# Learning Modular Simulations for Homogeneous Systems

**Anonymous Author(s)**
Affiliation
Address
email

## Abstract

Complex systems are often decomposed into modular subsystems for engineering tractability. Although various equation based white-box modeling techniques make use of such structure, learning based methods have yet to incorporate these ideas broadly. We present a modular simulation framework for modeling *homogeneous* multibody dynamical systems, which combines ideas from graph neural networks and neural differential equations. We learn to model the individual dynamical subsystem as a neural ODE module. Full simulation of the composite system is orchestrated via spatio-temporal message passing between these modules. An arbitrary number of modules can be combined to simulate systems of a wide variety of coupling topologies. We evaluate our framework on a variety of systems and show that message passing allows coordination between multiple modules over time for accurate predictions and in certain cases, enables zero-shot generalization to new system configurations. Furthermore, we show that our models can be transferred to new system configurations with lower data requirement and training effort, compared to those trained from scratch.

## 1 Introduction

Modeling and simulation are key enablers in engineering as viable alternatives to real world systems. Simulation allows for the testing and evaluation of large/complex systems in a risk-free environment and has been of use in many disciplines. Simulation also plays a key role in domains such as deep reinforcement learning, which requires a large volume of interactions with the environment to learn effective policies. However, traditional simulators often require a large amount of engineering effort to create and can be computationally intensive when high fidelity predictions are needed. This has created interest in data-driven simulations, where machine learning algorithms are used to learn system dynamics directly from observations [1, 2, 3, 4, 5]. While machine learning algorithms excel at learning correlations from data, such end-to-end learning approaches result in models that cannot be reused for different system configurations. Additionally, end-to-end learning also requires large amounts of data and training effort when trying to model real-world systems.

In this paper we propose an ML based framework to learn and compose modular simulators. We build upon the observations that complex systems comprises of multiple modules (e.g., multibody systems), where each module evolves according to its own set of dynamical equations with corresponding inputs and outputs [6]. Examples include multiple pendulums connected via a single thread and multiple quadrotors carrying a large structure or a beam. Instead of modeling the entire system jointly, it might be more beneficial to model individual subsystems (such as a single pendulum or one quadrotor) and then consider interaction effects. The ability to harness such modularity can help reduce the sample complexity for machine learning as well create transferable subsystems that enable rapid prototyping.

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

The technical methodology we describe here builds upon a class of models referred to as *mechanistic models*. Mechanistic models aim to introduce inductive biases related to the underlying mechanisms of the systems being modeled, which are in turn informed by domain knowledge. Some examples are neural networks motivated by differential equations [7, 8, 9, 10, 11, 12, 13], physics-informed neural networks [14, 15, 16, 17, 18, 19, 20] etc. As mechanistic models encode structural knowledge of the system at hand, they can be used to learn modular simulations, as opposed to treating the entire system as a singular block.

In this work, we propose an approach to simulate homogeneous systems from data. Under the assumption that the systems of interest can be modeled as ordinary differential equations, we present a class of neural ODEs which we call *message-passing neural ODEs* (MP-NODE). Instead of modeling the entire graph of a system through ODEs, the message passing neural ODE only learns the model of the subsystem that forms represents a node in the graph. In order to enable interaction between the nodes, the MP-NODE contains additional message variables along with the state variables. The entire system is thus composed of several MP-NODE modules with the weights shared between them, and the coordination as well as ability to account for model approximation errors over time is learnt through the messages. We show that MP-NODEs allow for expressive and efficient modular simulations which can be fine-tuned to new system configurations significantly faster than training a model from scratch. Our key contributions are listed below.

1. We present the message-passing neural ODE (MP-NODE), an augmented neural ODE formulation aimed at enabling interaction between multiple components through messages.
2. We test the MP-NODE on various connected systems and demonstrate its ability to learn complex dynamics and generalize to unseen system configurations.
3. We also show that this modular framework allows finetuning for new configurations with significantly lower training effort than training models from scratch.

## 2 Related Work

Lets assume $X_t \in \mathcal{X}$ is the state of the world at time $t$. Then, dynamical stepping over $T$ timesteps with optional control input $U_t \in \mathcal{U}$ gives a trajectory of states $\mathbf{X}_{t_{0:T}} = (X_{t_0}, \ldots, X_{t_T})$. A *simulator* $s : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$, models the dynamics by mapping previous states and control inputs to future states. Traditional simulators often use a modular set of equations to define the dynamics information as differential equations or differentiable algebraic equations and use appropriate numerical solvers as the update mechanism.

For data-driven simulations, we parameterize the dynamics of the simulator as $f_\theta : \mathcal{X} \times \mathcal{U} \to \mathcal{Y}$. Here, the parameters $\theta$ can be optimized for some training objective. The semantics of the dynamics information $Y \in \mathcal{Y}$ is governed by an update function, usually an integrator.

Learning such simulations from data has been quite important in fields like physics [21] and graphics [22]. Learned simulations can often be faster and more efficient for complex phenomenon prediction than just equation based simulators [23, 24].

