# MODULAR LAGRANGIAN NEURAL NETWORKS: DESIGNING STRUCTURES OF NETWORKS WITH PHYSICAL INDUCTIVE BIASES

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Deep learning struggles at extrapolation in many cases. This issue happens when it comes to untrained data domains or different input dimensions and becomes more common in physical problems. Leveraging physical inductive biases can help relieve this issue and generalise the laws of physics. Based on this idea, we proposed a kind of structural neural network called Modular Lagrangian Neural Networks (ModLaNNs). This model can learn from the dynamics of simpler systems such as three-body systems and extrapolate to more complex ones like multi-body systems, which is not feasible using other relevant physical-informed neural networks. We tested our model on double-pendulum or three-body systems and reached the best results compared with our counterparts. At the same time, we directly applied our trained models to predict the motion of multi-pendulum and multi-body systems, demonstrating the intriguing performance in the extrapolation of our method.

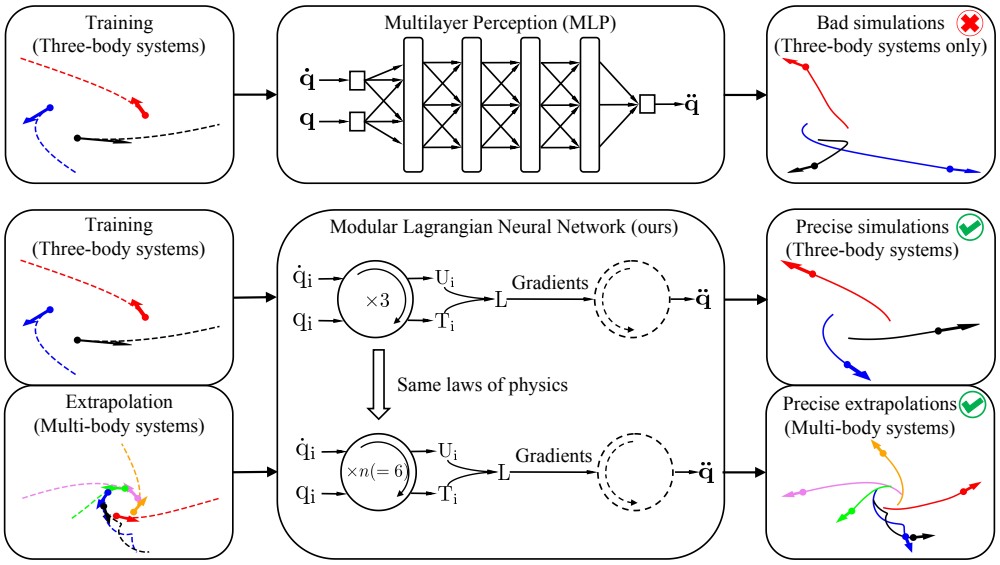



Figure 1: Learning from three-body systems and extrapolating to multi-body ($> 3$) systems. The inputs are the generalised position $\mathbf{q}$ and velocity $\dot{\mathbf{q}}$. The output is the acceleration $\ddot{\mathbf{q}}$. As for training, multilayer perception (MLP) can only accept data with fixed input size. After users train the MLP using a dataset of three-body systems, they can only implement this network to simulate the same system for testing. By contrast, our model can recurrently process input data to obtain the Lagrangian $L$, which describes the evolution of the whole system. This feature supports our model to accept data with different input sizes, extrapolate to multi-body systems with the same laws of physics, and predict the trajectories with high accuracy.

# 1 INTRODUCTION

Deep learning has been widely implemented in various kinds of physical problems such as flow control using reinforcement learning (Verma et al., 2018), nuclear learning and simulation (Pfau et al., 2020; Hermann et al., 2020), and super-resolution of the combustion process (Kipf et al., 2018). These deep-learning models are specialised in generalisation from high-dimensional data. But it is still hard to build neural networks to learn the laws of physics and inefficient for these models to extrapolate to regions where data is insufficient.

The fusion of physical knowledge and neural networks cast a light on this problem, which leads to physics-informed neural networks such as Hamiltonian neural network (HNN) (Greydanus et al., 2019) or Lagrangian Neural Networks (LNN) (Cranmer et al., 2020). This kind of model can learn laws of physics such as the conservation of energy to achieve more stable simulations. Despite this strength, after training, those laws are altogether implicitly encoded in networks, and the input size is also restrictively decided. These drawbacks still prohibit the performance of these models from extrapolation. New models are needed for practical use where the input is varying.

The contributions of this work exist in two aspects. First, we introduced Modular Lagrangian Neural Networks (ModLaNNs), where the network structure integrates with Euler-Lagrangian equations to describe the dynamical systems accurately. Also, our model will recurrently rebuild the energy of every element in a system and then construct the Lagrangian to describe the motion of the whole system. Therefore, the trained model can be applied for simulating a group of systems, as long as the dynamics of these elements are the same (see Figure 1 for graphic illustration). Previous models like HNN or LNN do not support this feature.

# 2 RELATED WORKS

**Neural Networks for Physical Problems.** Researchers have already built different neural networks for physical problems. For example, they implemented networks in precipitation forecasting (Shi et al., 2015), chains motions (de Avila Belbute-Peres et al., 2018), and multi-particles interactions (Battaglia et al., 2016; Kipf et al., 2018), where researchers designed recursive neural networks or graphical neural networks to induce the relations and interactions among objects. But these models do not encode any physical knowledge and can achieve nonphysical results.

**Dynamical System for Neural Networks.** Research also involved explaining neural networks in dynamical-system views and treating the iteration of the neural network in a continuous way (Weinan, 2017). Researchers further utilised the Pontryagin's maximum principle to formulate continuous systems with optimality (Li et al., 2017; Chen et al., 2018). These works expanded the field of deep learning and pointed out a way to combine the paradigm of engineering and statistics. However, it is still difficult to widely implement continuous system models in various types of tasks.

**Physical Inductive Biases.** Recently, several works have focused on designing neural networks with physical inductive biases (Lutter et al., 2018; 2019; Greydanus et al., 2019; Toth et al., 2019; Cranmer et al., 2020). These networks follow the problem-solving process in mechanics, learning specific physical quantities to describe the whole system. In this way, researchers have applied these networks in the control of the robotic arm (Lutter et al., 2018; 2019) and the prediction of the sea surface temperature (De Bézenac et al., 2019), but also achieved important features in mechanics such as energy conservation (Greydanus et al., 2019; Toth et al., 2019; Cranmer et al., 2020). However, their designs restrict that the network can only accept a fixed input size, so the learnt physical properties cannot be reused or extended to other situations.

