebraic variables in the NLP, is needed on the optimization side. This allows the new training workflow to seamlessly extend from ODEs to DAEs, far surpassing the current range of applications.

6 Conclusion

This paper proposes a formulation of PeN-ODE training as a collocation-based NLP, simultaneously optimizing states and NN parameters. The approach overcomes key limitations of ODE solver-based training in terms of order, stability, accuracy, and allowable step size. The NLP uses high order quadrature for the NN loss, potentially preserving the accuracy of the discretization. We demonstrate that known physical behavior can be trivially enforced.

We provide an open-source parallelized extension to GDOPT and on two example problems demonstrate exemplary accuracy, training times, and generalization with smaller NNs compared to other training techniques, even under significant noise. Furthermore, we show that the approach allows for efficient optimization of other parameter dependent surrogates.

Key limitations and challenges of the proposed method, including grid selection and general initialization strategies to increase stability, as well as training with larger datasets and networks are identified. Addressing and evaluating these issues in future work is essential to support broader applicability. To enable accessible training of Neural DAEs, without relying on external tools, work is underway to implement the method in OpenModelica.

Acknowledgements

This work was conducted as part of the OpenSCALING project (Grant No. 01IS23062E) at the University of Applied Sciences and Arts Bielefeld, in collaboration with Linköping University. The authors would like to express their sincere appreciation to both the OpenSCALING project and the Open Source Modelica Consortium (OSMC) for their support, collaboration, and shared commitment to advancing open-source modeling and simulation technologies.

References

Abdelhak, Karim and Bernhard Bachmann (2025). *Compiler Status and Development of the New Backend*. Presentation at the 17th OpenModelica Annual Workshop - February 3, 2025, accessed on 2025-04-25. URL: https://openmodelica.org/events/openmodelica-workshop/2025.

Abdelhak, Karim, Francesco Casella, and Bernhard Bachmann (2023-12). "Pseudo Array Causalization". In: pp. 177–188. DOI: 10.3384/ecp204177.

Åkesson, Johan et al. (2012). "Generation of Sparse Jacobians for the Function Mock-Up Interface 2.0". In: *Book of abstracts / 9th International Modelica Conferenc*. Vol. 76. Linköping Electronic Conference Proceedings. Linköping University Electronic Press, pp. 185–196. DOI: 10.3384/ecp12076185.

Amestoy, Patrick R. et al. (2001). "A Fully Asynchronous Multifrontal Solver Using Distributed Dynamic Scheduling". In: *SIAM Journal on Matrix Analysis and Applications* 23.1, pp. 15–41.

Andersson, Joel A E et al. (2019). "CasADi – A software framework for nonlinear optimization and optimal control". In: *Mathematical Programming Computation* 11.1, pp. 1–36. DOI: 10.1007/s12532-018-0139-4.

Becerra, V. M. (2010). "Solving complex optimal control problems at no cost with PSOPT". In: *2010 IEEE International Symposium on Computer-Aided Control System Design*, pp. 1391–1396. DOI: 10.1109/CACSD.2010.5612676.

Biegler, Lorenz T. (2010-01). *Nonlinear Programming: Concepts, Algorithms, and Applications to Chemical Processes*. ISBN: 9780898717020. DOI: 10.1137/1.9780898719383.

Boyd, Stephen et al. (2011). "Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers". In: *Foundations and Trends® in Machine Learning* 3.1, pp. 1–122. ISSN: 1935-8237. DOI: 10.1561/2200000016. URL: http://dx.doi.org/10.1561/2200000016.

Bruder, Frederic and Lars Mikelsons (2021-09). "Modia and Julia for Grey Box Modeling". In: pp. 87–95. DOI: 10.3384/ecp2118187.