Combining graph neural networks (GNN) [25, 26, 27] and differential equations [7, 8] has been explored in Poli et al. [28], Huang et al. [29] and Zang and Wang [30]. The dynamics of their test systems are relatively simple. Importantly, there is no study of transferability of the learned dynamics to changing graph structure. More recently, graph neural networks have been applied in the context of particle simulations by Sanchez-Gonzalez et al. [31] and mesh-based simulations by Pfaff et al. [32]. Moreover, unlike other graph neural net based dynamics learning works [28, 30] they also use edge based features. However, these largely ignore the existence of control inputs, limiting their usefulness for reinforcement learning and control applications.

Some work in the past has examined the idea of message passing for continuous time Bayesian networks [33], or as a combination of generative graphical models and GNNs [34]. The focus of these works was to model full system dynamics for a particular graph structure, where messages play a role in computing the interactions, but do not evaluate transfer performance to different graph structures. We distinguish our work from such prior methods primarily through the fact that our method is generalizable between different graph structures. An approach that is close to ours from the neural ODE body of work would be the augmented neural ODEs[8] which also contain extra dimensions

in the state, although these additional variables are zero-valued. In our method, we repurpose those extra dimensions to explicitly pass messages across multiple neural ODEs as per the graph structure.

# 3 Methodology

The proposed framework tackles the problem of learning modular simulations where (1) smaller subsystems are individually modeled and (2) can be composed to simulate larger systems and processes. Specifically, given observations from a complex system, the methodology seeks to learn both the individual simulation module as well the message passing mechanism that would best explain the data traces.

At the core of this method are Neural Differential Equations that use messages to coordinate amongst themselves. Such message passing enables the framework to model various interaction effects between the modules as well as account for errors due to model approximations and time discretization. The next subsection reviews Neural ODEs. We then present our MP-NODE which extends neural ODEs to incorporate message passing. Note that we assume that the graphs are homogeneous and the underlying graph structure is given. However, we do show that the learnt Neural ODEs can be reused across different given graph topologies. Handling heterogeneous subsystems along with learning the underlying graph structure is a more challenging problem and will be the focus of future work.

## 3.1 Background: Neural Differential Equations

Many continuous-time dynamical systems which can be found in physics (oscillators, mechanical systems), biology (population dynamics) etc. are best modeled by ordinary differential equations. Typically, such a system can be described using an ODE as $\dot{x} = f(x, u, t)$. A data-driven way of learning a simulation of this system would thus be to estimate $\theta$ such that $\dot{x} = f_\theta(x, u, t)$ fits a set of given observations.

Neural ordinary differential equations [7] are a class of neural networks that link the concepts of residual networks and dynamical systems. Given a set of observations of the system state over time, under the assumption that the system dynamics can be described through an ODE, neural ODEs can be used to approximate the time derivative directly. By integrating the model output using a black-box ODE solver for an arbitrarily long timestep and backpropagating appropriately, the model parameters $\theta$ can be updated to minimize the distance between the predicted states (Equation 2) and the observed ones.

$$\frac{d\hat{\mathbf{x}}(t)}{dt} = \mathbf{f}_\theta\left(\mathbf{x}(t), \mathbf{u}(t), t\right) \tag{1}$$

$$\hat{\mathbf{x}}(t+1) = \mathbf{x}(t) + \int_t^{t+1} \mathbf{f}_\theta\left(\mathbf{x}(t), \mathbf{u}(t), t\right) dt \tag{2}$$

## 3.2 Message Passing with Neural ODEs (MP-NODE)

Now, let us consider a connected system that evolves as $\frac{dX}{dt} = f(G, X, U, t)$, where $X(t) \in \mathbb{R}^{n \times d}$ represents the entire system state, constituting of $n$ connected nodes whose state is of length $d$ each. $G = \{\mathcal{V}, \mathcal{E}\}$ captures the graph structure and how the nodes are connected to each other, which we assume is known. $\mathcal{N}(k)$ refers to the neighbors of a node $k$ in the graph.

While neural ODEs have been successfully used to model dynamical systems, modeling large systems with complex connectivity and dynamics is still challenging and computationally intensive. Furthermore, even the slightest change to the connectivity of the system necessitates retraining of the entire system dynamics model. On the other hand, connected systems can often be decomposed into repeatable subsystems that communicate with each other, hence it can be more efficient to learn these individual blocks rather than attempting to model the connected system as a whole.

The first key aspect of our framework is that the focus is placed on the subsystems rather than the entire system. For a given homogeneous network, we essentially only learn the model corresponding to a single subsystem, or node. When orchestrating full simulation of the system, the weights are shared across all individual nodes.

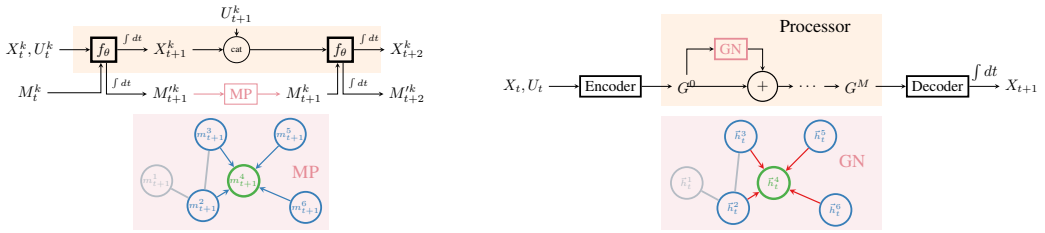



Figure 1: (Left) MP-NODE architecture: Individual nodes can simulate independently and coordinate via message passing. (Right) Learning to Simulate (L2S) architecture without edge features.

