Multibody System Dynamics
https://doi.org/10.1007/s11044-024-10012-6

METHODOLOGY

MBD-NODE: physics-informed data-driven modeling and simulation of constrained multibody systems

Jingquan Wang[1] · Shu Wang[1] · Huzaifa Mustafa Unjhawala[1] · Jinlong Wu[1] · Dan Negrut[1]

Received: 30 December 2023 / Accepted: 28 June 2024

Abstract
We describe a framework that can integrate prior physical information, e.g., the presence of kinematic constraints, to support data-driven simulation in multibody dynamics. Unlike other approaches, e.g., Fully Connected Neural Network (FCNN) or Recurrent Neural Network (RNN)-based methods, which are used to model the system states directly, the proposed approach embraces a Neural Ordinary Differential Equation (NODE) paradigm, which models the derivatives of the system states. A central part of the proposed methodology is its capacity to learn the multibody system dynamics from prior physical knowledge and constraints combined with data inputs. This learning process is facilitated by a constrained optimization approach, which ensures that physical laws and system constraints are accounted for in the simulation process. The models, data, and code for this work are publicly available as open source at https://github.com/uwsbel/sbel-reproducibility/tree/master/2024/MNODE-code.

Keywords Multibody dynamics · Neural ODE · Constrained dynamics · Scientific machine learning

1 Introduction

This contribution concerns the use of a data-driven approach to characterize the dynamics of multibody systems. Recently, data-driven modeling methods have been developed to

✉ D. Negrut
negrut@wisc.edu

J. Wang
jwang2373@wisc.edu

S. Wang
swang579@wisc.edu

H.M. Unjhawala
unjhawala@wisc.edu

J. Wu
jinlong.wu@wisc.edu

[1] Department of Mechanical Engineering, University of Wisconsin-Madison, 1513 University Avenue, 53706, Madison, USA

Published online: 23 July 2024

characterize multibody dynamics systems based on neural networks. For instance, fully connected neural networks (FCNNs) offer a straightforward way to model multibody dynamics, mapping input parameters (potentially including time for nonautonomous systems) to state values [11, 43]. FCNNs act as regressors or interpolators, predicting system states for any given time and input parameter. The simplicity of the method allows us to quickly estimate the state within the range of the training set. However, this approach requires expanding the input dimension to accommodate, for instance, changes in initial conditions, leading to an exponential increase in training data. An alternative method combines fixed-time increment techniques with principal component analysis (PCA) [20] to reduce training costs and data requirements. This method outputs time instances at fixed steps and employs PCA for dimensionality reduction. However, this results in discontinuous dynamics, limiting output to discrete time steps. Another approach uses dual FCNNs, one modeling dynamics and the other estimating errors, to enhance accuracy while reducing computational costs [25]. A limitation of FCNN is the strong assumption of independent and identically distributed (i.i.d.) data [6] required by the approach, which limits its ability to generalize, particularly in terms of extrapolation across time and phase space.

Time series-based approaches (e.g., LSTM [28]) have also been investigated for the data-driven modeling of multibody dynamics systems. These methods learn to map historical state values to future states, but they are inherently discrete and tied to specific time steps. For example, RNNs have been directly employed for predicting subsequent states for a drivetrain system [29] and a railway system [31]. The time-series approach works well for systems with strong periodic patterns, a scenario in which the approach yields high accuracy. RNNs turned out to be challenged by systems with more parameters than just time (e.g., accounting of different initial conditions). To mitigate its shortcomings, a combination of 3D Convolutional Neural Network(3DCNN), FCNN, and RNN has been used for tracked-vehicle system behavior prediction [54]. More examples using FCNNs and RNNs can be found in the review paper [26].

Neural ordinary differential equations (NODEs), recently introduced in [7], provide a framework that offers a more flexible data-driven approach to model continuous-time dynamics. Unlike traditional regression-based models, which assume a discretized time step (which is often fixed a priori), NODE employs a continuous way of modeling an ordinary differential equation (ODE) and allows flexible discretization in numerical simulations. This makes NODE capable of dealing with data in various temporal resolutions and emulating physical systems in continuous time. In terms of applications, NODE has seen good success in engineering applications, such as chemical reactions [42, 48], turbulence modeling [44], spintronic dynamics, [10], and vehicle dynamics problems [45]. Recent work has significantly advanced our understanding of NODE-based approaches, providing analysis of convergence [16], robustness [22], and generalization ability [5, 23, 55].

Conservation laws are important to be obeyed by the model used to characterize the response of the system. Hamiltonian Neural Networks (HNNs) [21], inspired by Hamiltonian mechanics, can factor in exact conservation laws by taking generalized positions and momenta as inputs to model the Hamiltonian function of a system. However, an HNN requires data in generalized coordinates with momentum, posing practical challenges, especially in multibody dynamics (MBD) systems as it necessitates the transformation of complex, often high-dimensional system dynamics into a reduced set of generalized coordinates. This transformation can be both computationally intensive and prone to inaccuracies, especially when dealing with intricate mechanical systems involving multiple interacting components whose dynamics is constrained through mechanical joints. In addition, even if an MBD system is nondissipative, it can still represent a nonseparable Hamiltonian system, for which

the integration process is more complicated, often necessitating implicit methods. Various follow-up works have expanded upon HNNs, addressing systems with dissipation [49, 56], generative networks [51], graph neural networks [46], symplectic integration [8, 37, 40], nonseparable Hamiltonian system [14], and the combination with probabilistic models [1].

To address the limitations of HNN, Lagrangian Neural Networks (LNNs) [12, 36] have been proposed to leverage Lagrangian mechanics. LNN models the system Lagrangian, with second-order state derivatives derived from the Euler–Lagrange equation. This approach also conserves the total energy and applies to a broader range of problems. However, it is computationally intensive and sometimes ill-posed due to its reliance on the inverse Hessian. Subsequent works on LNNs have explored various aspects, such as including constraints [17], extended use with graph neural networks [4], and model-based learning [24, 57].

In this study, our primary objective is to learn the dynamics of multibody systems from system states data using a NODE-based approach. We also explore the process of incorporating prior physical information, such as kinematic constraints, into the numerical solution through the use of a constrained optimization method in conjunction with standard NODEs. We compare the performance of the proposed approach with existing methodologies on various examples. Our contributions are as follows:

- We propose a method called Multibody NODE (MBD-NODE) by applying NODE to the data-driven modeling of general MBD problems and establishing a methodology to incorporate known physics and constraints in the model.
- We provide a comprehensive comparison of the performance of MBD-NODE with several other methods that have been applied to MBD problems.
- We build a series of MBD test problems, providing an open-source code base consisting of several data-driven modeling methods (FCNN, LSTM, HNN, LNN, MBD-NODE) and curating a well-documented summary of their performances.

2 Methodology

2.1 Multibody system dynamics

MBD is used in many mechanical engineering applications to analyze systems composed of interconnected bodies. Here we rely on the general form of the MBD problem [47], which accounts for the presence of constraint equations using Lagrange multipliers in the equations of motion:

$$\begin{bmatrix} \mathbf{M} & \mathbf{\Phi}_{\mathbf{q}}^{T} \\ \mathbf{\Phi}_{q} & 0 \end{bmatrix} \begin{bmatrix} \ddot{\mathbf{q}} \\ \lambda \end{bmatrix} = \begin{bmatrix} \mathbf{F_e} \\ \gamma_{\mathbf{c}} \end{bmatrix}, \tag{1}$$

where $\mathbf{M}$ represents the mass matrix, $\mathbf{\Phi_q}$ is the constraint Jacobian matrix, $\mathbf{q}$ denotes the vector of system states (generalized coordinates), $\ddot{\mathbf{q}}$ denotes the acceleration vector of the system, λ represents the Lagrangian multipliers, $\mathbf{F_e}$ is the combined vector of generalized external forces and quadratic velocity terms, and $\gamma_{\mathbf{c}}$ is the right-hand side of the kinematic constraint equations at the acceleration level. In practice, the set of differential-algebraic equations (DAE) in Eq. (1) can be numerically solved by several methods; see, for instance, [2].

2.2 Neural ordinary differential equations for multibody system dynamics

2.2.1 Neural ordinary differential equation (NODE)

NODE represents a class of deep learning models that train neural networks to approximate unknown vector fields in ordinary differential equations (ODEs) to characterize the continuous-time evolution of system states. Given a hidden state $\mathbf{z}(t), \mathbf{z} \in \mathbb{R}^{n_z}$, at time t, the NODE is defined by the equation

$$\frac{d\mathbf{z}(t)}{dt} = f(\mathbf{z}(t), t;\ \boldsymbol{\Theta}), \tag{2}$$

where $f : \mathbb{R}^{n_z} \times \mathbb{R}^+ \to \mathbb{R}^{n_z}$ corresponds to a neural network parameterized by $\boldsymbol{\Theta}$. For an arbitrary time $t > 0$, the state $\mathbf{z}(t)$ can be obtained by solving an initial value problem (IVP) through the forward integration:

$$\mathbf{z}(t) = \mathbf{z}(0) + \int_0^t f(\mathbf{z}(\tau), \tau;\ \boldsymbol{\Theta}) d\tau = \Phi(\mathbf{z}(0), f, t), \tag{3}$$

where Φ denotes an ODE solver.

NODE [7, 58] provides an efficient approach of calibrating the unknown parameters $\boldsymbol{\Theta}$ based on some observation data $\mathbf{z}(t_i)$, $i = 1, 2, \ldots, n$. Note that the time steps t_i do not have to be equidistant and thus we have flexibility in choosing numerical integrators for the forward integration in Eq. (3).

2.2.2 Extensions of neural ordinary differential equation

In the modeling of dynamical systems, it is quite common for equations to include parameters that significantly influence the system behavior, such as the Reynolds number in the Navier–Stokes equations or design parameters in MBD, e.g., lengths, masses, material properties. Enhancing NODEs to accommodate such variations would enable the simultaneous learning of a wide range of dynamics. A practical way to achieve this augmentation is to incorporate these parameters directly into the neural network inputs, known as PNODE, which is suggested in [32]:

$$\frac{d\mathbf{z}(t)}{dt} = f(\mathbf{z}(t), t, \boldsymbol{\mu};\ \boldsymbol{\Theta}), \tag{4}$$

where $\boldsymbol{\mu} = (\mu_1, \mu_2, \ldots, \mu_{n_{\boldsymbol{\mu}}})^T \in \mathbb{R}^{n_{\boldsymbol{\mu}}}$ is the parameter vector that can help better characterize the system, and $f : \mathbb{R}^{n_z} \times \mathbb{R}^+ \times \mathbb{R}^{n_{\boldsymbol{\mu}}} \to \mathbb{R}^{n_z}$ is the neural network.

For the systems whose governing equations are second-order, we can use the second-order neural ordinary differential equation (SONODE) [34, 39] to model them. Given an augmented state $\mathbf{Z}(t) = (\mathbf{z}(t), \dot{\mathbf{z}}(t))^T$, the SONODE is defined as

$$\frac{d\mathbf{Z}(t)}{dt} = f(\mathbf{Z}(t), t;\ \boldsymbol{\Theta}), \tag{5}$$

where $f : \mathbb{R}^{2n_z} \times \mathbb{R}^+ \to \mathbb{R}^{2n_z}$ is a neural network parameterized by $\boldsymbol{\Theta}$.

2.2.3 Multibody dynamics NODE (MBD-NODE)

Based on the above PNODE and SONODE, we extend the approach to make the NODE work with external inputs like external generalized forces, thus better fitting the MBD framework. Given the set of generalized coordinates $\mathbf{Z}(t,\boldsymbol{\mu}) = (\mathbf{z}^T(t,\boldsymbol{\mu}), \dot{\mathbf{z}}^T(t,\boldsymbol{\mu}))^T \in \mathbb{R}^{2n_z}$, the MBD-NODE is defined as

$$\frac{d\mathbf{Z}(t,\boldsymbol{\mu})}{dt} = f(\mathbf{Z}(t,\boldsymbol{\mu}), \mathbf{u}(t), t, \boldsymbol{\mu};\ \boldsymbol{\Theta}), \tag{6}$$

where

$$\mathbf{Z}(0,\boldsymbol{\mu}) = (\mathbf{z}^T(0,\boldsymbol{\mu}), \dot{\mathbf{z}}^T(0,\boldsymbol{\mu}))^T \tag{7}$$

are the initial values for the MBD,

$$\mathbf{z}(t,\boldsymbol{\mu}) = (z^1(t,\boldsymbol{\mu}), \ldots, z^{n_z}(t,\boldsymbol{\mu}))^T \in \mathbb{R}^{n_z} \tag{8}$$

are the generalized positions,

$$\dot{\mathbf{z}}(t,\boldsymbol{\mu}) = (\dot{z}^1(t,\boldsymbol{\mu}), \ldots, \dot{z}^{n_z}(t,\boldsymbol{\mu}))^T \in \mathbb{R}^{n_z} \tag{9}$$

are the generalized velocities;

$$\mathbf{u}(t) = (u^1(t), \ldots, u^{n_u}(t))^T \in \mathbb{R}^{n_u} \tag{10}$$

are n_u external loads like force/torque applied to the MBD at time t (note that time t can be included in the input $\mathbf{u}(t)$),

$$\boldsymbol{\mu} = (\mu_1, \mu_2, \ldots, \mu_{n_{\boldsymbol{\mu}}})^T \in \mathbb{R}^{n_{\boldsymbol{\mu}}} \tag{11}$$

are problem-specific parameters, and

$$f : \mathbb{R}^{2n_z} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_{\boldsymbol{\mu}}} \to \mathbb{R}^{2n_z} \tag{12}$$

is the neural network parameterized by $\boldsymbol{\Theta}$ with $(2n_z + n_u + n_\mu)$-dimensional input. For the forward pass to solve the initial value problem for $\mathbf{Z}(t,\boldsymbol{\mu})$, we can still use the integrator Φ:

$$\mathbf{Z}(t,\boldsymbol{\mu}) = \mathbf{Z}(0,\boldsymbol{\mu}) + \int_0^t f(\mathbf{Z}(\tau,\mu), \mathbf{u}(\tau), t, \boldsymbol{\mu};\ \boldsymbol{\Theta}) d\tau = \Phi(\mathbf{Z}(0,\boldsymbol{\mu}),\ f, \mathbf{u}, t). \tag{13}$$

For the backpass of the MBD-NODE, we can use the backpropagation or the adjoint method to design the corresponding adjoint state based on the property of second-order ODEs [34, 39]. We finally choose to use backpropagation, a step analyzed in detail in Sect. 2.2.3, which touches on the construction of loss function and optimization.

Figures 1 and 2 show the discretized version of the forward pass for the MBD with and without constraints. The constraint-related formulations are discussed in Sect. 2.2.3. Within the MBD-NODE framework, the initial state of the system is processed using an ODE solver evolving over a time span under the guidance of a neural network parameters. This neural network is trained to determine the optimal parameters that best describe the system dynamics. This continuous approach, in contrast to discrete-time models, often results in enhanced flexibility, efficiency, and good generalization accuracy. Based on the notation used for the definition of MBD-NODE in Eq. (6), we employ the corresponding three-layer neural network architecture in Table 1; the activation function can be Tanh and ReLU [41], and the initialization strategy used is that of Xavier [19] and Kaiming [27].

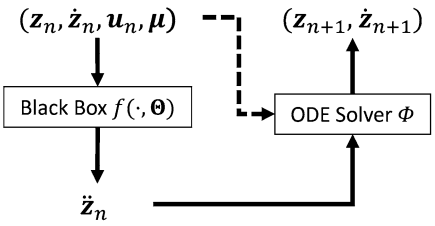


Fig. 1 The discretized forward pass for MBD-NODE for general MBD

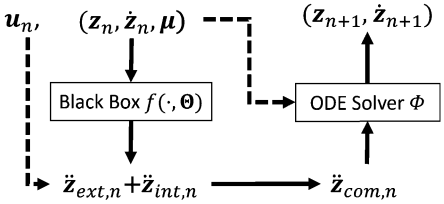


Fig. 2 The discretized forward pass for MBD-NODE for general MBD without hard constraints, which means that the external force/torque can be directly added to the acceleration from the NODE that models the internal acceleration without additional input channel for external force. Here $\ddot{\mathbf{z}}_{ext,n}$ is the acceleration caused by the external input, and $\ddot{\mathbf{z}}_{int,n}$ is the internal acceleration predicted by MBD-NODE

Table 1 MBD-NODE Architecture

Layer	Number of neurons	Activation function	Initialization
Input Layer	$2n_z + n_{\boldsymbol{\mu}} + n_u$	[Tanh,ReLU]	[Xavier, Kaiming]
Hidden Layer 1	d_{width}	[Tanh,ReLU]	[Xavier, Kaiming]
Hidden Layer 2	d_{width}	[Tanh,ReLU]	[Xavier, Kaiming]
Output Layer	$2n_z$	-	[Xavier, Kaiming]

2.2.4 Loss function and optimization without constraints

First, we discuss the loss function for the MBD without constraints. Without loss of generality, we assume there are no additional parameters $\boldsymbol{\mu}$ for notation simplicity. For a given initial state $\mathbf{Z}_0 = (\mathbf{z}_0, \dot{\mathbf{z}}_0)$, assume that the next system state $\mathbf{Z}_1 = (\mathbf{z}_1, \dot{\mathbf{z}}_1)$ is obtained with the integrator Φ used over a time interval Δt, which can be one or several numerical integration time steps. The loss function used for the MBD-NODE describes the mean square error (MSE) between the ground truth state and the predicted state:

$$\mathrm{L}(\boldsymbol{\Theta}) = \|\Phi(\mathbf{Z}_0, f, \Delta t) - \mathbf{Z}_1\|_2^2 = \|\hat{\mathbf{Z}}_1 - \mathbf{Z}_1\|_2^2 = \|(\hat{\mathbf{z}}_1, \hat{\dot{\mathbf{z}}}_1) - (\mathbf{z}_1, \dot{\mathbf{z}}_1)\|_2^2 \,, \tag{14}$$

where $\hat{\mathbf{Z}}_1 = (\hat{\mathbf{z}}_1^T, \hat{\dot{\mathbf{z}}}_1^T)^T$ is the predicted state by integration with the derivatives from MBD-NODE.