Byrd, Richard H., Jorge Nocedal, and Richard A. Waltz (2006). "Knitro: An Integrated Package for Nonlinear Optimization". In: *Large-Scale Nonlinear Optimization*. Ed. by G. Di Pillo and M. Roma. Boston, MA: Springer US, pp. 35–59. ISBN: 978-0-387-30065-8. DOI: 10.1007/0-387-30065-1_4. URL: https://doi.org/10.1007/0-387-30065-1_4.

Chandra, Rohit et al. (2001). *Parallel programming in OpenMP*. Morgan Kaufmann.

Chen, Tian Qi et al. (2018). "Neural Ordinary Differential Equations". In: *CoRR* abs/1806.07366. arXiv: 1806.07366. URL: http://arxiv.org/abs/1806.07366.

Codecà, Fabio and Francesco Casella (2006-09). "Neural Network Library in Modelica". In: *Proceedings of the 5th International Modelica Conference*. Vol. 2. Modelica Association. Vienna, Austria, pp. 549–557.

Fritzson, Peter, Adrian Pop, Karim Abdelhak, et al. (2020-10). "The OpenModelica Integrated Environment for Modeling, Simulation, and Model-Based Development". In: *Modeling, Identification and Control: A Norwegian Research Bulletin* 41, pp. 241–295. DOI: 10.4173/mic.2020.4.1.

Gill, P. E. et al. (2007). *SNOPT 7.7 User's Manual*. Tech. rep. CCoM Technical Report 18-1. San Diego, CA: Center for Computational Mathematics, University of California, San Diego.

Gill, Philip E., Walter Murray, and Michael A. Saunders (2005-01). "SNOPT: An SQP Algorithm for Large-Scale Constrained Optimization". In: *SIAM Rev.* 47.1, pp. 99–131. ISSN: 0036-1445. DOI: 10.1137/S0036144504446096. URL: https://doi.org/10.1137/S0036144504446096.

HSL (2013). *HSL: A collection of Fortran codes for large-scale scientific computation*. Available at http://www.hsl.rl.ac.uk. Accessed: 15-04-2025.

Kamp, Tobias, Johannes Ultsch, and Jonathan Brembeck (2023-11). "Closing the Sim-to-Real Gap with Physics-Enhanced Neural ODEs". In: *20th International Conference on Informatics in Control, Automation and Robotics, ICINCO 2023*. Ed. by Guiseppina Gini, Henk Nijmeijer, and Dimitar Filev. Vol. 2. Proceedings of the 20th International Conference on Informatics in Control, Automation and Robotics. SCITEPRESS, pp. 77–84. URL: https://elib.dlr.de/200100/.

Langenkamp, Linus (2024-12). *Adaptively Refined Mesh for Collocation-Based Dynamic Optimization*. Master's thesis. DOI: 10.13140/RG.2.2.18499.72484.

Lehtimäki, Mikko, Lassi Paunonen, and Marja-Leena Linne (2024-01). "Accelerating Neural ODEs Using Model Order Reduction". In: *IEEE Transactions on Neural Networks and Learning Systems* 35.1, pp. 519–531. ISSN: 2162-2388. DOI: 10.1109/tnnls.2022.3175757. URL: http://dx.doi.org/10.1109/TNNLS.2022.3175757.

Liu, Fengjin, William Hager, and Anil Rao (2015-05). "Adaptive Mesh Refinement Method for Optimal Control Using Nonsmoothness Detection and Mesh Size Reduction". In: *Journal of the Franklin Institute* 47. DOI: 10.1016/j.jfranklin.2015.05.028.

Lueg, Laurens R. et al. (2025). *A Simultaneous Approach for Training Neural Differential-Algebraic Systems of Equations*. arXiv: 2504.04665 [cs.LG]. URL: https://arxiv.org/abs/2504.04665.

Magnusson, Fredrik and Johan Åkesson (2015). "Dynamic Optimization in JModelica.org". In: *Processes* 3.2, pp. 471–496. ISSN: 2227-9717. DOI: 10.3390/pr3020471. URL: https://www.mdpi.com/2227-9717/3/2/471.

