Noname manuscript No.
(will be inserted by the editor)

FNODE: Flow-Matching for data-driven simulation of constrained multibody systems

Hongyu Wang · Jingquan Wang · Dan Negrut

Received: date / Accepted: date

Abstract Data-driven modeling of constrained multibody systems faces two persistent challenges: high computational cost and limited long-term prediction accuracy. To address these issues, we introduce the Flow-Matching Neural Ordinary Differential Equation (FNODE), a framework that learns acceleration vector fields directly from trajectory data. By reformulating the training objective to supervise accelerations rather than integrated states, FNODE eliminates the need for backpropagation through an ODE solver, which represents a bottleneck in traditional Neural ODEs. Acceleration targets are computed efficiently using numerical differentiation techniques, including a hybrid Fast Fourier Transform (FFT) and Finite Difference (FD) scheme. We evaluate FNODE on a diverse set of benchmarks, including the single and triple mass-spring-damper systems, double pendulum, slider-crank, and cart-pole. Across all cases, FNODE consistently outperforms existing approaches such as Multi-Body Dynamic Neural ODE (MBD-NODE), Long Short-Term Memory (LSTM) networks, and Fully Connected Neural Networks (FCNN), demonstrating good accuracy, generalization, and computational efficiency.

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

Hongyu Wang
Department of Electrical and Computer Engineering, University of Wisconsin-Madison, 1415 Engineering Dr, 53706, Madison, USA
E-mail: hwang2487@wisc.edu

Jingquan Wang
Department of Mechanical Engineering, University of Wisconsin-Madison, 1513 University Avenue, 53706, Madison, USA
E-mail: jwang2373@wisc.edu

Dan Negrut
Department of Mechanical Engineering, University of Wisconsin-Madison, 1513 University Avenue, 53706, Madison, USA
E-mail: negrut@wisc.edu

Mathematics Subject Classification (2020) 34C60 · 37M05

1 Introduction

Multibody dynamics simulation is relevant in engineering design and scientific computation, with applications in numerous fields, e.g., robotics [1], vehicle and train dynamics [2], biomechanics [3], and molecular dynamics [4], to name a few. Traditionally, modeling multibody systems has relied on first-principles approaches such as Newton-Euler [5], Lagrangian [6], or Hamiltonian mechanics [7]. While these classical methods provide accurate and interpretable models, they are challenged in scenarios where the underlying dynamics are partially unknown, highly complex, or subject to unmodeled effects such as friction, contact forces, or material nonlinearities [8].

Machine learning has opened the door to data-driven simulation of dynamical systems, offering the potential to learn complex behaviors directly from observational data [9,10]. Early approaches in this domain employed classical machine learning architectures such as Long Short-Term Memory (LSTM) networks [11,12] and Fully Connected Neural Networks (FCNNs) [13] to learn discrete-time mappings from current states to future states. While these methods have shown promise in short-term prediction tasks, they show limited out-of-distribution generalization. Specifically, these black-box predictors learn fixed-timestep transitions $f : s_t \mapsto s_{t+\Delta t}$, which not only limits their temporal resolution but also causes error accumulation that grows exponentially with the prediction horizon [14]. Furthermore, such discrete-time approaches often fail to capture the continuous nature of physical dynamics or enforce fundamental conservation laws of conservative systems [15].

Neural Ordinary Differential Equations (NODEs) [16] marked a major step in continuous-time learning by representing system dynamics as neural vector fields integrated with standard ODE solvers. This formulation handles irregular time sampling and directly connects learning with established numerical integration theory [17]. This approach has been successfully applied to various physical systems, with extensions including Hamiltonian Neural Networks [18], Lagrangian Neural Networks [19], and constrained formulations for systems with known invariants [20]. Despite their strengths, NODEs suffer from a bottleneck: the adjoint sensitivity method for backpropagation requires solving an additional backward ODE, which increases computational cost and memory use, and limits scalability for stiff or high-dimensional systems [21,22].

Recent advances in flow-matching algorithms [23,24] show that vector fields can be learned by directly supervising derivatives at sampled points, avoiding backward ODE solves [25]. Applying this idea to dynamical systems, we train neural networks to approximate accelerations directly, eliminating costly integration in the learning loop [26]. We propose Flow-Matching Neural Ordinary Differential Equations (FNODE), a framework for efficient data-driven modeling of multibody systems. Instead of predicting future states via time integration, FNODE trains a neural network to learn directly the acceleration

vector field – the natural quantity in mechanical systems governed by force laws [27]. This reformulation removes the need for ODE solvers in backpropagation, improving computational efficiency while preserving accuracy. By learning accelerations directly, the model captures the instantaneous dynamics of the system, leading to better generalization and more stable long-term predictions. Moreover, FNODE naturally accommodates various numerical differentiation schemes for computing acceleration targets from position data, including finite differences and spectral methods, allowing practitioners to balance between noise robustness and accuracy based on their data characteristics [28].

Directly learning accelerations requires accurate derivative information, which can be difficult to obtain from noisy trajectory data. To address this, we use a hybrid scheme that combines the accuracy of Fast Fourier Transform (FFT)-based spectral differentiation [29] with the stability of finite differences (FD) near trajectory boundaries. Our numerical experiments show that, with proper differentiation techniques, the advantages of acceleration-based learning outweigh the challenges of derivative estimation.

This paper makes the following contributions:

1. We introduce FNODE, an approach for data-driven simulation of dynamical systems that learns acceleration vector fields directly from trajectory data. This removes the backpropagation bottleneck of conventional NODEs and reduces training cost while preserving accuracy.
2. We present a numerical differentiation framework for constructing high-quality acceleration targets, combining FFT-based spectral differentiation with finite differences to balance accuracy and stability.
3. We evaluate FNODE on a diverse set of multibody dynamics benchmarks, showing consistent improvements over baselines—including Neural ODEs, Hamiltonian Neural Networks, LSTMs, and FCNNs—in long-term prediction accuracy, out-of-distribution generalization, and computational efficiency.
4. To support reproducibility, we provide an open-source implementation of FNODE together with benchmark problems, implementations of baseline methods, data generation tools, and evaluation protocols.

2 Methodology

2.1 The Multibody Dynamics Problem

We employ a MBD formulation in redundant generalized coordinates wherein Lagrange multipliers are used to incorporate constraint equations directly into the equations of motion [2].

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

Above, the mass matrix $\mathbf{M}$ encodes the system's inertia, while $\mathbf{\Phi_q}$ is the Jacobian of the kinematic constraints. The generalized coordinates are collected

in $\mathbf{q}$, with $\ddot{\mathbf{q}}$ denoting accelerations. The Lagrange multipliers λ correspond to the constraint forces. The right-hand side includes $\mathbf{F_e}$, the vector of external and velocity-dependent forces, and $\gamma_\mathbf{c}$, the contribution from the second time derivative of the constraints.

2.2 Neural Ordinary Differential Equations for Multibody System Dynamics

2.2.1 Neural Ordinary Differential Equation (NODE)

For a general dynamical system, the NODE framework assumes the existence of an underlying continuous-time process that governs the evolution of the system states. To that end, we assume a hidden state $\mathbf{z}(t) \in \mathbb{R}^{n_z}$ with dynamics

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

where $f : \mathbb{R}^{n_z} \times \mathbb{R}^+ \to \mathbb{R}^{n_z}$ is a neural network parameterized by $\boldsymbol{\Theta}$. The future state $\mathbf{z}(t)$ is obtained by solving the initial value problem

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

with Φ denoting the chosen numerical ODE solver. The parameters $\boldsymbol{\Theta}$ are identified from trajectory data $\mathbf{z}(t_i)_{i=1}^n$. Since time steps need not be uniform, adaptive and variable-step integrators can be used, which is particularly advantageous in multibody dynamics where time scales vary widely.

2.2.2 Flow-Matching in Neural Ordinary Differential Equation

FNODE operates on an augmented state space defined as $\boldsymbol{Z}(t) = (\boldsymbol{z}(t), \dot{\boldsymbol{z}}(t))^T$ that is used to model the acceleration directly:

$$\ddot{\boldsymbol{z}}(t, \boldsymbol{\mu}) = f(\boldsymbol{Z}(t, \boldsymbol{\mu});\ \boldsymbol{\Theta})\ , \tag{4}$$

where the initial conditions for positions and velocities are defined as:

$$\boldsymbol{Z}(0, \boldsymbol{\mu}) = (\boldsymbol{z}^T(0, \boldsymbol{\mu}), \dot{\boldsymbol{z}}^T(0, \boldsymbol{\mu}))^T\ , \tag{5}$$

and the parameter vector capturing problem-specific characteristics such as material properties, geometric parameters, or external loading conditions is

$$\boldsymbol{\mu} = (\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_{n_{\boldsymbol{\mu}}})^T \in \mathbb{R}^{n_{\boldsymbol{\mu}}}\ .$$

The generalized coordinates and velocity are

$$\boldsymbol{z}(t, \boldsymbol{\mu}) = (\boldsymbol{z}^1(t, \boldsymbol{\mu}), ..., \boldsymbol{z}^{n_z}(t, \boldsymbol{\mu}))^T \in \mathbb{R}^{n_z}, \tag{6}$$

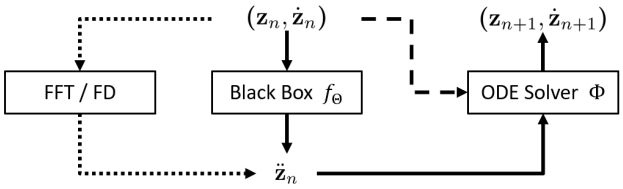

Fig. 1 The discretized forward pass for FNODE for general MBD

$$\dot{\boldsymbol{z}}(t, \boldsymbol{\mu}) = (\dot{\boldsymbol{z}}^1(t, \boldsymbol{\mu}), \ldots, \dot{\boldsymbol{z}}^{n_z}(t, \boldsymbol{\mu}))^T \in \mathbb{R}^{n_z} \ . \tag{7}$$

The neural network function processes this enriched input space that is modulated by the parameter vector $\boldsymbol{\mu}$ to predict acceleration values directly

$$f : \mathbb{R}^{2n_z} \times \mathbb{R}^{n_{\boldsymbol{\mu}}} \to \mathbb{R}^{n_z} \ . \tag{8}$$

In inference phase, the forward integration process maintains the same mathematical structure as traditional NODEs with the definition from Eq. (3) and Eq. (4).

$$\boldsymbol{Z}(t, \boldsymbol{\mu}) = \boldsymbol{Z}(0, \boldsymbol{\mu}) + \int_0^t f(\boldsymbol{Z}(\tau, \boldsymbol{\mu});\ \boldsymbol{\Theta}) d\tau = \Phi(\boldsymbol{Z}(0, \boldsymbol{\mu}), f, t). \tag{9}$$

Figure 1 shows the discretized forward pass of FNODE. The network maps the system state to accelerations using the three-layer architecture in Table 1, with Tanh/ReLU activations and Xavier/Kaiming initialization.

Table 1 MBD-NODE Architecture

Layer	Number of Neurons	Activation Function	Initialization
Input Layer	$2n_z$	[Tanh,ReLU]	[Xavier, Kaiming]
Hidden Layer 1	d_{width}	[Tanh,ReLU]	[Xavier, Kaiming]
Hidden Layer 2	d_{width}	[Tanh,ReLU]	[Xavier, Kaiming]
Output Layer	n_z	-	[Xavier, Kaiming]

2.3 Calculation of Acceleration

Two primary approaches are employed depending on the characteristics of the available dataset: Fast Fourier Transform (FFT) spectral differentiation and Finite Difference (FD) methods.

2.3.1 FFT Spectral Differentiation

For dynamic systems which satisfies periodicity assumptions, FFT-based spectral differentiation provides is known for high accuracy in time gradient calculation. Given a periodic function $z(t)$ with time interval $L = N\Delta t$ and uniform time step Δt, the Fourier series representation is:

$$\boldsymbol{z}(t) = \sum_{k=-\infty}^{\infty} \boldsymbol{Z}_k e^{2\pi i k t/L} \ . \tag{10}$$

The time gradient computed by the spectral representation assumes then the expression:

$$\frac{d\boldsymbol{z}}{dt} = \sum_{k=-\infty}^{\infty} \frac{2\pi i k}{L} \boldsymbol{Z}_k e^{2\pi i k t/L} \ . \tag{11}$$

Due to aliasing errors and Gibbs phenomenon, the results may contain spurious high-frequency oscillations. The Gibbs error arises from discontinuity in periodical expansion:

$$\Delta_{boundary} = z(N-1) - z(0) \neq 0 \ . \tag{12}$$

When the boundary discontinuity of magnitude Δ_{boundary} exists as defined in Eq. (12), the Fourier coefficients exhibit asymptotic behavior

$$Z(k) \sim \frac{\Delta_{boundary}}{2\pi i k/N} \ . \tag{13}$$

The derivative operation multiplies each Fourier coefficient in Eq. (13) by $\frac{2\pi i k}{N\Delta t}$, resulting in

$$Z'(k) = \frac{2\pi i k}{N\Delta t} \cdot Z(k) \sim \frac{\Delta_{boundary}}{2\Delta t} \ . \tag{14}$$

This creates a constant-amplitude error across all high frequency components as shown in Eq. (14), leading to a total Gibbs error

$$\epsilon_{Gibbs} \approx \frac{|\Delta_{boundary}|}{2\Delta t} \ . \tag{15}$$

The error is independent of N but inversely proportional to the time step, which means it would be particularly problematic for fine temporal resolution. To solve the Gibbs phenomenon problem described in Eq. (15), *least-square detrending* is applied

$$\ell(t) \ = \ a + bt, \quad \begin{bmatrix} a & b \end{bmatrix} \ = \ \underset{a,b}{\operatorname{argmin}} \ \sum_i \big(z_i - (a + b\, t_i)\big)^2 \ , \tag{16}$$

where $\ell(t)$ is the linear trend function to approximate the trend of the raw dataset. The detrended dataset is formulated using Eq.(16) as

$$z_d(t) \ = \ z(t) - \ell(t) \ . \tag{17}$$

The new discontinuity value is formulated from Eq. (17) as

$$\Delta_{new} \;=\; z_d(N-1) - z_d(0) \;=\; \Delta_{\text{boundary}} - b\,L\;. \tag{18}$$