For a trajectory of states $\mathbf{z}_0, \mathbf{z}_1, \ldots, \mathbf{z}_T$, the common way [7] is to treat the first state as an initial condition and all other states as targets, so the loss function can be defined as

$$\mathrm{L}(\boldsymbol{\Theta}) = \sum_{i=0}^{T-1} \|\Phi(\mathbf{Z}_0, f, \Delta t_i) - \mathbf{Z}_{i+1}\|_2^2 = \sum_{i=0}^{T-1} \|\hat{\mathbf{Z}}_{i+1} - \mathbf{Z}_{i+1}\|_2^2. \tag{15}$$

The training phase is refining the neural network parameters, ensuring that the predicted states mirror the true future states, which yields the optimization problem

$$\boldsymbol{\Theta}^* = \underset{\boldsymbol{\Theta}}{\operatorname{argmin}} \quad \mathrm{L}(\boldsymbol{\Theta}). \tag{16}$$

Similarly to most deep learning models, the parameter optimization of MBD-NODE can be conducted by backpropagation via stochastic gradient descent (SGD). The key for NODE-based frameworks is that the objective is to fit the entire trajectory, which necessitates the storage of intermediate gradients through the integration of the whole trajectory by backpropagation. This process needs a memory cost of $O(NCL)$, where N represents the number of time steps of the trajectory, C is the number of neural network calls per integration step, and L is the number of layers in the NODE. To solve this, adjoint methods [7] and their adaptive enhancements [58] were implemented in the NODE-based model, achieving gradient approximation with only $O(L)$ memory costs. Further, specialized adjoint methods have been proposed for symplectic integrators [38] and SONODE [39], each tailored for specific applications.

In practice, optimizing parameters to fit lengthy trajectories from highly nonlinear dynamics did not work well for our problems. To address this, we partition the long trajectory, consisting of n states, into $[n/w]+1$ shorter subtrajectories of length w. The training process is then moved to these subtrajectories. Although this strategy makes the optimization easier, it may slightly impair the neural network capacity for long-term prediction. In practice, we set w to be 1 for our numerical test, and we did not find the obvious loss of capacity for long-term prediction. In this case, the loss function will be the sum of the loss of each subtrajectory:

$$\mathrm{L}(\boldsymbol{\Theta}) = \sum_{i=0}^{T-1} \|\Phi(\mathbf{Z}_i, f, \Delta t_i) - \mathbf{Z}_{i+1}\|_2^2 = \sum_{i=0}^{T-1} \|\hat{\mathbf{Z}}_{i+1} - \mathbf{Z}_{i+1}\|_2^2. \tag{17}$$

The constant C, the number of neural network calls per integration step, depends on the integrator used. For the Runge–Kutta 4th-order method, $C = 4$, because we need to evaluate the acceleration at intermediate states during one step of integration, whereas $C = 1$ for the forward Euler method. Also, for the implicit solvers, C is the same as their explicit version, because in the training stage, we already have the next state.

Given these considerations, the memory cost for optimization via backpropagation remains within acceptable limits. For the system subject to constraints (discussed in Sect. 2.2.3), the adjoint method may not align with the used method. Upon reviewing recent literature, we found no instances of the adjoint methods being applied to constrained problems. Based on these, we finally choose backpropagation to optimize the neural network. The main process for training the MBD-NODE without constraints is summarized in the Algorithm 1 of Appendix A.

2.2.5 Loss function and optimization with constraints

For MBD problems, accounting for constraints in the evolution of a system is imperative. These constraints capture not only physical design attributes (e.g., a spherical joint requires two points to coincide), but also factor in conservation laws, e.g., energy, numerical Hamiltonian. Accounting for these constraints is important in MBD. However, integrating constraints within deep neural network models is still an open problem, and further research and exploration are necessary.

From a high vantage point, constraints fall into one of two categories. Holonomic constraints depend solely on the coordinates without involving the time derivatives of the latter and can be represented as $\phi(\mathbf{z}, t) = 0$. Nonholonomic constraints, involving the time derivatives of the coordinates and which cannot be time-integrated into a holonomic constraint, are denoted as $\phi(\mathbf{z}, \dot{\mathbf{z}}, t) = 0$.

Additionally, constraints can be categorized based on their temporal dependency. Scleronomic constraints, or geometric constraints, do not explicitly depend on time and are expressed as $\phi(\mathbf{z}) = 0$. In contrast, rheonomic constraints, which depend on time, can also be framed in the form $\phi(\mathbf{z}, t) = 0$.

In summary, using the same notation as in Sect. 2.2.3, the MBD constraints can be expressed in a generalized form $\phi(\mathbf{z}, \dot{\mathbf{z}}, \mathbf{u}, \boldsymbol{\mu}) = 0$, and the optimization problem solved can be posed as

$$\min_{\boldsymbol{\Theta}} \quad \mathrm{L}(\boldsymbol{\Theta}) \tag{18}$$

$$\text{s.t. } \phi_i(\mathbf{z}, \dot{\mathbf{z}}, \mathbf{u}, \boldsymbol{\mu}) = 0, \forall(\mathbf{z}, \dot{\mathbf{z}}, \mathbf{u}, \boldsymbol{\mu}) \in \mathbb{R}^{2n_z} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_\mu} \cap \Omega, i = 1, .., n_c, \tag{19}$$

where Ω is the area from the prior physical knowledge that the MBD should have constraints.

There are two common ways to handle hard constraints. One is to relax this hard constrained problem to a soft constraint problem by adding the constraints to the loss function as a penalty term [15, 18, 33, 35]. The loss function then becomes

$$\mathrm{J}(\boldsymbol{\Theta}) = \mathrm{L}(\boldsymbol{\Theta}) + \sum_i g_i(\phi_i(\mathbf{z}, \dot{\mathbf{z}}, \mathbf{u}, \boldsymbol{\mu})) \,, \tag{20}$$

where g_i represents the function for the ith constraint, typically comprising a quadratic and a linear term, as is common in the well-known augmented Lagrangian method [15, 18, 35]. The primary advantage of this approach is its ease of implementation, requiring only the addition of constraints as a regularization term. However, there are several drawbacks to it: the optimization process may not always converge, and the use of regularization can often diminish accuracy. Most critically, the constraints are applied exclusively within the phase space of the training set, rendering them ineffective in domains beyond this phase space.

The alternative is to enforce the constraints in both the training and inference stages without adding a constraint loss term. Based on a coordinates partition technique [53], we denote the minimal (or independent) coordinates as $\mathbf{Z}^M$ and the dependent coordinates as $\mathbf{Z}^D$. Then the dependent coordinates can be obtained from the independent coordinates and the prior knowledge of constraints:

$$\mathbf{Z}^D = \phi^{-1}(\mathbf{Z}^M, \boldsymbol{\mu}), \tag{21}$$

where ϕ^{-1} is defined as the inverse function that maps the value of minimal coordinates to the dependent coordinates and typically does not have a closed form, yet it can be evaluated given $\mathbf{Z}^M$. If the MBD system has n_z generalized coordinates and n_c position constraints ($n_z - n_c =$ DOF), then we build the MBD-NODE only with the minimal coordinates $\mathbf{Z}^M$. Depending on whether we have the ground truth data of the dependent coordinates $\mathbf{Z}^D$, the training stage can be divided into two cases:

(1) If we have the complete information of the dependent coordinates $\mathbf{Z}^D$, then we can first input the minimal state $\mathbf{Z}_n^M$ to get the acceleration for integration for getting the minimal state at the next time step $\hat{\mathbf{Z}}_{n+1}^M$. Then we can solve the dependent state $\hat{\mathbf{Z}}_{n+1}^D$ by solving the

Table 2 Summary of comparison of the methods. The compared methods are MBD-NODE, HNN, LNN, LSTM, and FCNN

	MBD-NODE	HNN	LNN	LSTM	FCNN
Works on energy-conserving system	✓	✓	✓		
Works on general coordinates	✓		✓	✓	✓
Works on dissipative systems	✓			✓	✓
Works with constraints	✓				
No need for second-order derivatives	✓		✓		
Scalability for long time simulation	✓	✓	✓		
Learn continuous dynamics	✓	✓	✓		✓

constraint equation $\hat{\mathbf{Z}}^D_{n+1} = \phi^{-1}(\hat{\mathbf{Z}}^M_{n+1}, \mathbf{u}, \boldsymbol{\mu})$. We could use the minimal coordinates $\hat{\mathbf{Z}}^M_{n+1}$ and dependent state $\hat{\mathbf{Z}}^D_{n+1}$ to get the full combined states $\tilde{\mathbf{Z}}_{n+1} = (\hat{\mathbf{Z}}^M_{n+1}, \hat{\mathbf{Z}}^D_{n+1})^T \in \mathbb{R}^{2n_z}$. By the difference between the combined predicted state $\tilde{\mathbf{Z}}_{n+1}$ and the ground truth state $\mathbf{Z}_{n+1}$ we can optimize the MBD-NODE. A similar concept has been explored in [3, 13] for enforcing hard constraints within data-driven models. Beucler et al. [3] have approached this by designing a constraint layer, whereas Daems et al. [13] encoded holonomic constraints directly into the Euler–Lagrange equations. Our method can address more general nonholonomic constraints. Given the initial state $\mathbf{Z}_0$ and the ground truth state $\mathbf{Z}_1$, the corresponding loss function for one data pair could be written as

$$\begin{aligned} \mathrm{L}(\boldsymbol{\Theta}) &= \|(\Phi(\mathbf{Z}^M_0, f, \Delta t), \phi^{-1}(\Phi(\mathbf{Z}^M_0, f, \Delta t)))^T - \mathbf{Z}_1\|_2^2 \\ &= \|(\hat{\mathbf{Z}}^M_1, \hat{\mathbf{Z}}^D_1)^T - \mathbf{Z}_1\|_2^2 = \|\tilde{\mathbf{Z}}_1 - \mathbf{Z}_1\|_2^2. \end{aligned} \tag{22}$$

(2) If we have access to only the minimal state information ($\mathbf{Z}^M_n$) – for instance, if we prefer not to expend effort in collecting data on dependent coordinates due to potential costs – then we can construct and train the MBD-NODE using solely the minimal coordinates ($\mathbf{Z}^M$). During the inference, we could use MBD-NODE to predict minimal states and then solve all the states. In this case, the loss function could be written as

$$\mathrm{L}(\boldsymbol{\Theta}) = \|\Phi(\mathbf{Z}^M_0, f, \Delta t) - \mathbf{Z}^M_1\|_2^2 = \|\hat{\mathbf{Z}}^M_1 - \mathbf{Z}^M_1\|_2^2. \tag{23}$$

By solving the constraint equation in both the training (with dependent coordinates data) and inference stage the hard constraints are satisfied in both phases. The algorithm for constraints equation-based optimization is summarized in Algorithm 2 (which utilizes dependent coordinates data) and Algorithm 3 (which uses only minimal coordinates data), both found in Appendix A.

2.2.6 Baseline models

Table 2 summarizes the models used in the numerical tests discussed in this paper along with some of their salient attributes. Code for all of these methods is provided with this contribution.

Table 3 Summary of the numerical examples and modeling methods

Test case	Model A	Model B	Model C
Single Mass-Spring	MBD-NODE	HNN	LNN
Single Mass-Spring-Damper	MBD-NODE	LSTM	FCNN
Triple Mass-Spring-Damper	MBD-NODE	LSTM	FCNN
Single Pendulum	MBD-NODE	LSTM	FCNN
Double Pendulum	MBD-NODE	LSTM	FCNN
Cart-pole	MBD-NODE	LSTM	FCNN
Slider Crank	MBD-NODE	-	-

Table 4 Summary of the numerical errors for different models in various test cases. The detailed information about the models is included in Table 3

Test case	Error		
	Model A	Model B	Model C
Single Mass-Spring	1.3e-6	1.9e-2	9.1e-6
Single Mass-Spring-Damper	8.6e-4	1.8e-2	9.9e-2
Triple Mass-Spring-Damper	8.2e-3	1.8e-1	4.2e-2
Single Pendulum	2.0e-3	3.4e-3	8.0e-1
Double Pendulum	2.0e-1	6.4e-1	2.2e0
Cart-pole	6.0e-5	3.2e-4	4.7e-2
Slider Crank	3.2e-2	-	-

3 Numerical experiments

We study the performance of the methods in Table 2 with seven numerical examples, reflecting method attributes such as energy conservation, energy dissipation, multiscale dynamics, generalization to different parameters and external force, model-based control, chaotic dynamics, and constraint enforcement. One or more of these attributes often comes into play in engineering applications that rely on MBD simulation. We use these numerical examples to compare the performance of the proposed MBD-NODE methodology with state-of-the-art data-driven modeling methods. The numerical examples and modeling methods are summarized in Table 3. The model performance is evaluated via the MSE ϵ defined as

$$\epsilon = \frac{1}{N}\sum_{i=1}^{N}\left(\|\mathbf{Z}_i - \hat{\mathbf{Z}}_i\|^2\right) = \frac{1}{N}\sum_{i=1}^{N}\left(\|\mathbf{z}_i - \hat{\mathbf{z}}_i\|^2 + \|\dot{\mathbf{z}}_i - \hat{\dot{\mathbf{z}}}_i\|^2\right), \tag{24}$$

where i indicates the index of a test sample, $\mathbf{z}_i$ and $\dot{\mathbf{z}}_i$ are the ground truth of the coordinate and its time derivative, and $\hat{\mathbf{z}}_i$ and $\hat{\dot{\mathbf{z}}}_i$ denote the predicted results by a trained model. Here $\|\cdot\|$ corresponds to the standard vector 2-norm.

In Table 4, we summarize the MSE error made by each method on the test data of all the numerical examples. Sections 3–3.4 present more detail about the setup of each test case and the performance of our method in comparison to the others. The training costs for the MBD-NODE, HNN, LNN, LSTM, and FCNN models with different integrators used for each test case are recorded in Appendix A. Python code is provided publicly for all models and all test cases for unfetter used and reproducibility studies [52].

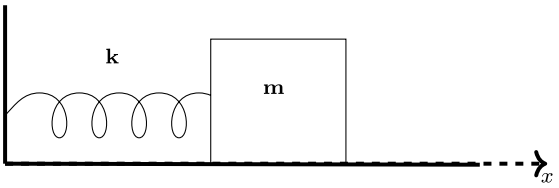


Fig. 3 Single mass-spring system; k and m denote the spring constant and the mass of the object, respectively. Only the motion along the x-direction is considered

3.1 Single mass-spring system

This system is relevant as it does not model viscous damping and serves as a numerical example to evaluate the predictive attribute of the trained models on an energy-conserving system [9, 21, 40]. Figure 3 illustrates the setup of the single mass-spring system. The equation of motion is formulated as

$$\frac{d^2x}{dt^2} = -\frac{k}{m}x, \tag{25}$$

where x represents the displacement of the mass from its equilibrium position, k, the spring constant, is set to 50 N/m, and m, the mass of the object, is set to 10 kg. The system Hamiltonian, which describes its total energy, is

$$T(p) = \frac{p^2}{2m}, \tag{26}$$

$$V(q) = 1/2kq^2, \tag{27}$$

$$H(p,q) = T(p) + V(q), \tag{28}$$

where q is the generalized position, p is the generalized momentum, which, in this context, is $m \times \dot{q}$, with $\dot{q}$ being the generalized velocity, and T and V represent the kinetic and potential energies.

We choose a time step of 0.01 s in both training and testing for the single mass-spring system. The training data consists of a trajectory analytically solved over 300 time steps with initial conditions $x_0 = 1$ m and $v_0 = 0$ m/s.

For this system, which possesses a separable Hamiltonian as shown in Eq. (28), the MBD-NODE model employed the leapfrog method as the symplectic integrator of choice. We also show the performance of MBD-NODE when used with the more common RK4 integrator. We also benchmark against the HNN and LNN methods (see Table 3 for a summary). The Hamiltonian-based methods used data in generalized coordinates, whereas the others (including a numerical method) were tested using Cartesian coordinates. We also provide a baseline test by numerically solving the system of ODEs in Eq. (25) with the RK4 integrator. The specific configurations of each model, including the choice of coordinate systems and integrators, are detailed in Table 5. Additionally, the hyperparameters used for the neural network-based tests are summarized in Table 6. These settings and tests were designed to evaluate the efficiency and accuracy of different modeling approaches and integrators in predicting and understanding the dynamics of the single mass-spring system.

Figure 4 shows the dynamic response in terms of position x and velocity v for the test data, and the MSE of each method is shown in Table 5. More specifically, Figs. 4 (a)–(d) demonstrate the performance of the MBD-NODE model with the RK4 integrator and the results obtained from a purely numerical solution using the RK4 method. The ground truth

Table 5 Numerical tests with corresponding MSE for the single-mass spring system

Model	Coordinate system	Integrator	MSE
MBD-NODE	Generalized	Leapfrog	1.3e-6
HNN	Generalized	RK4	2.0e-3
LNN	Cartesian	RK4	9.1e-6
Numerical	Cartesian	RK4	2.0e-3
MBD-NODE	Cartesian	RK4	9.2e-1

Table 6 Hyperparameters for the single mass-spring system

Hyperparameters	Model			
	MBD-NODE$_{\text{LF}}$	MBD-NODE$_{\text{RK4}}$	HNN	LNN
No. of hidden layers	2	2	2	2
No. of nodes per hidden layer	256	256	256	256
Max. epochs	450	300	30,000	400
Initial learning rate	1e-3	1e-3	1e-3	1e-4
Learning rate decay	0.99	0.98	0.98	0.98
Activation function	Tanh	Tanh	Sigmoid,Tanh	Softmax
Loss function	MSE	MSE	MSE	MSE
Optimizer	Adam	Adam	Adam	Adam

for comparison is obtained by analytically solving Eq. (25). We can see in Figs. 4 (c) and (d) that the direct usage of the RK4 integrator provides results that gradually deviate from the true system. This issue of gradually increased errors becomes more severe in the MBD-NODE results with the RK4 integrator in Figs. 4 (a) and (b), highlighting their limitations in accurately modeling Hamiltonian systems. In contrast, both the LNN and HNN models, despite utilizing the RK4 integrator, demonstrate stable behavior in solving the mass-spring system, as shown in Figs. 4 (g)–(j). The more stable simulations of these two models can be attributed to the underlying equations of these models, which ensure energy conservation in the system. Notably, the HNN performance, as shown in Figs. 4 (i) and (j), show a deviation from the expected trajectory around the 30-second mark, leading to a higher MSE when compared to the MBD-NODE with the leapfrog integrator and the LNN model. Among all the methods we studied, the MBD-NODE with the leapfrog integrator outperforms other models, achieving the lowest MSE of $\epsilon = 1.3\text{e-}6$, with detailed trajectories of x and v presented in Figs. 4 (e) and (f).