# 3 PRELIMINARIES

In Lagrangian mechanics, the motion of the whole system is decided by its Lagrangian $L$, which is a combination of the kinetic energy $T$ and the potential energy $U$.

$$L(t, \mathbf{q}, \dot{\mathbf{q}}) \coloneqq T(t, \mathbf{q}, \dot{\mathbf{q}}) - U(t, \mathbf{q}, \dot{\mathbf{q}}), \tag{1}$$

where $t$ is time, $\mathbf{q}$ is the generalised position, and $\dot{\mathbf{q}}$ is the generalised velocity. To describe dynamical systems, we will introduce the Euler-Lagrange equations with or without constraints and then derive the related dynamic equations.

**Euler-Lagrange Equation.** In Calculus of variations, the first-order of necessary conditions for weak extrema leads to *the Euler-Lagrange equation* of the system (Liberzon, 2011):

$$\frac{\partial L(t, \mathbf{q}, \dot{\mathbf{q}})}{\partial \mathbf{q}} = \frac{\mathrm{d}}{\mathrm{d}t} \frac{\partial L(t, \mathbf{q}, \dot{\mathbf{q}})}{\partial \dot{\mathbf{q}}}. \tag{2}$$

From Equation 2, we can get the differential equations concerning $\mathbf{q}$ and $\dot{\mathbf{q}}$, thereby describing the evolution of the system.

For generalisation, we will consider variational problems with constraints in physics. These constraints can be divided into two groups, *integral constraint* in Equation 3 and *non-integral constraint* in Equation 4:

$$\int_a^b M_i(t, \mathbf{q}, \dot{\mathbf{q}}) dt = C_i, \tag{3}$$

$$M_j(t, \mathbf{q}, \dot{\mathbf{q}}) = C_j. \tag{4}$$

where $C_i$ and $C_j$ are constants corresponding to the $i$-th and $j$-th constraint equation $M_i$ and $M_j$. An approach to deal with these constraints is to utilise a Lagrange multiplier $\lambda^\star$, defining augmented Lagrangian $\bar{L}(t, \mathbf{q}, \dot{\mathbf{q}}) = (L + \lambda^\star M)(t, \mathbf{q}, \dot{\mathbf{q}})$ to substitute $L$ in Equation 2. The multiplier $\lambda^\star$ is a number for an integral constraint, and $\lambda^\star$ is a function $\lambda^\star(t)$ for a non-integral constraint. $\lambda^\star = 0$ relates to the situation without any constraint. $\lambda^\star$ can be solved by combining Euler-Lagrangian equation (Equation 2), constraints (Equation 3 and 4) and boundary conditions.

**Dynamic Equations.** When parameters in Equation 2 such as mass and size are not related to time $t$, we can expand this equation as:

$$\frac{\partial L(\mathbf{q}, \dot{\mathbf{q}})}{\partial \mathbf{q}} = \frac{\mathrm{d}}{\mathrm{d}t} \frac{\partial L(t, \mathbf{q}, \dot{\mathbf{q}})}{\partial \dot{\mathbf{q}}} = \frac{\partial^2 L}{\partial \mathbf{q} \partial \dot{\mathbf{q}}} \frac{\mathrm{d}\mathbf{q}}{\mathrm{d}t} + \frac{\partial^2 L}{\partial \dot{\mathbf{q}}^2} \frac{\mathrm{d}\dot{\mathbf{q}}}{\mathrm{d}t}.$$

Then the resultant force and the acceleration can be calculated in Equation 5:

$$\begin{aligned} \mathbf{f} &= \frac{\partial^2 L}{\partial \dot{\mathbf{q}}^2} \ddot{\mathbf{q}} = \left( \frac{\partial L}{\partial \mathbf{q}} - \frac{\partial^2 L}{\partial \mathbf{q} \partial \dot{\mathbf{q}}} \dot{\mathbf{q}} \right), \\ \ddot{\mathbf{q}} &= \left( \frac{\partial^2 L}{\partial \dot{\mathbf{q}}^2} \right)^{-1} \left( \frac{\partial L}{\partial \mathbf{q}} - \frac{\partial^2 L}{\partial \mathbf{q} \partial \dot{\mathbf{q}}} \dot{\mathbf{q}} \right). \end{aligned} \tag{5}$$

## 4 METHOD

With Equation 5, we can describe the motion of a dynamical system once obtaining its Lagrangian. We will propose the Modular Lagrangian Neural Networks (ModLaNN) framework to show how to construct the Lagrangian and how to simulate the dynamics formulated by Equation 5.

Lagrangian is made up of kinetic energy $T$ and potential energy $U$, and correspondingly, ModLaNN contains two branches to regress the forms of these two kinds of energy. In each branch, at least one RelationCell layer is designed to process the input data. Each RelationCell contains one multilayer perception (MLP) for regression and a recurrent structure to accept inputs with different dimensions. The outputs will be the global velocity and the potential field used to calculate $T$ and $U$. Then we can construct the Lagrangian with Equation 1, calculate its derivatives, and derive the accelerations using Equation 5. The whole structure is shown in Figure 2.

The advantages of our framework are that we can directly learn the properties of the elements from the dataset of a type of system and then construct Lagrangian to describe the whole system. Our model can describe other systems with the same elements. This extrapolation feature enables us to process inputs with different sizes rather than a fixed input size.

For optimisation, because our experiments are all for simulations, we calculated accelerations of systems and optimised them using L2-loss

$$\mathcal{L} = \left\| \hat{\ddot{\mathbf{q}}} - \ddot{\mathbf{q}} \right\|_2 + \lambda \left\| \hat{E} - E \right\|_2, \tag{6}$$

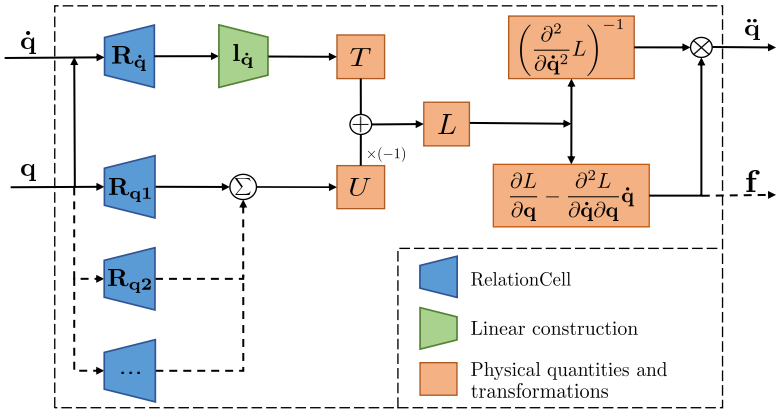
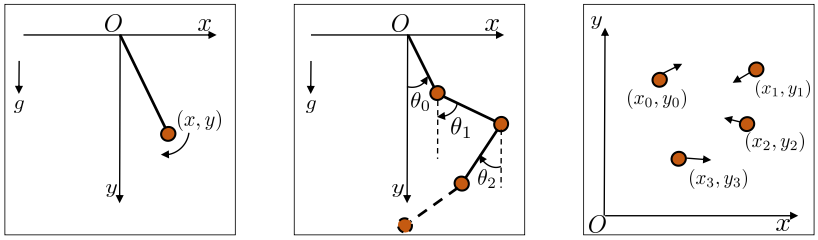