Even though the detrend narrows the gap of the two ends of dataset as described in Eq. (18), the first order derivative of the dataset is still not continuous. In order to further reduce Gibbs error while maintain the spectrum information as mush as possible, we adopt *mirror reflection* and *cosine taper*.

$$z_e(t) = \begin{cases} \tau_L(t)\, z_d(-t), & -M \le t < 0, \\ z_d(t), & 0 \le t \le N, \\ \tau_R(t)\, z_d(2N-t), & N < t \le N+M, \end{cases} \qquad M = \frac{N}{4}, \tag{19}$$

$$\tau(t) = \tfrac{1}{2}\left(1 - \cos\left(\frac{\pi t}{M}\right)\right), \quad t = 0, 1, \ldots, M.$$

Above, $\tau(i)$ is the cosine window weight and M is the length of Mirror Reflection. The amplitude of the dataset can be smoothly transited through this process. A Tukey window is then applied, tapering the signal to zero at both ends so that it appears periodic and suitable for FFT-based differentiation:

$$w(t) \;=\; \begin{cases} \dfrac{1}{2}\Big[1 + \cos\Big(\pi\,\dfrac{\alpha L/2 - t}{\alpha L/2}\Big)\Big], & 0 \le t < \frac{\alpha L}{2}, \\ 1, & \frac{\alpha L}{2} \le t \le L - \frac{\alpha L}{2}, \quad \alpha = 0.2 \\ \dfrac{1}{2}\Big[1 + \cos\Big(\pi\,\dfrac{t - (L - \alpha L/2)}{\alpha L/2}\Big)\Big], & L - \frac{\alpha L}{2} < t \le L, \end{cases} \tag{20}$$

Detrending, mirror reflection, and Tukey windowing smooth structural discontinuities, but FFT differentiation still amplifies residual noise and high-frequency artifacts. To suppress these effects, we apply a Gaussian low-pass filter in the frequency domain, defined as

$$G(k) \;=\; \exp\Big[-\tfrac{1}{2}\Big(\tfrac{k}{\sigma}\Big)^2\Big], \quad \sigma \;\approx\; \frac{N}{20}\;, \tag{21}$$

where k is the frequency index and σ controls the filter bandwidth. The choice of $\sigma \approx N/20$ ensures that the filter effectively suppresses high-frequency noise while preserving the essential spectral content of the signal.

The complete FFT-based differentiation process is formulated as follows. First, we compute the discrete Fourier transform (DFT) of the preprocessed signal $\mathbf{z}_e[n]$ with the applied window function $w[n]$ from Eq. (20):

$$\mathbf{Z}_e(k) = \sum_{n=0}^{N-1} \mathbf{z}_e[n]\, w[n]\, e^{-j2\pi kn/N}\;. \tag{22a}$$

Next, we apply the differentiation operator in the frequency domain, together with the Gaussian filter $G(k)$ from Eq. (21):

$$\mathbf{Z}'_e(k) = j\omega_k\, \mathbf{Z}_e(k)\, G(k)\;, \tag{22b}$$

where $\omega_k = \frac{2\pi k}{N\Delta t}$ is the angular frequency. Finally, the filtered time-domain derivative is obtained by the inverse FFT:

$$\dot{\mathbf{z}}[n] = \mathcal{F}^{-1}\big[\mathbf{Z}'_e(k)\big] = \frac{1}{N}\sum_{k=-N/2}^{N/2-1} \mathbf{Z}'_e(k)\, e^{j2\pi kn/N} \ . \tag{22c}$$

This approach combines spectral differentiation with noise suppression, yielding smooth and accurate derivative estimates for FNODE training.

2.3.2 Finite Difference Method

For non-periodic datasets, finite difference (FD) schemes provide a simple and robust alternative to FFT-based differentiation. Using Taylor expansions, the first derivative can be approximated by [30]:

$$z'(t) \approx \frac{z(t+\Delta t) - z(t-\Delta t)}{2\Delta t} \quad \text{(central, second order)} \ , \tag{23a}$$

with forward and backward differences used at boundaries:

$$z'(t) \approx \frac{z(t+\Delta t) - z(t)}{\Delta t} \ , \quad z'(t) \approx \frac{z(t) - z(t-\Delta t)}{\Delta t} \ . \tag{23b}$$

The central scheme has truncation error $\mathcal{O}(\Delta t^2)$, while forward and backward schemes are $\mathcal{O}(\Delta t)$. Although less accurate than spectral differentiation, FD methods are more stable for short trajectories and at dataset boundaries, making them complementary in our hybrid FFT-FD approach.

2.4 Error Propagation in Integration

While FNODE avoids numerical integration during training, its long-term accuracy depends on how errors propagate through the ODE solver. To carry out a basic error analysis following a classical framework [31,32], let the true system state at step n be Z_n, advanced by the exact evolution operator Φ_g, and let $\hat{Z}_n$ denote the predicted state advanced by the learned solver Φ_f. The global error is $E_n = \hat{Z}_n - Z_n$. The exact and predicted updates are

$$Z_{n+1} = \Phi_g(Z_n), \qquad \hat{Z}_{n+1} = \Phi_f(\hat{Z}_n),$$

so that

$$E_{n+1} = \hat{Z}_{n+1} - Z_{n+1} = \Phi_f(\hat{Z}_n) - \Phi_g(Z_n).$$

Substituting $\hat{Z}_n = Z_n + E_n$ gives

$$E_{n+1} = \Phi_f(Z_n + E_n) - \Phi_g(Z_n).$$

A first-order Taylor expansion around Z_n yields

$$\Phi_f(Z_n + E_n) \approx \Phi_f(Z_n) + \left.\frac{\partial \Phi_f}{\partial Z}\right|_{Z_n} E_n \ ,$$

where $\frac{\partial \Phi_f}{\partial Z}\big|_{Z_n}$ denotes the Jacobian of the learned flow map Φ_f with respect to the state, evaluated at Z_n. Inserting this into the expression for E_{n+1} gives

$$E_{n+1} \approx \underbrace{\big(\Phi_f(Z_n) - \Phi_g(Z_n)\big)}_{\text{local error}} + \underbrace{\left.\frac{\partial \Phi_f}{\partial Z}\right|_{Z_n} E_n}_{\text{propagated error}} .$$

Thus, each step combines a local error (discrepancy between Φ_f and Φ_g) and a propagated error (amplification of the previous error by the solver Jacobian), showing how small inaccuracies in the learned acceleration field can accumulate over time and underscoring the need for accurate acceleration targets during training.

2.5 Loss Function and Optimization

The FNODE loss directly supervises accelerations. Given predicted accelerations $\hat{\ddot{\mathbf{z}}}_i = f(\mathbf{z}_i, \dot{\mathbf{z}}_i; \boldsymbol{\Theta})$ and reference accelerations $\ddot{\mathbf{z}}_i$ from trajectory data, we minimize the mean squared error (MSE)

$$\mathcal{L}(\boldsymbol{\Theta}) = \frac{1}{N} \sum_{i=0}^{N-1} \|\hat{\ddot{\mathbf{z}}}_i - \ddot{\mathbf{z}}_i\|_2^2. \tag{24a}$$

The optimal parameters are obtained as

$$\boldsymbol{\Theta}^* = \arg\min_{\boldsymbol{\Theta}} \mathcal{L}(\boldsymbol{\Theta}). \tag{24b}$$

We train FNODE using the Adam optimizer, which provides stable and efficient convergence in practice. The training algorithm is shown in Appendix A. A discussion of the training costs is deferred to Section 4.

3 Numerical Experiments

3.1 Single-Mass-Spring-Damper System

As a first benchmark, we consider a simple single-mass-spring-damper system. Its dynamics are governed by

$$\ddot{x} = -\tfrac{k}{m}x - \tfrac{d}{m}\dot{x}, \tag{25}$$

with $m = 10$ kg, $k = 50$ N/m, and $d = 2$ Ns/m. The system is initialized with $x(0) = 1$ m, $\dot{x}(0) = 0$, and simulated with RK4 at $\Delta t = 0.01$ s. A trajectory of 700 steps was used for training, with an additional 300 steps reserved for extrapolation. Model hyperparameters are listed in Table 2.

Figure 2 compares predicted displacement and velocity. All methods fit well within the training interval, but differences emerge in extrapolation. FNODE maintains close agreement with the ground truth (ϵ = 1.9e-2), while

Table 2 Hyper-parameters for the single mass-spring-damper system.

Hyper-parameters	Model			
	FNODE	MBD-NODE	LSTM	FCNN
No. of hidden layers	2	2	2	2
No. of nodes per hidden layer	256	256	256	256
Max. epochs	500	450	500	500
Initial learning rate	1e-3	1e-3	1e-3	1e-3
Learning rate decay	0.98	0.98	0.98	0.98
Activation function	Tanh	Tanh	Sigmoid,Tanh	Tanh
Loss function	MSE	MSE	MSE	MSE
Optimizer	Adam	Adam	Adam	Adam

MBD-NODE degrades more quickly (ϵ = 1.3e-1). LSTM fails to capture the dissipative behavior (ϵ = 7.0e-1), and FCNN diverges entirely (ϵ = 4.2e1). Phase-space trajectories in Fig. 3 confirm FNODE's stable long-term prediction, in contrast to the drift or stagnation observed in the baselines.

3.2 Multiscale Triple-Mass-Spring-Damper System

3.3 Triple-Mass-Spring-Damper System

We next consider a three-mass-spring-damper system with strong scale separation (Fig. 4). The masses are $m_1 = 100$ kg, $m_2 = 10$ kg, and $m_3 = 1$ kg. All springs have stiffness $k_i = 50$ N/m and all dampers $d_i = 2$ Ns/m. The governing equations are

$$\begin{aligned}
\ddot{x}_1 &= -\tfrac{k_1}{m_1}x_1 - \tfrac{d_1}{m_1}(\dot{x}_1 - \dot{x}_2) + \tfrac{k_2}{m_1}(x_2 - x_1) + \tfrac{d_2}{m_1}(\dot{x}_2 - \dot{x}_1),\\
\ddot{x}_2 &= -\tfrac{k_2}{m_2}(x_2 - x_1) - \tfrac{d_2}{m_2}(\dot{x}_2 - \dot{x}_1) + \tfrac{k_3}{m_2}(x_3 - x_2) + \tfrac{d_3}{m_2}(\dot{x}_3 - \dot{x}_2),\\
\ddot{x}_3 &= -\tfrac{k_3}{m_3}(x_3 - x_2) - \tfrac{d_3}{m_3}(\dot{x}_3 - \dot{x}_2).
\end{aligned} \tag{26}$$

The system was simulated with RK4 at a fixed step of $\Delta t = 0.01$ s. Training trajectories consisted of 300 steps from the initial state $x_1 = 1$, $x_2 = 2$, $x_3 = 3$, $v_1 = v_2 = v_3 = 0$ (SI units). Performance was then evaluated by extrapolation for an additional 100 steps. Model hyperparameters are listed in Table 3.

Figure 5 shows the trajectories of x_1, x_2, x_3 and v_1, v_2, v_3 during training and testing. All models fit the training data, though FCNN exhibits minor oscillations. In extrapolation ($t > 3$), FNODE achieves the best agreement with ground truth (ϵ = 3.0e-2). MBD-NODE remains reasonable but drifts due to solver error accumulation. LSTM largely repeats training patterns and fails to generalize, while FCNN diverges under multiscale dynamics.

Figure 6 shows phase-space trajectories on the test set. FNODE tracks the dynamics well, though small discrepancies appear for the lighter masses due to irregular acceleration fields and error accumulation. MBD-NODE performs poorly for Mass 1, where low variance in the signal reduces its contribution to

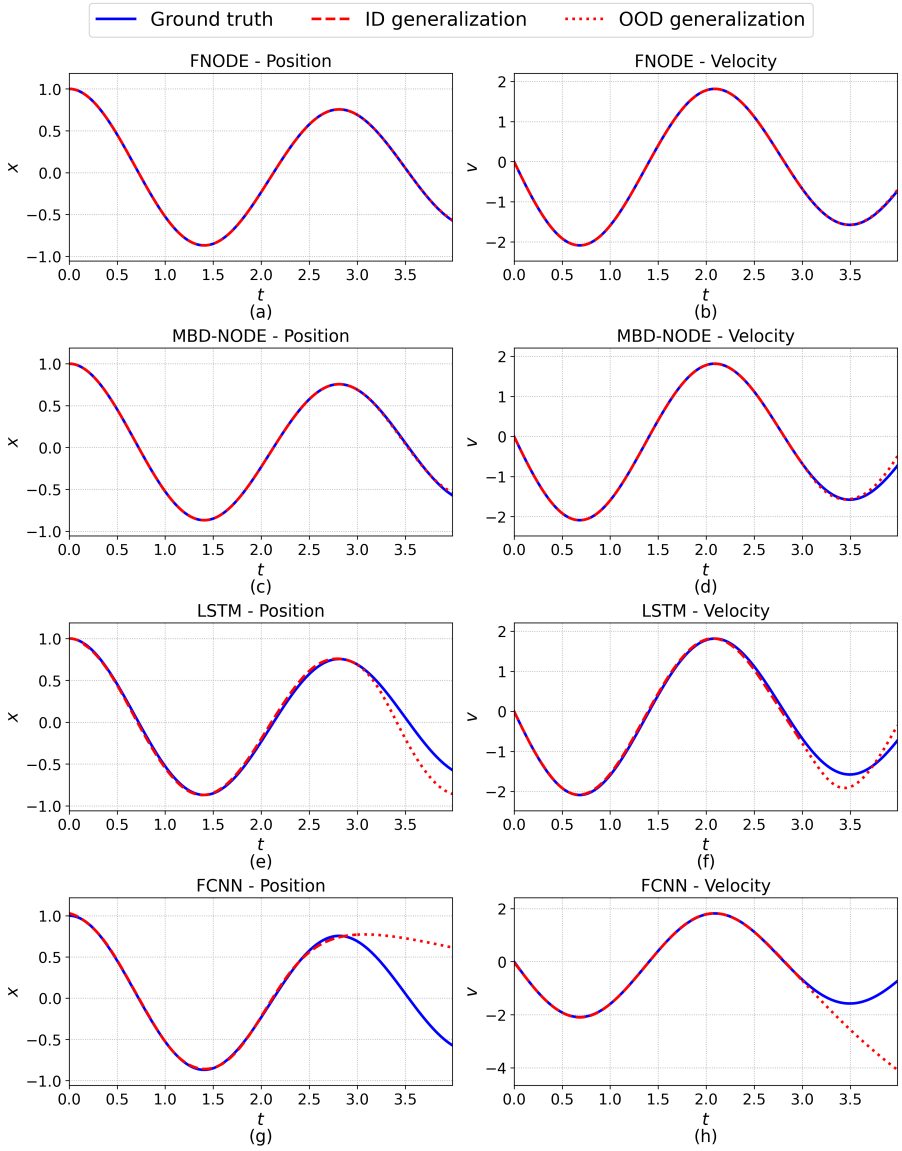

Fig. 2 Temporal evolution of state variables for the single-mass-spring-damper system. The left and right columns depict the displacement (x) and velocity (v), respectively. Dashed lines indicate model predictions over the training interval ($t \in [0, 3]$), while dotted lines represent extrapolated predictions on the test interval. The mean squared error (MSE) for each model is provided: FNODE (ϵ = 1.9e-2), MBD-NODE (ϵ = 1.3e-1), LSTM (ϵ = 7.0e-1), and FCNN (ϵ = 4.2e1).