Figure 5 presents the phase space trajectory and energy profile for the test set. It confirms the instability issues with the RK4 solver and the MBD-NODE model with the RK4 integrator, particularly in terms of energy drift accumulating over time. In comparison, both the LNN and HNN models, as well as the MBD-NODE model with the leapfrog integrator, demonstrate stable solutions without any noticeable energy drift. The results in Figs. 4 and 5 confirm the effectiveness of the MBD-NODE model with a symplectic integrator in accurately learning the Hamiltonian structure of the system.

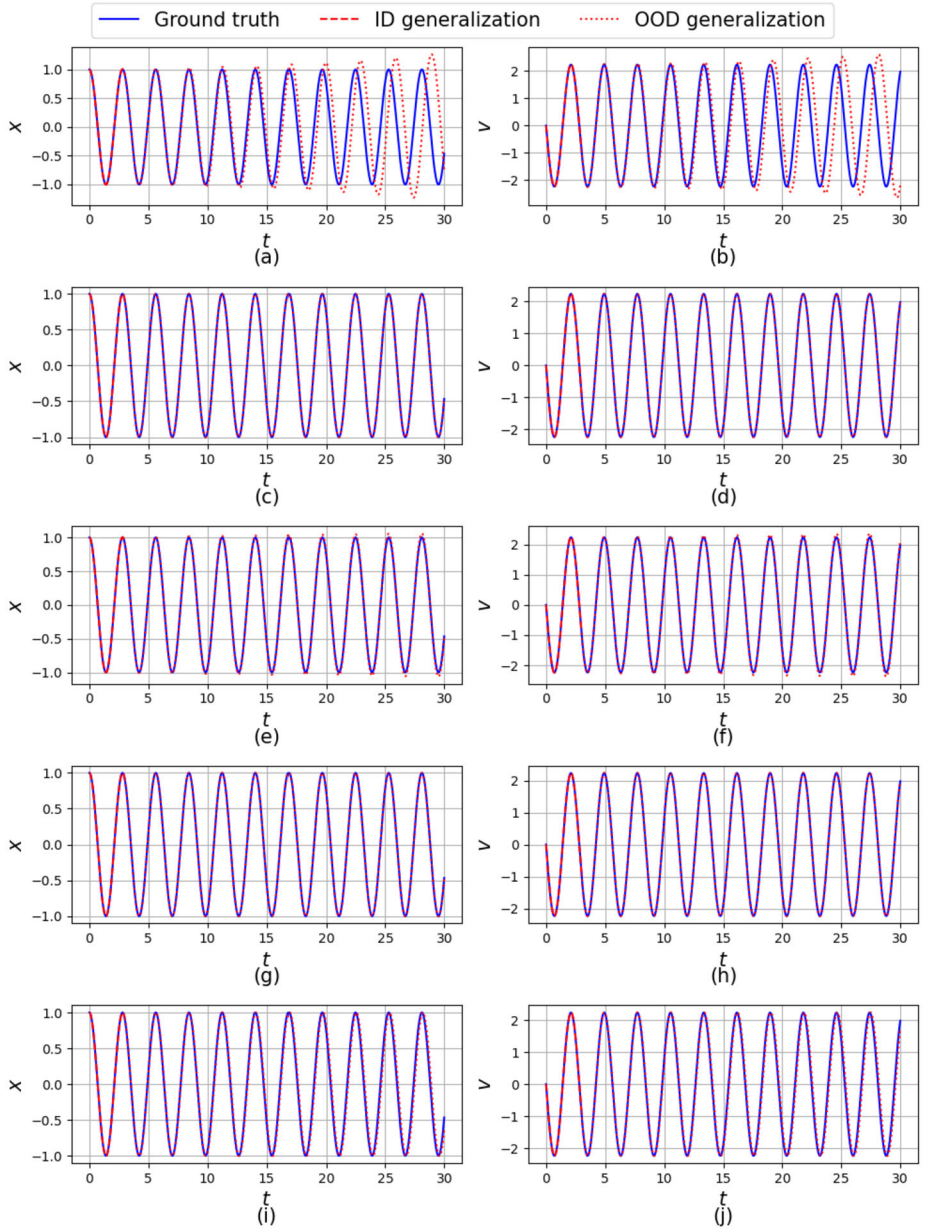


Fig. 4 Comparison of t vs x (left column) and t vs v (right column) for the different model and integrator combinations (rows) for the single-mass-spring system. Notice that the dashed lines represent performance on the training data set $t \in [0, 3]$, after which the dotted lines represent performance on the testing data set $t \in [0, 30]$. (a) and (b) are for the MBD-NODE with RK4, (c) and (d) are for the MBD-NODE with leapfrog integrator, (e) and (f) are for the RK4 integrator, (g) and (h) are for the LNN, and (i) and (j) are for the HNN

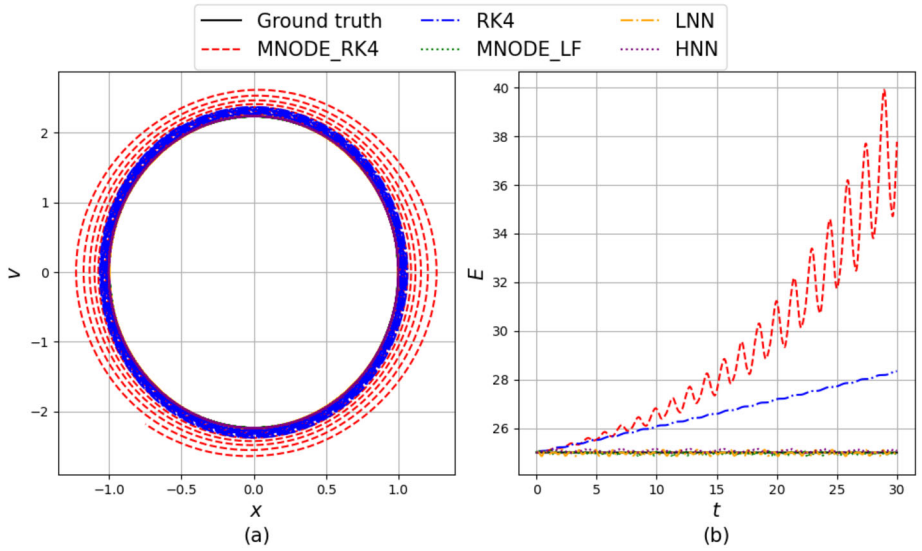


Fig. 5 (a) The phase space x vs v and (b) the system energy for the test data for the single mass-spring system

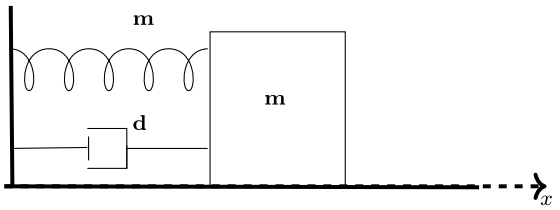


Fig. 6 The single mass-spring-damper system. The setup is similar to the example in Fig. 3, except for the addition of a damper with coefficient d

3.2 Single mass-spring-damper system

The second numerical test involves a single-mass-spring-damper system, as shown in Fig. 6. Compared with the first numerical test, there is a damper between the mass and wall, which causes the mass to slow down over time. It is important to note that the models designed for energy-conserving systems, like the LNN and HNN, are generally not applicable for dissipative systems without further modification. Therefore, we compare the performance of our method to LSTM and FCNN models, which are commonly employed in multibody dynamics problems.

The equation of motion for the single-mass-spring-damper system is given by:

$$\frac{d^2x}{dt^2} = \frac{-k}{m}x - \frac{d}{m}\frac{dx}{dt}, \tag{29}$$

where x represents the displacement of the mass from its equilibrium position, m is the mass of the object, set to 10 kg in this test, d is the damping coefficient, set to 2 Ns/m in this test, and k is the coefficient of stiffness of the spring, set to 50 N/m.

We choose the time step as 0.01 s for both the training and testing. The training dataset consists of a trajectory numerically solved by the RK4 solver for 300 time steps. In the testing phase the models are tested by predicting the system state within the training range and

Table 7 Hyperparameters for the single mass-spring-damper system

Hyperparameters	Model		
	MBD-NODE	LSTM	FCNN
No. of hidden layers	2	2	2
No. of nodes per hidden layer	256	256	256
Max. epochs	350	400	600
Initial learning rate	1e-3	5e-4	5e-4
Learning rate decay	0.98	0.98	0.98
Activation function	Tanh	Sigmoid,Tanh	Tanh
Loss function	MSE	MSE	MSE
Optimizer	Adam	Adam	Adam

extrapolation to predict system behavior for additional 100 time steps. The initial condition for this problem is $x = 1$ m, $v = 0$ m/s. Note that the system is no longer a Hamiltonian system, so we use Cartesian coordinates for all the methods. The hyperparameters used for each model are summarized in Table 7.

Figure 7 presents the position x and velocity v for all the trained models. In the first 300 time steps, which correspond to the training range, all three models exhibit accurate predictions, indicating an effective training process. However, differences in model performance start to show up in the testing regime (i.e., $t > 3$). More specifically, Figs. 7(a) and (b) show that the MBD-NODE gives a reasonable prediction that closely matches the ground truth with the lowest MSE of $\epsilon = 8.6\text{e-}4$, which demonstrates its predictive capability.

On the other hand, the LSTM predictions tend to just replicate historical data patterns (see Figs. 7 (c) and (d)), rather than learning and adapting to the underlying dynamics of the system. This limitation makes LSTM fail to correctly capture the decay of energy for an energy-dissipative system. The FCNN model struggles with extrapolation as well, mainly because the good extrapolation performance of FCNN heavily relies on the closeness of training and testing data in their distributions. This limitation of FCNN leads to errors in the extrapolation task of this example, resulting in the largest MSE of $\epsilon = 9.9\text{e-}2$ as shown in Figs. 7 (e) and (f).

More insights into the system dynamics are provided by the phase space trajectories illustrated in Fig. 8. The predictions of the MBD-NODE, as depicted in Fig. 8 (a), are closely aligned with the observed behavior of the system. In contrast, the LSTM trajectory, shown in Fig. 8 (b), exhibits stagnation and fails to reflect the system eventual halt. The FCNN performance, presented in Fig. 8 (c), is lacking during the extrapolation test – it merely yields predictions in the tangent direction, resulting in a significant divergence from the anticipated trajectory.

3.3 Multiscale triple mass-spring-damper system

This system, shown in Fig. 9, has three masses. The largest mass is 100 times larger than the smallest one. The main purpose of this example is to gauge method performance on multiscale systems. The equations of motion for the triple mass-spring-damper system are

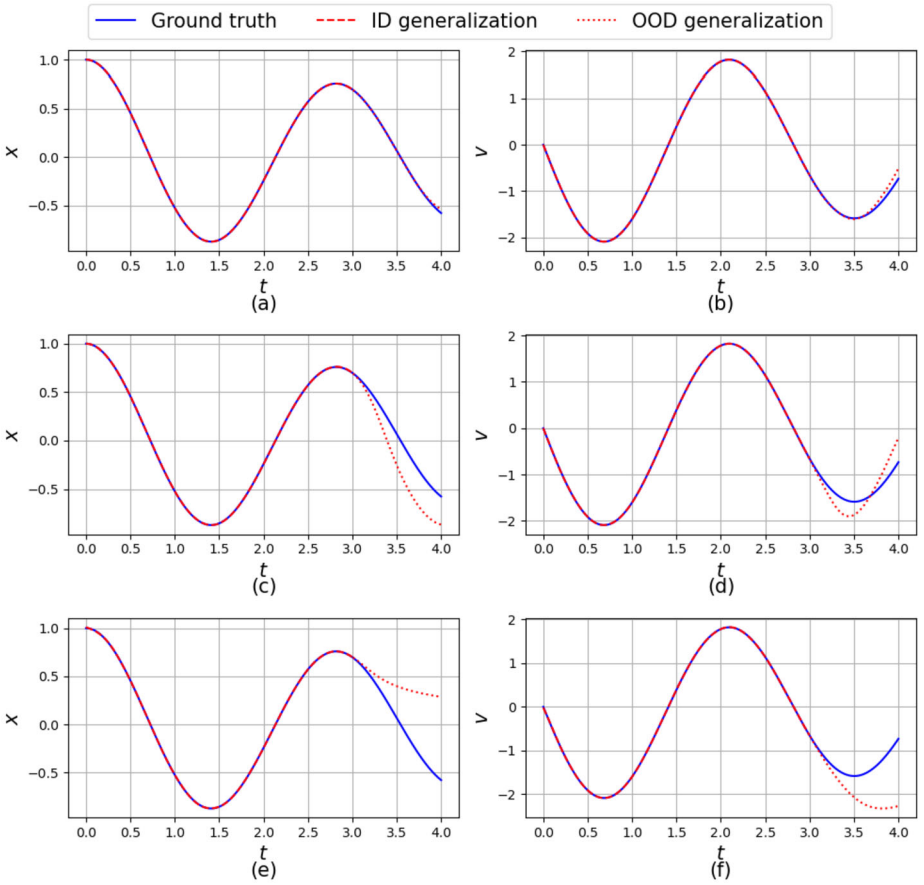


Fig. 7 Comparison of t vs x (left column) and t vs v (right column) for different models (rows) for the single-mass-spring-damper system. Notice that the dashed lines represent performance on the training data set $t \in [0, 3]$, after which the dotted lines represent performance on the testing data set. (a) and (b) are for the MBD-NODE with MSE $\epsilon = 8.6\text{e-}4$; (c) and (d) are for the LSTM with MSE $\epsilon = 1.8\text{e-}2$, and (e) and (f) are for the FCNN with MSE $\epsilon = 9.9\text{e-}2$

as follows:

$$\begin{aligned}
\frac{d^2x_1}{dt^2} &= -\frac{k_1}{m_1}x_1 - \frac{d_1}{m_1}(v_1 - v_2) + \frac{k_2}{m_1}(x_2 - x_1) + \frac{d_2}{m_1}(v_2 - v_1),\\
\frac{d^2x_2}{dt^2} &= -\frac{k_2}{m_2}(x_2 - x_1) - \frac{d_2}{m_2}(v_2 - v_1) + \frac{k_3}{m_2}(x_3 - x_2) + \frac{d_3}{m_2}(v_3 - v_2),\\
\frac{d^2x_3}{dt^2} &= -\frac{k_3}{m_3}(x_3 - x_2) - \frac{d_3}{m_3}(v_3 - v_2),
\end{aligned} \tag{30}$$

where x_1, x_2, x_3 are the positions of the masses, respectively, m_1, m_2, m_3 are the masses of the object with values of 100 kg, 10 kg, and 1 kg, respectively, d_1, d_2, d_3 are the damping coefficients, each set to 2 Ns/m, and k_1, k_2, k_3 are the spring stiffness values, all set to 50 N/m.

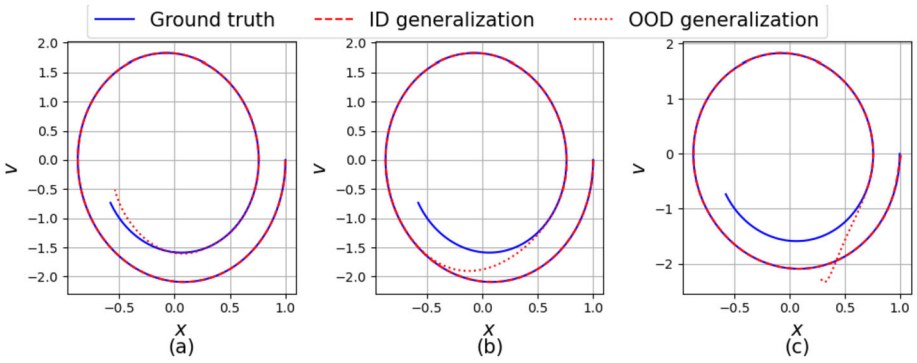


Fig. 8 The phase space x vs v for the single mass spring damper system. Dashed lines represent performance on the training data, and the dotted lines on the test data. (a) is for the MBD-NODE, (b) is for the LSTM, and (c) is for the FCNN

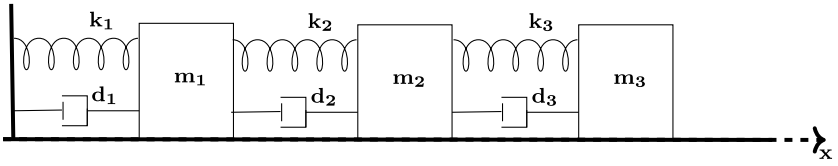


Fig. 9 Triple mass-spring-damper system. The setup is similar to the example in Fig. 3, except for the addition of two more masses, springs, and dampers

Table 8 Hyperparameters for the triple mass-spring-damper system

Hyperparameters	Model		
	MBD-NODE	LSTM	FCNN
No. of hidden layers	2	2	2
No. of nodes per hidden layer	256	256	256
Max. epochs	350	400	600
Initial learning rate	6e-4	5e-4	5e-4
Learning rate decay	0.98	0.98	0.98
Activation function	Tanh	Sigmoid,Tanh	Tanh
Loss function	MSE	MSE	MSE
Optimizer	Adam	Adam	Adam

For the numerical settings of the triple mass-spring-damper system, we choose the time step as 0.01 s for both training and testing. The training dataset has a trajectory numerically computed by the RK4 solver for 300 time steps. The initial conditions are set as $x_1 = 1$, $x_2 = 2$, $x_3 = 3$, $v_1 = v_2 = v_3 = 0$ (all units are SI). The models are tested by extrapolating for 100 more time steps. The hyperparameters used for the models are summarized in Table 8.