Figure 2: The compositional graph of the Modular Lagrangian Neural Network (ModLaNN). The generalised position $\mathbf{q}$ and generalised velocity $\dot{\mathbf{q}}$ are inputted into two branches and recursively processed. Each branch can contain more than one RelationCell $\mathbf{R}$ to deal with inputs with different combination forms. For example, the input of $\mathbf{R}_{\mathbf{q1}}$ can be position $q_i$ for loop $i$ to regress gravitational potential, and the input of $\mathbf{R}_{\mathbf{q2}}$ may be positions $(q_i, q_j)$ for loop $i$ and $j$ to represent electric potential field. The outputs are used to calculate kinetic energy $T$, potential energy $U$, and Lagrangian $L$. The derivatives of $L$ will be utilised for the accelerations $\ddot{\mathbf{q}}$ or resultant force $\mathbf{f}$ following Equation 5.




(a) Single pendulum system (b) Multi-pendulum system (c) Multi-body system

Figure 3: Three dynamical systems for illustrations. (a) shows a single pendulum system. Both multi-pendulum systems (b) and multi-body systems (c) are highly chaotic systems whose dynamics are hard to simulate (Shinbrot et al., 1992; Vaidyanathan & Volos, 2016).

where $\lambda$ is a hyper-parameter, $\hat{\ddot{\mathbf{q}}}$ and $\hat{E}$ represent the acceleration vector and energy outputted from the network, and $\ddot{\mathbf{q}}$ and $E$ represent those of the ground truth. $\|\hat{E} - E\|_2$ is a regulation term to guarantee the optimisation of two branches can go through the correct gradient direction together.

To introduce with more details, we present three cases to show how to implement our framework.

**Case 1: Single pendulum systems.** A way to involve constraints is to view this system in Cartesian coordinates shown in Figure 3(a), the Lagrangian then will be:

$$\bar{L}(x, y, \dot{x}, \dot{y}) = \frac{1}{2} m\dot{x}^2 + \frac{1}{2} m\dot{y}^2 - mgy + m\lambda M(x, y),$$

where $x$ and $y$ are positions, $\dot{x}$ and $\dot{y}$ are velocities, and $\lambda$ and $M(x, y)$ can be expressed as:

$$\lambda(t) = \frac{1}{2}\left(gy(t) - (\dot{x}^2(t) + \dot{y}^2(t))\right),$$
$$M(x, y) = \left[(x/l)^2 + (y/l)^2\right].$$

Here $l$ is the length of the pendulum and can be calculated directly from $l = \sqrt{x^2 + y^2}$.

Two RelationCell layers are separately used in two branches. One will reconstruct kinetic energy and the other for gravitational potential. One additional branch consists of the dimensionless constraint function $M(x, y)$ multiplying an MLP to model the constraint part. Inputs of this MLP are derivatives of the kinetic and potential energy and related variations so that the base unit of the output is the same as energy.

**Case 2: Multi-pendulum systems.** A multi-pendulum system is $n$ single pendulums that are linked one by one like Figure 3(b), the Lagrangian of this system in polar coordinates is:

$$L(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}) = \left( \sum_{i=0}^{n-1} \sum_{j=0}^{i-1} (\frac{1}{2} m_i l_j^2 \dot{\theta}_j^2 + m_i g l_j \cos\theta_j) + \sum_{i=0}^{n-1} \sum_{j,k=0, j\neq k}^{i-1} m_i l_j l_k \dot{\theta}_j \dot{\theta}_k \cos(\theta_j - \theta_k) \right),$$

where $m_i$, $l_i$, $\theta_i$ and $\dot{\theta}_i$ are mass, pendulum length, angle position and angular velocity related to the $i$-th pendulum. $g$ is the standard gravitational acceleration.

Following the ModLaNN structure, in $T$ branch, one RelationCell $\mathbf{R}_{\dot{\mathbf{q}}}$ will accept pair $(\theta_i, \dot{\theta}_i)$ to regress the velocity expression. Then we will linearly reconstruct these velocities in the global coordinates to obtain the $T$. In $U$ branch, another RelationCell is to obtain the global position with a $for$ loop. Then we can construct $L$ and its derivatives from $T$ and $U$.

**Case 3: Multi-body systems.** For multi-body systems, $n$ particles move within the frame shown in Figure 3(c). The Lagrangian is

$$L(\mathbf{x}, \mathbf{y}, \dot{\mathbf{x}}, \dot{\mathbf{y}}) = \sum_{i=0}^{n-1} \frac{1}{2} m_i (x_i^2 + y_i^2) - \sum_{i=0}^{n-1} \sum_{j=0}^{i} \frac{m_i m_j}{\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}},$$

where $m_i$, $(x_i, y_i)$, and $(\dot{x}_i, \dot{y}_i)$ are mass, positions and velocities related to the $i$-th particle.

For modelling this system, one RelationCell $\mathbf{R}_{\dot{\mathbf{q}}}$ will accept velocities $(\dot{x}_i, \dot{y}_i)$ to calculate $T$ and another RelationCell $\mathbf{R}_{\mathbf{q}}$ will process the positions pair $(\dot{x}_i, \dot{y}_i, \dot{x}_j, \dot{y}_j)$ with $for$ loop to calculate $U$. The outputs will be used to construct $L$ along with its derivatives.

## 5 EXPERIMENTS

Based on the three cases in the last section, we designed three experiments to reveal the performance of our model compared to counterparts and to examine the extrapolation feature of our model. Our model was built using PyTorch (Paszke et al., 2019) and experiments were conducted in Ubuntu 20.04 using single core i7@3.7GHz. These experiments can run without the usage of GPU.

**Ex.1: Pendulum systems with different lengths.** Ex.1 consists of two steps. First, we trained models using datasets with fixed pendulum lengths ($l = 1$ m or $4$ m) and made comparison. Second, we trained our ModLaNN model in the dataset with varying lengths. The training and testing data were collected with lengths in the range of $[1, 7]$ and $[7, 9]$, respectively. For settings, the noise was set to be $0$ or $0.1$. The learning rate was set to be $10^{-2}$ and the training epochs were 2000. For loss function in Equation 6, $\lambda$ was chosen as $10^{-1}$.

Training results are shown in Table 1. Our method outperforms others in the first step. ModLaNN can also handle data with varying lengths in the second step, where the loss is smaller compared with that in the first step. This result can be explained that training with more data also improves the model's performance in return.