Secondly, due to this modular nature, the individual nodes need to learn to coordinate with each other and understand how their long term evolution changes based on interaction according to the graph structure. We facilitate such communication between the nodes by augmenting each subsystem state with additional variables which we call *messages* with the augmented state for node $k$ being $\mathbf{X}^k = \begin{bmatrix} \mathbf{x}^k \\ \mathbf{m}^k \end{bmatrix}$. With the neural ODE operating on the augmented state, each neural subsystem outputs a message along with its predicted state at each timestep. Similarly, each node also receives a message at the input, which results from an aggregation operation between all output messages of the node neighbors which are identified according to $G$. We call this model the *message passing neural ODE* (MP-NODE) and describe the structure in Figure 1(Left). The full formulation of the MP-NODE can thus be written as below, where $\mathbf{m}'_k(t)$ is the input message at node $k$ and $\mathbf{m}_k$ is the output message.

$$\mathbf{m}'_k(t) = \text{MEAN}\left(\{\mathbf{m}_j(t), j \in \mathcal{N}(k)\}\right) \tag{3}$$

$$\frac{d}{dt}\begin{bmatrix} \mathbf{x}_k(t) \\ \mathbf{m}_k(t) \end{bmatrix} = \mathbf{f}_\theta\left(\begin{bmatrix} \mathbf{x}_k(t) \\ \mathbf{m'}_k(t) \end{bmatrix}, \mathbf{u}(t)\right) \tag{4}$$

$$\hat{\mathbf{X}}_k(t+dt) = \text{ODESOLVE}\Big(\mathbf{f}_\theta(\begin{bmatrix} \mathbf{x}_k(t) \\ \mathbf{m'}_k(t) \end{bmatrix}, \mathbf{u}(t), dt)\Big) \tag{5}$$

While the outputs from the ODE solver contain both the predicted state and a message, the supervision and loss computation only happen on the predicted states, allowing the messages to evolve freely. Over time, the model is able to use these additional variables to potentially encode the effects of different aggregations and interactions between neighbors, finally reaching an optimum configuration that is applicable for all the nodes. Learning such a generalizable subsystem allows the MP-NODE framework to be directly applicable for another system configuration with higher or lower number of nodes, as opposed to attempting to learn dynamics for the entire system state such as is done with a conventional neural ODE, which needs retraining for new configurations.

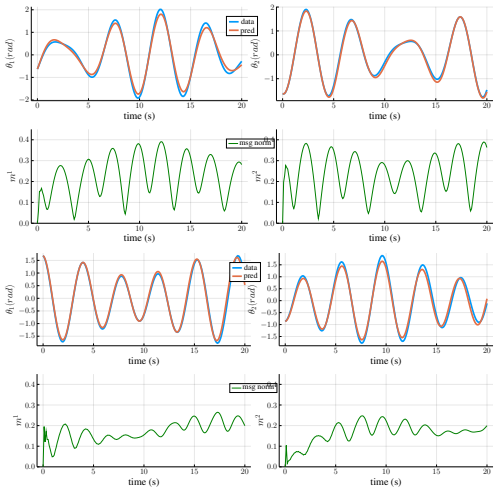
The length of the message can be varied, and we treat it as a hyperparameter that changes the expressivity of the model. In our implementation, we use the mean as the aggregator for incoming messages from the neighbors of a node.

## 4 Experiments

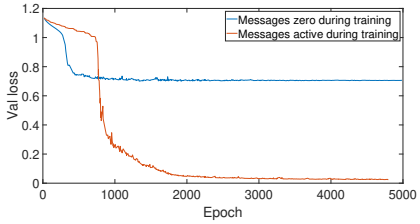
We evaluate the performance of MP-NODE on five different systems: two of them selected to focus on robotic regimes and three of them as representative of standard graph neural network benchmarks, for all of which we generate datasets of trajectories. We implement our method in Julia [35] and make use of the SciML ecosystem [36, 37]. We list training-specific details such as hyperparameters, learning rates, optimizers in the appendix.

We list the systems used below, and provide additional details in the appendix.

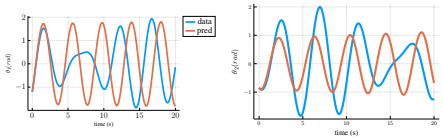
- *Coupled Pendulum:* This system is a combination of two identical simple pendula connected by a string, behaving in an undamped manner. Hence, we train an MP-NODE that models a single pendulum, which is instantiated twice and the communication is enabled through the messages.




(a) The figure compares the predicted (red) with observed (blue) state evolution. Rows 1 and 3 are two different instance of trajectories. Each column represent one pendulum and the green curve (rows 2 and 4) represents outgoing messages. The messages enable interaction between the connected pendulums and enable accurate modeling.




(b) Comparison of the messages passed normally vs. set to zero during training. We achieve much lower training loss when messages are active (red) compared to when they are set to zero (blue).




(c) When message passing is disabled at test time, each node acts similar to an unconnected, simple pendulum. The two graphs represent the two nodes and we plot the prediction (red) and the observed (blue) state evolution.