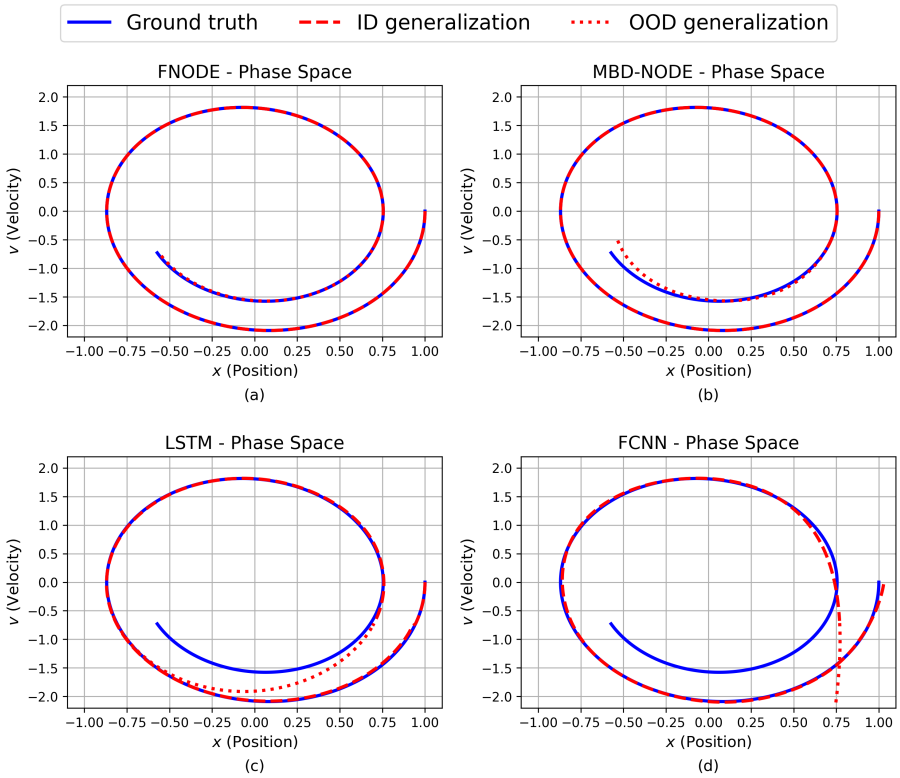

Fig. 3 Comparative phase-space trajectories (v vs. x) for the single-mass-spring-damper system. Model performance on the training and test data is differentiated by dashed and dotted lines, respectively. The subplots correspond to the predictions of (a) FNODE, (b) MBD-NODE, (c) LSTM, and (d) FCNN.

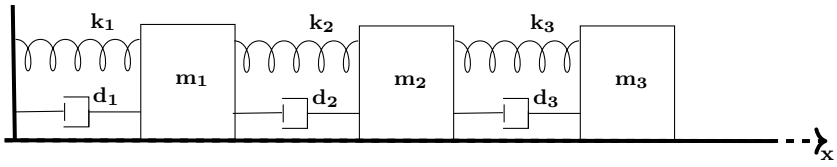

Fig. 4 Triple mass-spring-damper system. The setup is similar to the single mass-spring-damper system, except for the addition of two more masses, springs, and dampers.

the loss. LSTM largely repeats training patterns and fails to generalize, while FCNN is inaccurate both in- and out-of-distribution.

Table 3 Hyper-parameters for the triple-mass-spring-damper system.

Hyper-parameters	Model			
	FNODE	MBD-NODE	LSTM	FCNN
No. of hidden layers	2	2	2	2
No. of nodes per hidden layer	256	256	256	256
Max. epochs	5000	450	500	500
Initial learning rate	1e-3	1e-3	1e-3	1e-3
Learning rate decay	0.98	0.98	0.98	0.98
Activation function	Tanh	Tanh	Sigmoid,Tanh	Tanh
Loss function	MSE	MSE	MSE	MSE
Optimizer	Adam	Adam	Adam	Adam

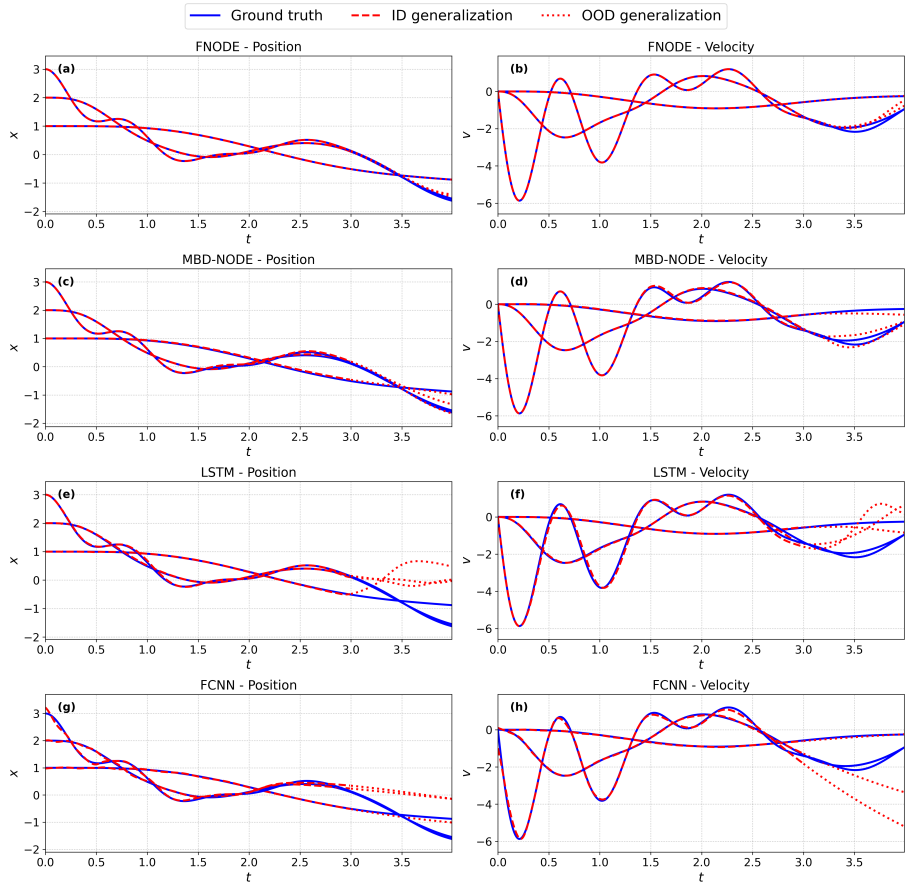

Fig. 5 Predicted temporal evolution for the triple-mass-spring-damper benchmark. The left column presents the displacement (x) and the right column presents the velocity (v) for each of the three masses. Dashed and dotted lines denote predictions on the training ($t \in [0, 3]$) and test sets, respectively. The associated MSEs are: FNODE (ϵ = 3.0e-2), MBD-NODE (ϵ = 5.0e-2), LSTM (ϵ = 1.0e0), and FCNN (ϵ = 1.3e0).

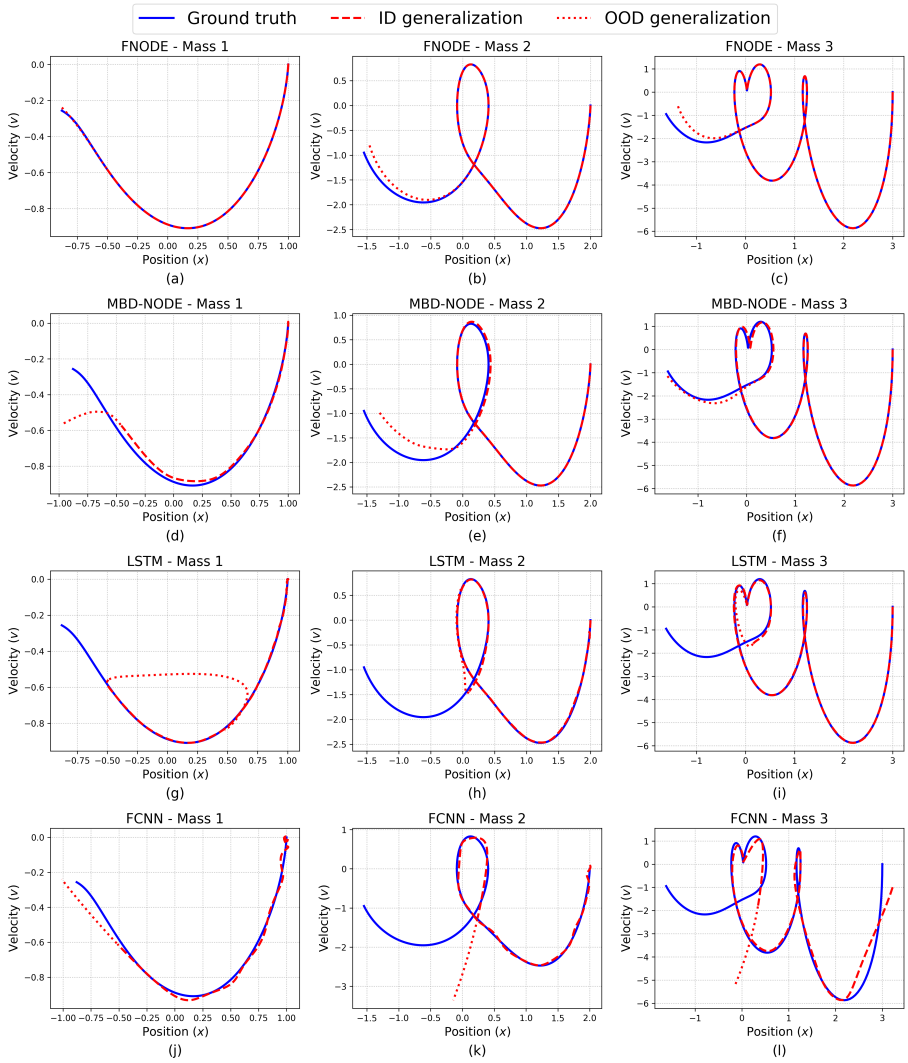

Fig. 6 Phase-space trajectories for the three constituent masses of the triple-mass-spring-damper system. Each column corresponds to a different mass. Model performance on the training and test datasets is indicated by dashed and dotted lines, respectively. The rows correspond to the results from (a-c) FNODE, (d-f) MBD-NODE, (g-i) LSTM, and (j-l) FCNN.

3.4 Double Pendulum

We next evaluate FNODE on a chaotic benchmark: the double pendulum with $m_1 = m_2 = 1$ kg, $l_1 = l_2 = 1$ m, and $g = 9.81$ m/s^2. The dynamics follow from the Hamiltonian formulation (see Appendix B for details).

The double pendulum was simulated with $L_1 = L_2 = 1$ m, $m_1 = m_2 = 1$ kg, and $g = 9.81$ m/s^2. The initial state was $\theta_1 = 3\pi/7$, $\theta_2 = 3\pi/4$, and $\dot{\theta}_1 = \dot{\theta}_2 = 0$. Trajectories were generated using RK4 with $\Delta t = 0.01$ s for 300 training steps and 100 extrapolation steps. Model hyperparameters are listed in Table 4.

Table 4 Hyper-parameters for the double pendulum system

Hyper-parameters	Model			
	FNODE	MBD-NODE	LSTM	FCNN
No. of hidden layers	3	3	3	3
No. of nodes per hidden layer	256	256	256	256
Max. epochs	10000	450	400	600
Initial learning rate	1e-3	1e-3	5e-4	5e-4
Learning rate decay	0.7	0.98	0.98	0.99
Activation function	Tanh	Tanh	Sigmoid,Tanh	Tanh
Loss function	MSE	MSE	MSE	MSE
Optimizer	Adam	Adam	Adam	Adam

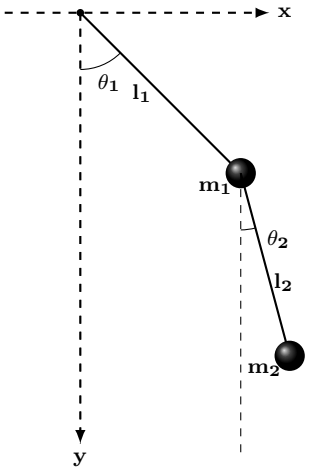

Fig. 7 Schematic of the double pendulum.