Table 1: Final loss of three models after training in Ex.1 ($\sigma = 0.1$).

| | Train Loss | | | Test Loss | | |
|---|---|---|---|---|---|---|
| Type | ModLaNN | HNN (Toth et al., 2019) | Baseline (Hornik et al., 1989) | ModLaNN | HNN | Baseline |
| $l = 1$ | $\mathbf{3.7 \times 10^{-1}}$ | $2.1 \times 10^{0}$ | $3.3 \times 10^{0}$ | $\mathbf{4.2 \times 10^{-1}}$ | $5.9 \times 10^{0}$ | $4.8 \times 10^{0}$ |
| $l = 4$ | $\mathbf{2.6 \times 10^{-3}}$ | $2.9 \times 10^{-1}$ | $3.4 \times 10^{0}$ | $\mathbf{2.3 \times 10^{-3}}$ | $3.5 \times 10^{-1}$ | $4.8 \times 10^{-1}$ |
| $l \in [1, 9]$ | $\mathbf{2.4 \times 10^{-4}}$ | | | $\mathbf{3.3 \times 10^{-4}}$ | | |

For step one, Figure 4 illustrates the simulation performance of different models. The length of the pendulum is 1 m. The dynamics are integrated with models using the ODE solver offered in SciPy (Virtanen et al., 2020). Our model can ensure the smallest mean squared errors (MSEs) in coordinates and energy. HNN also performs well but slightly diverges from the ground truth. Baseline performs the worst, with large MSEs in trajectory and energy.

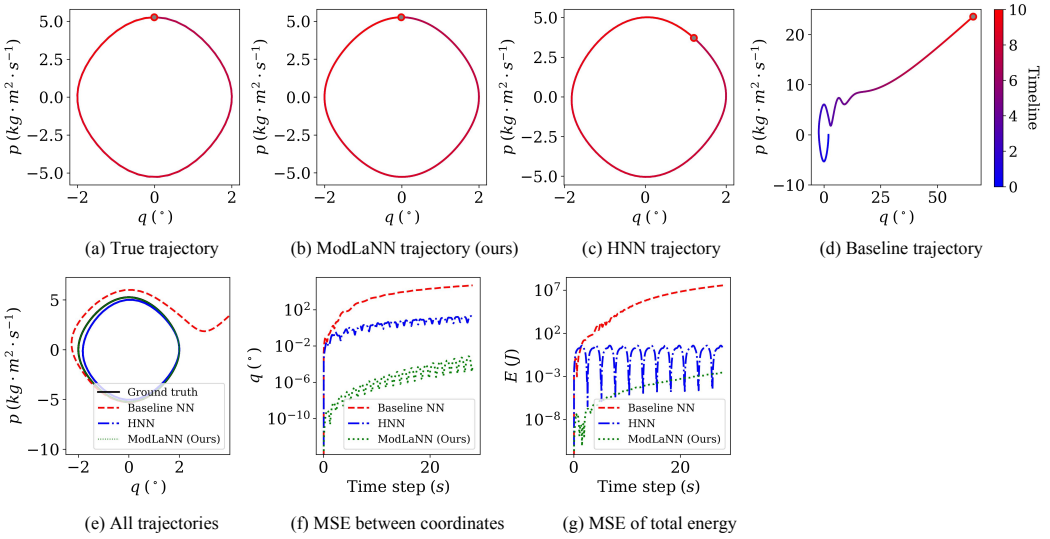



Figure 4: Simulation results of a single pendulum system (best view in colour). Colours represent the change of time marked in the colour bar. Points mark the end of the trajectory. Our prediction (b) is very close to the ground truth (a). The endpoint of HNN's trajectory (c) implies the difference between the HNN simulation and the ground truth. (d) reveals that the baseline's trajectory quickly diverges. (e) puts all trajectories together for better comparison, where HNN's orbit is smaller than the ground truth. (f) and (g) show our model can maintain minor mean squared errors (MSEs) in coordinates and energy.

For step two, Figure 5 reveals results when our model simulates pendulums with different lengths. Our model can achieve stable results even when the length is out of the training range ($[7, 10]$). The constraint part already contained the information of the length, so once trained with enough data, it can remain stable to those domains not covered during training.

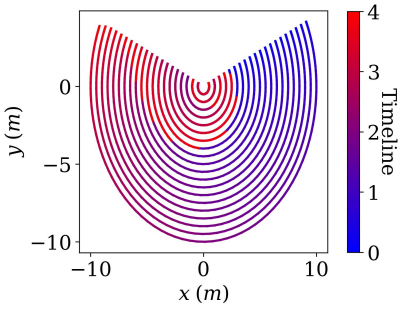



Figure 5: Simulation of pendulum systems with varying lengths (best view in colour). Each curve represents one trajectory of a pendulum. The colour bar marks the time change.

**Ex.2: Double pendulum systems to multi-pendulum systems.** This experiment was divided into two parts. First, we compared the performance of different methods based on double-pendulum datasets. Then we applied our ModLaNN model to multi-pendulum systems to show its ability in extrapolation. The length and mass of each object were set to be 1 m and 1 kg. For training settings, the training epochs were 10000, and the learning rate was chosen the best from $[10^{-4}, 10^{-1}]$.

Training results are shown in Table 2. The training of the other two models remained divergent after great efforts. Figure 6 also clearly shows the training results of three models. Our model can achieve an accurate prediction of the trajectory. The coordinates error and the energy error are all under control, below $10^{-4}$ and $10^{-1}$, respectively. The performances of the other two models are not optimistic, with significant MSEs in coordinates and energy.

For the extension to multi-pendulum systems, we directly loaded the trained model in the first step and simulated a triple-pendulum and a quadruple-pendulum system. The initial conditions were chosen arbitrarily. The results are shown in Figure 7. The trajectories of objects in the pendulum systems are very close to the ground truth, which means our model truly learned the dynamics within systems. MSEs show that the simulation of higher-order pendulum systems diverges with time, which is because of the chaotic property of these systems (Shinbrot et al., 1992). Even slight variances of chaotic systems will lead to extremely different dynamic behaviours.