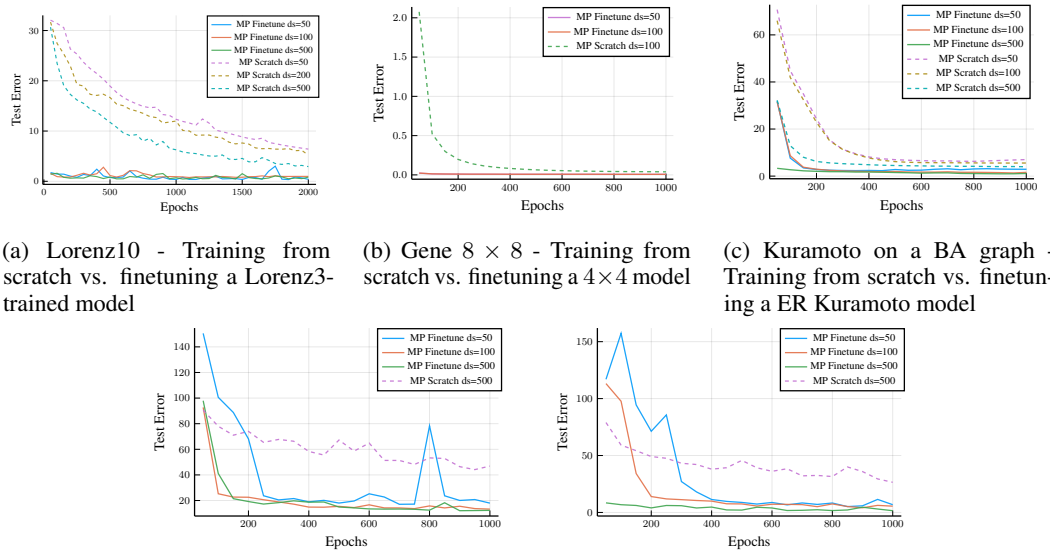
Figure 2: MP-NODE applied to the coupled pendulum system. We show how the messages evolve over time and help minimize drift and compare the characteristics of the system when messages are disabled.

- *Lorenz Attractor:* A Lorenz system is a simplified model of atmospheric convection, described through a 3-dimensional system that exhibits chaotic behavior in specific cases [38]. In our case, we consider multiple Lorenz systems coupled in a fully connected manner. We denote Lorentz3 and Lorentz10 as fully connected systems with 3 and 10 nodes respectively.
- *Gene Dynamics:* This system simulates a continuous time system from biology: the gene regulatory dynamics governed by the Michaelis-Menten equation[39] over grid-like configurations with varying connectivity. Similar to one of the test cases in [30], we generate data for three types of graph topologies: a) Erdós-Rényi (ER) [40]; b) Barabási-Albert (BA) [41] and c) Wattz-Strogatz (WS) [42]. We use two configurations in our experiments, where the dynamics are simulated over a $4 \times 4$ grid and an $8 \times 8$ grid.
- *Kuramoto System:* The Kuramoto model [43] describes the behavior of large sets of coupled oscillators. We simulate the Kuramoto dynamics over the same three types of networks mentioned above, for 10 nodes, which we refer to as Kuramoto10. We use Lindner et al. [44] to build our reference simulators to generate our datasets.
- *Quadrotor swarm:* This system involves a more challenging dynamics learning task of multiple quadrotors carrying a heavy load (that a single quadrotor cannot lift) while passing through a narrow doorway [45]. We consider two versions of this: one with 3 quadrotors, another with 6.

We compare our proposed approach to the one proposed by Sanchez-Gonzalez et al. [31] that generalized different graph based dynamics learning methods, which we refer to as the L2S baseline. We do not use edge features in our implementation of L2S (Figure 1(Right)).

### 4.1 Effect of messages

When the MP-NODE method of modeling is applied to systems with connected nodes, we find that over time, the messages evolve to enable coordination between the nodes. Figure 2a compares predictions (red) with observations (blue) of state evolution for the coupled pendulum system. Each column corresponds to a pendulum and we plot two different trajectories (rows 1 and 3) in Figure 2a.




(a) Lorenz10 - Training from scratch vs. finetuning a Lorenz3-trained model

(b) Gene $8 \times 8$ - Training from scratch vs. finetuning a $4 \times 4$ model

(c) Kuramoto on a BA graph - Training from scratch vs. finetuning a ER Kuramoto model

(d) Finetuning MP-NODE trained on Lorenz3 to Lorenz10 with higher coupling

(e) Finetuning MP-NODE trained on Lorenz3 to Lorenz10 with longer trajectory time horizon

Figure 3: MP-NODE performance when trained from scratch vs. finetuning an existing model

We observe that the predictions closely match the ground truth observations. We also plot the norm of outgoing messages (green) (rows 2 and 4) in Figure 2a and notice that they dynamically change in order to appropriately coordinate across the two nodes.

We also investigate the importance of message passing by comparing performance of MP-NODE with and without messages. Specifically, we turn the messages off by clamping the message outputs to zero. Note that such forcing the messages to zero is similar to the augmented ODE applied to individual nodes. Figure 2b shows that we achieve lower training loss with messages on (red) vs when off (blue).

Similarly, Figure 2c highlights that if no messages are passed between the nodes at test time, each node starts behaving like a simple pendulum that is unconnected to any neighbors. This indicates that the message parameters are important and help the model by encoding the interaction between the nodes.

