# AUTOREGRESSIVE GRAPH NETWORK FOR LEARNING MULTI-STEP PHYSICS

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

## ABSTRACT

In this work, we propose an *Autoregressive* Graph Network (AGN) that learns forward physics using a temporal inductive bias. Currently, temporal state space information is provided as additional input to a GN when generating roll-out physics simulations. While this relatively increases the network's predictive performance over multiple time steps, a temporal model enables the network to induce and learn temporal biases. In dynamical systems, the arrow of time simplifies possible interactions in the sense that we can assume current observations to be dependent on preceding states. The autoregressive property naturally induces the arrow of time and can further constrain physics-induced GNs to conserve symmetries over long time-steps. Our proposed GN encodes temporal state information using an autoregressive encoder that can parallelly compute latent temporal embeddings over multiple time steps during a single forward pass. We perform case studies that compare multi-step forward predictions against baseline data-driven GNs across diverse datasets that feature different particle interactions. Our approach outperforms the state of the art GN and physics-induced GNs in 9 out of 10 and in 7 out of 10 particle physics datasets when conditioned on optimal historical states.

## 1 INTRODUCTION

In the recent years, there has been a growing interest in learning physics with the help of deep learning coupled with other techniques such as inductive biases, physics informed loss functions and meta-learning (Fragkiadaki et al. (2016); Battaglia et al. (2016); Xu et al. (2019); Hall et al. (2021)). Relational networks such as Graph Networks (GNs) can decompose and learn the dynamics of a physics system on the basis of particle interactions within their neighborhoods (Battaglia et al. (2016); Li et al. (2018); Sanchez-Gonzalez et al. (2020)). Across science and engineering, particle states often contain system and particle specific properties such as mass, density, velocity, particle type, etc. that are required to approximate the dynamics of a system.

In general, it is possible to apply GNs to pose the following fundamental question: Given the current state of a system of particles along with particle specific local properties and global system properties, can the trajectory of the system be predicted? For example, predicting the trajectory of a system of particles subject to Newtonian forces (Sanchez-Gonzalez et al. (2018; 2020)). Often referred to as the forward problem, it assumes knowledge about the physical properties of particles and therefore utilizes the observations to construct a suitable model that predicts the trajectory of the system of particles.

The solution to a typical forward dynamics problem governed by an ODE involving particles can be parameterized using a GNN by learning from the current state or by using a history of previous particle states. There are strong benefits to training on entire sequences or multiple time-steps (Mohajerin (2017); Xu et al. (2019)) as one-step GNs tend to be unstable and accumulate error in the long-run. While prior work has shown that concatenating history of previous states enables a trained simulator such as a GNN to predict the next state more accurately, a sequential model captures certain symmetries, e.g., arrow of time, conservation of energy, momentum .etc. Sequential models such as RNN, LSTM, GRU and Transformers have been applied to 1D time series and $N$-body systems (Chen et al. (2018); Zhang et al. (2020); Han et al. (2022)). While appealing choices to model dynamical systems, they require sequential computations, come with significant memory overhead as the lookback length and/or the dimensionality of the problem increases.

In this paper, we propose an Autoregressive Graph Encoder that induces time dependence via the autoregressive property while avoiding expensive sequential computations required by classical auto-regressive recurrent models. The vector autoregressive property offers two advantages in the proposed approach. First, the encoder learns the arrow of time (i.e. previous states affect the future states) which in turns allows it to project the state space data onto a high-dimensional spatio-temporal vector space in a parallel manner. Second, the encoder computes spatio-temporal embeddings at a similar computation time in comparison to a single-step GN. We present a case study that compares current physics induced (Lagrangian and Hamiltonian) GNs as well as general purpose baseline GNs. We find that our computationally efficient AGN outperforms the baselines in multi-time step forward simulation predictions.

## 2 Related Work

Engineering simulations developed for prediction and control of complex physical systems can be built based on empirical or theoretical findings. Compared to analytical models, a learned simulator can be more efficient at predicting complex phenomena (He et al. (2019); Sanchez-Gonzalez et al. (2020)). Further, domain knowledge can be encoded as physics priors in the form of loss function regularization (Raissi et al. (2019)) and data featurization to improve accuracy and performance. However, explicitly providing knowledge may limit generalizability to different datasets that are fundamentally governed by the same underlying dynamics. Data-driven approaches learn the dynamics of a system without explicitly solving an ODE or a PDE. The physical interactions between particles are learned implicitly by nature of the model construction (Battaglia et al. (2018); Chen et al. (2018); Greydanus et al. (2019); Cranmer et al. (2020a;b); Sanchez-Gonzalez et al. (2020); Pfaff et al. (2021); Rubanova et al. (2022)), therefore enabling generlization across a similar class of problems. Such neural networks have shown to approximate solutions of ordinary differential equations (ODE) that govern the dynamics of a system and have also shown to be compatible with most off-the-shelf ODE explicit solvers (i.e., Euler, Runge-Kutta). While successful, such meta-models have only been trained to learn the forward dynamics of the system based on the current state information or by concatenating history of states.