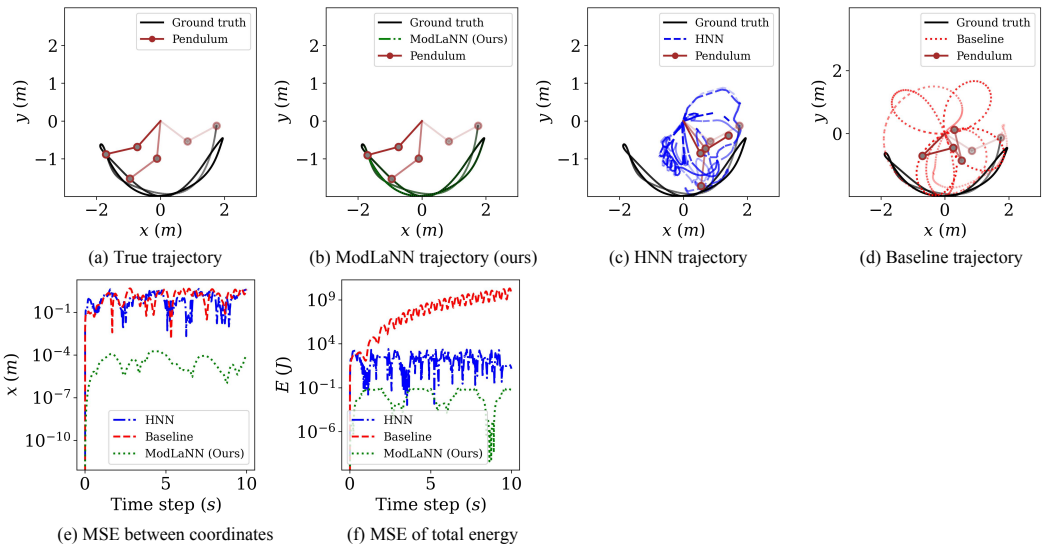



Figure 6: Simulation results of a double pendulum system (best view in colour). Only our model (b) can make precise predictions close to the ground truth (a), while the HNN model (c) and baseline model (d) cannot converge. MSEs related to our model in (e) and (f) are much smaller than those of the other two models.

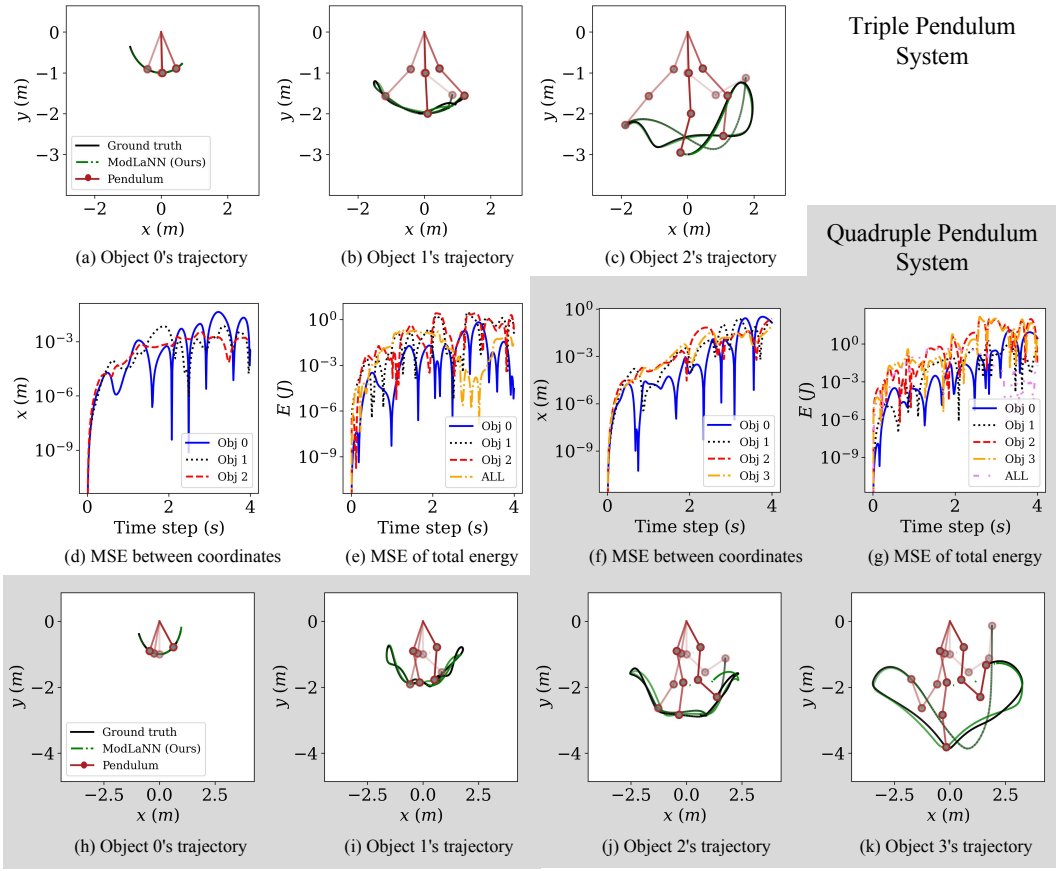



Figure 7: Simulation results of multi-pendulum ($> 2$) systems for extrapolations (best view in colour). The trajectories of the triple pendulum (a-c) and the quadruple pendulum (h-k) are listed compared with the ground truth. MSEs of different components shown in (d-g) indicate that predictions by our model are close to the ground truth.

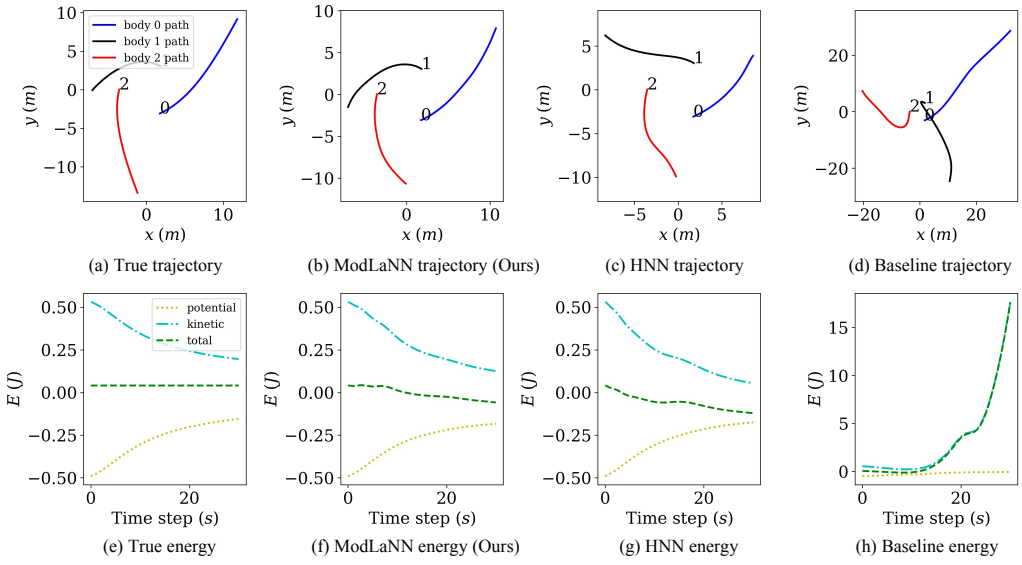
**Ex.3: Three-body systems to multi-body systems.** Multi-body systems are also complicated chaotic systems (Vaidyanathan & Volos, 2016). The procedure of Ex.3 is the same as Ex.2, with the scenario changing to multi-body systems. We first trained three models using the dataset of three-body systems and then applied ModLaNN to multi-body systems for extrapolation. The training epochs were set to be 2000, and the learning rate was chosen the best from $[10^{-4}, 10^{-1}]$.