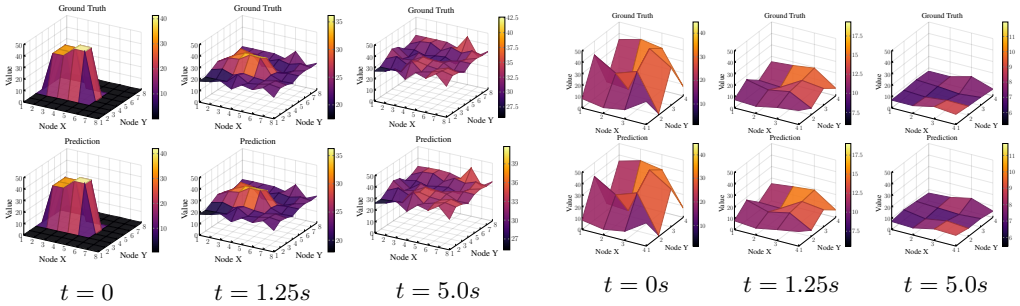
### 4.2 Finetuning

The modular nature of the MP-NODE also allows for easy transfer to different configurations of systems. We examine the ability and performance of finetuning in several cases.

*Increasing number of nodes:*

A common use case would be to take a trained MP-NODE representing a subsystem and finetune it for a larger graph than what it was originally trained on. We use the MP-NODE that was trained on Lorenz3, and finetune this model for Lorenz10. We compare this training effort with that of MP-NODE models trained from scratch just for Lorenz10. We show in Figure 3a that even with a small dataset of 50 trajectories, finetuning a pretrained MP-NODE reaches a low error significantly faster than the models trained from scratch with larger training sets.

Similarly, we put the modular nature of the MP-NODE to test in the gene dynamics system by taking MP-NODE models trained with smaller grid sizes and finetuning them on data from larger grid sizes. In Figure 3b, we show the finetuning performance when a model trained on a $4 \times 4$ grid is transferred to an $8 \times 8$ grid, compared to one trained from scratch on the $8 \times 8$ grid. Similar to the trend with the Lorenz systems, we see that finetuning a trained MP-NODE allows for accurate predictions faster




(a) MP-NODE predictions on a $8 \times 8$ grid of gene regulatory dynamics finetuned over a model trained on $4 \times 4$ grid data.

(b) MP-NODE predictions on a $4 \times 4$ grid of gene regulatory dynamics using the ER network, a topology unseen during training.

Figure 4: Performance of the MP-NODE on the gene dynamics model: finetuning to larger grids, as well as zero-shot generalization to unseen graph structures.

Table 1: Zero-shot generalization performance of MP-NODE

| **Validation** | | **Test** | | |
|---|---|---|---|---|
| BA | BA - L2S | BA | ER (unseen) | WS (unseen) |
| 0.0056 | 0.0106 | 0.0049 | 0.0035 | 0.0025 |

than training a model from scratch. Furthermore, in Figure 4a, we show a qualitative comparison of ground truth and predicted state over the finetuned Gene $8 \times 8$ grid at different time steps.

This shows that due to the modular characteristic of our approach, finetuning a pretrained MP-NODE module allows for rapid adaptation to arbitrary number of nodes in a graph.

*Higher system complexity:*

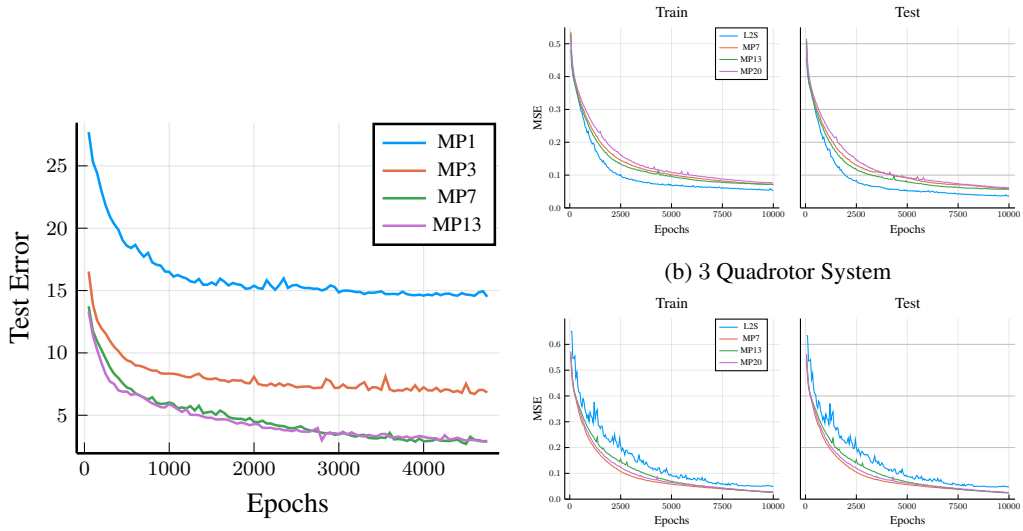
We also evaluate the possibility of finetuning MP-NODEs for different system parameters. As an example, we finetune a MP-NODE model trained on Lorenz3 to Lorenz10 but with different coupling intensity and show the results in Figure 3d. We observe that finetuning requires a lot less data to achieve better performance than training from scratch. Similarly, when we finetune the MP-NODE model trained on Lorenz3 to Lorenz10 for longer time horizon of 10s in Figure 3e, we again find that a lot less data is required to achieve better performance than training from scratch.