Figure 10 presents the position x and velocity v of the triple mass-spring-damper system during training and testing. In terms of accuracy, the MBD-NODE outperforms other models with an MSE $\epsilon = 8.2\text{e-}3$. More specifically, the MBD-NODE and LSTM models provide

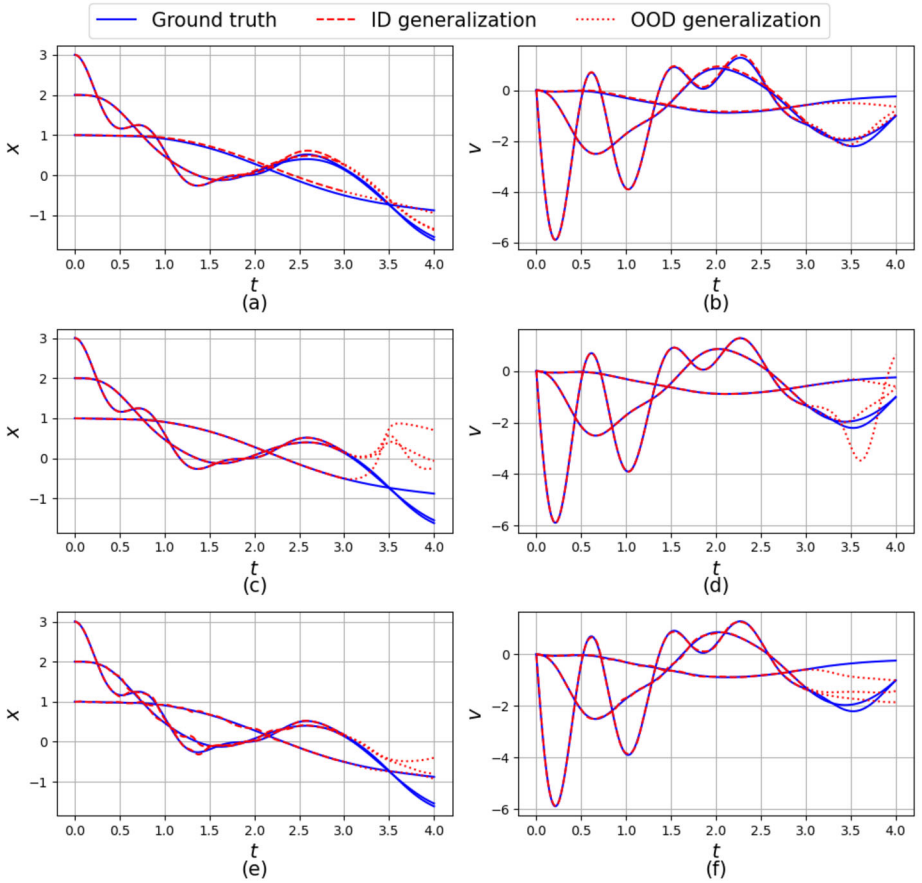


Fig. 10 Comparison of t vs x (left column) and t vs v (right column) for the different models (rows) for the triple-mass-spring-damper system. Notice that the dashed lines represent performance on the training data set $t \in [0, 3]$, after which the dotted lines represent performance on the testing data set. (a) and (b) are for the MBD-NODE with MSE $\epsilon = 8.2\text{e-}3$, (c) and (d) are for the LSTM with MSE $\epsilon = 1.8\text{e-}2$, and (e) and (f) are for the FCNN with MSE $\epsilon = 4.2\text{e-}2$

accurate results in the range of training data (i.e., $t < 3$), whereas the results of FCNN model show small oscillation mainly due to the multiscale setting of the dynamics shown in Fig. 10 (e). In the testing data (i.e., $t > 3$) the performance of trained models starts to differ more. The MBD-NODE can still give a reasonable prediction for the triple mass-spring-damper system shown in Figs. 10 (a) and (b), although the predicted trajectories slowly deviate from the true ones, mainly because of the accumulation of numerical errors. On the other hand, the LSTM tends to replicate some of the historical patterns. The testing performance of the FCNN model is more reasonable than the LSTM model in this example but still less satisfactory compared with the MBD-NODE model.

The trajectory for the triple mass-spring-damper system for the test set is shown in Fig. 11. We can see that for the MBD-NODE, the trajectory of the first body shown in Fig. 11 (a) has some mismatch with the ground truth. This is caused by the multiscale property that the first mass has the largest mass, which leads to the slightest change in the position x and velocity v, whereas the MBD-NODE learns the dynamics from the difference between the

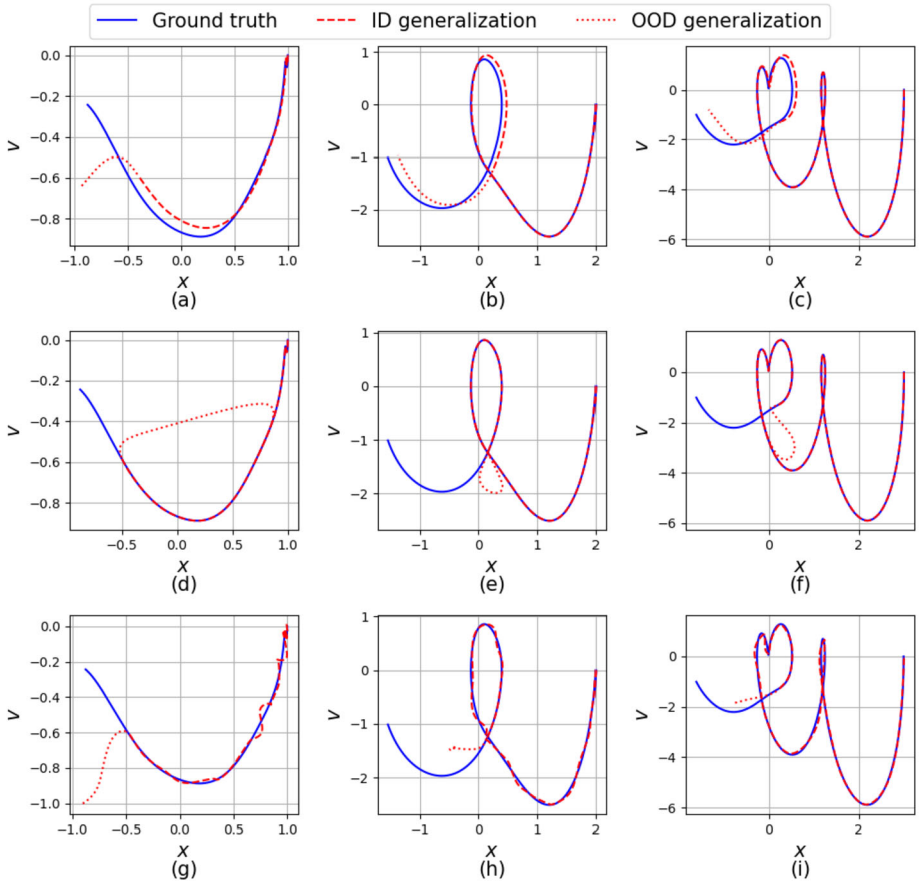


Fig. 11 The phase space trajectories for the triple mass-spring-damper system. The left, middle, and right columns correspond to the first, second, and third masses. Dashed lines represent performance on the training data, and the dotted lines on the test data. (a), (b), and (c) are for the MBD-NODE. (d), (e), and (f) are for the LSTM, and (g), (h), and (i) are for the FCNN

state at two nearby times. So the largest mass will contribute the least to the loss, which causes the MBD-NODE to learn the dynamics of the first body inadequately. The numerical integration error also accumulates during the inference, which makes the error larger. For LSTM, we can more clearly see that its prediction trends converge to the historical data, which does not work well during OOD generalization. For FCNN, we note the oscillation for the first body in ID generalization, and for the OOD generalization, which leads to a lackluster predictive performance.

3.4 Damped single pendulum

In this section, we test the MBD-NODE ability to generalization on different initial conditions and external forces using the damped single pendulum as shown in Fig. 12. The equation of motion (31) for a damped single pendulum, including the gravitational and damping

Fig. 12 Single pendulum system

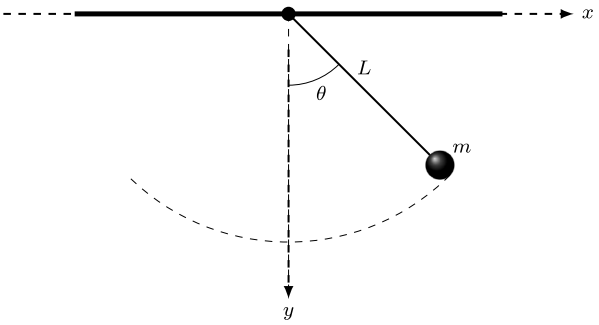


Table 9 Hyperparameters for the single pendulum system

Hyperparameters	Model		
	MBD-NODE	LSTM	FCNN
No. of layers	2	2	2
No. of nodes per hidden layer	256	256	256
Max. epochs	400	400	600
Initial learning rate	6e-4	5e-4	5e-4
Learning rate decay	0.98	0.98	0.98
Activation function	Tanh	Sigmoid,Tanh	Tanh
Loss function	MSE	MSE	MSE
Optimizer	Adam	Adam	Adam

forces, can be represented as the second-order ODE

$$\ddot{\theta}(t) + \frac{g}{L}\sin(\theta(t)) + \frac{c}{mL}\dot{\theta}(t) = 0, \tag{31}$$

where $\theta(t)$ is the angular displacement as a function of time, g is the acceleration due to gravity and external force, $L = 1$ m is the length of the pendulum, $c = 0.1$ Ns/m is the damping coefficient, and $m = 1$ kg is the mass of the pendulum bob.

Initially, we examine a scenario where the pendulum is released from its lowest point with initial angular velocity $\omega = \pi$. We employ various models to predict the trajectory of the pendulum using identical training and testing datasets. In practice, the midpoint method is utilized to solve the ODE (31), adopting a time step of 0.01 seconds. The dataset for training spans the initial 3 seconds, whereas the testing dataset covers the subsequent 1 second. The hyperparameters applied across the models are detailed in Table 9.

Figs. 13 and 14 present the dynamics response and phase space of the single pendulum system during the ID generalization and OOD generalization. The MBD-NODE outperforms other models with an MSE $\epsilon = 2.0\text{e-}3$. Although LSTM has a small MSE $\epsilon = 3.4\text{e-}3$, it tends to replicate some of the historical patterns and fails to capture the damping effect for OOD generalization. The FCNN model has a larger MSE $\epsilon = 8.0\text{e-}1$, associated with the lackluster OOD generalization ability of the FCNN model.

Beyond the first setting, we test MBD-NODE ability to generalize under varying initial conditions and external forces. Importantly, we use the MBD-NODE trained in the first setting directly without adding new training data – a significant challenge for OOD generalization. FCNNs and LSTMs are not suitable for handling time-varying external forces.

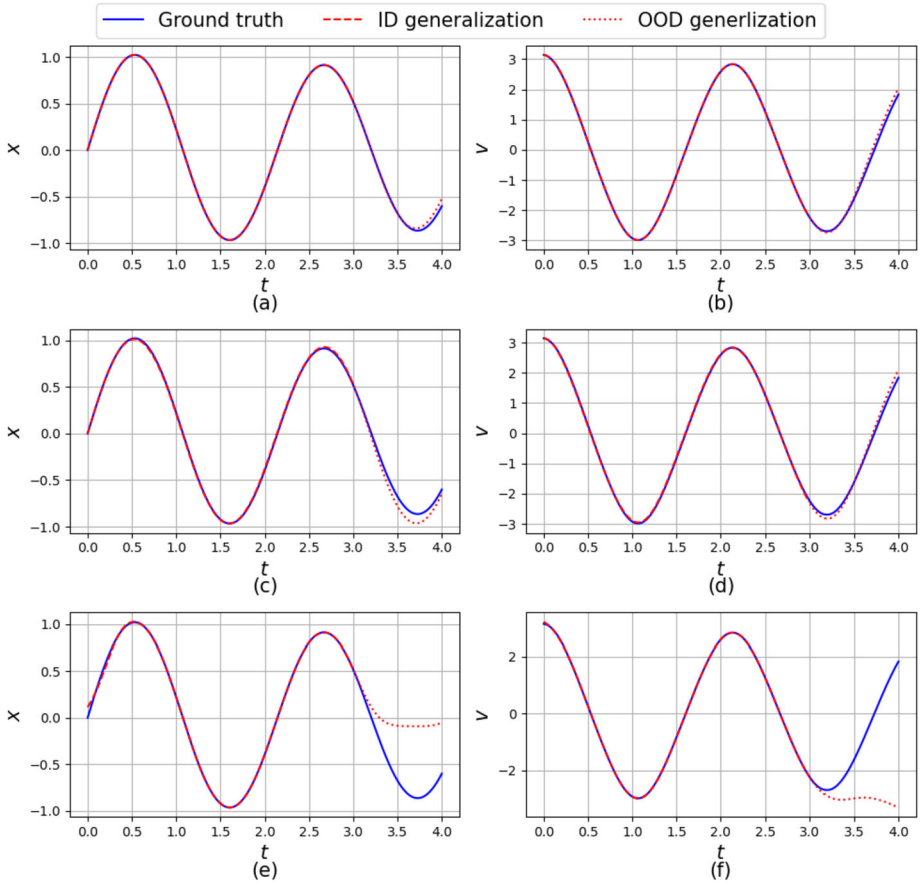


Fig. 13 Comparison of t vs x (left column) and t vs v (right column) for the different models (rows) for the single pendulum system. Notice the dashed lines represent performance on the training data set $t \in [0, 3]$, after which the dotted lines represent performance on the testing data set. (a) and (b) are for the MBD-NODE with MSE $\epsilon = 2.0\text{e-}3$, (c) and (d) are for the LSTM with MSE $\epsilon = 3.4\text{e-}3$, and (e) and (f) are for the FCNN with MSE $\epsilon = 8.0\text{e-}1$

They can only work with different parameters that do not change with respect to time. For FCNNs, accommodating changes in initial conditions would require a larger model, additional data, and retraining. Therefore we only test MBD-NODE in this setting. Additionally, MBD-NODE nature allows us to directly calculate acceleration from external forces and incorporate it, simplifying integration with gravity as shown in Fig. 2.

Fig. 15 presents the dynamics response of the single pendulum system with four different unseen initial conditions given in four quadrants. Because the MBD-NODE learns the dynamics related to the range of the phase space covered in the training set and does so independently of the initial condition, MBD-NODE yields a reasonable prediction for any of the four different initial conditions. Fig. 16 presents the dynamics response of the single pendulum system with random external force. Here we sample the external force from the normal distribution $F \sim \mathcal{N}(0, 25)$ and apply it to the single pendulum at every time step. We predict 300 time steps for the single pendulum with random excitation. Because the force will push the pendulum to unseen state space, this is a good test to probe the MBD-NODE

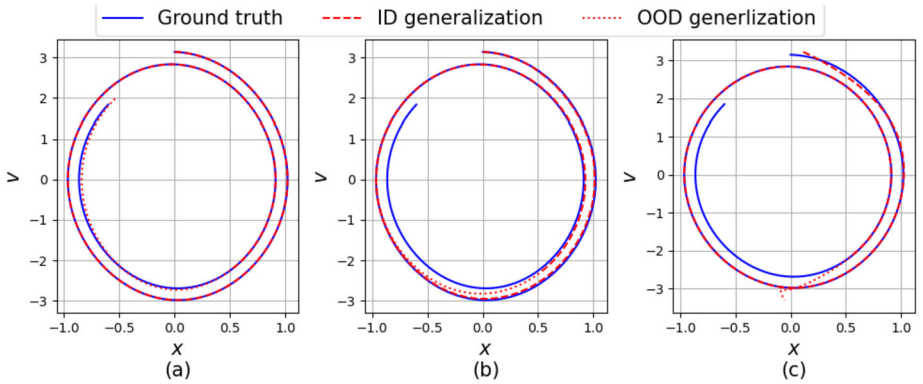


Fig. 14 The phase space trajectories for the single pendulum system. Dashed lines represent performance on the training data and the dotted lines on the test data. (a) is for the MBD-NODE, (b) is for the LSTM, and (c) is for the FCNN