Figure 8 shows the double pendulum trajectories. All models fit the training data, but diverge in extrapolation. FNODE, despite error accumulation in the ODE solver, achieves the best accuracy ($\epsilon = 1.9$e-2).

Phase-space plots in Fig. 9 confirm this trend: FNODE reproduces the main chaotic patterns, with only minor discrepancies. MBD-NODE loses accuracy in testing, LSTM largely repeats training patterns without generalization, and FCNN fails to capture the dynamics outside the training regime.

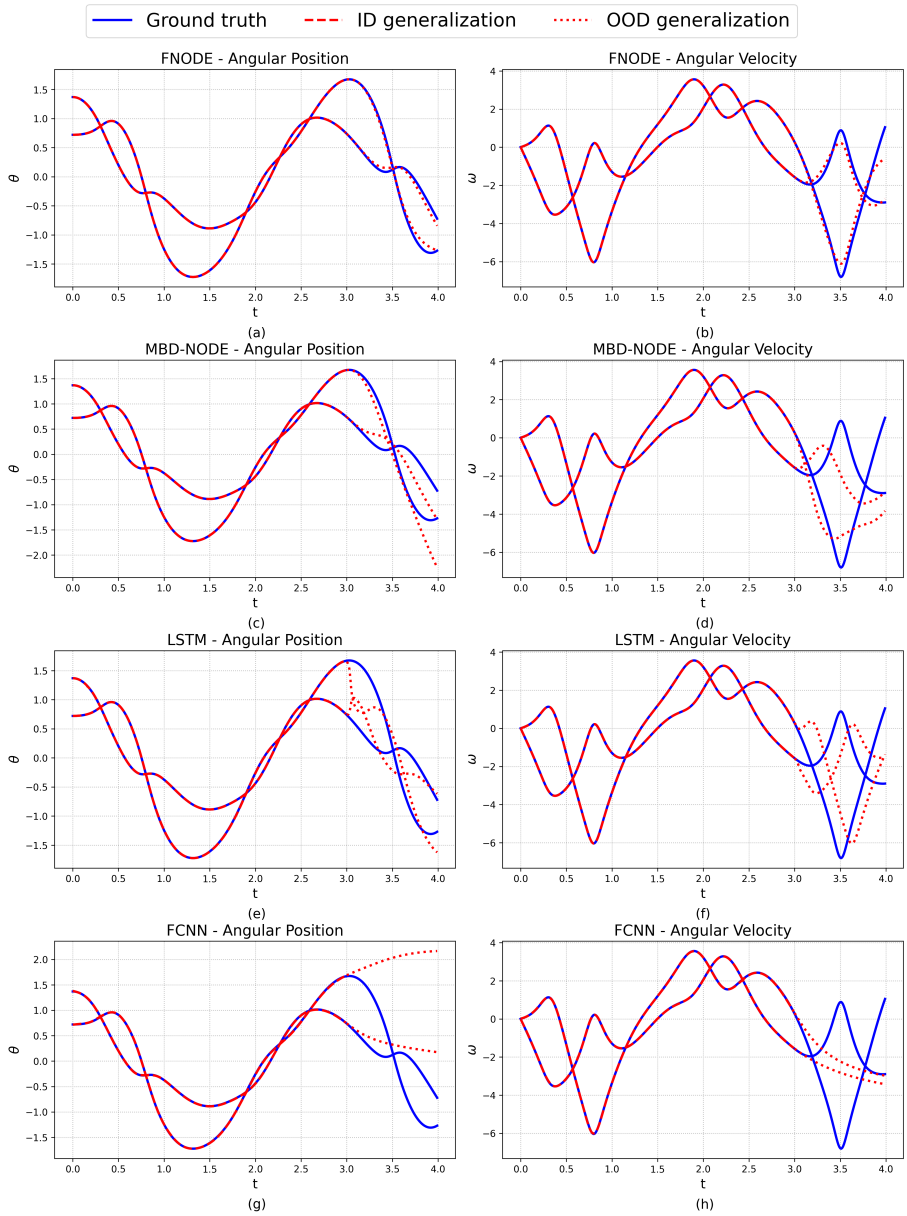

Fig. 8 Dynamic response predictions for the double pendulum system. The left and right columns show the temporal evolution of angular position (θ) and angular velocity (ω), respectively. Dashed lines represent the fit to the training data ($t \in [0, 3]$), while dotted lines show the extrapolation on test data. The reported MSE values are: FNODE (ϵ = 1.9e-2), MBD-NODE (ϵ = 3.7e-1), LSTM (ϵ = 2.2e0), and FCNN (ϵ = 8.7e0).

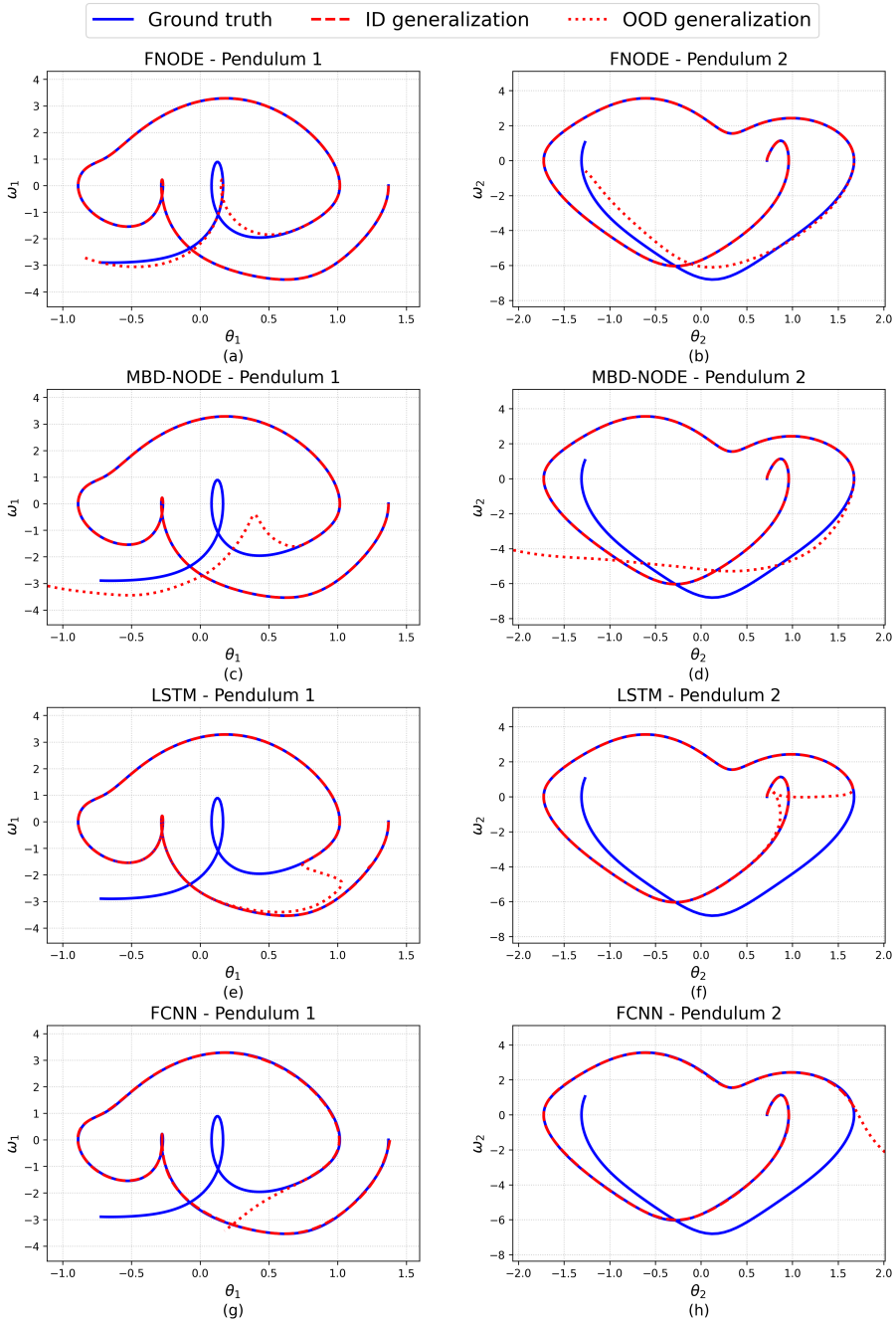

Fig. 9 Phase-space of double pendulum system. The left column corresponds to the first mass and the right to the second. Dashed lines indicate the trajectory over the training interval, and dotted lines show the predicted trajectory over the test interval. The models evaluated are (a,b) FNODE, (c,d) MBD-NODE, (e,f) LSTM, and (g,h) FCNN.

3.5 Slider Crank

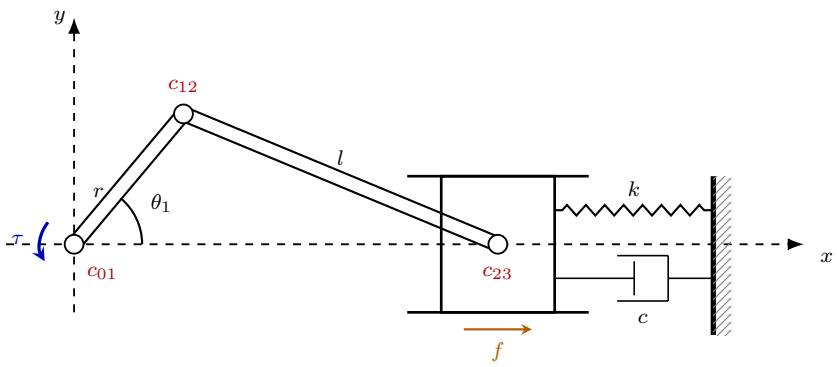

Fig. 10 Slider-crank mechanism with motor torque τ_1, rotational dampers at all joints, and slider friction.

We next test long-term prediction on a slider–crank mechanism (Fig. 10). The system is simulated for 45 s (4500 steps), with 1500 steps used for training and the remainder for testing. The mechanism consists of three rigid bodies with a slider connected to a spring and damper, forming a constrained three-body problem. The generalized coordinates are $q = (x_1, y_1, \theta_1, x_2, y_2, \theta_2, x_3, y_3, \theta_3)$.

1. A crank (body 1) with mass $m_1 = 1$ kg and moment of inertia $I_1 = 0.1\ \text{kg} \cdot \text{m}^2$ is connected to the ground by a revolute joint. Its length is $l = 2$ m.
2. A connecting rod (body 2) with mass $m_2 = 1$ kg and moment of inertia $I_2 = 0.1\ \text{kg} \cdot \text{m}^2$ is attached to the crank via a revolute joint. Its length is $r = 1$ m.
3. A slider (body 3) with mass $m_3 = 1$ kg and moment of inertia $I_3 = 0.1\ \text{kg} \cdot \text{m}^2$ is joined to the connecting rod by a revolute joint. It is also connected to a fixed wall via a spring with a torsional constant of $k = 1$ N/m.

FNODE learns a mapping from the full 18-dimensional state space $(q, \dot{q})$ to accelerations $\ddot{q}$. Since the mechanism has only one DOF, the dynamics can be represented compactly with the minimal coordinates $(\theta_1, \dot{\theta}_1)$, with all other states dependent. The model hyperparameters are summarized in Table 5.

Figure 11 shows the slider-crank responses. All models fit the training data, but diverge in extrapolation. FNODE maintains accurate long-term predictions. MBD-NODE, while structurally similar to FNODE, is less stable and incurs higher computational cost due to its solver-based training. In contrast, LSTM and FCNN fail to generalize beyond the training window. Figure 15 illustrates the remaining state variables, which inherit errors from the minimal coordinates $(\theta_1, \dot{\theta}_1)$.

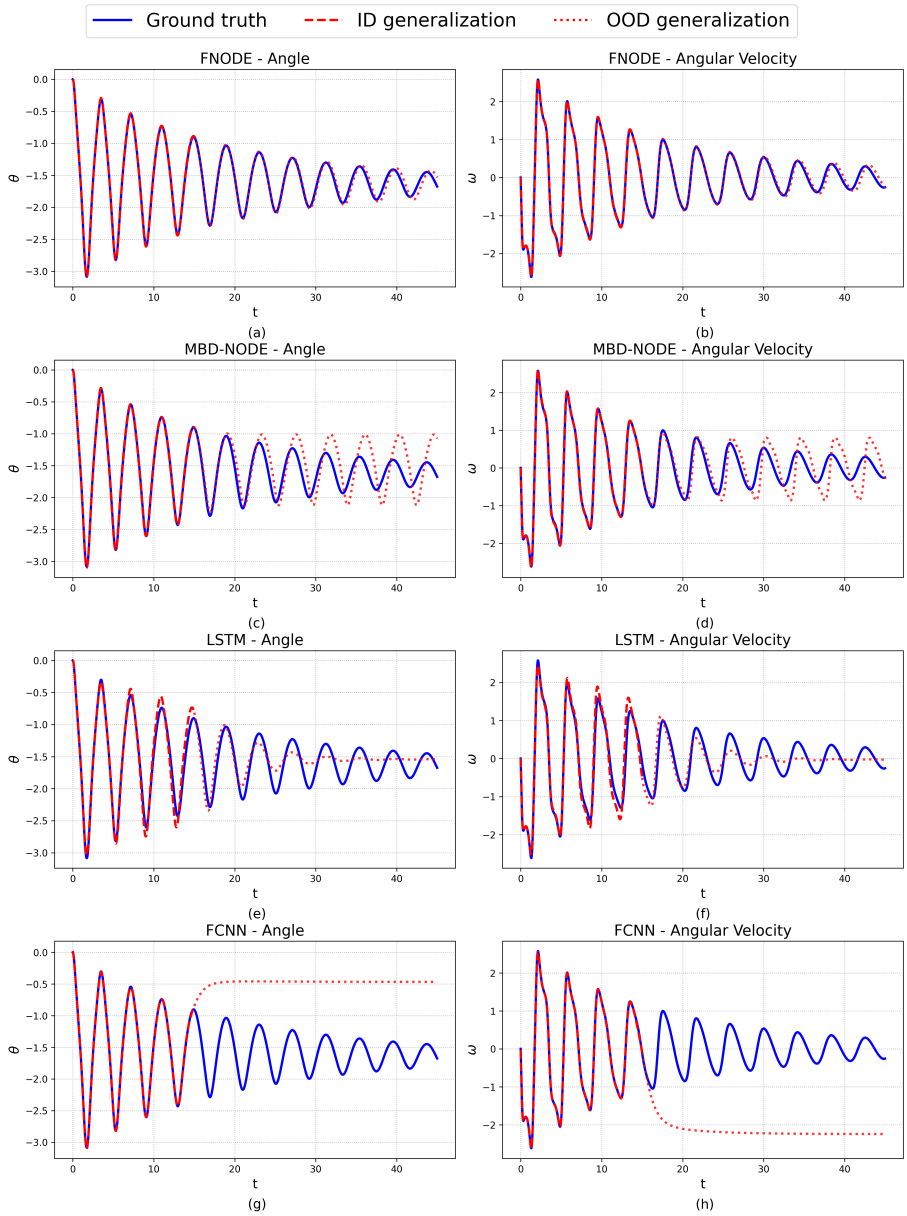

Fig. 11 Long-term prediction results for the slider-crank's independent coordinate (θ_1). The left column presents angular position versus time; the right column presents angular velocity versus time. Model performance on the training interval ($t \in [0, 15]$) is shown with dashed lines, while extrapolated performance on the test interval is shown with dotted lines. The corresponding MSEs are: FNODE (ϵ = 1.9e-3), MBD-NODE (ϵ = 8.2e-2), LSTM (ϵ = 4.5e-2), and FCNN (ϵ = 2.03e0).