Misener, Ruth and Lorenz Biegler (2023). "Formulating data-driven surrogate models for process optimization". In: *Computers & Chemical Engineering* 179, p. 108411. ISSN: 0098-1354. DOI: https://doi.org/10.1016/j.compchemeng.2023.108411. URL: https://www.sciencedirect.com/science/article/pii/S0098135423002818.

Múčka, Peter (2018-01). "Simulated Road Profiles According to ISO 8608 in Vibration Analysis". In: *Journal of Testing and Evaluation* 46, p. 20160265. DOI: 10.1520/JTE20160265.

Patterson, Michael A. and Anil V. Rao (2014-10). "GPOPS-II: A MATLAB Software for Solving Multiple-Phase Optimal Control Problems Using hp-Adaptive Gaussian Quadrature Collocation Methods and Sparse Nonlinear Programming". In: *ACM Trans. Math. Softw.* 41.1. ISSN: 0098-3500. DOI: 10.1145/2558904. URL: https://doi.org/10.1145/2558904.

Rackauckas, Chris et al. (2020-01). *Universal Differential Equations for Scientific Machine Learning*. DOI: 10.21203/rs.3.rs-55125/v1.

Ramadhan, Ali et al. (2023). *Capturing missing physics in climate model parameterizations using neural differential equations*. arXiv: 2010.12559 [physics.ao-ph]. URL: https://arxiv.org/abs/2010.12559.

Roesch, Elisabeth, Chris Rackauckas, and Michael Stumpf (2021-07). "Collocation based training of neural ordinary differential equations". In: *Statistical Applications in Genetics and Molecular Biology* 20. DOI: 10.1515/sagmb-2020-0025.

Ruge, Vitalij et al. (2014). "Efficient Implementation of Collocation Methods for Optimization using OpenModelica and ADOL-C". In: *Proceedings of the 10th International Modelica Conference, March 10-12, 2014, Lund, Sweden*. Vol. 96. Linköping Electronic Conference Proceedings. Linköping University Electronic Press, pp. 1017–1025. DOI: 10.3384/ecp140961017.

Schneider, C. and W. Werner (1986). "Some New Aspects of Rational Interpolation". In: *Math. Comp.* 47.175, pp. 285–299. DOI: 10.1090/S0025-5718-1986-0842136-8. URL: https://www.ams.org/journals/mcom/1986-47-175/S0025-5718-1986-0842136-8/.

Shapovalova, Mariia and Calvin Tsay (2025). *Training Neural ODEs Using Fully Discretized Simultaneous Optimization*. arXiv: 2502.15642 [cs.LG]. URL: https://arxiv.org/abs/2502.15642.

Sorourifar, Farshud et al. (2023-09). "Physics-Enhanced Neural Ordinary Differential Equations: Application to Industrial Chemical Reaction Systems". In: *Industrial & Engineering Chemistry Research* 62. DOI: 10.1021/acs.iecr.3c01471.