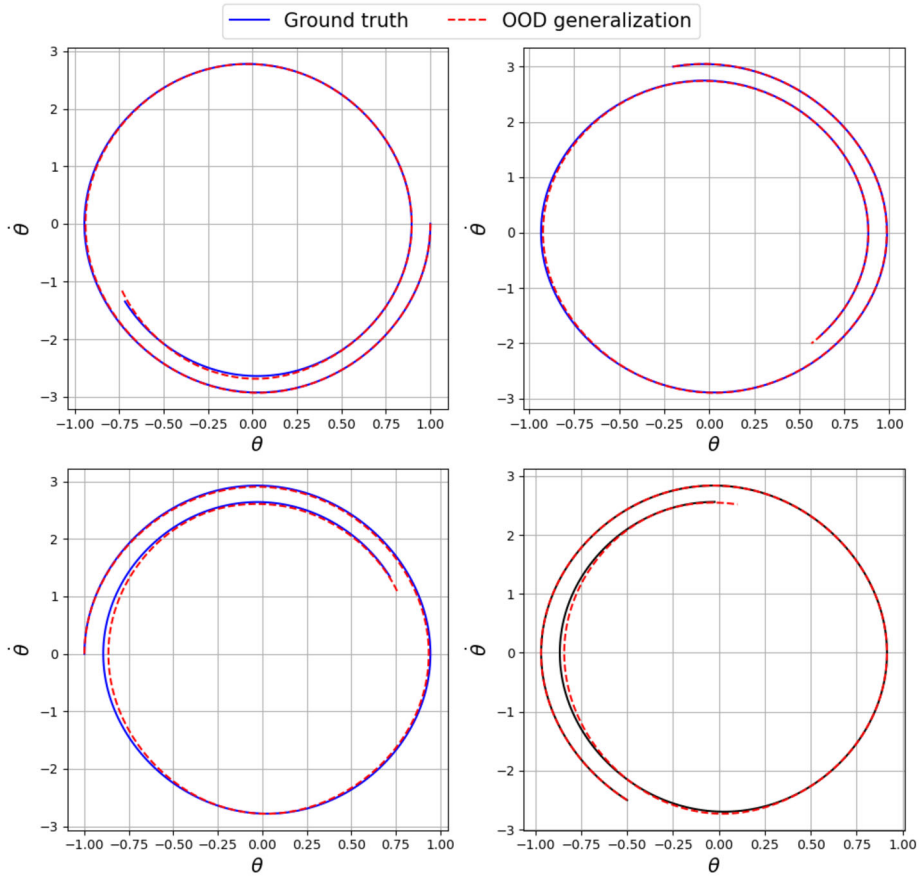


Fig. 15 The prediction trajectory θ vs ω plot for the single pendulum by MBD-NODE with different initializations. (a): $(\theta(0), \omega(0)) = (1, 0)$; (b): $(\theta(0), \omega(0)) = (-0.2, 3)$; (c): $(\theta(0), \omega(0)) = (-1, 0)$; (d): $(\theta(0), \omega(0)) = (-0.5, -2.5)$

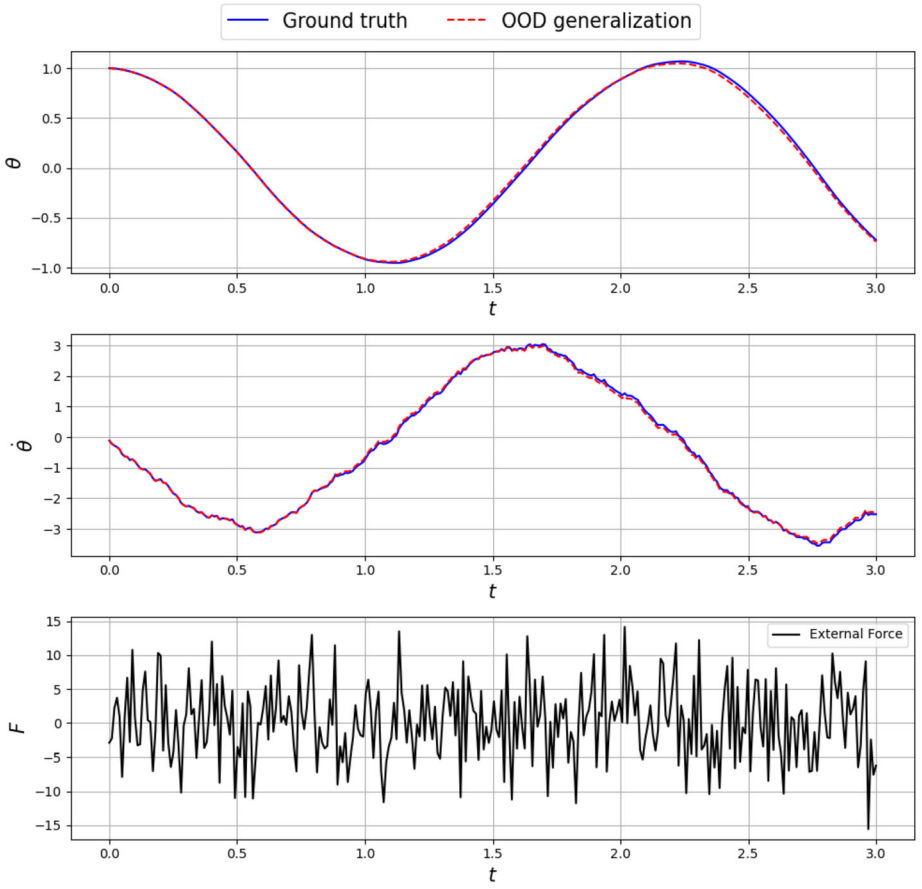


Fig. 16 The dynamics response plot for t vs θ, t vs ω plot, and the value of random force applied to the single pendulum

OOD generalization ability. MBD-NODE can continue to give an accurate prediction for the single pendulum system under random external force.

3.5 Double pendulum

To gauge the performance of our model on chaotic systems, we study the double pendulum system (see Fig. 17) as a numerical example. This pendulum system has two masses m_1 and m_2, lengths l_1 and l_2, and two angles $q_1 = \theta_1$ and $q_2 = \theta_2$. The generalized momenta corresponding to these angles are p_{θ_1} and p_{θ_2}, which needs to be calculated by using the Lagrangian. The Hamiltonian H for this system is given by

$$H(q_1, q_2, p_{\theta_1}, p_{\theta_2}) = T(q_1, q_2, p_{\theta_1}, p_{\theta_2}) + V(q_1, q_2), \tag{32}$$

where T is the kinetic energy, and V is the potential energy. For the double pendulum system, the kinetic energy T and potential energy U are given by

$$T = \frac{1}{2} m_1 l_1^2 \dot{\theta}_1^2 + \frac{1}{2} m_2 \left(l_1^2 \dot{\theta}_1^2 + l_2^2 \dot{\theta}_2^2 + 2 l_1 l_2 \dot{\theta}_1 \dot{\theta}_2 \cos(\theta_1 - \theta_2) \right), \tag{33}$$

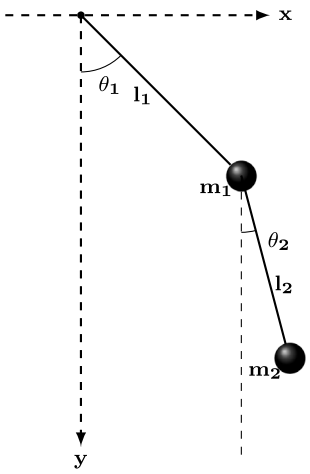


Fig. 17 Double pendulum

$$U = -m_1 g l_1 \cos(\theta_1) - m_2 g \left(l_1 \cos(\theta_1) + l_2 \cos(\theta_2)\right). \tag{34}$$

The Hamiltonian H can be expressed in terms of q_1, q_2, p_{θ_1}, and p_{θ_2} as

$$\begin{aligned} H(q_1, q_2, p_{\theta_1}, p_{\theta_2}) = & \frac{p_{\theta_1}^2}{2m_1 l_1^2} + \frac{p_{\theta_2}^2}{2m_2 l_2^2} + m_2 l_1 l_2 \cos(q_1 - q_2) p_{\theta_1} p_{\theta_2} \\ & - m_1 g l_1 \cos(q_1) - m_2 g \left(l_1 \cos(q_1) + l_2 \cos(q_2)\right). \end{aligned} \tag{35}$$

The gradients of the Hamiltonian, $\nabla_q H$ and $\nabla_p H$, can be used to derive the Hamilton equations of motion:

$$\dot{q}_i = \frac{\partial H}{\partial p_{\theta_i}}, \tag{36}$$

$$\dot{p_{\theta_i}} = -\frac{\partial H}{\partial q_i}. \tag{37}$$

The specific Hamilton equations of motion for the double pendulum system are

$$\dot{\theta}_1 = \frac{l_2 p_{\theta_1} - l_1 p_{\theta_2} \cos(\theta_1 - \theta_2)}{l_1^2 l_2 \left[m_1 + m_2 \sin^2(\theta_1 - \theta_2)\right]}, \tag{38}$$

$$\dot{\theta}_2 = \frac{-m_2 l_2 p_{\theta_1} \cos(\theta_1 - \theta_2) + (m_1 + m_2) l_1 p_{\theta_2}}{m_2 l_1 l_2^2 \left[m_1 + m_2 \sin^2(\theta_1 - \theta_2)\right]}, \tag{39}$$

$$\dot{p}_{\theta_1} = -(m_1 + m_2) g l_1 \sin\theta_1 - h_1 + h_2 \sin\left[2(\theta_1 - \theta_2)\right], \tag{40}$$

$$\dot{p}_{\theta_2} = -m_2 g l_2 \sin\theta_2 + h_1 - h_2 \sin\left[2(\theta_1 - \theta_2)\right], \tag{41}$$

where

$$h_2 = \frac{m_2 l_2^2 p_{\theta_1}^2 + (m_1 + m_2) l_1^2 p_{\theta_2}^2 - 2 m_2 l_1 l_2 p_{\theta_1} p_{\theta_2} \cos(\theta_1 - \theta_2)}{2 l_1^2 l_2^2 \left[m_1 + m_2 \sin^2(\theta_1 - \theta_2)\right]^2}, \tag{42}$$

Table 10 Hyperparameters for the double pendulum system

Hyperparameters	Model		
	MBD-NODE	LSTM	FCNN
No. of hidden layers	2	2	2
No. of nodes per hidden layer	256	256	256
Max. epochs	450	400	600
Initial learning rate	1e-3	5e-4	5e-4
Learning rate decay	0.98	0.98	0.99
Activation function	Tanh	Sigmoid,Tanh	Tanh
Loss function	MSE	MSE	MSE
Optimizer	Adam	Adam	Adam

$$h_1 = \frac{p_{\theta_1} p_{\theta_2} \sin(\theta_1 - \theta_2)}{l_1 l_2 \left[m_1 + m_2 \sin^2(\theta_1 - \theta_2)\right]}. \tag{43}$$

The double pendulum system is defined as follows: the rod lengths $L_1 = L_2 = 1$ m, the concentrated masses $m_1 = m_2 = 1$ kg, the gravitational acceleration $g = 9.81$ m/s^2, tne initial angular displacement of the first mass $\theta_1(0) = \frac{3\pi}{7}$, the initial angular velocity of the first mass $\dot{\theta}_1(0) = 0$, the initial angular displacement of the second mass $\theta_2(0) = \frac{3\pi}{4}$, and the initial angular velocity of the second mass $\dot{\theta}_2(0) = 0$. We set the time step as 0.01 s for both training and testing. The training dataset has a trajectory numerically computed via RK4 for 300 time steps. The models are tested by extrapolating for 100 more time steps. The hyperparameters used for the models are summarized in Table 10.

Figure 18 shows the dynamic response of the different methods for a double pendulum. We can observe that all three models can give good predictions in the range of the training set. In the extrapolation range the three models gradually diverge. The challenge for integrator-based methods like MBD-NODE is particularly pronounced due to the inherently chaotic nature of the double pendulum system, which tends to amplify integration errors rapidly, leading to significant discrepancies. For a discussion about the limitations of numerical integration methods like the Runge–Kutta and integration-based neural networks like Physics-Informed Neural Networks (PINN), the reader is referred to [50]. For the double pendulum problem, these two approaches give large divergence for small initial perturbation. Despite this, the MBD-NODE outperforms the two other models with an MSE of $\epsilon = 2.0\text{e-}1$.

The phase space trajectories obtained by the three models are shown in Fig. 19. We can observe that the MBD-NODE model overall outperforms the other two models in the testing data regime. Although there are noticeable differences between the prediction and ground truth for the MBD-NODE model, it is still trying to capture the patterns of ground truth in the testing regime, especially for the second mass. On the contrary, the LSTM model tends to replicate the historical trajectories as shown in Figs. 19 (c) and (d). For example, the FCNN fails to demonstrate predictive attributes outside the training regime.

3.6 Cart-pole system

In this section, we consider the cart-pole system, which is a classical benchmark problem in control theory. As shown in Fig. 20, the system consists of a cart that can move horizontally along a frictionless track and a pendulum that is attached to the cart. The pendulum is free

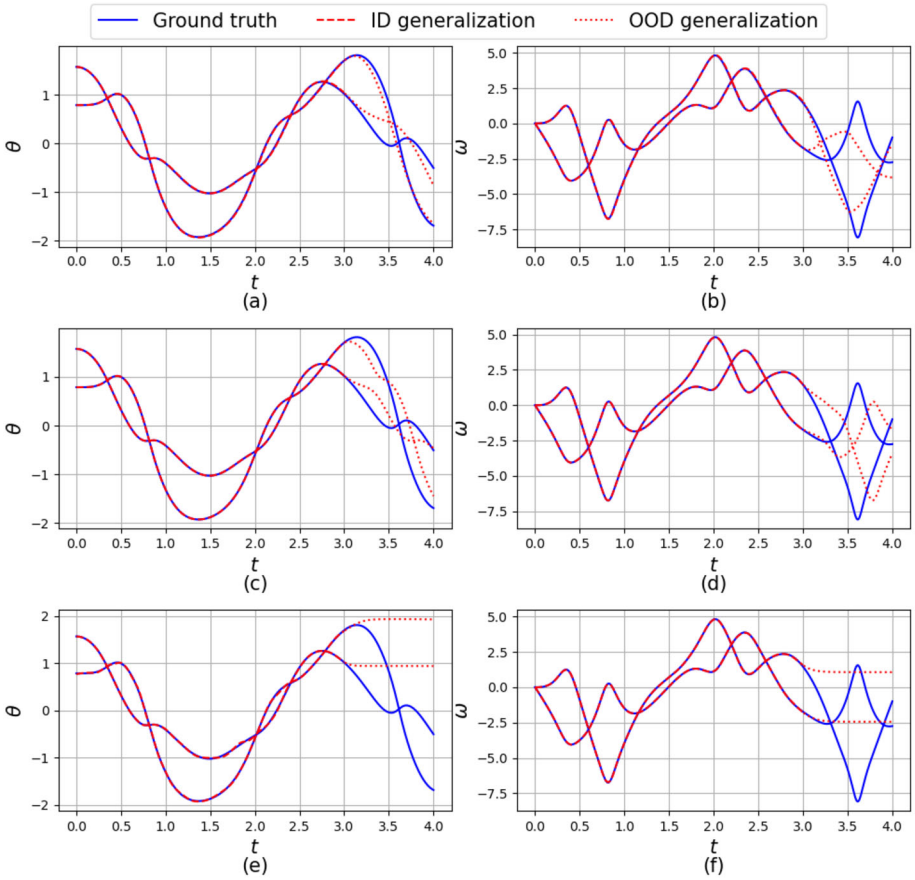


Fig. 18 Comparison of t vs θ (left column) and t vs ω (right column) for different models (rows) of the double pendulum system. Notice that the dashed lines represent performance on the training data set $t \in [0, 3]$, after which the dotted lines represent performance on the testing data set. (a) and (b) are for the MBD-NODE with MSE $\epsilon = 2.0\text{e-}1$, (c) and (d) are for the LSTM with MSE $\epsilon = 6.4\text{e-}1$, and (e) and (f) are for the FCNN with MSE $\epsilon = 2.2\text{e+}0$

to rotate about its pivot point. The system state is described by the position of the cart x, the velocity of the cart v, the angle of the pendulum θ, and the angular velocity of the pendulum ω. The equations of motion for the cart-pole system are given by the following second-order nonlinear ODEs:

$$\begin{aligned} ml^2\ddot{\theta} + ml\cos\theta\ddot{x} - mgl\sin\theta = 0, \\ ml\cos\theta\ddot{\theta} + (M+m)\ddot{x} - ml\dot{\theta}^2\sin\theta = u, \end{aligned} \tag{44}$$

where $M = 1$ kg is the mass of the cart, and $m = 1$ kg is the mass of the pole, $l = 0.5$ m is the length of the pole, and u is the external force horizontally applied to the cart with unit N. We first consider the case in which the cart-pole system is set to an initial position, and then we let the system evolve without any external force being applied. The initial conditions are

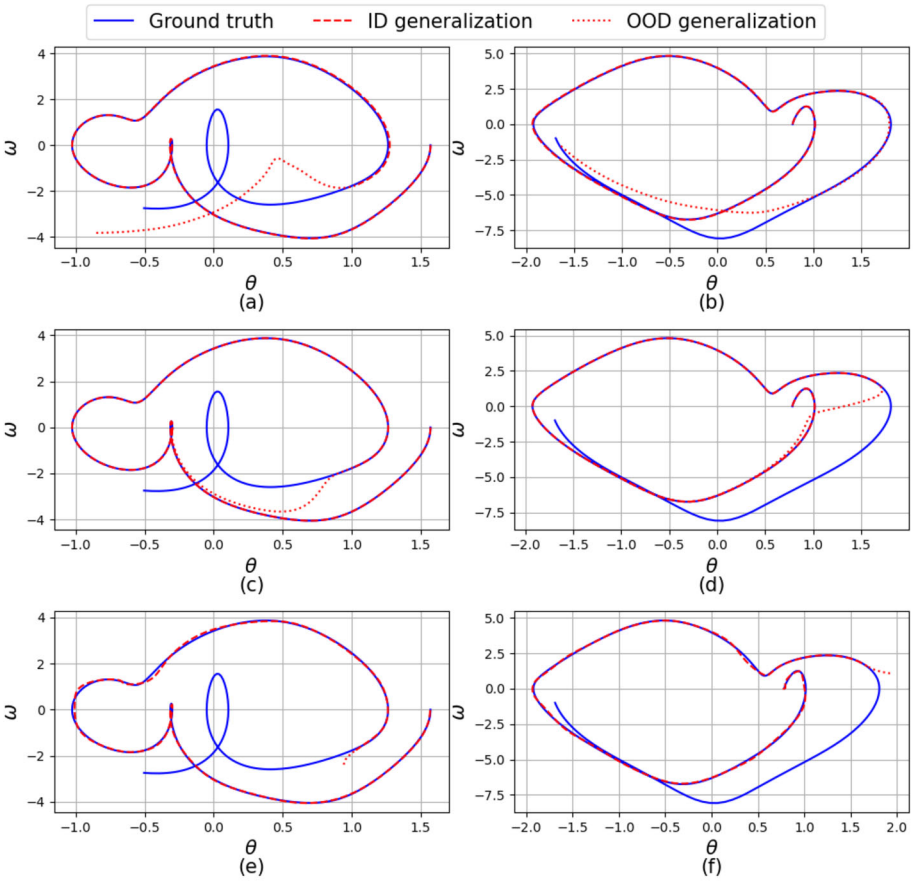
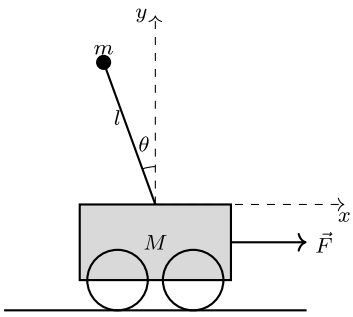