Table 5 Hyper-parameters for the slider-crank mechanism.

Hyper-parameters	Model			
	FNODE	MBD-NODE	LSTM	FCNN
No. of hidden layers	3	3	3	3
No. of nodes per hidden layer	256	256	256	256
Max.epochs	10000	400	5000	3000
Initial learning rate	1e-3	1e-3	1e-3	1e-3
Learning rate decay	0.98	0.98	0.98	0.98
Activation function	Tanh	Tanh	Tanh	Tanh
Loss function	MSE	MSE	MSE	MSE
Optimizer	Adam	Adam	Adam	Adam

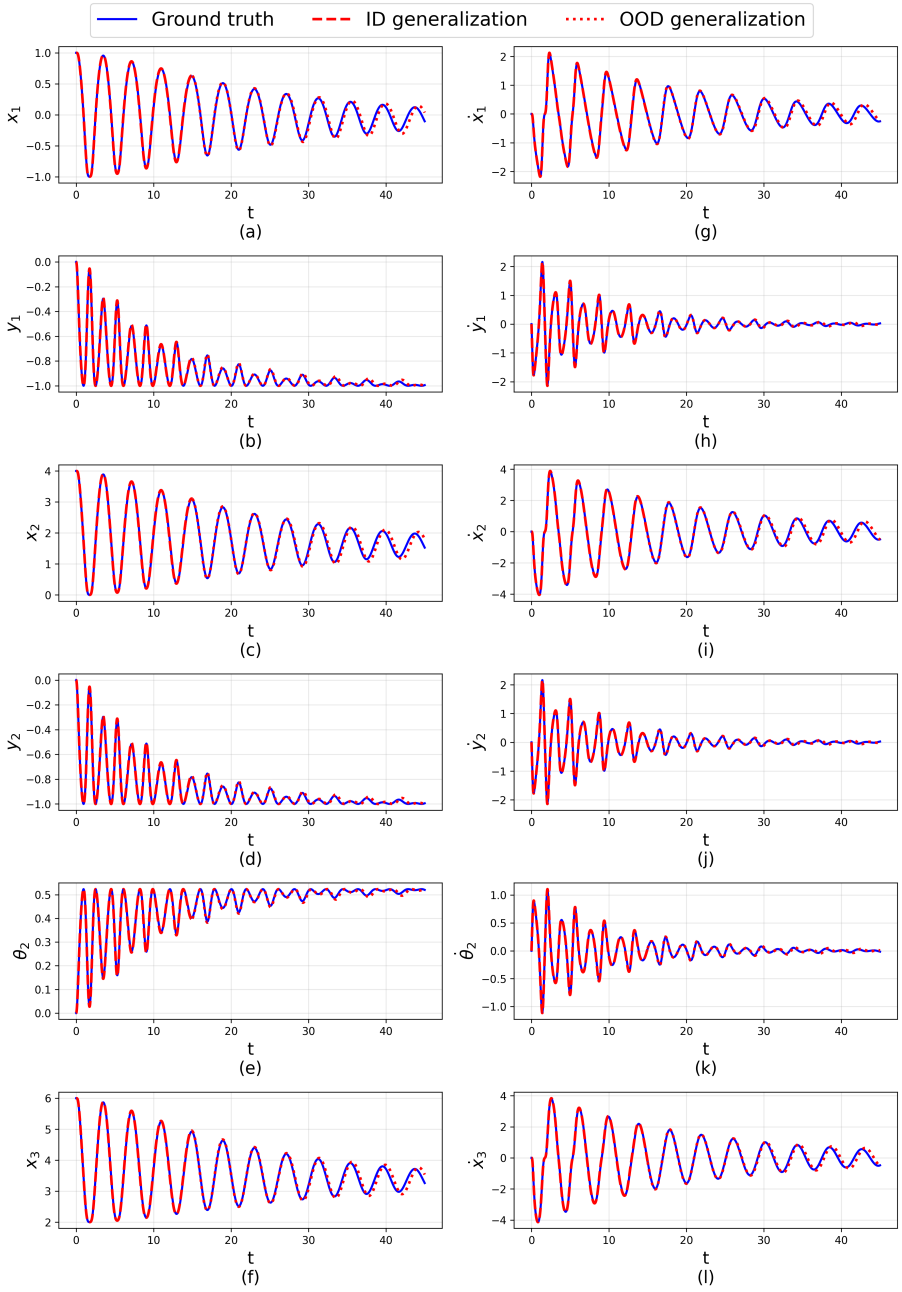

Fig. 12 Temporal evolution of the dependent state variables for the slider-crank mechanism, as reconstructed from the FNODE model's prediction of the minimal coordinates.

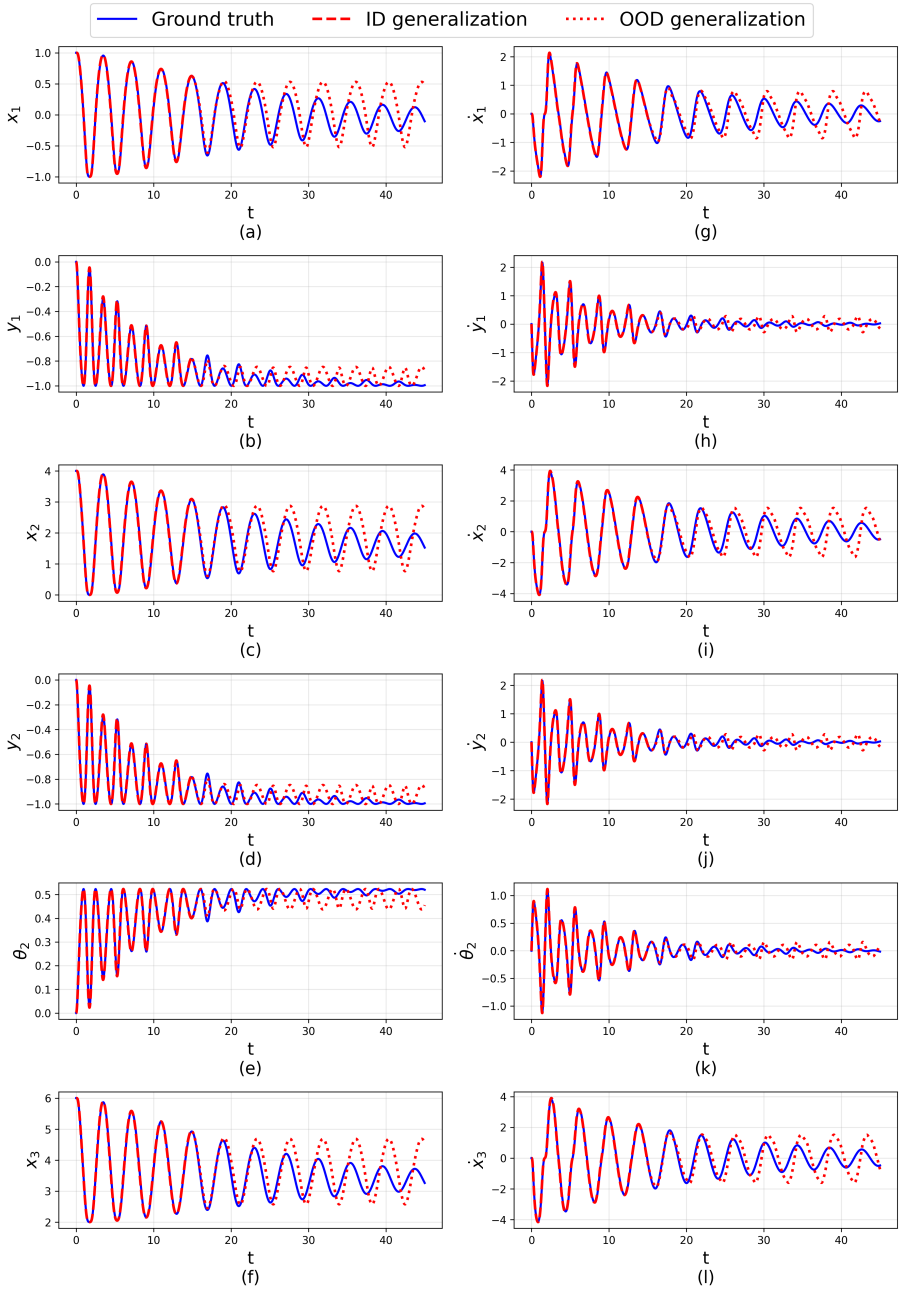

Fig. 13 Temporal evolution of the dependent state variables for the slider-crank mechanism, as reconstructed from the MBD-NODE model's prediction of the minimal coordinates.

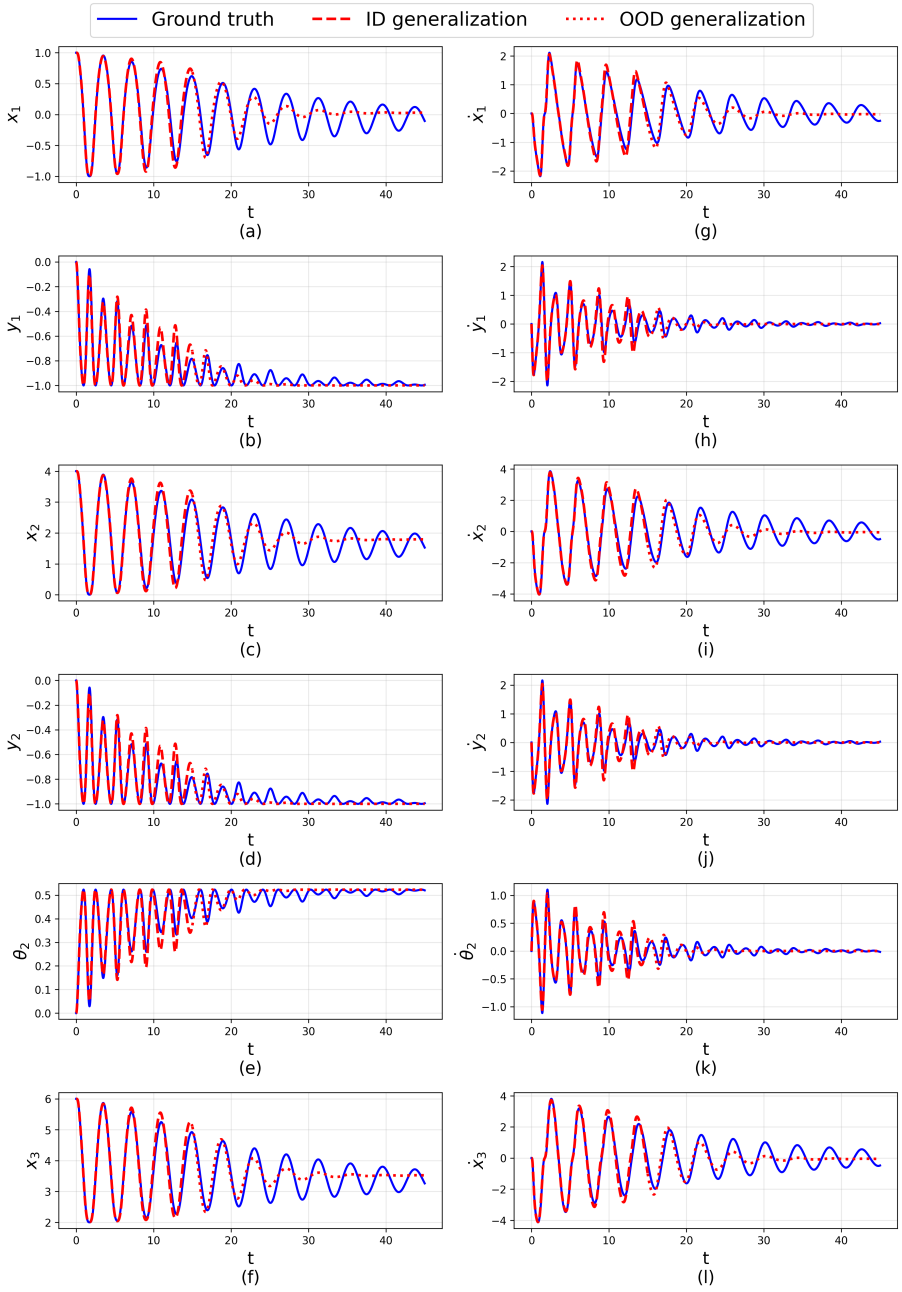

Fig. 14 Temporal evolution of the dependent state variables for the slider-crank mechanism, as reconstructed from the LSTM model's prediction of the minimal coordinates.