*Changing graph structure:*
We also examine the ability of MP-NODE to be finetuned for different graph structures. To this end, we train an MP-NODE on a Kuramoto10 system connected according to the Barabasi-Albert (BA) network and attempt to finetune it for systems of other network types. Similar to above, we see that finetuning is more efficient at adapting to new network types than training models from scratch. A comparison of training effort is shown in Figure 3c when transferring the Kuramoto10 model trained on the BA network to the ER network.

### 4.3 Zero-shot generalization

Considering that the modular nature of MP-NODE allows us to connect arbitrary number of subsystems together for a given graph structure, we can investigate the performance for new configurations without explicitly finetuning the model. When tested on the Lorenz system, we find that an MP-NODE trained only on Lorenz3 exhibits reasonable zero-shot generalization performance on a higher number of Lorenz attractors, such as Lorenz7 and Lorenz10 (see appendix) without requiring any additional training.

We notice that this ability holds for not only changing numbers of nodes, but also different graph topologies. We train the MP-NODE on the Gene $4 \times 4$ system connected according to the Barabasi-Albert (BA) topology, with the dataset comprising of data corresponding to five different adjacency matrices in the BA topology. By training on data coming from different types of connections, the




(a) Effect of message size on test performance in Kuramoto system

(b) 3 Quadrotor System

(c) 6 Quadrotor System

Figure 5: Effect of message dimensionality on train/test performance of the MP-NODE examined for Kuramoto and Quadrotor systems

MP-NODE learns how node dynamics are are affected by different graph structures. After training, we observe that the MP-NODE model that was trained only on BA topology generalizes to the other unseen topologies as well (ER and WS). Table 1 shows the MP-NODE's performance on both seen and unseen network topologies. We observe that MP-NODE exhibits low test error on unseen topologies as well, while achieving better validation performance than L2S. In Figure 4b, we take a qualitative look at the predictions for gene dynamics on the ER network using the zero-shot transferred model.

### 4.4 Effect of message size

In order to further benefit the performance when modeling generally complex dynamics of systems such as Kuramoto or the quadrotor swarm, we investigate whether having a larger message size helps. We train MP-NODE models on the Kuramoto10 system with message dimensionalities of $1, 3, 7$, and $13$. By default, each Kuramoto node has a state length of 3. We show this comparison in Figure 5a. We notice that larger messages result in higher accuracy, but after a certain threshold (messsage size $= 7$ in this case), the improvement becomes minimal.

In the case of the quadrotor swarm, we choose message sizes of $7, 13, 20$ for analysis. Through Section 4.4, we observe that all tested message sizes for MP-NODE lead to similar final test errors. Although L2S seems to do slightly better on the 3 Quadrotor System, it performs worse on the 6 Quadrotor System.

Similar to the trend in other datasets, we find in Figure 6 that finetuning MP-NODE on even limited data allows learning more accurate simulations much faster than learning from scratch, and this holds true for different message sizes as well. Although, we note that higher message dimensions can lead to higher data requirements.

To better demonstrate the advantages of MP-NODE when it comes to finetuning for different system configurations, we consider a metric: for a given test configuration, we compute the 'number of epochs taken to reach minimum test error multiplied by the value of the minimum test error observed'. Intuitively, we can see that a lower number is better. We summarize our findings in Table 2, where we compare MP-NODE finetuning vs. training from scratch, and for some configurations, we also compare with L2S. Through Figure 6, we observe that MP-NODE finetuning consistently achieves a lower combination of test error and number of epochs required to reach it.

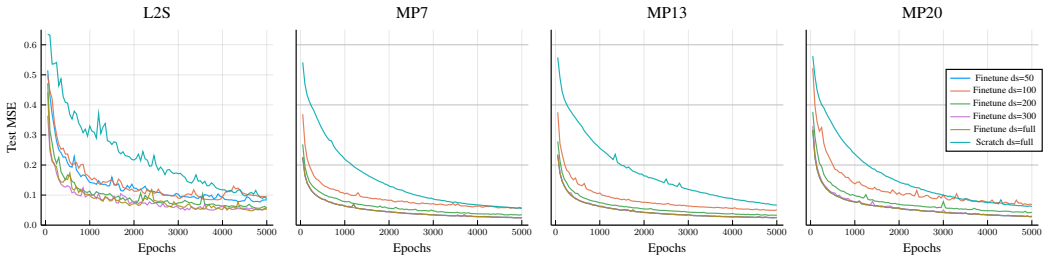



Figure 6: Performance of a 3 Quadrotor MP-NODE finetuned on the 6 Quadrotor System

Table 2: Summary of the performance of MP-NODE finetuning vs. other baselines, using the metric of minimum test error multiplied by number of epochs taken to reach the minimum. Lower is better.

| System | MP-NODE finetuned | MP-NODE scratch | L2S |
|---|---|---|---|
| Lorenz10 (ds=50) | **632.70** | 12800.0 | N/A |
| Lorenz10 (ds=100) | **833.91** | 10454.0 | N/A |
| Lorenz10 (ds=500) | **477.75** | 5856.0 | N/A |
| Gene Dynamics | **12.53** | 28.0 | 46.64 |
| Kuramoto (ds=50) | **1358.50** | 5008.0 | N/A |
| Kuramoto (ds=100) | **1264.83** | 5268.4 | N/A |
| Kuramoto (ds=500) | **874.80** | 3973.9 | N/A |
| Quadrotors | **236.00** | N/A | 476.0 |