Fig. 19 The phase space trajectories for the double pendulum system. The left and right columns correspond to the first and second masses, respectively. Dashed lines represent performance on the training data, and the dotted lines on the test data. (a) and (b) are for the MBD-NODE, (c) and (d) are for the LSTM, and (e) and (f) are for the FCNN

Fig. 20 The cart-pole system



set as

$$x(0) = 1, \quad v(0) = 0, \quad \theta(0) = \frac{\pi}{6}, \quad \omega(0) = 0. \tag{45}$$

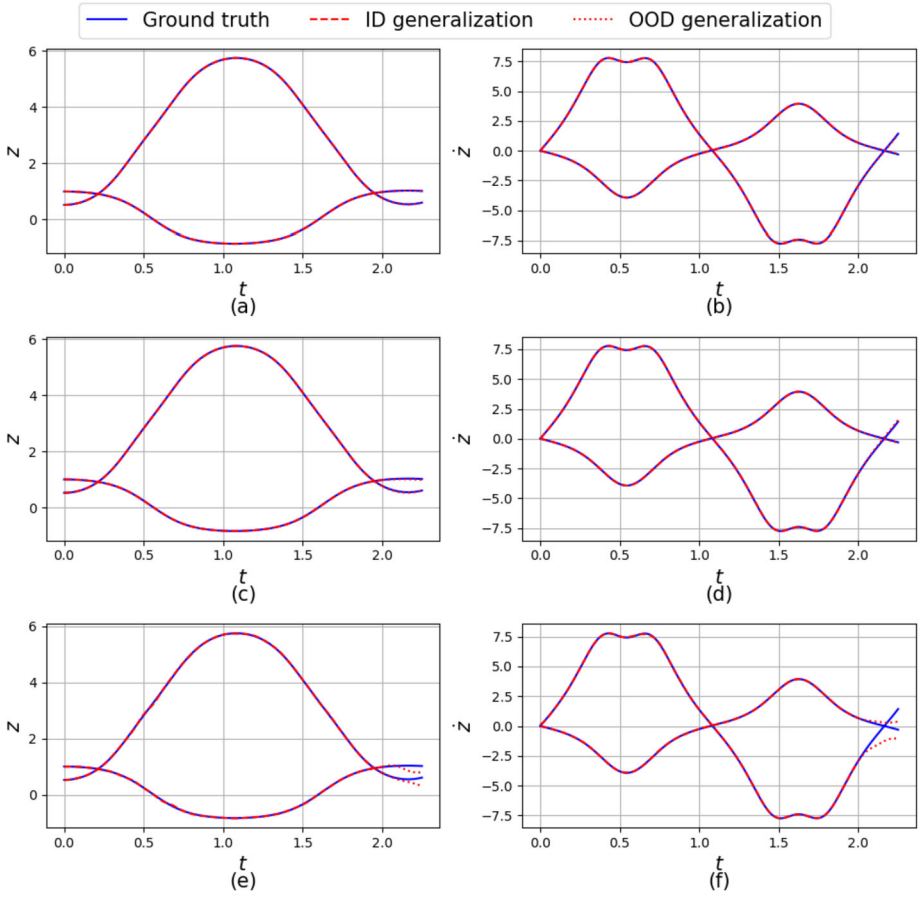


Fig. 21 Comparison of t vs z (left colum) and t vs $\dot{z}$ (right column) for different models (rows) of the cart-pole system. The dashed lines represent the ID generation, and the dotted lines represent the OOD generalization. (a) and (b) are for the MBD-NODE with MSE $\sigma = 6.0\text{e-}5$, (c) and (d) are for the LSTM with $\sigma = 3.2\text{e-}4$, and (e) and (f) are for the FCNN with $\sigma = 4.7\text{e-}2$

The system is simulated using the midpoint method with a time step of 0.005 s. We generate the training data by simulating the system for 400 time steps and the testing data by simulating the system for 100 time steps. As shown in Figs. 21 and 22, the MBD-NODE can accurately predict the system dynamics with $\sigma = 6.0\text{e-}5$. Because this case is a periodic system that time series data can fully describe, the LSTM model can also provide accurate predictions with $\sigma = 3.0\text{e-}4$. However, the FCNN model still gives lackluster OOD generalization performance with $\sigma = 4.7\text{e-}2$. The hyperparameters for each model are summarized in Table 11.

Furthermore, we consider the case that the cart-pole system is set to the initial position (45), and then we apply the external force u to the cart to balance the pole and keep the cart-pole system at the origin point. In general control theory, model predictive control (MPC) is a popular method for solving this kind of control problem by linearizing the nonlinear system dynamics and solving a quadratic convex optimization problem over a finite time horizon at each time step. Specifically, for a linearized system dynamics $\dot{z} = Az + Bu$, the

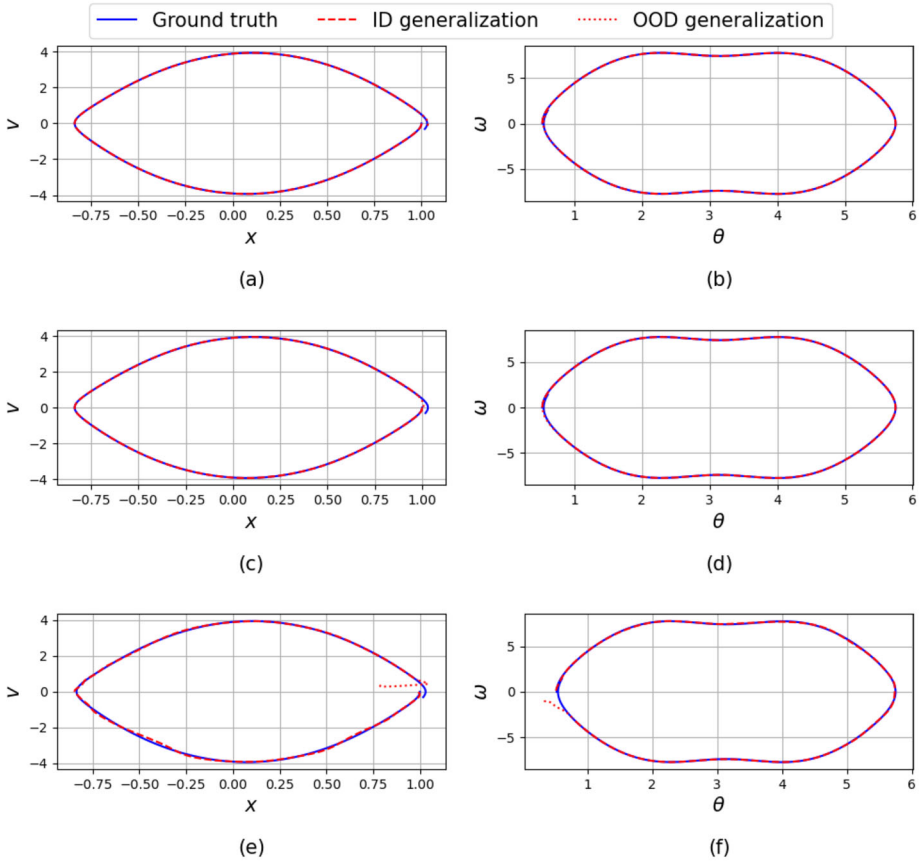


Fig. 22 The phase space x vs v (left Column) and θ vs $\dot{\theta}$ (right column) of the cart-pole system. The dashed line represents the ID generation, and the dotted line represents the ODD generalization. (a) and (b) are for the MBD-NODE, (c) and (d) are for the LSTM, and (e) and (f) are for the FCNN

optimization problem can be formulated as a convex optimization problem as follows [30]:

$$\min_{u} \sum_{k=0}^{N-1} z_k^T Q z_k + u_k^T R u_k \tag{46}$$

$$\text{s.t.} z_{k+1} = A z_k + B u_k, \quad k = 0, 1, \ldots, N-1, \tag{47}$$

$$z_k \in Z, \quad u_k \in U, \quad k = 0, 1, \ldots, N-1, \tag{48}$$

where $z_k = (\theta_k, x_k, \omega_k, v_k)$ is the state of the system at time step k, $N = 50$ is the time horizon for optimization, u_k is the control input at time step k, Q and R are the weighting matrices, which are set to the identity matrix, and Z and U are the constraints for the state and control input, respectively.

Table 11 Hyperparameters for the cart-pole system

Hyperparameters	Model		
	MBD-NODE	LSTM	FCNN
No. of hidden layers	2	2	2
No. of nodes per hidden layer	256	256	256
Max. epochs	450	400	600
Initial learning rate	1e-3	5e-4	5e-4
Learning rate decay	0.98	0.98	0.99
Activation function	Tanh	Sigmoid,Tanh	Tanh
Loss function	MSE	MSE	MSE
Optimizer	Adam	Adam	Adam

For the cart-pole system, the matrix A and B can be easily derived from the system dynamics Eq. (44) by the first-order Taylor series approximation, which are:

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ \frac{g(m+M)}{Ml} & 0 & 0 & 0 \\ -\frac{mg}{M} & 0 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 0 \\ -\frac{1}{Ml} \\ \frac{1}{M} \end{bmatrix}. \tag{49}$$

As a high-accuracy and differentiable model, the MBD-NODE can be used to directly linearize the system dynamics by calculating the Jacobian matrix of the well-trained MBD-NODE. In this case, MBD-NODE captures the system dynamics by learning the map $f(\theta_k, x_k, \omega_k, v_k, u)$ to the angular acceleration $\ddot{\theta}_k$ and the acceleration $\ddot{x}_k$. In practice, the Jacobian matrix can be calculated by automatic differentiation, which is used to replace the matrix A and B in the MPC optimization problem. To get the well-trained MBD-NODE, we train the model with 10^5 uniformly sampling data points in the range of $(\theta_k, x_k, \omega_k, v_k, u) \in [0, 2\pi] \times [-1.5, 1.5] \times [-8, 8] \times [-4, 4] \times [-10, 30]$ for the state space and the control input. We limit our analysis to the MBD-NODE model as FCNN and LSTM models cannot work with time-evolving external input.

Fig. 23 shows the trajectories and the obtained control input for the MPC methods and the MBD-NODE-based MPC method. We can see that the MBD-NODE-based MPC can provide high-accuracy control input and trajectory as the analytic equation of motion-based MPC, which also shows MBD-NODE's strong ability to capture the system dynamics.

3.7 Slider-Crank mechanism

We assessed MBD-NODE's capacity for long-term, high-accuracy predictions using the slider-crank mechanism (Fig. 24). The test involved generating predictions for up to 10,000 time steps (100 s), while encompassing generalization to arbitrary external forces and torques applied to both the slider and crank. We did not include LSTM and FCNN models in the comparison as their inherent structure does not readily accommodate the representation of system dynamics with variable external forces and torques. Additionally, LSTM and FCNN models face challenges in long-term prediction. Their training data requirements and computational costs scale linearly with the time horizon, whereas MBD-NODE's performance depends on the phase space and external inputs, not directly on the time horizon. Previous FCNN- and LSTM-based approaches in related work [11, 20, 25, 54] typically

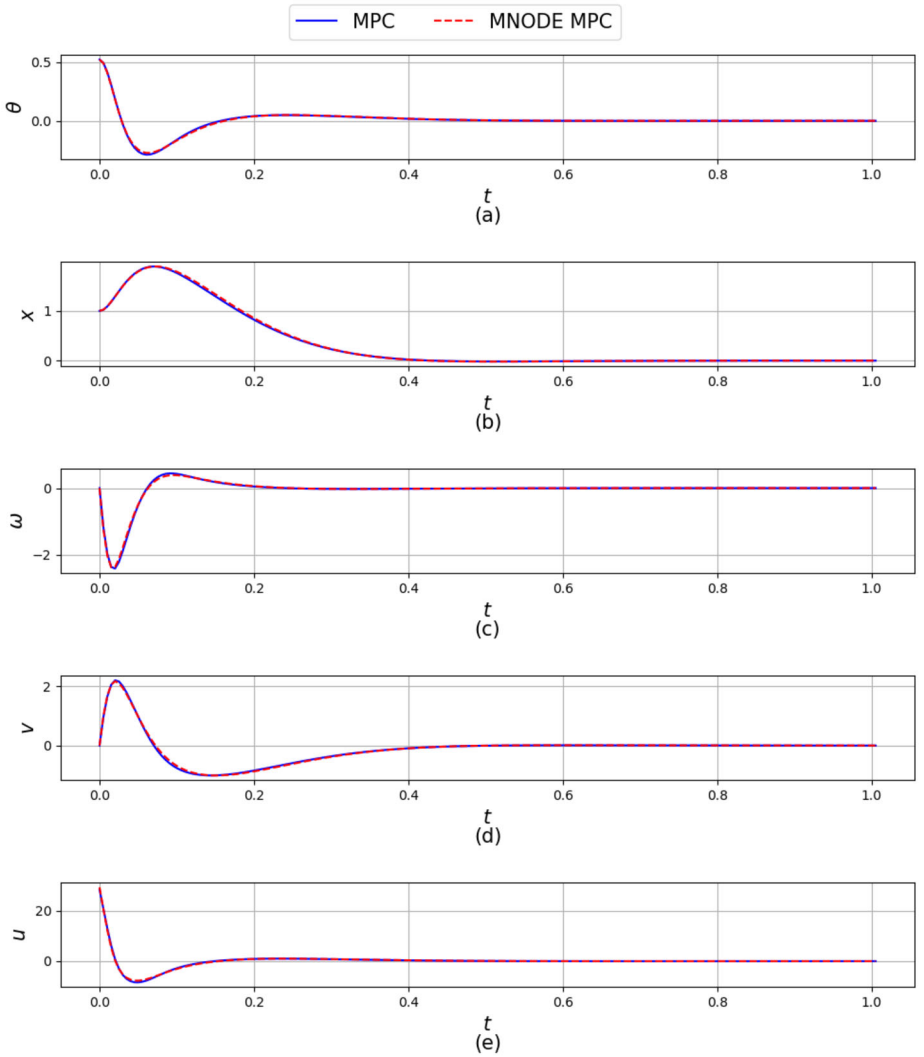


Fig. 23 The trajectory and control input for the MPC methods and the MBD-NODE-based MPC method. (a): the θ vs t; (b): the x vs t; (c): the ω vs t. (d): the v vs t; (e): the u vs t

Fig. 24 The slider-crank system

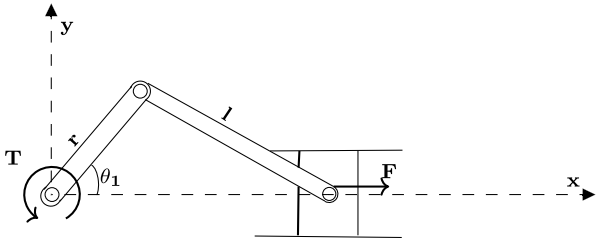


Table 12 Hyperparameters for the slider-crank mechanism

Hyperparameters	Model		
	MBD-NODE	LSTM	FCNN
No. of hidden layers	2	-	-
No. of nodes per hidden layer	256	-	-
Max. epochs	500	-	-
Initial learning rate	1e-3	-	-
Learning rate decay	0.98	-	-
Activation function	Tanh	-	-
Loss function	MSE	-	-
Optimizer	Adam	-	-

demonstrate short-term prediction capabilities, limited to durations of several seconds or hundreds of time steps.

We formulate the slider-crank mechanism as a three-body problem with hard constraints as follows:

1. The crank is connected to ground with a revolute joint with mass $m_1 = 3$ kg, moment of inertia $I_1 = 4\ \mathrm{kg\,m^2}$. The center of mass of the crank in the global reference frame is (x_1, y_1, θ_1), and the length of the crank is $l = 2$ m.
2. The rod is connected to the crank with a revolute joint with mass $m_2 = 6$ kg, moment of inertia $I_2 = 32\ \mathrm{kg\,m^2}$. The center of mass of the rod in the global reference frame is expressed as (x_2, y_2, θ_2), and the length of the connecting rod is $r = 4$ m.
3. The slider is connected to the rod with a revolute joint and constrained to move horizontally with mass $m_3 = 1$ kg and moment of inertia $I_3 = 1\ \mathrm{kg\,m^2}$. The center of mass of the slider in the global reference frame is (x_3, y_3, θ_3).

The generalized coordinates $q = (x_1, y_1, \theta_1, x_2, y_2, \theta_2, x_3, y_3, \theta_3)$ are used to describe the system dynamics. Given q, $\dot{q}$ and some values of the external force/torque (F, T) at a time point, we seek to produce the generalized acceleration $\ddot{q}$, i.e., we have a total of $9 \cdot 2 = 18$ system states and 2 external inputs to describe the system dynamics. Because the slider-crank mechanism is a one-DOF system, we take the minimum coordinates as $(\theta_1, \dot{\theta}_1)$ with the external input (F, T); these four variables fully determine the system dynamics. All other coordinates are treated as dependent coordinates. The detailed formulation is given in Appendix A.

For the training part, we uniformly sampled 10^7 data points as $(\theta_1, \dot{\theta}_1, F, T) \in [0, 2\pi] \times [-4, 4] \times [-10, 10] \times [-10, 10]$ providing the training data. The training used the hyperparameters shown in Table 12. In the testing part, we set the initial condition to be $(\theta_1, \dot{\theta}_1) = (1, 1)$, the simulation time step as 0.01 s, and the external force and torque $(F, T) \sim U[-10, 10] \times U[-10, 10]$ sampled from uniform distribution are applied to the system for each time step; note that there is no requirement for smoothness in F and T, although if one is present, which would only help. We run the prediction for 10,000 steps (100 s) to test the MBD-NODE long-time prediction ability.