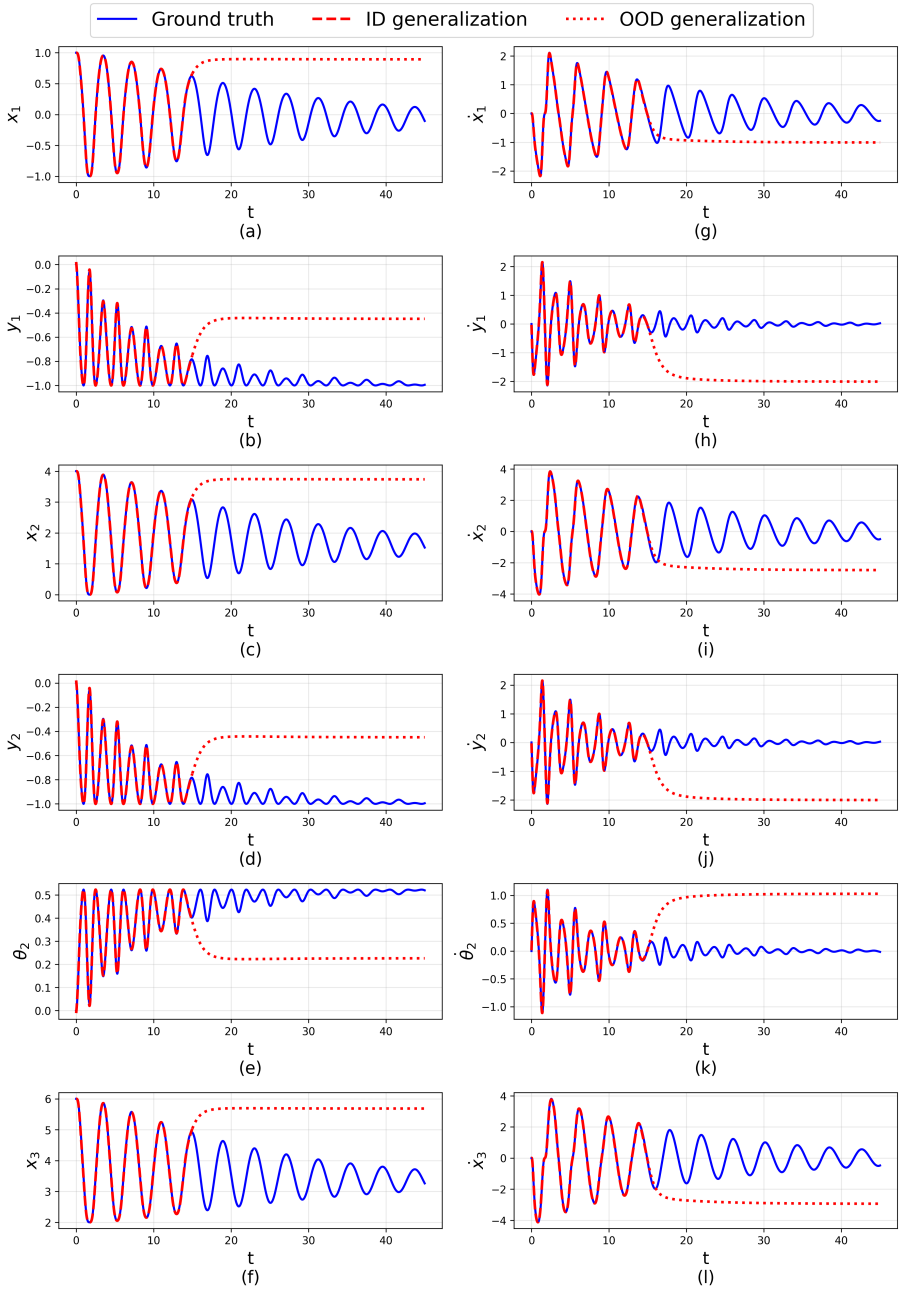

Fig. 15 Temporal evolution of the dependent state variables for the slider-crank mechanism, as reconstructed from the FCNN model's prediction of the minimal coordinates.

3.6 Cart Pole

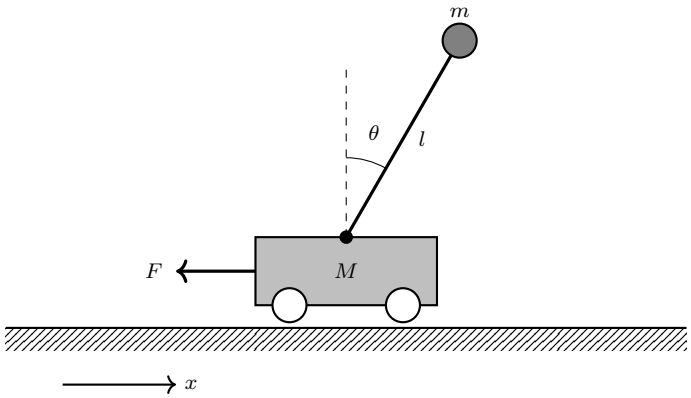

Fig. 16 Cart-pole system.

3.7 Cart-Pole System

We finally evaluate model performance on the cart-pole system, a canonical control benchmark (Fig. 16). The cart moves horizontally on a frictionless track, while the pendulum rotates freely about its pivot, with motion restricted to $[-\pi/2, \pi/2]$. The system state is described by four variables: cart position x, velocity v, pendulum angle θ, and angular velocity ω. The dynamics are governed by the nonlinear ODEs in Eqs. (27).

$$\begin{aligned} ml^2\ddot{\theta} + ml\cos\theta\ddot{x} - mgl\sin\theta = 0 \\ ml\ddot{\theta}\cos\theta + (M+m)\ddot{x} - ml\dot{\theta}^2\sin\theta = u \end{aligned} \tag{27}$$

The cart-pole was first tested in free evolution (no external input), initialized at

$$x(0) = 1,\ v(0) = 0,\ \theta(0) = \tfrac{\pi}{6},\ \omega(0) = 0.$$

Trajectories were generated with a midpoint integrator at $\Delta t = 0.01$ s. The first 200 steps (2 s) were used for training and the following 50 steps (0.5 s) for testing. Model hyperparameters are listed in Table 6.

Figures 17-18 compare model performance. FNODE and MBD-NODE both match the ground truth closely, with low MSEs (1.4×10^{-5} and 8.6×10^{-6}, respectively). MBD-NODE is slightly more accurate, but FNODE remains highly competitive while training far more efficiently (Table 4). In contrast, LSTM largely reproduces training patterns without generalization, and FCNN diverges significantly in extrapolation.

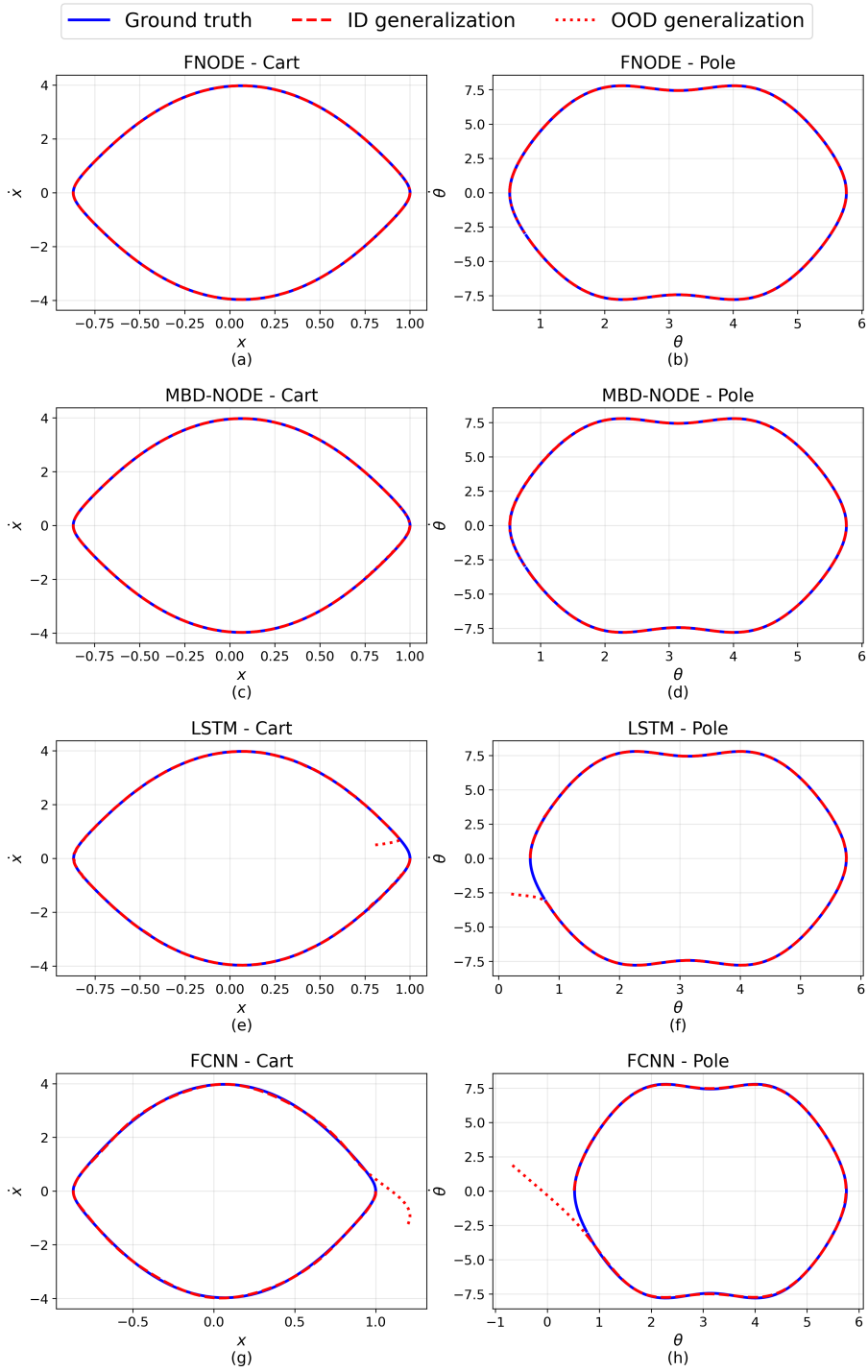

Fig. 17 Phase-space trajectories for the cart-pole free-evolution scenario. The left subplot shows the cart's phase portrait (v vs. x), while the right subplot shows the pole's phase portrait (ω vs. θ). Dashed lines represent the fit to training data; dotted lines represent test data predictions. The models are: (a,b) FNODE, (c,d) MBD-NODE, (e,f) LSTM, and (g,h) FCNN.

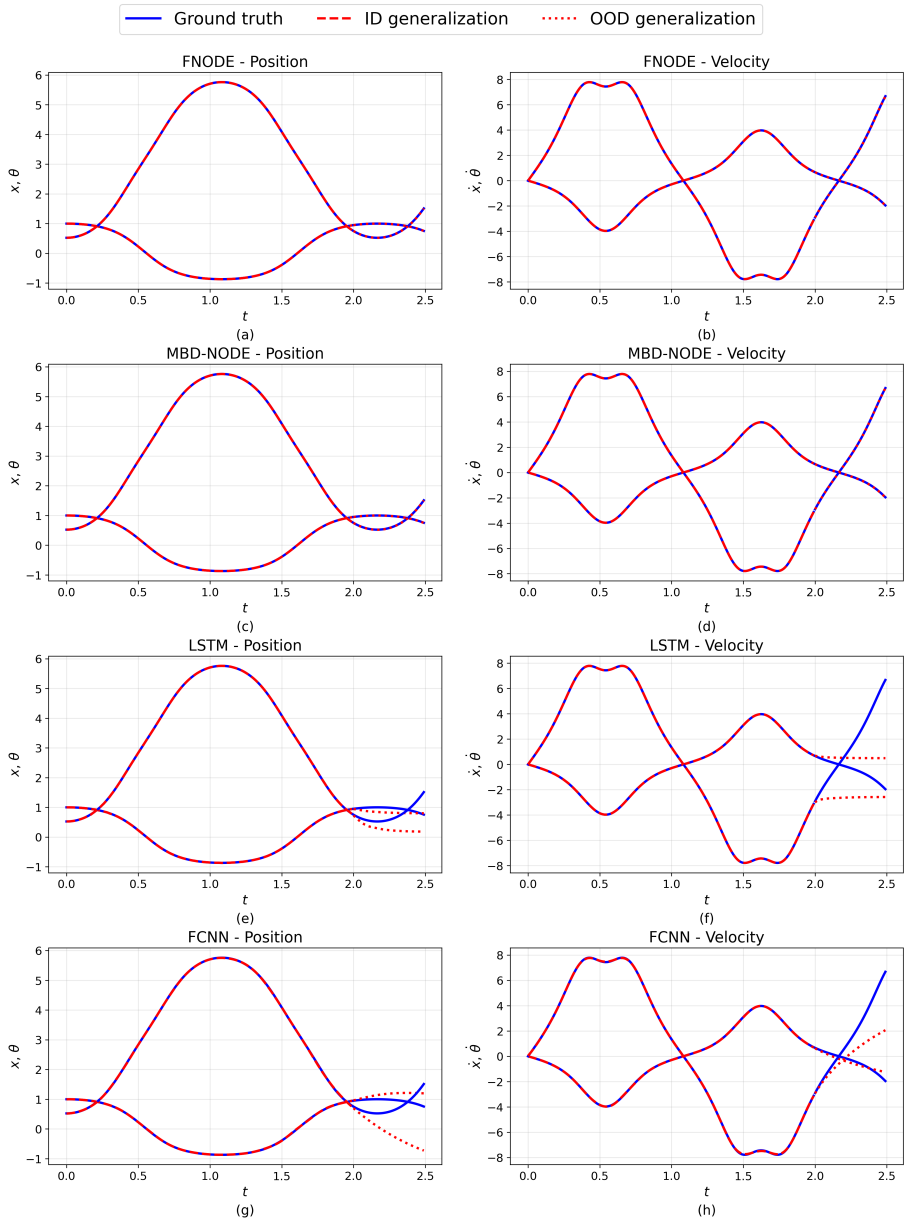

Fig. 18 State variable trajectories over time for the cart-pole system. The left column shows position variables (x, θ), and the right column shows velocity variables (v, ω). Dashed lines correspond to the training interval ($t \in [0, 2]$), and dotted lines to the test interval. The final MSEs are: FNODE (ϵ = 1.4e-5), MBD-NODE (ϵ = 8.6e-6), LSTM (ϵ = 6.6e0), and FCNN (ϵ = 1.4e0).