Table 2 lists the training results and Figure 8 gives a clear illustration of how these three models perform. The first row of Figure 8 demonstrates the trajectory and the second row reveals the change of energy. Even though the training and testing losses of every model are lower than $10^{-3}$ in Table 2, the prediction of the baseline network diverges fast from the ground truth, causing an intensive increase in kinetic energy and total energy. ModLaNN and HNN can ensure that total energy is roughly conservative, while the trajectory of body 1 implies that our ModLaNN model outperforms HNN.

We can also see from Figure 9 for the extrapolation effects in 4, 5, 6-body systems using ModLaNN. Each body's trajectory and each part of the energy can be close to the ground truth.

Table 2: Final loss after training in Ex.2 and Ex.3.

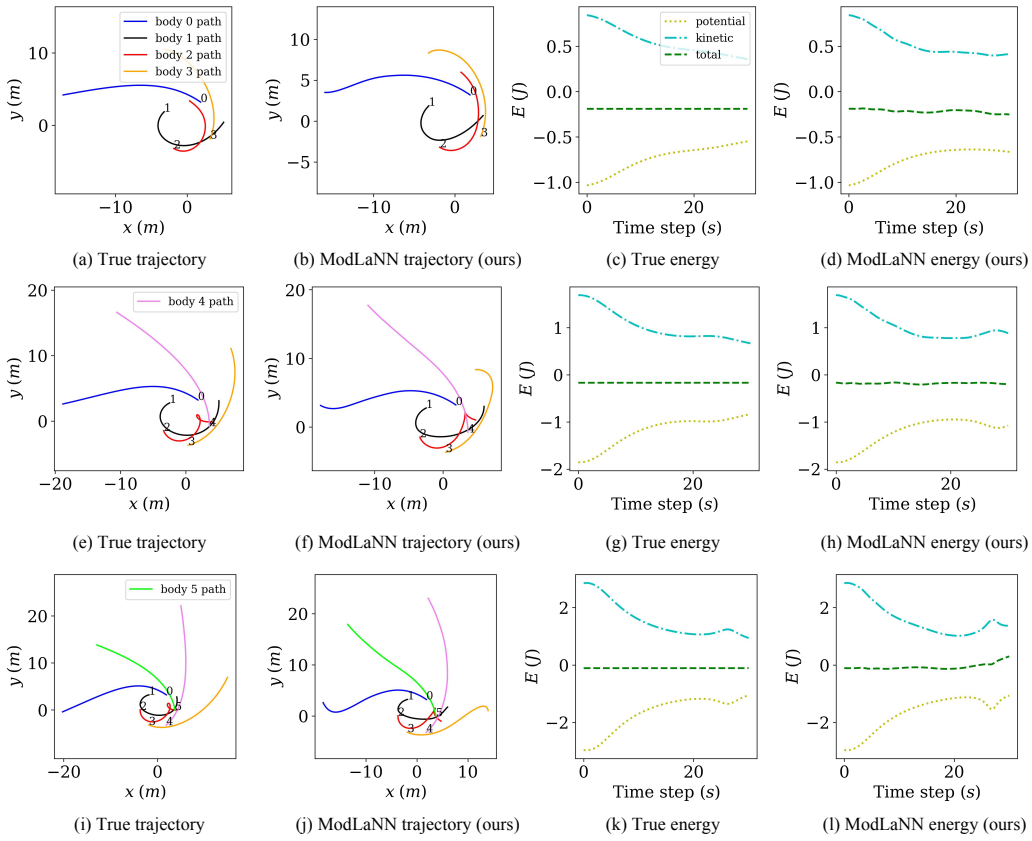
| | Train Loss | | | Test Loss | | |
|---|---|---|---|---|---|---|
| Type | ModLaNN | HNN (Toth et al., 2019) | Baseline (Hornik et al., 1989) | ModLaNN | HNN | Baseline |
| Ex.2 | $\mathbf{4.0 \times 10^{-2}}$ | $1.0 \times 10^{1}$ | $9.0 \times 10^{-4}$ | $\mathbf{4.0 \times 10^{-2}}$ | $2.3 \times 10^{3}$ | $4.4 \times 10^{2}$ |
| Ex.3 | $\mathbf{7.7 \times 10^{-5}}$ | $2.2 \times 10^{-4}$ | $4.5 \times 10^{-4}$ | $\mathbf{8.3 \times 10^{-5}}$ | $3.1 \times 10^{-4}$ | $4.7 \times 10^{-4}$ |




(a) True trajectory (b) ModLaNN trajectory (Ours) (c) HNN trajectory (d) Baseline trajectory

(e) True energy (f) ModLaNN energy (Ours) (g) HNN energy (h) Baseline energy

Figure 8: Simulation results of a three-body system (best view in colour). Our prediction (b) here is closest to the ground truth (a). The trajectory of object 1 predicted by HNN (c) is away from the ground truth. The baseline performance is the worst, with the orbit (d) and energy (h) divergence. Our ModLaNN (f) and HNN model (g) are also able to predict the change of the energy, while our method can achieve more precise predictions to the ground truth (e).




(a) True trajectory (b) ModLaNN trajectory (ours) (c) True energy (d) ModLaNN energy (ours)

(e) True trajectory (f) ModLaNN trajectory (ours) (g) True energy (h) ModLaNN energy (ours)

(i) True trajectory (j) ModLaNN trajectory (ours) (k) True energy (l) ModLaNN energy (ours)

Figure 9: Simulation results of multi-body ($> 3$) systems for extrapolations (best view in colour). The trajectories corresponding to 4, 5, 6-body systems (b, f, j) simulated using the ModLaNN model can grasp the main motion features similar to the ground truth (a, e, i). Our model can also predict energy change and retain the conservation of total energy (d, h, l) like the ground truth (c, g, k). As a result, our model can make great extensions to multi-body systems with the same law of physics, predict the correct motion and ensure the stability of the energy.

## 6 Conclusion

We proposed a kind of structural neural network called Modular Lagrangian Neural Networks. Compared with relevant models such as LNN and HNN that directly output Lagrangians or Hamiltonians, our model aims at learning to regress different kinds of energy for objects in a system and then constructs the Lagrangian to describe the dynamics of this system. In this way, our models can simulate the dynamics of more complicated systems as long as they consist of the same objects after training. Three experiments were conducted to reveal the importance of this feature, where our model has achieved the best results during training and testing than the counterparts. At the same time, these trained models could directly make generalisations in multi-pendulum and multi-body systems with robust extrapolation performance.

## References

Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, and Koray kavukcuoglu. Interaction networks for learning about objects, relations and physics. In *Advances in Neural Information Processing Systems (NeurIPS 2016)*, pp. 4509–4517, 2016.

Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K. Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems (NeurIPS 2018)*, pp. 6572–6583, 2018.