Fig. 25 shows the dynamics response of the minimal coordinates $(\theta_1, \dot{\theta}_1)$ under the external force and torque. MBD-NODE accurately predicts the system dynamics for the random external force and torque in the predefined range. Figure 26 shows the dynamics response of the dependent coordinates calculated from the minimal coordinates $(\theta_1, \dot{\theta}_1)$ under the same external force and torque. With the combination of Figs. 26 and 25, we can see that the

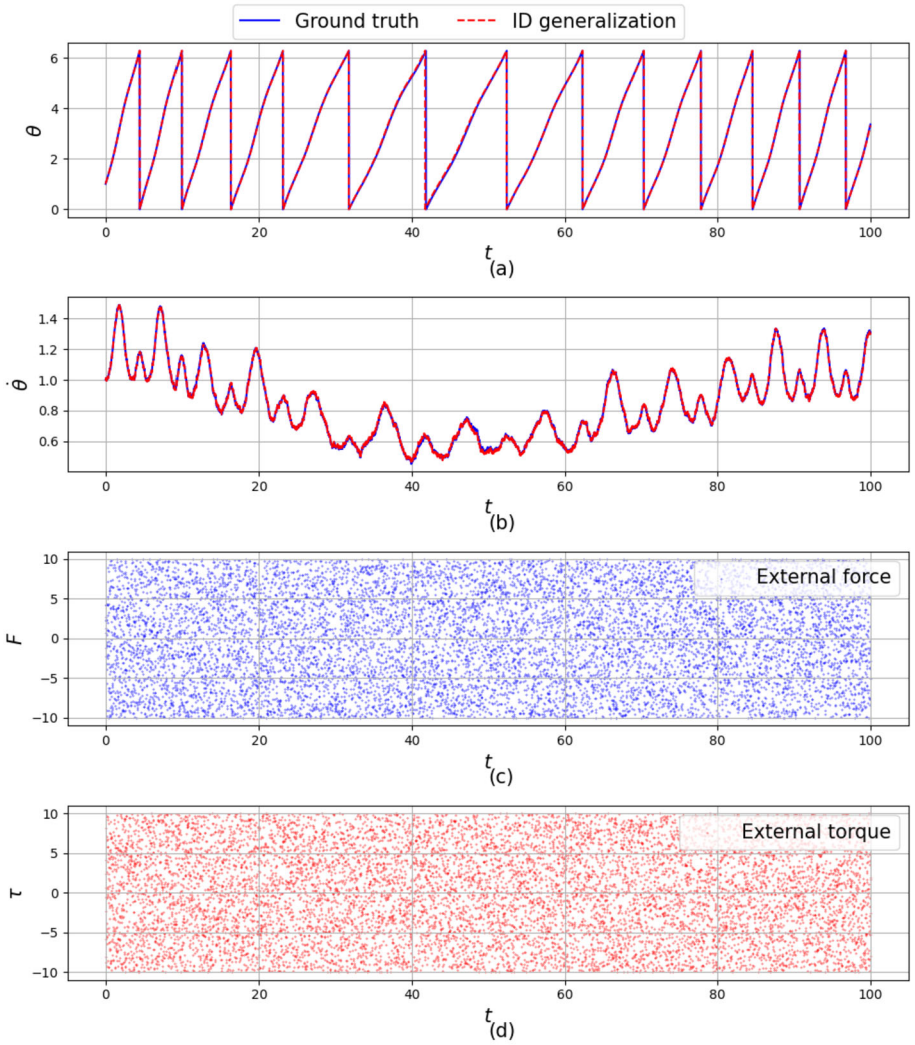


Fig. 25 The dynamics response of the slider-crank mechanism under the external force and torque. (a): the θ_1 vs t; (b): the $\dot{\theta}_1$ vs t; (c): the applied external force F vs t; and (d): the applied external torque T vs t

MBD-NODE provides good-accuracy, long-time prediction for all states. We do not show the coordinates $(y_3, \theta_3, \dot{\theta}_3)$ because they are zeros for all time.

4 Conclusions

Drawing on the NODE methodology, this work introduces MBD-NODE, a method for the data-driven modeling of MBD problems. The performance of MBD-NODE is compared against that of several state-of-the-art data-driven modeling methods by means of seven numerical examples that display attributes encountered in common real-life systems,

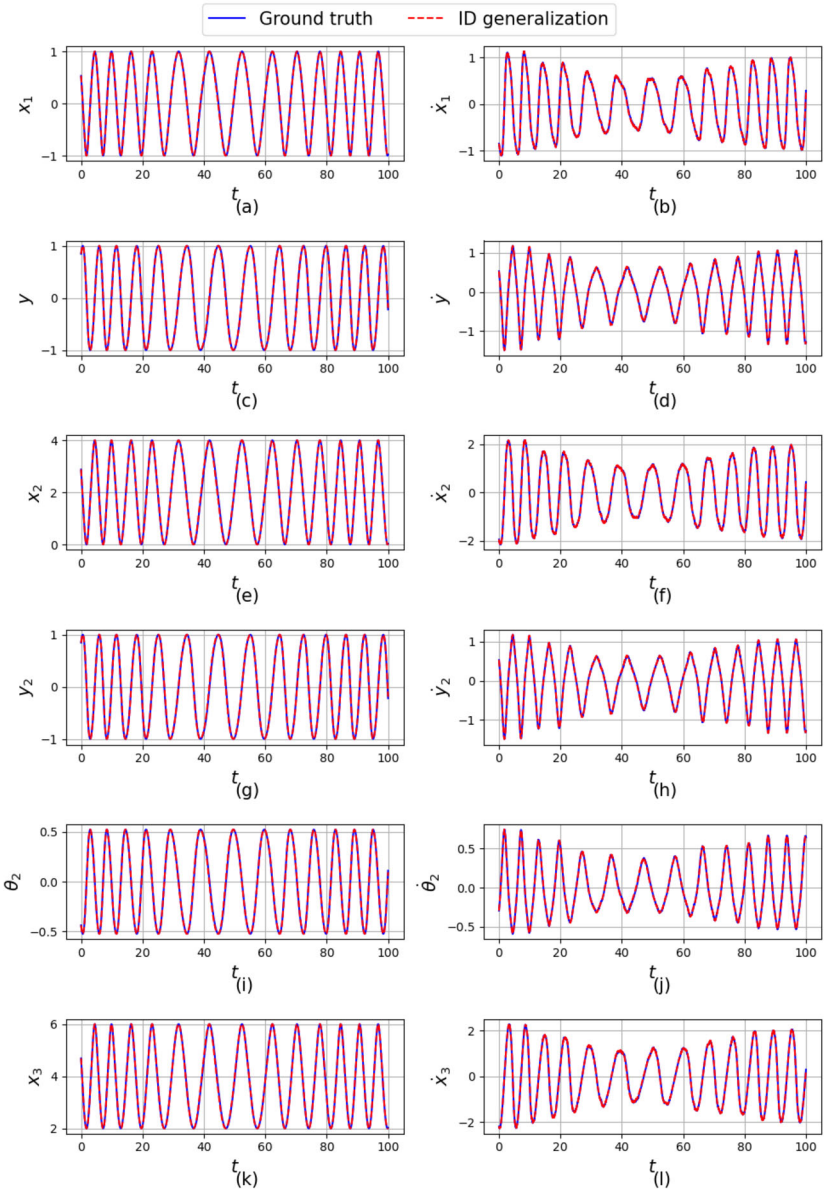


Fig. 26 The dynamics response of the slider-crank mechanism under the external force and torque for 10,000 time steps. (a): the x_1 vs t; (b): the $\dot{x}_1$ vs t; (c): the y_1 vs t; (d): the $\dot{y}_1$ vs t; (e): the x_2 vs t; (f): the $\dot{x}_2$ vs t; (g): the y_2 vs t; (h): the $\dot{y}_2$ vs t; (i): the θ_2 vs t; (j): the $\dot{\theta}_2$ vs t; (k): the x_3 vs t; and (l): the $\dot{x}_3$ vs t

e.g., energy conservation (single mass-spring system), energy dissipation (single mass-spring-damper system), multiscale dynamics (triple mass-spring-damper system), generalization to different parameters (single pendulum system), MPC-based control problem (cart-pole system), chaotic behavior (double pendulum system), and presence of constraints with long time prediction (slider-crank mechanism). The results demonstrate an

overall superior performance of the proposed MBD-NODE method in the following aspects:

1. Generalization Capability: MBD-NODE demonstrates superior accuracy in both in-distribution (ID) and out-of-distribution (OOD) scenarios, a significant advantage over the ID-focused generalization typically observed with FCNN and LSTM models.
2. Model-Based Control Application: The structure of MBD-NODE, mapping system states and external inputs to accelerations, combined with its high generalization accuracy, makes it suitable for model-based control challenges, as demonstrated in the cart-pole control problem.
3. Efficiency in Data Usage and Time Independence: Unlike FCNN, MBD-NODE integration-based learning does not require extensive time-dependent data, enabling accurate long-term dynamics predictions with less data, as demonstrated in the slider-crank problem.
4. Independence from Second-Order Derivative Data: MBD-NODE can predict second-order derivatives based on position and velocity data alone, avoiding the need for direct second-order derivative data required by FCNN and LSTM.

For reproducibility studies, we provided the open-source code base of MBD-NODE, which includes all the numerical examples and all the trained models used in this study [52]. To the best of our knowledge, this represents the first time mechanical system models, and machine learning models are made publicly and unrestrictedly available for reproducibility studies and further research purposes. This can serve as a benchmark testbed for the future development of data-driven modeling methods for multibody dynamics problems.

5 Limitations and future work

The proposed model has several limitations that remain to be addressed in the future. Firstly, although the extrapolation capabilities of MBD-NODE have been tested on several problems in this work and have shown superior performance compared to traditional models like LSTM and FCNN, additional testing will paint a better picture in relation to the out-of-distribution performance of MBD-NODE. Secondly, although MBD-NODE is efficient in terms of data usage and does not rely on second-order derivative data, as detailed in this contribution, other competing methods come with less computational costs. A study to gauge the MBD-NODE trade-off between computational cost and quality of results would be justified and insightful.

Future work should also focus on optimizing the training process to reduce the MBD-NODE computational costs. Another area for improvement is extending MBD-NODE to work with flexible multibody system dynamics problems. Exploring these directions stands to enhance the practical applicability of MBD-NODE and contribute to its broader adoption.

Appendix A: Algorithms for training the MNODE

Algorithm 1 The training algorithm for MBD without constraints

Initialize: Randomly initialized MNODE $f(\cdot, \boldsymbol{\Theta})$; choose integrator Φ

Input: Ground truth trajectories $\mathcal{T} = \{\mathbf{Z}_i\}_{i=0}^{T}$ with parameters $\boldsymbol{\mu}$ and external inputs $\mathbf{u}$, optimizer and its settings

for each epoch $e = 1, 2, \ldots, E$ **do**

  for each time step $i = 0, 1, \ldots, T-1$ **do**

    Prepare input state $\mathbf{Z}_i$ and target state $\mathbf{Z}_{i+1}$

    Forward pass by integrator Φ and $f(\cdot, \boldsymbol{\Theta})$ get the predicted state $\hat{\mathbf{Z}}_{i+1} = \Phi(\mathbf{Z}_i, f, \Delta t_i)$

    Compute loss $\mathrm{L} = \|\mathbf{Z}_{i+1} - \hat{\mathbf{Z}}_{i+1}\|_2^2$

    Backpropagate the loss to compute gradients $\nabla_{\boldsymbol{\Theta}}\mathrm{L}$

    Update the parameters using optimizer: $\boldsymbol{\Theta} = \mathrm{Optimizer}(\boldsymbol{\Theta}, \nabla_{\boldsymbol{\Theta}}\mathrm{L})$

  Decay the learning rate using exponential schedule

Output: Trained MBD $f(\cdot, \boldsymbol{\Theta}^*)$

Algorithm 2 The training algorithm for MBD with constraints equation-based optimization and dependent coordinates data

Initialize: Randomly initialized MNODE $f(\cdot, \boldsymbol{\Theta})$. Choose integrator Φ, identify constraint equation ϕ and the map ϕ^{-1} from the independent/minimal coordinates to the dependent coordinates.

Input: Ground truth trajectories $\mathcal{T} = \{\mathbf{Z}_i\}_{i=0}^{T}$, optimizer and its settings.

for each epoch $e = 1, 2, \ldots, E$ **do**

  for each time step $i = 0, 1, \ldots, T-1$ **do**

    Prepare input state $\mathbf{Z}_i = (\mathbf{Z}_i^M, \mathbf{Z}_i^R)^T \in \mathbf{R}^{2n_z}$ and target state $\mathbf{Z}_{i+1} = (\mathbf{Z}_{i+1}^M, \mathbf{Z}_{\mathbf{i+1}}^{\mathbf{R}})^T \in \mathbf{R}^{2n_z}$

    Forward pass the minimal coordinates $\mathbf{Z}_i^M$ to $f(\cdot, \boldsymbol{\Theta})$ with integrator Φ and get the predicted minimal coordinates at next time step $\hat{\mathbf{Z}}_{i+1}^M = \Phi(\mathbf{Z}_i^M, f, \Delta t)$

    Recover the dependent coordinates $\hat{\mathbf{Z}}_{i+1}^R$ using the independent coordinates $\hat{\mathbf{Z}}_{i+1}^M$ with ϕ^{-1}

    Combine the minimal and dependent coordinates to get the full coordinates $\tilde{\mathbf{Z}}_{i+1} = (\hat{\mathbf{Z}}_{i+1}^M, \hat{\mathbf{Z}}_{i+1}^R)^T$

    Compute loss $\mathrm{L} = \|\mathbf{Z}_{i+1} - \tilde{\mathbf{Z}}_{i+1}\|_2^2$

    Backpropagate the loss to compute gradients $\nabla_{\boldsymbol{\Theta}}\mathrm{L}$

    Update the parameters using optimizer: $\boldsymbol{\Theta} = \mathrm{Optimizer}(\boldsymbol{\Theta}, \nabla_{\boldsymbol{\Theta}}\mathrm{L})$

  Decay the learning rate using exponential schedule

Output: Trained MBD $f(\cdot, \boldsymbol{\Theta}^*)$

Algorithm 3 The training algorithm for MBD; presence of constraints handled by using minimal/independent coordinates; kinematic constraints used to recover the dependent ones

Initialize: Randomly initialized MNODE $f(\cdot, \boldsymbol{\Theta})$; choose integrator Φ; uses pior knowledge of constraint equation ϕ and the minimal coordinates.

Input: Ground truth minimal coordinates trajectories $\mathcal{T} = \{\mathbf{Z}_i^M\}_{i=0}^T$, optimizer and its settings.

for each epoch $e = 1, 2, \ldots, E$ **do**

  for each time step $i = 0, 1, \ldots, T-1$ **do**

    Prepare input state $\mathbf{Z}_i^M$ and target state $\mathbf{Z}_{i+1}^M$

    Forward pass the minimal coordinates $\mathbf{Z}_i^M$ to $f(\cdot, \boldsymbol{\Theta})$ with integrator Φ and to get the predicted minimal coordinates at next time step $\hat{\mathbf{Z}}_{i+1}^M = \Phi(\mathbf{Z}_i^M, f, \Delta t)$

    Compute loss $\mathrm{L} = \|\mathbf{Z}_{i+1}^M - \hat{\mathbf{Z}}_{i+1}^M\|_2^2$

    Backpropagate the loss to compute gradients $\nabla_{\boldsymbol{\Theta}}\mathrm{L}$

    Update the parameters using optimizer: $\boldsymbol{\Theta} = \mathrm{Optimizer}(\boldsymbol{\Theta}, \nabla_{\boldsymbol{\Theta}}\mathrm{L})$

  Decay the learning rate using exponential schedule

Output: Trained MBD $f(\cdot, \boldsymbol{\Theta}^*)$

Appendix B: Training time cost for different models and integrators

Table 13 Time cost for training the models with different integrators. Here the FE1 represents the 1st-order forward Euler, LF2 represents the 2nd-order Leapfrog method, MP2 represents the 2nd-order midpoint method, RK4 represents the 4th-order Runge–Kutta method, YS4 represents the 4th-order Yoshida method, and FK6 represents the 6th-order Fukushima method. Please note that compared with the MNODE and LNN, the second-order derivative data is provided to the HNN

Test case	Model	Integrator	Time cost (s)
Single Mass Spring	MNODE	LF2	507.73
	MNODE	YS4	874.05
	MNODE	FK6	1461.52
	HNN	RK4	218.02
	LNN	RK4	988.45
Single Mass Spring Damper	MNODE	FE1	316.86
	MNODE	MP2	518.09
	MNODE	RK4	1254.13
	FCNN	-	220.13
	LSTM	-	500.63
Triple Mass Spring Damper	MNODE	FE1	358.55
	MNODE	MP2	608.55
	MNODE	RK4	915.68
	FCNN	-	214.13
	LSTM	-	460.33
Single Pendulum	MNODE	FE1	250.52
	MNODE	MP2	276.12
	MNODE	RK4	838.37
	FCNN	-	210.18
	LSTM	-	348.36
Double Pendulum	MNODE	FE1	253.34
	MNODE	MP2	368.75
	MNODE	RK4	854.06
	FCNN	-	176.51
	LSTM	-	402.62
Cart-pole	MNODE	FE1	255.80
	MNODE	MP2	285.04
	MNODE	RK4	776.08
	FCNN	-	214.26
	LSTM	-	358.14

Appendix C: The detail formulation of the equation of motion for the slider-crank mechanism

Following the setting mentioned in Sect. 3.4, the equation of motion for the slider-crank mechanism can be formulated as follows:

The mass matrix $M \in \mathbb{R}^{9\times 9}$ is

$$M = \begin{bmatrix} M_1 & 0_{3\times 3} & 0_{3\times 3} \\ 0_{3\times 3} & M_2 & 0_{3\times 3} \\ 0_{3\times 3} & 0_{3\times 3} & M_3 \end{bmatrix}, \tag{50}$$

where

$$M_1 = \begin{bmatrix} m_1 & 0 & 0 \\ 0 & m_1 & 0 \\ 0 & 0 & I_1 \end{bmatrix} = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 4 \end{bmatrix},$$

$$M_2 = \begin{bmatrix} m_2 & 0 & 0 \\ 0 & m_2 & 0 \\ 0 & 0 & I_2 \end{bmatrix} = \begin{bmatrix} 6 & 0 & 0 \\ 0 & 6 & 0 \\ 0 & 0 & 32 \end{bmatrix},$$

$$M_3 = \begin{bmatrix} m_3 & 0 & 0 \\ 0 & m_3 & 0 \\ 0 & 0 & I_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

The states of the slider-crank mechanism $(x_1, y_1, \theta_1, x_2, y_2, \theta_2, x_3, y_3, \theta_3)$ follows the following constraints $\Phi : \mathbb{R}^9 \to \mathbb{R}^8$ on the position:

$$\Phi(q) = \begin{bmatrix} x_1 - \cos(\theta_1) \\ y_1 - \sin(\theta_1) \\ x_1 + \cos(\theta_1) - x_2 + 2\cos(\theta_2) \\ y_1 + \sin(\theta_1) - y_2 + 2\sin(\theta_2) \\ x_2 + 2\cos(\theta_2) - x_3 \\ y_2 + 2\sin(\theta_2) - y_3 \\ y_3 \\ \theta_3 \end{bmatrix}. \tag{51}$$

We also have the following constraints $\Phi_q \in \mathbb{R}^{8\times 9}$ on the velocit:

$$\Phi_q = \begin{bmatrix} 1 & 0 & \sin(\theta_1) & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -\cos(\theta_1) & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & -\sin(\theta_1) & -1 & 0 & -2\sin(\theta_2) & 0 & 0 & 0 \\ 0 & 1 & \cos(\theta_1) & 0 & -1 & 2\cos(\theta_2) & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & -2\sin(\theta_2) & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2\cos(\theta_2) & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{52}$$

The vector $F_e \in \mathbb{R}^9$ from external forces is

$$F_e = \begin{bmatrix} F_{e1} \\ F_{e2} \\ F_{e3} \end{bmatrix}, \tag{53}$$

where

$$F_{e1} = \begin{bmatrix} 0 \\ 0 \\ T \end{bmatrix} \in \mathbb{R}^3,$$

$$F_{e2} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \in \mathbb{R}^3,$$

$$F_{e3} = \begin{bmatrix} F \\ 0 \\ 0 \end{bmatrix} \in \mathbb{R}^3.$$

We can rearrange the constraint equations on the acceleration:

$$\ddot{x}_1 + \ddot{\theta}_1 \sin(\theta_1) + \dot{\theta}_1^2 \cos(\theta_1) = 0, \tag{54a}$$

$$\ddot{y}_1 - \ddot{\theta}_1 \cos(\theta_1) + \dot{\theta}_1^2 \sin(\theta_1) = 0, \tag{54b}$$

$$\ddot{x}_1 - \ddot{\theta}_1 \sin(\theta_1) - \dot{\theta}_1^2 \cos(\theta_1) - \ddot{x}_2 - 2\ddot{\theta}_2 \sin(\theta_2) - 2\dot{\theta}_2^2 \cos(\theta_2) = 0, \tag{54c}$$

$$\ddot{y}_1 + \ddot{\theta}_1 \cos(\theta_1) - \dot{\theta}_1^2 \sin(\theta_1) - \ddot{y}_2 + 2\ddot{\theta}_2 \cos(\theta_2) - 2\dot{\theta}_2^2 \sin(\theta_2) = 0, \tag{54d}$$

$$\ddot{x}_2 - 2\ddot{\theta}_2 \sin(\theta_2) - 2\dot{\theta}_2^2 \cos(\theta_2) - \ddot{x}_3 = 0, \tag{54e}$$

$$\ddot{y}_2 + 2\ddot{\theta}_2 \cos(\theta_2) - 2\dot{\theta}_2^2 \sin(\theta_2) - \ddot{y}_3 = 0, \tag{54f}$$

$$\dot{\theta}_3 = 0, \tag{54g}$$

$$\ddot{\theta}_3 = 0 \tag{54h}$$

to get the γ_c as

$$\gamma_c = \begin{bmatrix} -\dot{\theta}_1^2 \cos(\theta_1) \\ -\dot{\theta}_1^2 \sin(\theta_1) \\ \dot{\theta}_1^2 \cos(\theta_1) + 2\dot{\theta}_2^2 \cos(\theta_2) \\ \dot{\theta}_1^2 \sin(\theta_1) + 2\dot{\theta}_2^2 \sin(\theta_2) \\ 2\dot{\theta}_2^2 \cos(\theta_2) \\ 2\dot{\theta}_2^2 \sin(\theta_2) \\ 0 \\ 0 \end{bmatrix}. \tag{55}$$

Finally, we plug the above equations into Eq. (1) to get the equation of motion for the slider-crank mechanism.

Author contributions J.W.(Jingquan Wang) wrote the main manuscript text. S.W. and H.U. prepared figures 1,2,3,6,9,12,15. S.W., H.U., J.W.(Jinlong Wu), and D.N. helped polish the manuscript. All authors reviewed the manuscript.

Funding This work was carried out in part with support from National Science Foundation project CMMI2153855.

Data Availability No datasets were generated or analysed during the current study.

Declarations

Competing interests The authors declare no competing interests.

References

1. Bacsa, K., Lai, Z., Liu, W., Todd, M., Chatzi, E.: Symplectic encoders for physics-constrained variational dynamics inference. Sci. Rep. **13**(1), 2643 (2023)
2. Bauchau, O.A., Laulusa, A.: Review of contemporary approaches for constraint enforcement in multibody systems. J. Comput. Nonlinear Dyn. **3**(1), 011005 (2007)
3. Beucler, T., Pritchard, M., Rasp, S., Ott, J., Baldi, P., Gentine, P.: Enforcing analytic constraints in neural networks emulating physical systems. Phys. Rev. Lett. **126**(9), 098302 (2021)
4. Bhattoo, R., Ranu, S., Krishnan, N.A.: Learning the dynamics of particle-based systems with Lagrangian graph neural networks. Mach. Learn.: Sci. Technol. **4**(1), 015003 (2023)
5. Bishnoi, S., Bhattoo, R., Ranu, S., Krishnan, N.M.: Enhancing the inductive biases of graph neural ODE for modeling dynamical systems (2022). arXiv preprint arXiv:2209.10740
6. Borovykh, A., Oosterlee, C.W., Bohté, S.M.: Generalization in fully-connected neural networks for time series forecasting. J. Comput. Sci. **36**, 101020 (2019)
7. Chen, R.T.Q., Rubanova, Y., Bettencourt, J., Duvenaud, D.K.: Neural ordinary differential equations. In: Advances in Neural Information Processing Systems, vol. 31 (2018)
8. Chen, Z., Zhang, J., Arjovsky, M., Bottou, L.: Symplectic recurrent neural networks (2020)
9. Chen, Y., Matsubara, T., Yaguchi, T.: Neural symplectic form: learning Hamiltonian equations on general coordinate systems. In: Beygelzimer, A., Dauphin, Y., Liang, P., Vaughan, J.W. (eds.) Advances in Neural Information Processing Systems (2021)
10. Chen, X., Araujo, F.A., Riou, M., Torrejon, J., Ravelosona, D., Kang, W., Zhao, W., Grollier, J., Querlioz, D.: Forecasting the outcome of spintronic experiments with neural ordinary differential equations. Nat. Commun. **13**(1), 1016 (2022)
11. Choi, H.S., An, J., Han, S., Kim, J.G., Jung, J.Y., Choi, J., Orzechowski, G., Mikkola, A., Choi, J.H.: Data-driven simulation for general-purpose multibody dynamics using deep neural networks. Multibody Syst. Dyn. **51**(4), 419–454 (2021)
12. Cranmer, M.D., Greydanus, S., Hoyer, S., Battaglia, P.W., Spergel, D.N., Ho, S.: Lagrangian neural networks (2020). CoRR arXiv:2003.04630
13. Daems, R., Taets, J., Crevecoeur, G., et al.: Keycld: learning constrained Lagrangian dynamics in keypoint coordinates from images (2022). arXiv preprint arXiv:2206.11030
14. DiPietro, D.M., Xiong, S., Zhu, B.: Sparse symplectically integrated neural networks. In: Advances in Neural Information Processing Systems, vol. 34 (2020)
15. Djeumou, F., Neary, C., Goubault, E., Putot, S., Topcu, U.: Neural networks with physics-informed architectures and constraints for dynamical systems modeling. In: Firoozi, R., Mehr, N., Yel, E., Antonova, R., Bohg, J., Schwager, M., Kochenderfer, M. (eds.) Proceedings of the 4th Annual Learning for Dynamics and Control Conference. Proceedings of Machine Learning Research, vol. 168, pp. 263–277 (2022)
16. Finlay, C., Jacobsen, J.-H., Nurbekyan, L., Oberman, A.: How to train your neural ODE: the world of Jacobian and kinetic regularization. In: Daumé, H.III, Singh, A. (eds.) Proceedings of the 37th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 119, pp. 3154–3164 (2020)
17. Finzi, M., Wang, K.A., Wilson, A.G.: Simplifying Hamiltonian and Lagrangian neural networks via explicit constraints. Adv. Neural Inf. Process. Syst. **33**, 13880–13889 (2020)
18. Fortin, M., Glowinski, R.: Augmented Lagrangian Methods: Applications to the Numerical Solution of Boundary-Value Problems. Elsevier, Amsterdam (2000)
19. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: Teh, Y.W., Titterington, M. (eds.) Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, Chia Laguna Resort, Sardinia, Italy 13–15 May 2010. Proceedings of Machine Learning Research, vol. 9, pp. 249–256 (2010). PMLR. Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010
20. Go, M.S., Han, S., Lim, J.H., Kim, J.G.: An efficient fixed-time increment-based data-driven simulation for general multibody dynamics using deep neural networks. In: Engineering with Computers (2023)
21. Greydanus, S., Dzamba, M., Yosinski, J.: Hamiltonian neural networks. In: Advances in Neural Information Processing Systems, vol. 32 (2019)

22. Grunbacher, S., Hasani, R., Lechner, M., Cyranka, J., Smolka, S.A., Grosu, R.: On the verification of neural ODEs with stochastic guarantees. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35, pp. 11525–11535 (2021)
23. Gruver, N., Finzi, M., Stanton, S., Wilson, A.G.: Deconstructing the inductive biases of Hamiltonian neural networks (2022). arXiv preprint arXiv:2202.04836
24. Gupta, J.K., Menda, K., Manchester, Z., Kochenderfer, M.J.: A general framework for structured learning of mechanical systems (2019). CoRR arXiv:1902.08705
25. Han, S., Choi, H.-S., Choi, J., Choi, J.H., Kim, J.-G.: A DNN-based data-driven modeling employing coarse sample data for real-time flexible multibody dynamics simulations. Comput. Methods Appl. Mech. Eng. **373**, 113480 (2021)
26. Hashemi, A., Orzechowski, G., Mikkola, A., McPhee, J.: Multibody dynamics and control using machine learning. Multibody Syst. Dyn. **58**, 1–35 (2023)
27. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: surpassing human-level performance on ImageNet classification. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 1026–1034 (2015)
28. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
29. Koutsoupakis, J., Giagopoulos, D.: Drivetrain response prediction using AI-based surrogate and multibody dynamics model. Machines **11**(5), 514 (2023)
30. Kouvaritakis, B., Cannon, M.: Model Predictive Control, vol. 38, pp. 13–56. Springer, Switzerland (2016)
31. Kraft, S., Causse, J., Martinez, A.: Black-box modelling of nonlinear railway vehicle dynamics for track geometry assessment using neural networks. Int. J. Veh. Mech. and Mobil. **57**(9), 1241–1270 (2019)
32. Lee, K., Parish, E.J.: Parameterized neural ordinary differential equations: applications to computational physics problems. Proc. R. Soc. A **477**(2253), 20210162 (2021)
33. Lim, Y.H., Kasim, M.F.: Unifying physical systems' inductive biases in neural ODE using dynamics constraints. In: ICML 2022 2nd AI for Science Workshop (2022)
34. Liu, G.-H., Chen, T., Theodorou, E.: Second-order neural ODE optimizer. Adv. Neural Inf. Process. Syst. **34**, 25267–25279 (2021)
35. Lu, L., Pestourie, R., Yao, W., Wang, Z., Verdugo, F., Johnson, S.G.: Physics-informed neural networks with hard constraints for inverse design. SIAM J. Sci. Comput. **43**(6), B1105–B1132 (2021)
36. Lutter, M., Ritter, C., Peters, J.: Deep Lagrangian networks: using physics as model prior for deep learning. In: International Conference on Learning Representations (ICLR 2019) (2019). OpenReview.net
37. Marco, D., Méhats, F.: Symplectic learning for Hamiltonian neural networks. J. Comput. Phys. **494**, 112495 (2023)
38. Matsubara, T., Miyatake, Y., Yaguchi, T.: Symplectic adjoint method for exact gradient of neural ODE with minimal memory. In: Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P.S., Vaughan, J.W. (eds.) Advances in Neural Information Processing Systems, vol. 34, pp. 20772–20784. Curran Associates, Red Hook (2021)
39. Norcliffe, A., Bodnar, C., Day, B., Simidjievski, N., Lió, P.: On second order behaviour in augmented neural odes. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M.F., Lin, H. (eds.) Advances in Neural Information Processing Systems, vol. 33, pp. 5911–5921. Curran Associates, Red Hook (2020)
40. Noren, H., Eidnes, S., Celledoni, E.: Learning dynamical systems from noisy data with inverse-explicit integrators (2023)
41. Nwankpa, C., Ijomah, W.L., Gachagan, A., Marshall, S.: Activation functions: comparison of trends in practice and research for deep learning (2018). arXiv:1811.03378
42. Owoyele, O., Pal, P.: ChemNODE: a neural ordinary differential equations framework for efficient chemical kinetic solvers. Energy AI **7**, 100118 (2022)
43. Pan, Y., Nie, X., Li, Z., Gu, S.: Data-driven vehicle modeling of longitudinal dynamics based on a multibody model and deep neural networks. Measurement **180**, 109541 (2021)
44. Portwood, G.D., Mitra, P.P., Ribeiro, M.D., Nguyen, T.M., Nadiga, B.T., Saenz, J.A., Chertkov, M., Garg, A., Anandkumar, A., Dengel, A., et al.: Turbulence forecasting via neural ode (2019). arXiv preprint arXiv:1911.05180
45. Quaglino, A., Gallieri, M., Masci, J., Koutník, J.: SNODE: spectral discretization of neural ODEs for system identification. In: International Conference on Learning Representations (2019)
46. Sanchez-Gonzalez, A., Bapst, V., Cranmer, K., Battaglia, P.: Hamiltonian graph networks with ODE integrators (2019). arXiv preprint arXiv:1909.12790
47. Shabana, A.A.: Dynamics of Multibody Systems. Cambridge university press, Cambridge (2020)
48. Sorourifar, F., Peng, Y., Castillo, I., Bui, L., Venegas, J., Paulson, J.A.: Physics-enhanced neural ordinary differential equations: application to industrial chemical reaction systems. Ind. Eng. Chem. Res. **62**(38), 15563–15577 (2023)

49. Sosanya, A., Greydanus, S.: Dissipative Hamiltonian neural networks: learning dissipative and conservative dynamics separately (2022). arXiv:2201.10085
50. Steger, S., Rohrhofer, F.M., Geiger, B.: How PINNs cheat: predicting chaotic motion of a double pendulum. In: The Symbiosis of Deep Learning and Differential Equations II @ the 36th Neural Information Processing Systems (NeurIPS) Conference (2022)
51. Toth, P., Rezende, D.J., Jaegle, A., Racanière, S., Botev, A., Higgins, I.: Hamiltonian generative networks. In: International Conference on Learning Representations (2019)
52. Wang, J., Wang, S., Unjhawala, H., Wu, J., Negrut, D.: Models, scripts, and meta-data: physics-informed data-driven modeling and simulation of constrained multibody systems (2024). https://github.com/uwsbel/sbel-reproducibility/tree/master/2024/MNODE-code
53. Wehage, R.A., Haug, E.J.: Generalized coordinate partitioning for dimension reduction in analysis of constrained dynamic systems. J. Mech. Des. **104**, 247–255 (1982)
54. Ye, Y., Huang, P., Sun, Y., Mbsnet, D.S.: A deep learning model for multibody dynamics simulation and its application to a vehicle-track system. Mech. Syst. Signal Process. **157**, 107716 (2021)
55. Zhang, H., Gao, X., Unterman, J., Arodz, T.: Approximation capabilities of neural ODEs and invertible residual networks. In: Daumé, H.III, Singh, A. (eds.) Proceedings of the 37th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 119, pp. 11086–11095 (2020)
56. Zhong, Y.D., Dey, B., Chakraborty, A.: Dissipative symODEN: encoding Hamiltonian dynamics with dissipation and control into deep learning. In: ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations (2019)
57. Zhong, Y.D., Dey, B., Chakraborty, A.: Extending Lagrangian and Hamiltonian neural networks with differentiable contact models. In: Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P.S., Vaughan, J.W. (eds.) Advances in Neural Information Processing Systems, vol. 34, pp. 21910–21922. Curran Associates, Red Hook (2021)
58. Zhuang, J., Dvornek, N., Li, X., Tatikonda, S., Papademetris, X., Duncan, J.: Adaptive checkpoint adjoint method for gradient estimation in neural ODE. In: Daumé, H.III, Singh, A. (eds.) Proceedings of the 37th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 119, pp. 11639–11649 (2020)