Table 6 Hyper-parameters for the cart-pole system.

Hyper-parameters	Model			
	FNODE	MBD-NODE	LSTM	FCNN
No. of hidden layers	3	3	3	3
No. of nodes per hidden layer	256	256	256	256
Max.epochs	10000	400	5000	3000
Initial learning rate	1e-3	1e-3	1e-3	1e-3
Learning rate decay	0.98	0.98	0.98	0.98
Activation function	Tanh	Tanh	Tanh	Tanh
Loss function	MSE	MSE	MSE	MSE
Optimizer	Adam	Adam	Adam	Adam

We also test FNODE in a model predictive control (MPC) task, where an external force u is applied to stabilize the pole and return the cart to the origin. MPC proceeds by linearizing the dynamics at each step and solving a quadratic program over a finite horizon. For a linearized system $\dot{z} = Az + Bu$, the optimization problem is given in Eq. (28).

$$\begin{aligned} \min_{u} \quad & \sum_{k=0}^{N-1} z_k^T Q z_k + u_k^T R u_k \\ \text{s.t.} \quad & z_{k+1} = A z_k + B u_k, \quad k = 0, 1, \ldots, N-1 \\ & z_k \in Z, \quad u_k \in U, \quad k = 0, 1, \ldots, N-1 \end{aligned} \tag{28}$$

At each step, the system state is $z_k = (\theta_k, x_k, \omega_k, v_k)$ with control input u_k. The MPC problem is solved over a horizon of length N, subject to state and input constraints $\mathcal{Z}$ and $\mathcal{U}$. The cost function weights are Q and R, both chosen as identity matrices. The linearized dynamics are written as $\dot{z} = Az + Bu$, where A and B are obtained from a first-order Taylor expansion of the nonlinear cart–pole equations (Eq. 29).

$$A = \begin{bmatrix} 0 & 0\ 1\ 0 \\ 0 & 0\ 0\ 1 \\ \frac{g(m+M)}{Ml} & 0\ 0\ 0 \\ -\frac{mg}{M} & 0\ 0\ 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 0 \\ -\frac{1}{Ml} \\ \frac{1}{M} \end{bmatrix} \tag{29}$$

A trained FNODE model can be directly linearized by computing the Jacobian of its learned dynamics. In this task, FNODE learns the mapping from state and control input to accelerations, $f(\theta_k, x_k, \omega_k, v_k, u) \mapsto (\ddot{\theta}_k, \ddot{x}_k)$. The Jacobian is obtained via automatic differentiation and used as the linearized matrices A and B within the MPC framework.

Figure 19 shows the MPC results for the cart-pole. The five subplots depict the pole angle θ, cart position x, angular velocity ω, cart velocity $\dot{x}$, and control input u. The FNODE-based controller (red dotted) closely matches the analytical baseline (blue solid), demonstrating that FNODE learns the dynamics with sufficient accuracy for MPC.

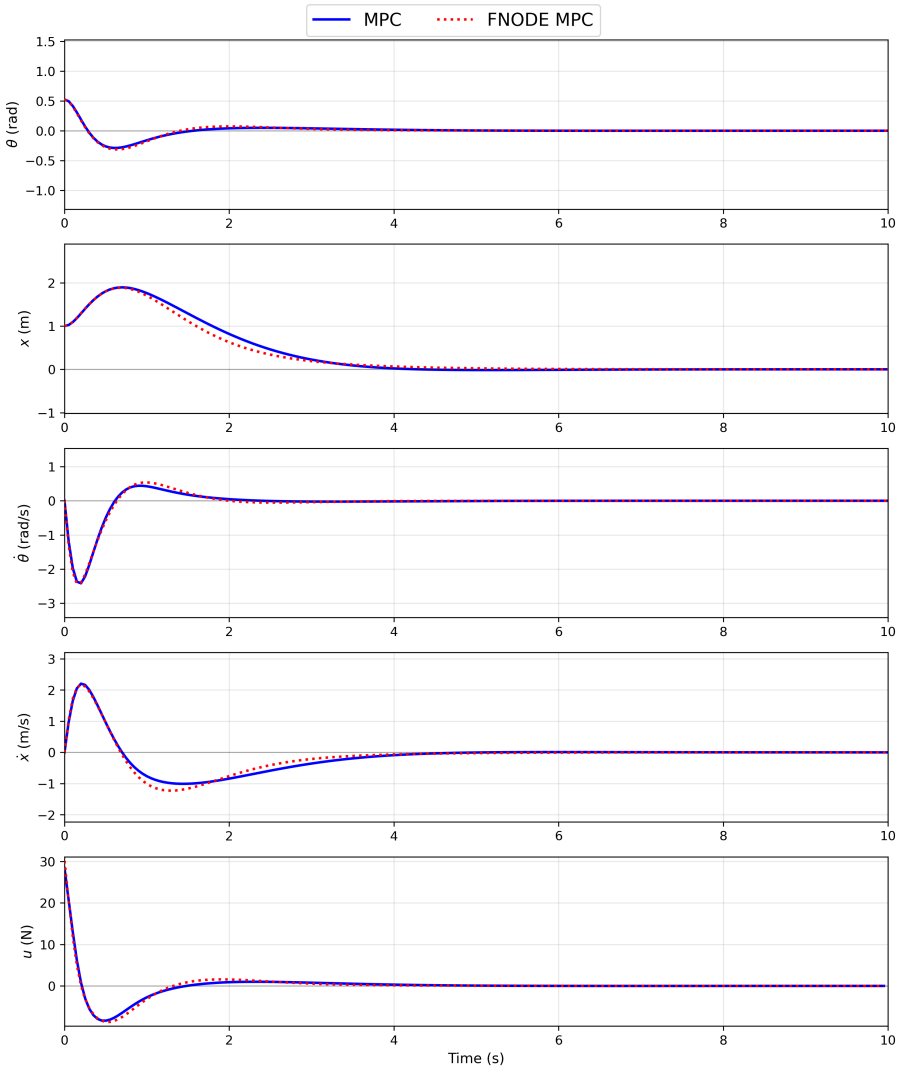

Fig. 19 Comparative performance of the MPC strategy. The analytical model (solid blue) and FNODE-based model (dotted red) are compared across: (a) pole angle θ, (b) cart position x, (c) pole angular velocity ω, (d) cart velocity v, and (e) the control force u.

4 Discussion

Tables 4 and 4 summarize the training cost and predictive accuracy of the four models across all benchmarks. Two main observations emerge.

First, FNODE consistently trains much faster than MBD-NODE, often by one to two orders of magnitude. This gap arises because FNODE bypasses the adjoint sensitivity method and does not embed an ODE solver in the backprop-

agation loop. On larger problems such as the slider-crank, the efficiency gain is especially pronounced: FNODE completes training in ∼119 s versus over 2700 s for MBD-NODE. Compared to purely data-driven baselines (FCNN, LSTM), FNODE's training times are of the same order of magnitude, demonstrating that its physics-informed formulation does not incur significant overhead.

Second, FNODE achieves consistently lower prediction error than the black-box baselines and is competitive with, or superior to, MBD-NODE. In dissipative and chaotic systems (single/triple mass-spring-damper, double pendulum), FNODE outperforms all baselines by large margins, maintaining stable long-term predictions. For the slider-crank mechanism, FNODE achieves the best accuracy (1.9×10^{-3} MSE), significantly outperforming both MBD-NODE and the data-driven models. Only in the cart-pole case does MBD-NODE obtain a slightly lower error than FNODE, though at nearly $10\times$ the training cost. These results underscore the importance of accurate acceleration targets: FFT provides precision for smooth trajectories, while FD ensures robustness for non-periodic signals and trajectory endpoints.

Ultimately, FNODE provides a favorable balance between efficiency and accuracy: it is orders of magnitude faster to train than MBD-NODE while delivering consistently higher fidelity than LSTM and FCNN, and near-parity or better accuracy than MBD-NODE across benchmarks. These results highlight FNODE's potential as a *scalable* framework for data-driven simulation of constrained multibody systems.

Test Case	**Model**	**Integrator**	**Time Cost (s)**
Single Mass Spring Damper	FNODE	–	18.17
	MBD-NODE	RK4	540.18
	FCNN	–	16.96
	LSTM	–	500.63
Triple Mass Spring Damper	FNODE	–	11.45
	MBD-NODE	RK4	529.86
	FCNN	–	9.67
	LSTM	–	5.73
Double Pendulum	FNODE	–	15.04
	MBD-NODE	RK4	553.96
	FCNN	–	8.94
	LSTM	–	5.66
Slider Crank	FNODE	–	118.98
	MBD-NODE	RK4	2735.80
	FCNN	–	27.81
	LSTM	–	33.05
Cart Pole	FNODE	–	14.70
	MBD-NODE	RK4	167.87
	FCNN	–	6.52
	LSTM	–	6.98

Table 7 Training time cost for the models. RK4 denotes the 4th-order Runge-Kutta method. The models are trained on Nvidia GeForce 5070Ti GPU with 16 GB memory.

Test Case	Error			
	FNODE	**MBD-NODE**	**LSTM**	**FCNN**
Single Mass-Spring-Damper	1.9e-2	1.3e-1	7.0e-1	4.2e1
Triple Mass-Spring-Damper	3.0e-2	5.0e-2	1.0e0	1.3e0
Double Pendulum	1.9e-2	3.7e-1	2.2e0	8.7e0
Slider Crank	1.9e-3	8.2e-2	4.5e-2	2.03e0
Cart Pole	1.4e-5	8.6e-6	6.6e0	1.4e0

Table 8 Summary of the numerical MSE for different models.

5 Conclusions and Future Work

We presented FNODE, a framework for data-driven modeling of multibody dynamics that learns accelerations directly rather than states. By reformulating the training objective and leveraging FFT- and finite-difference-based differentiation, FNODE eliminates the adjoint bottleneck of traditional Neural ODEs. Across benchmarks ranging from simple oscillators to chaotic and constrained mechanisms, FNODE trained one to two orders of magnitude faster than MBD-NODE while outperforming black-box baselines such as LSTM and FCNN. Key outcomes include:

- **High accuracy and generalization**: FNODE achieved significantly lower MSE than all baselines, performing well in both interpolation and extrapolation.
- **Data efficiency**: Accurate long-term predictions were obtained from short training trajectories, e.g., thousands of steps extrapolated in the slider-crank test.
- **Multiscale robustness**: FNODE captured dynamics across widely varying time scales, as shown in the triple-mass-spring system.
- **Reproducibility**: An open-source implementation and benchmark suite are provided for future comparisons.

A key ingredient of FNODE is the construction of accurate acceleration targets. FFT-based spectral differentiation offers high precision when trajectories are smooth, but performs poorly at the endpoints of finite, non-periodic data. Finite differences, though lower order, are robust in those cases. By employing these two schemes in a complementary way, FNODE achieves the necessary balance of accuracy and stability, addressing the limitations of using either method alone [33].

Two main limitations remain. First, FNODE relies on accurate acceleration estimates from trajectory data; both FFT and FD methods can suffer from noise and, in the case of FFT, Gibbs artifacts. Second, integration error compounds over time, reducing long-term accuracy in chaotic systems such as the double pendulum.

Future work should address the two main limitations noted above. One direction is developing noise-resilient methods for estimating accelerations from sparse or imperfect data. Another is reducing long-term error growth, for ex-

ample by embedding physical constraints or conservation laws into training or integration. Extending FNODE to higher-dimensional, contact-rich multibody systems would further broaden its impact in engineering applications. Beyond multibody dynamics, applying FNODE ideas to PDE-governed systems, integrating uncertainty quantification, and exploring closed-loop control contexts are promising avenues.

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

Conflict of interest

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

References

1. Roy Featherstone. *Rigid body dynamics algorithms*. Springer, 2008.
2. Ahmed A Shabana. *Dynamics of multibody systems*. Cambridge University Press, 2013.
3. Scott L Delp, Frank C Anderson, Allison S Arnold, Peter Loan, Ayman Habib, Chand T John, Eran Guendelman, and Darryl G Thelen. Opensim: open-source software to create and analyze dynamic simulations of movement. *IEEE Transactions on Biomedical Engineering*, 54(11):1940–1950, 2007.
4. Tamar Schlick. *Molecular modeling and simulation: an interdisciplinary guide*. Springer, 2010.
5. Jens Wittenburg. *Dynamics of multibody systems*. Springer, 2016.
6. Herbert Goldstein, Charles Poole, and John Safko. *Classical mechanics*. Addison Wesley, 2002.
7. Jerrold E Marsden and Tudor S Ratiu. *Introduction to mechanics and symmetry*. Springer, 1999.
8. Christoph Glocker. *Set-valued force laws: dynamics of non-smooth systems*. Springer, 2001.
9. Scott Baker, Cheng Xiang, and Alexandre M Tartakovsky. PDE-based machine learning for discovering hidden physics from noisy data. *Journal of Computational Physics*, 457:111061, 2022.
10. George E Karniadakis, Ioannis G Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021.
11. Pantelis R Vlachas, Wonmin Byeon, Zhong Y Wan, Themistoklis P Sapsis, and Petros Koumoutsakos. Data-driven forecasting of high-dimensional chaotic systems with long short-term memory networks. *Proceedings of the Royal Society A*, 474(2213):20170844, 2018.
12. Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
13. Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. Learning to simulate complex physics with graph networks. In *International Conference on Machine Learning*, pages 8459–8468. PMLR, 2020.
14. Yi Ren and Danica J Sutherland. Learning dynamics of llm finetuning. In *International Conference on Learning Representations (ICLR)*, 2025.