Miles Cranmer, Sam Greydanus, Stephan Hoyer, Peter Battaglia, David Spergel, and Shirley Ho. Lagrangian neural networks. In *International Conference on Learning Representations Workshop on Integration of Deep Neural Models and Differential Equations*, 2020.

Filipe de Avila Belbute-Peres, Kevin Smith, Kelsey Allen, Josh Tenenbaum, and J Zico Kolter. End-to-end differentiable physics for learning and control. *Advances in Neural Information Processing Systems (NeurIPS 2018)*, 31:7178–7189, 2018.

Emmanuel De Bézenac, Arthur Pajot, and Patrick Gallinari. Deep learning for physical processes: Incorporating prior scientific knowledge. *Journal of Statistical Mechanics: Theory and Experiment*, 2019(12):124009, 2019.

Sam Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian neural networks. In *Advances in Neural Information Processing Systems (NeurIPS 2019)*, 2019.

Jan Hermann, Zeno Schätzle, and Frank Noé. Deep-neural-network solution of the electronic schrödinger equation. *Nature Chemistry*, 12(10):891–897, 2020.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. Neural relational inference for interacting systems. In *International Conference on Machine Learning*, pp. 2688–2697. PMLR, 2018.

Qianxiao Li, Long Chen, Cheng Tai, and E Weinan. Maximum principle based algorithms for deep learning. *The Journal of Machine Learning Research*, 18(1):5998–6026, 2017.

Daniel Liberzon. *Calculus of variations and optimal control theory*. Princeton university press, 2011.

Michael Lutter, Christian Ritter, and Jan Peters. Deep lagrangian networks: Using physics as model prior for deep learning. In *International Conference on Learning Representations*, 2018.

Michael Lutter, Kim Listmann, and Jan Peters. Deep lagrangian networks for end-to-end learning of energy-based control for under-actuated systems. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2019)*, pp. 7718–7725. IEEE, 2019.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems (NeurIPS 2019)*, pp. 8024–8035. Curran Associates, Inc., 2019.

David Pfau, James S Spencer, Alexander GDG Matthews, and W Matthew C Foulkes. Ab initio solution of the many-electron schrödinger equation with deep neural networks. *Physical Review Research*, 2(3):033429, 2020.

Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-kin Wong, and Wang-chun Woo. Convolutional lstm network: a machine learning approach for precipitation nowcasting. In *International Conference on Neural Information Processing Systems (NeurIPS 2015)*, pp. 802–810, 2015.

Troy Shinbrot, Celso Grebogi, Jack Wisdom, and James A Yorke. Chaos in a double pendulum. *American Journal of Physics*, 60(6):491–499, 1992.

Peter Toth, Danilo J Rezende, Andrew Jaegle, Sébastien Racanière, Aleksandar Botev, and Irina Higgins. Hamiltonian generative networks. In *International Conference on Learning Representations*, 2019.

Sundarapandian Vaidyanathan and Christos Volos. *Advances and applications in chaotic systems*, volume 636. Springer, 2016.

Siddhartha Verma, Guido Novati, and Petros Koumoutsakos. Efficient collective swimming by harnessing vortices through deep reinforcement learning. *Proceedings of the National Academy of Sciences*, 115(23):5849–5854, 2018.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

E Weinan. A proposal on machine learning via dynamical systems. *Communications in Mathematics and Statistics*, 5(1):1–11, 2017.

# A APPENDIX

## A.1 EX.1: SINGLE PENDULUM SYSTEMS

Figure 10 is an addition of Figure 4 in Ex.1. It shows the performances of different models in simulating single pendulum systems. Each model is separately trained twice using two datasets. The only difference between datasets is the pendulum's length.