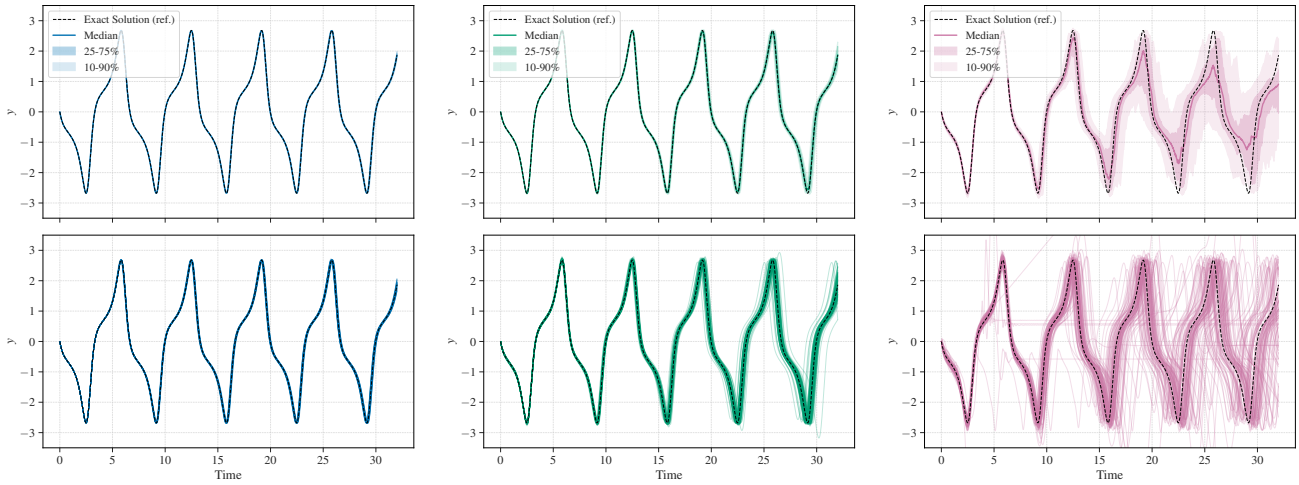
Thebelt, Alexander et al. (2022-02). "Maximizing information from chemical engineering data sets: Applications to machine learning". In: *Chemical Engineering Science* 252, p. 117469. DOI: 10.1016/j.ces.2022.117469.

Thummerer, Tobias, Johannes Stoljar, and Lars Mikelsons (2022). "NeuralFMU: Presenting a Workflow for Integrating Hybrid NeuralODEs into Real-World Applications". In: *Electronics* 11.19. ISSN: 2079-9292. DOI: 10.3390/electronics11193202. URL: https://www.mdpi.com/2079-9292/11/19/3202.

Wächter, Andreas and Lorenz T. Biegler (2006-03). "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming". In: *Mathematical Programming* 106.1, pp. 25–57. ISSN: 1436-4646. DOI: 10.1007/s10107-004-0559-y. URL: https://doi.org/10.1007/s10107-004-0559-y.

Zhao, Jisong and Teng Shang (2018-09). "Dynamic Optimization Using Local Collocation Methods and Improved Multiresolution Technique". In: *Applied Sciences* 8, p. 1680. DOI: 10.3390/app8091680.

Appendix: Sensitivity Analysis and Robustness of VdP Neural ODEs



(a) No noise: $\sigma = 0$ **(b)** Low noise: $\sigma = 0.1$ **(c)** High noise: $\sigma = 0.5$

Figure 10. Sensitivity analysis of the learned Van-der-Pol (VdP) Neural ODEs to different levels of Gaussian noise. For each noise level, 100 training sessions were performed with random initial guesses for the neural network parameters and a random seed for the data perturbation. The plots show the simulation results for the observable state y(t) over an extended time horizon of 32 seconds. Each subplot consists of two panels: The top panel visualizes the reference solution (dashed black line), the median of the 100 learned solutions, and the 25-75% and 10-90% percentile bands. The bottom panel shows all 100 individual learned trajectories (faint lines). **(a) No noise** ($\sigma = 0$)**:** The method demonstrates full robustness and perfect convergence, with all 100 solutions converging to the exact reference solution. **(b) Low noise** ($\sigma = 0.1$)**:** The method remains highly robust. A vast majority of the solutions (approximately 95%) form a tight band around the reference, showing excellent agreement. **(c) High noise** ($\sigma = 0.5$)**:** As expected under severe noise, the robustness is reduced. While many solutions remain relatively close to the reference, some diverge significantly, indicating a failure to find a good local optimum during training. The solutions that do converge often show a period mismatch or diverge after a few periods, but still capture the general oscillatory behavior. The combined effects of reduced solution quality and an inaccurate period are reflected in the median and percentile bands, which exhibit a noticeable deviation in amplitude and phase after three periods compared to the true trajectory.