# 5 Conclusions

We present a framework for modular modeling of complex homogeneous systems called the message passing neural ODE (MP-NODE). MP-NODE builds upon neural ODEs, and learns a model for the individual node within a connected homogeneous system, which can then be shared across all instances in the system. Augmenting the system state with additional message variables allows the model to encode the features required for coordination and accurate prediction over long horizons. This modular way of thinking about complex systems opens up several possibilities such as zero-shot generalization to unseen configurations and rapid finetuning to more complex configurations. We apply our framework to a wide variety of systems such as coupled pendulum, Lorenz attractors, biological networks, Kuramoto systems and a swarm of quadrotors, and showcase its ability to learn complex dynamics in a modular fashion while outperforming existing methods. We also show that pretrained MP-NODE models can be finetuned for new configurations significantly faster than training from scratch which has the potential to greatly amortize data and compute requirements for data-driven simulations.

**Limitations:** Current implementation of MP-NODE is limited to homogeneous systems only. For better ability to adapt to more complex systems, an extension that can handle heterogeneous systems would be needed. Although encoder-decoder architectures could help in extending to nodes with different node state dimensions, different time-stepping requirements for the nodes might require a different messaging setup. Moreover, the approach currently requires full knowledge of the graph structure which may not always be available for large complex systems. In our current problem setup, we also do not consider potential issues such as delays in communication between nodes, which we leave for future work.

**Broader Impact:** Pretrained simulation nodes that are reusable can minimize data, compute, and energy requirements. Data-driven simulations can often speed up conventional slow simulations while also being capable of capturing real world phenomena ignored in traditional simulations. We show through our analysis of finetuning and generalization that our approach of modeling subsystems that are inherently reusable as opposed to specific configurations of systems has the potential to alleviate data and compute requirements. However, limited data and lack of physical constraints can lead to arbitrarily bad predictions for unseen simulation regimes. Nevertheless, there is a potential to enable large variety of existing simulations to work with heterogeneous compute hardware and data via surrogate modeling [46] building upon this approach.

# References


[1] Radek Grzeszczuk, Demetri Terzopoulos, and Geoffrey Hinton. Neuroanimator: Fast neural network emulation and control of physics-based models. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 9–20, 1998.

[2] Ivan Markovsky and Paolo Rapisarda. Data-driven simulation and control. *International Journal of Control*, 81(12):1946–1959, December 2008. doi: 10.1080/00207170801942170.

[3] Xiaolin Hu. Dynamic data-driven simulation: Connecting real-time data with simulation. In *Concepts and Methodologies for Modeling and Simulation*, pages 67–84. Springer, 2015.

[4] J Nathan Kutz, Steven L Brunton, Bingni W Brunton, and Joshua L Proctor. *Dynamic mode decomposition: data-driven modeling of complex systems*. SIAM, 2016.

[5] Steven L. Brunton and J. Nathan Kutz. *Data-Driven Science and Engineering*. Cambridge University Press, January 2019. doi: 10.1017/9781108380690.

[6] R Kübler and W Schiehlen. Modular simulation in multibody system dynamics. *Multibody System Dynamics*, 4(2):107–127, 2000.

[7] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 6572–6583, 2018.

[8] Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. Augmented neural ODEs. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, 2019.

[9] Belinda Tzen and Maxim Raginsky. Neural stochastic differential equations: Deep latent gaussian models in the diffusion limit. *arXiv preprint arXiv:1905.09883*, 2019.

[10] Junteng Jia and Austin R Benson. Neural jump stochastic differential equations. *arXiv preprint arXiv:1905.10403*, 2019.

[11] Xuechen Li, Ting-Kam Leonard Wong, Ricky TQ Chen, and David Duvenaud. Scalable gradients for stochastic differential equations. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 3870–3882. PMLR, 2020.

[12] Xuanqing Liu, Si Si, Qin Cao, Sanjiv Kumar, and Cho-Jui Hsieh. Neural SDE: Stabilizing neural ode networks with stochastic noise. *arXiv preprint arXiv:1906.02355*, 2019.

[13] Philipp Holl, Vladlen Koltun, and Nils Thuerey. Learning to control PDEs with differentiable physics. *arXiv preprint arXiv:2001.07457*, 2020.

[14] 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.

[15] M Lutter, C Ritter, and Jan Peters. Deep Lagrangian Networks: Using physics as model prior for deep learning. In *International Conference on Learning Representations (ICLR)*, 2019.

[16] Miles Cranmer, Sam Greydanus, Stephan Hoyer, Peter Battaglia, David Spergel, and Shirley Ho. Lagrangian neural networks. *arXiv preprint arXiv:2003.04630*, 2020.

[17] Jayesh K Gupta, Kunal Menda, Zachary Manchester, and Mykel J Kochenderfer. A general framework for structured learning of mechanical systems. *arXiv preprint arXiv:1902.08705*, 2019.