15. Yaofeng Zhong, Joseph Dulny, III, Bethany Lusch, and Wing Yu. SympNets: A unified framework for physics-constrained learning in Hamiltonian systems. *Journal of Computational Physics*, 492:112441, 2023.
16. Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in Neural Information Processing Systems*, 31, 2018.
17. James Morrill, Patrick Kidger, Cristopher Salvi, James Foster, and Terry Lyons. Neural-ODE processes. In *International Conference on Learning Representations (ICLR)*, 2021.
18. Sam Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian neural networks. *Advances in Neural Information Processing Systems*, 32, 2019.
19. Miles Cranmer, Sam Greydanus, Stephan Hoyer, Peter Battaglia, David Spergel, and Shirley Ho. Lagrangian neural networks. In *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations*, 2020.
20. Bofeng Xiao, Taiming Chen, and Kayhan Batmanghelich. Learning physical constraints with neural projections. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pages 25011–25024, 2022.
21. Amir Gholami, Kurt Keutzer, and George Biros. Anode: Unconditionally accurate memory-efficient gradients for neural odes. *arXiv preprint arXiv:1902.10298*, 2019.
22. Pinaki Pal and Opeoluwa Owoyele. ChemNODE: A neural ordinary differential equations framework for efficient chemical kinetic solvers. *Energy and AI*, 7:100118, 2022.
23. Xing Liu, Chenlin Gong, and Qiang Liu. Flow matching for generative modeling. In *International Conference on Machine Learning*, pages 13876–13896. PMLR, 2022.
24. Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2023.
25. Alexander Tong, Nikolay Malkin, Guillaume Huguet, Yanlei Zhang, Jarrid Rector-Brooks, Kilian Fatras, Guy Wolf, and Yoshua Bengio. Improving and generalizing flow-based generative models with minibatch optimal transport. *arXiv preprint arXiv:2302.00482*, 2023.
26. Xin Li, Jingdong Zhang, Qunxi Zhu, Chengli Zhao, Xue Zhang, Xiaojun Duan, and Wei Lin. From Fourier to Neural ODEs: Flow matching for modeling complex systems. *arXiv preprint arXiv:2402.11566*, 2024.
27. Vladimir I Arnold. *Mathematical methods of classical mechanics*. Springer, 1989.
28. Bengt Fornberg. Generation of finite difference formulas on arbitrarily spaced grids. *Mathematics of Computation*, 51(184):699–706, 1988.
29. Lloyd N Trefethen. *Spectral methods in MATLAB*. SIAM, 2000.
30. K.E. Atkinson. *An introduction to numerical analysis*. John Wiley and Sons, USA, 1989.
31. E. Hairer, S. P. Nørsett, and G. Wanner. *Solving ordinary differential equations I. Non-stiff problems*, volume 18 of *Comput. Math.* Springer, Berlin, Second Revised Edition, 1993.
32. K. E. Brenan, S. L. Campbell, and L. R. Petzold. *Numerical solution of initial-value problems in differential-algebraic equations*. SIAM Classics in Appl. Math. SIAM, Philadelphia, Second Edition, 1996.
33. Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

A Algorithms for training the FNODE

Algorithm 1 Training Algorithm for FNODE without Constraints

Initialize: FNODE network $f(\cdot, \boldsymbol{\Theta})$ predicting accelerations
Input: Ground truth state-acceleration pairs $\mathcal{D} = \{(\mathbf{S}_i, \mathbf{a}_i)\}_{i=0}^{T-1}$ where $\mathbf{S}_i \in \mathbb{R}^{2n_b}$ and $\mathbf{a}_i \in \mathbb{R}^{n_b}$, optimizer and its settings
for each epoch $e = 1, 2, \ldots, E$ **do**
    for each training sample $(\mathbf{S}_i, \mathbf{a}_i) \in \mathcal{D}$ **do**
        Prepare input state $\mathbf{S}_i = [\mathbf{s}_i^{\text{pos}}, \mathbf{s}_i^{\text{vel}}]$ and target acceleration $\mathbf{a}_i$
        Predict Acceleration: $\hat{\mathbf{a}}_i = f(\mathbf{S}_i, \boldsymbol{\Theta})$
        Compute loss $L = \|\mathbf{a}_i - \hat{\mathbf{a}}_i\|_2^2$
        Backpropagate the loss to compute gradients $\nabla_{\boldsymbol{\Theta}} L$
        Update the parameters using optimizer: $\boldsymbol{\Theta} = \text{Optimizer}(\boldsymbol{\Theta}, \nabla_{\boldsymbol{\Theta}} L)$
    Decay the learning rate using exponential schedule
Output: Trained FNODE $f(\cdot, \boldsymbol{\Theta}^*)$

Algorithm 2 Training Algorithm for FNODE with Constraints using Only Minimal Coordinates

Initialize: FNODE network $f(\cdot, \boldsymbol{\Theta})$ for minimal coordinates predicting accelerations; Choose integrator Φ, uses prior knowledge of constraint equation ϕ and the minimal coordinates.
Input: Minimal coordinate state-acceleration pairs $\mathcal{D}^M = \{(\mathbf{S}_i^M, \mathbf{a}_i^M)\}_{i=0}^{T-1}$ where $\mathbf{S}_i^M \in \mathbb{R}^{2m}$ and $\mathbf{a}_i^M \in \mathbb{R}^m$, constraint equations φ, optimizer and its settings
for each epoch $e = 1, 2, \ldots, E$ **do**
    for each training sample $(\mathbf{S}_i^M, \mathbf{a}_i^M) \in \mathcal{D}^M$ **do**
        Prepare minimal state $\mathbf{S}_i^M = [\mathbf{s}_i^{M,\text{pos}}, \mathbf{s}_i^{M,\text{vel}}]$ and target acceleration $\mathbf{a}_i^M$
        Predict Minimal Acceleration: $\hat{\mathbf{a}}_i^M = f(\mathbf{S}_i^M, \boldsymbol{\Theta})$
        Compute minimal loss $L = \|\mathbf{a}_i^M - \hat{\mathbf{a}}_i^M\|_2^2$
        Backpropagate the loss to compute gradients $\nabla_{\boldsymbol{\Theta}} L$
        Update the parameters using optimizer: $\boldsymbol{\Theta} = \text{Optimizer}(\boldsymbol{\Theta}, \nabla_{\boldsymbol{\Theta}} L)$
    Decay the learning rate using exponential schedule
Output: Trained FNODE $f(\cdot, \boldsymbol{\Theta}^*)$ for minimal coordinates

B Double Pendulum Hamiltonian Formulation

For completeness, we outline the Hamiltonian derivation of the double pendulum dynamics used in Section 3.

The total energy is

$$H(q_1, q_2, p_{\theta_1}, p_{\theta_2}) = T(q_1, q_2, p_{\theta_1}, p_{\theta_2}) + V(q_1, q_2), \tag{30}$$

with kinetic and potential energy

$$T = \tfrac{1}{2} m_1 l_1^2 \dot{\theta}_1^2 + \tfrac{1}{2} m_2 \big(l_1^2 \dot{\theta}_1^2 + l_2^2 \dot{\theta}_2^2 + 2 l_1 l_2 \dot{\theta}_1 \dot{\theta}_2 \cos(\theta_1 - \theta_2)\big), \tag{31}$$

$$V = -m_1 g l_1 \cos\theta_1 - m_2 g \big(l_1 \cos\theta_1 + l_2 \cos\theta_2\big). \tag{32}$$

From Hamilton's equations

$$\dot{q}_i = \frac{\partial H}{\partial p_{\theta_i}}, \qquad \dot{p}_{\theta_i} = -\frac{\partial H}{\partial q_i},$$

we obtain the explicit form of the governing equations:

$$\dot{\theta}_1 = \frac{l_2 p_{\theta_1} - l_1 p_{\theta_2} \cos(\theta_1 - \theta_2)}{l_1^2 l_2 \big[m_1 + m_2 \sin^2(\theta_1 - \theta_2)\big]}, \tag{33}$$

$$\dot{\theta}_2 = \frac{-m_2 l_2 p_{\theta_1} \cos(\theta_1 - \theta_2) + (m_1 + m_2) l_1 p_{\theta_2}}{m_2 l_1 l_2^2 \big[m_1 + m_2 \sin^2(\theta_1 - \theta_2)\big]}, \tag{34}$$

$$\dot{p}_{\theta_1} = -(m_1 + m_2) g l_1 \sin\theta_1 - h_1 + h_2 \sin\big(2(\theta_1 - \theta_2)\big), \tag{35}$$

$$\dot{p}_{\theta_2} = -m_2 g l_2 \sin\theta_2 + h_1 - h_2 \sin\big(2(\theta_1 - \theta_2)\big), \tag{36}$$

with auxiliary terms

$$h_2 = \frac{m_2 l_2^2 p_{\theta_1}^2 + (m_1 + m_2) l_1^2 p_{\theta_2}^2 - 2 m_2 l_1 l_2 p_{\theta_1} p_{\theta_2} \cos(\theta_1 - \theta_2)}{2 l_1^2 l_2^2 \big[m_1 + m_2 \sin^2(\theta_1 - \theta_2)\big]^2}, \tag{37}$$

$$h_1 = \frac{p_{\theta_1} p_{\theta_2} \sin(\theta_1 - \theta_2)}{l_1 l_2 \big[m_1 + m_2 \sin^2(\theta_1 - \theta_2)\big]}. \tag{38}$$

C A slider-crank mechanism, with rigid bodies

Based on the settlement illustrated in Section 3.5, we formulates the equation of motion of slider-crank mechanism:

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

$$M = \begin{bmatrix} M_1 & 0_{3\times 3} & 0_{3\times 3} \\ 0_{3\times 3} & M_2 & 0_{3\times 3} \\ 0_{3\times 3} & 0_{3\times 3} & M_3 \end{bmatrix}, \tag{39}$$

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 below constraints $\Phi : \mathbb{R}^9 \to \mathbb{R}^8$ on the position:

$$\Phi(q) = \begin{bmatrix} x_1 - r\cos(\theta_1) \\ y_1 - r\sin(\theta_1) \\ x_1 + r\cos(\theta_1) - x_2 + l\cos(\theta_2) \\ y_1 + r\sin(\theta_1) - y_2 + l\sin(\theta_2) \\ x_2 + l\cos(\theta_2) - x_3 \\ y_2 + l\sin(\theta_2) - y_3 \\ y_3 \\ \theta_3 \end{bmatrix}. \tag{40}$$

Then the constraints on velocit should be:

$$\Phi_q = \begin{bmatrix} 1 & 0 & r\sin(\theta_1) & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -r\cos(\theta_1) & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & -r\sin(\theta_1) & -1 & 0 & -l\sin(\theta_2) & 0 & 0 & 0 \\ 0 & 1 & r\cos(\theta_1) & 0 & -1 & l\cos(\theta_2) & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & -l\sin(\theta_2) & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & l\cos(\theta_2) & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{41}$$

The external forces vector $F_e \in \mathbb{R}^9$ is formulated as:

$$F_e = \begin{bmatrix} F_{e1} \\ F_{e2} \\ F_{e3} \end{bmatrix}, \tag{42}$$

where:

$$F_{e1} = \begin{bmatrix} 0 \\ 0 \\ \tau - c_{01}\omega_1 \end{bmatrix} \in \mathbb{R}^3,$$

$$F_{e2} = \begin{bmatrix} 0 \\ -c_{12}(\omega_1 - \omega_2) \\ 0 \end{bmatrix} \in \mathbb{R}^3,$$

$$F_{e3} = \begin{bmatrix} -k\Delta x_3 - f - c_{23}\omega_3 - c\dot{x}_3 \\ 0 \\ 0 \end{bmatrix} \in \mathbb{R}^3.$$

The rearranged constraint equations on the acceleration can be formulated from (40):

$$\ddot{x}_1 + r\ddot{\theta}_1\sin(\theta_1) + r\dot{\theta}_1^2\cos(\theta_1) = 0 \tag{43a}$$

$$\ddot{y}_1 - r\ddot{\theta}_1\cos(\theta_1) + r\dot{\theta}_1^2\sin(\theta_1) = 0 \tag{43b}$$

$$\ddot{x}_1 - r\ddot{\theta}_1\sin(\theta_1) - r\dot{\theta}_1^2\cos(\theta_1) - \ddot{x}_2 - 2l\ddot{\theta}_2\sin(\theta_2) - 2l\dot{\theta}_2^2\cos(\theta_2) = 0 \tag{43c}$$

$$\ddot{y}_1 + r\ddot{\theta}_1\cos(\theta_1) - r\dot{\theta}_1^2\sin(\theta_1) - \ddot{y}_2 + 2l\ddot{\theta}_2\cos(\theta_2) - 2l\dot{\theta}_2^2\sin(\theta_2) = 0 \tag{43d}$$

$$\ddot{x}_2 - 2l\ddot{\theta}_2\sin(\theta_2) - 2l\dot{\theta}_2^2\cos(\theta_2) - \ddot{x}_3 = 0 \tag{43e}$$

$$\ddot{y}_2 + 2l\ddot{\theta}_2\cos(\theta_2) - 2l\dot{\theta}_2^2\sin(\theta_2) - \ddot{y}_3 = 0 \tag{43f}$$

$$\dot{\theta}_3 = 0 \tag{43g}$$

$$\ddot{\theta}_3 = 0 \tag{43h}$$

Then we can get the γ_c as follows:

$$\gamma_c = \begin{bmatrix} -r\dot{\theta}_1^2\cos(\theta_1) \\ -r\dot{\theta}_1^2\sin(\theta_1) \\ r\dot{\theta}_1^2\cos(\theta_1) + 2l\dot{\theta}_2^2\cos(\theta_2) \\ r\dot{\theta}_1^2\sin(\theta_1) + 2l\dot{\theta}_2^2\sin(\theta_2) \\ 2l\dot{\theta}_2^2\cos(\theta_2) \\ 2l\dot{\theta}_2^2\sin(\theta_2) \\ 0 \\ 0 \end{bmatrix}. \tag{44}$$