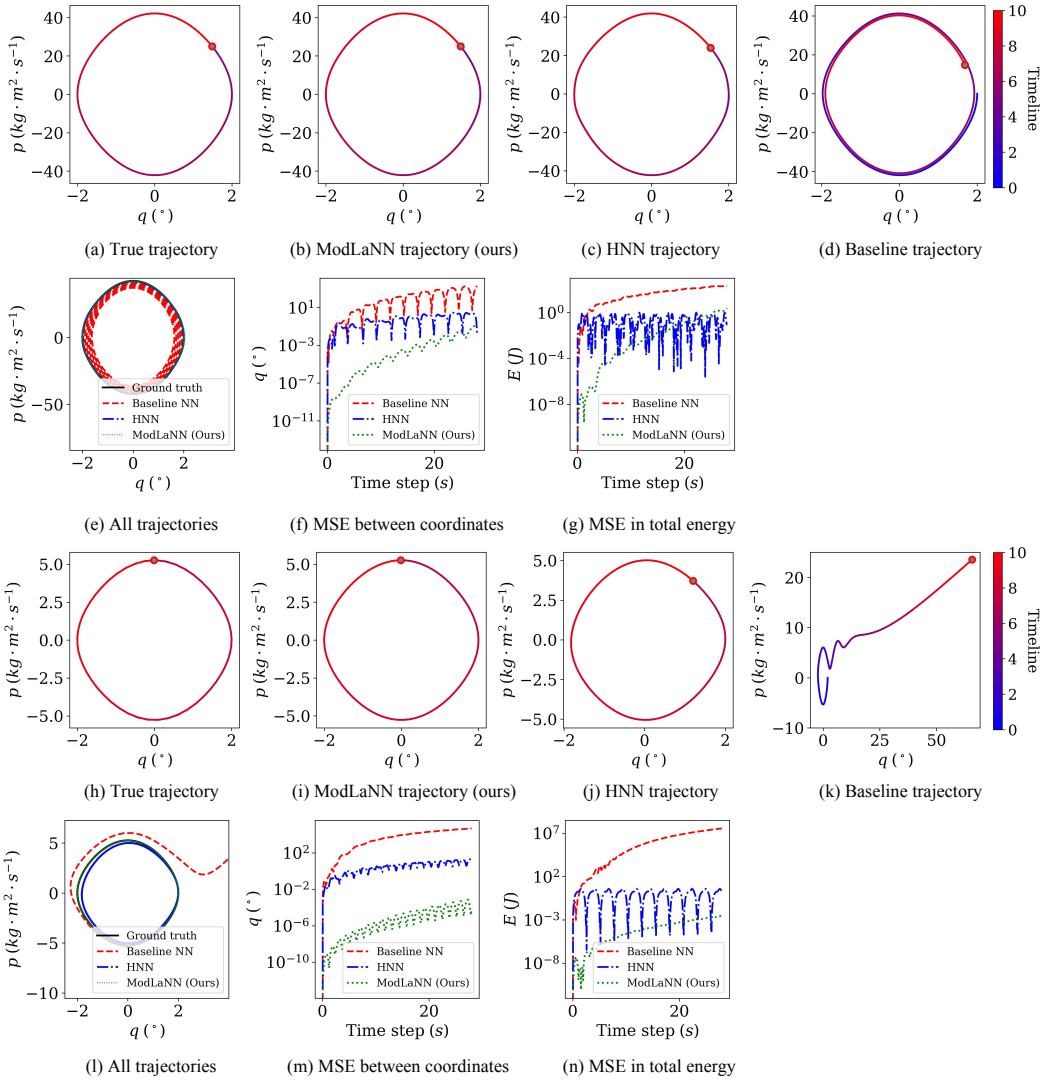
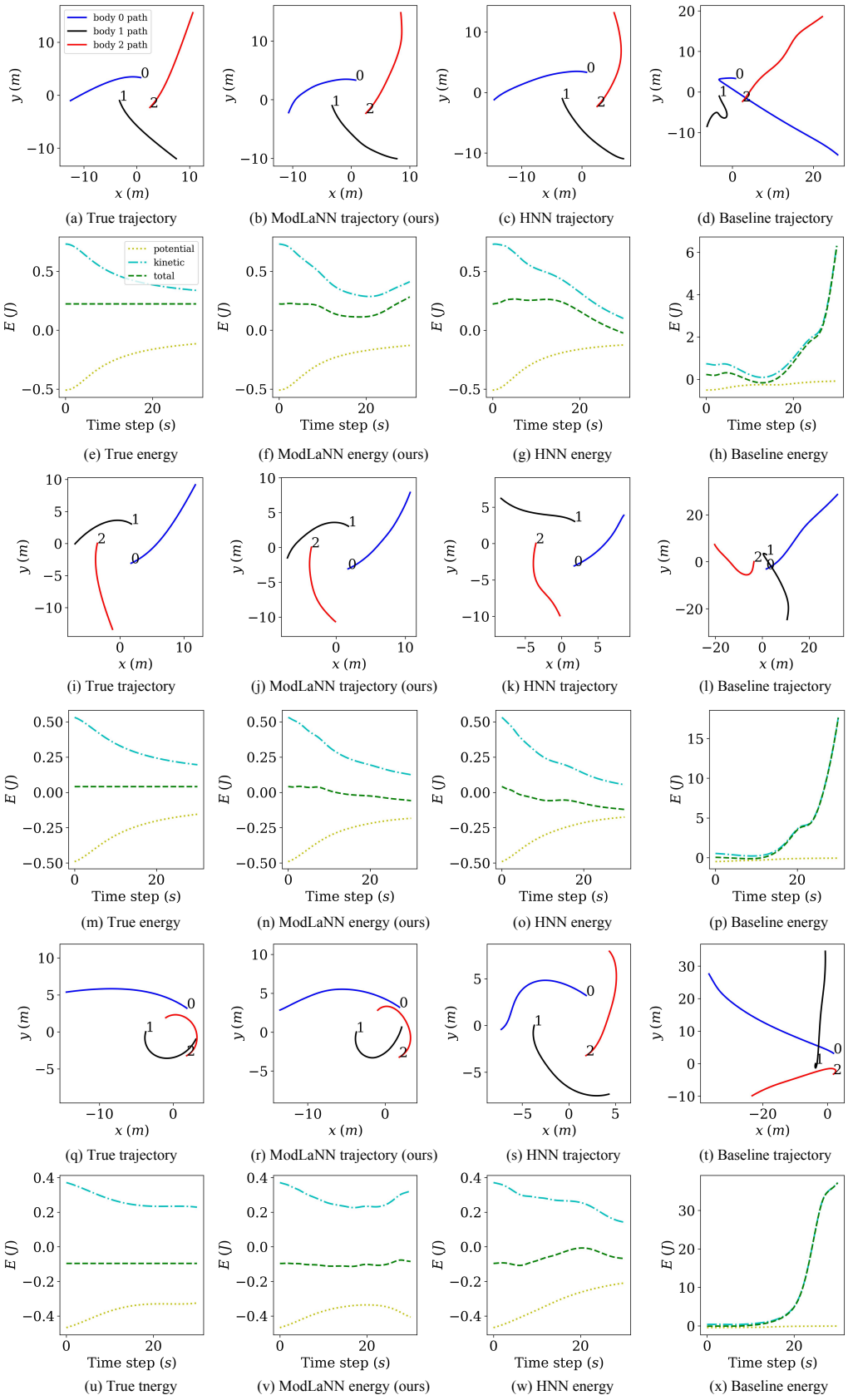



Figure 10: Training results of pendulum systems with different datasets. In (a-g), models are trained using the dataset in which the pendulum's length is 4 m. The performances of HNN (c) and baseline model (d) are close to the ground truth (a). In (h-n), models are trained when the length is 1 m. More significant divergences exist in trajectories predicted by HNN (j) and baseline model (k) compared with the ground truth (h). By contrast, our model made precise predictions in both situations (b) and (i). This difference can be seen using MSEs in (f-g) and (m-n). These results show that HNN and baseline models are sensitive to the chosen datasets, while our models are robust.

## A.2 EX.3: MULTI-BODY SYSTEMS

Figure 11 is an addition of Figure 8 in Ex.3. In this figure, we show the simulation results of three-body systems with three different initial conditions. The results contain the trend of trajectories and energy. Our models get better predictions in all three situations and capture the trend of movements of the whole system.




(a) True trajectory (b) ModLaNN trajectory (ours) (c) HNN trajectory (d) Baseline trajectory

(e) True energy (f) ModLaNN energy (ours) (g) HNN energy (h) Baseline energy

(i) True trajectory (j) ModLaNN trajectory (ours) (k) HNN trajectory (l) Baseline trajectory

(m) True energy (n) ModLaNN energy (ours) (o) HNN energy (p) Baseline energy

(q) True trajectory (r) ModLaNN trajectory (ours) (s) HNN trajectory (t) Baseline trajectory

(u) True tnergy (v) ModLaNN energy (ours) (w) HNN energy (x) Baseline energy

Figure 11: Simulation results of three-body systems in three cases. Three models make their predictions in three initial conditions (a-h), (i-p), and (q-x). Within these cases, our model made the most precise predictions of the change of motion and energy.

### A.3 ARCHITECTURE DETAILS

Our model is implemented and trained using PyTorch (Paszke et al., 2019). The model contains two input branches for calculating the kinetic energy and the potential energy. Each branch consists of several RelationCells, and Each RelationCell includes an MLP network with three fully connected layers and two $tanh$ activation layers. The hidden dimension of the MLP is set to be 16 or 50. For kinetic energy $T$, the branch will accept data of each object with a $for$ loop. Then the outputs will be linear reconstructed to generate the velocity in global coordinates and lead to $T$. For potential energy $U$, a different RelationCell with the same setting will accept the combinations of positions to reconstruct the potential field. Then we will add the outputs all together to get $U$. Lagrangian is calculated by $L = T - U$, and the differentiations of $L$ will be used to generated acceleration through equation 5.