[18] Jayesh K Gupta, Kunal Menda, Zachary Manchester, and Mykel Kochenderfer. Structured mechanical models for robot learning and control. In *Learning for Dynamics and Control*, pages 328–337. PMLR, 2020.

[19] George S Misyris, Andreas Venzke, and Spyros Chatzivasileiadis. Physics-informed neural networks for power systems. In *2020 IEEE Power & Energy Society General Meeting (PESGM)*, pages 1–5. IEEE, 2020.

[20] Marc Finzi, Ke Alexander Wang, and Andrew Gordon Wilson. Simplifying hamiltonian and lagrangian neural networks via explicit constraints. *arXiv preprint arXiv:2010.13581*, 2020.

[21] Siyu He, Yin Li, Yu Feng, Shirley Ho, Siamak Ravanbakhsh, Wei Chen, and Barnabás Póczos. Learning to predict the cosmological structure formation. *Proceedings of the National Academy of Sciences*, 116(28):13825–13832, 2019.

[22] L'ubor Ladickỳ, SoHyeon Jeong, Barbara Solenthaler, Marc Pollefeys, and Markus Gross. Data-driven fluid simulations using regression forests. *ACM Transactions on Graphics (TOG)*, 34(6):1–9, 2015.

[23] 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.

[24] Dmitrii Kochkov, Jamie A Smith, Ayya Alieva, Qing Wang, Michael P Brenner, and Stephan Hoyer. Machine learning–accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences*, 118(21), 2021.

[25] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

[26] Peter W Battaglia, Razvan Pascanu, Matthew Lai, Danilo Rezende, and Koray Kavukcuoglu. Interaction networks for learning about objects, relations and physics. *arXiv preprint arXiv:1612.00222*, 2016.

[27] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[28] Michael Poli, Stefano Massaroli, Junyoung Park, Atsushi Yamashita, Hajime Asama, and Jinkyoo Park. Graph neural ordinary differential equations. *arXiv preprint arXiv:1911.07532*, 2019.

[29] Zijie Huang, Yizhou Sun, and Wei Wang. *Coupled Graph ODE for Learning Interacting System Dynamics*, page 705715. Association for Computing Machinery, New York, NY, USA, 2021. ISBN 9781450383325. URL `https://doi.org/10.1145/3447548.3467385`.

[30] Chengxi Zang and Fei Wang. Neural dynamics on complex networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 892–902, 2020.

[31] 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 (ICML)*, pages 8459–8468. PMLR, 2020.

[32] Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W Battaglia. Learning mesh-based simulation with graph networks. *arXiv preprint arXiv:2010.03409*, 2020.

[33] Tal El-Hay, Ido Cohn, Nir Friedman, and Raz Kupferman. Continuous-time belief propagation. In *ICML*, 2010.

[34] Victor Garcia Satorras, Zeynep Akata, and Max Welling. Combining generative and discriminative models for hybrid inference. *arXiv preprint arXiv:1906.02547*, 2019.

[35] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98, 2017.

[36] Christopher Rackauckas and Qing Nie. DifferentialEquations.jl a performant and feature-rich ecosystem for solving differential equations in Julia. *The Journal of Open Research Software*, 5 (1), 2017. doi: 10.5334/jors.151.

[37] Michael Innes, Elliot Saba, Keno Fischer, Dhairya Gandhi, Marco Concetto Rudilosso, Neethu Mariya Joy, Tejan Karmali, Avik Pal, and Viral Shah. Fashionable modelling with flux. *arXiv preprint arXiv:1811.01457*, 2018.

[38] Edward N Lorenz. Deterministic nonperiodic flow. *Journal of atmospheric sciences*, 20(2): 130–141, 1963.

[39] Uri Alon. *An introduction to systems biology: design principles of biological circuits*. CRC press, 2019.

[40] P. Erds and A Rényi. On the evolution of random graphs. In *Publication of the Mathematical Institute of the Hungarian Academy of Sciences*, pages 17–61, 1960.

[41] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

[42] Duncan J Watts and Steven H Strogatz. Collective dynamics of small-worldnetworks. *Nature*, 393(6684):440–442, 1998.

[43] Yoshiki Kuramoto. *Chemical oscillations, waves, and turbulence*. Courier Corporation, 2003.

[44] Michael Lindner, Lucas Lincoln, Fenja Drauschke, Julia Monika Koulen, Hans Würfel, Anton Plietzsch, and Frank Hellmann. Networkdynamics.jl – Composing and simulating complex networks in julia. *arXiv preprint arXiv:2012.12696*, 2020.

[45] Brian E Jackson, Taylor A Howell, Kunal Shah, Mac Schwager, and Zachary Manchester. Scalable cooperative transport of cable-suspended loads with UAVs using distributed trajectory optimization. *IEEE Robotics and Automation Letters*, 5(2):3368–3374, 2020.

[46] Chris Rackauckas, Ranjan Anantharaman, Alan Edelman, Shashi Gowda, Maja Gwozdz, Anand Jain, Chris Laughman, Yingbo Ma, Francesco Martinuzzi, Avik Pal, et al. Composing modeling and simulation with machine learning in julia. *arXiv preprint arXiv:2105.05946*, 2021.

# Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 5.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See supplementary material. We include the training pipeline for a closer look at our proposed technique. Evaluation pipeline/code to generate individual figures is not included.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Appendix.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Appendix.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes]
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]