Deep implicit models (Amos & Kolter (2017); Bai et al. (2019; 2020)) are a powerful class of implicit learning methods proposed for learning constraints as well as learning to solve constraints. Often these models work around the computational expense of computing first and higher order gradients required by numerical optimization solvers such as gradient descent, Newton's method, .etc. Classical mechanics present a mechanistic understanding of reality that can be captured using such implicit or constrained models. Symmetries such as conservation of energy and momentum can be implicitly learned by constraining the gradients of the networks to satisfy Euler-Lagrange or Hamilton's equations (Chen et al. (2018); Lutter et al. (2019); Greydanus et al. (2019); Cranmer et al. (2020a)). Recent work (Yang et al. (2020); Rubanova et al. (2022)) have proposed implicit neural network strategies to learn physics-based constraints. These methods frame the forward problem as a constrained optimization problem. The Neural Projections method iteratively proposes a prediction of the next state of a system with an explicit Euler step to solve a root-finding problem, then projects the prediction onto a learned constraint manifold that is implemented as a multilayer perceptron (MLP). A recent work (Han et al. (2022)) proposes a sequential model with a transformer operating in the latent space to perform autoregression on latent temporal embeddings of particles. While their approach helps manage the dimensionality of the problem so as to limit memory consumption of the sequence model, it still requires $t$ sequential computations.

In this work, we induce an arrow of time constraint on the encoder parameterization by modifying it as an autoregressive encoder (Germain et al. (2015)) that encodes the nodes and edges separately with modifications to suit $n$ dimensional physics problems. The Masked Autoencoder for Distribution Estimation (MADE) model proposed by (Germain et al. (2015)) only enforces a scalar autoregressive property that may not suit vectors denoting a single state of a particle. By changing the manner in which the masking scheme is implemented, we generalize the MADE model to have a vector autoregressive property that can be used to generate temporal latent node and edge embeddings. Put together with a message passing network, we show that our model is able to model time dependencies better and lead to slower error accumulation over long time-steps.

## 3 APPROACH

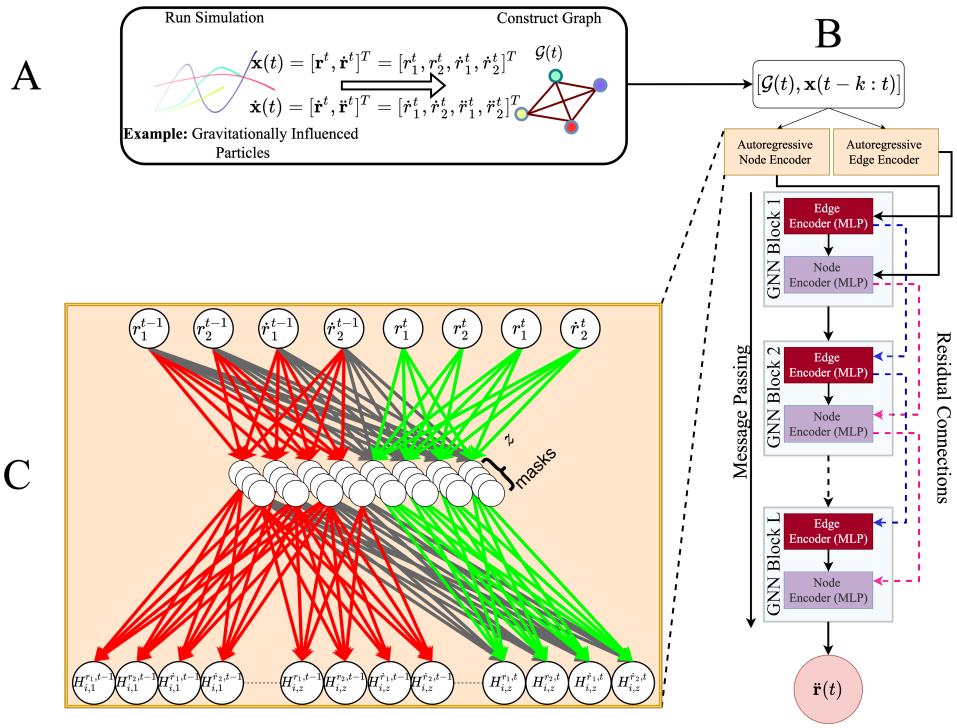


Figure 1: Schematics: **A)** Illustrates data and corresponding particle graph generation, **B)** Autoregressive Graph Network, **C)** A single layer Autoregressive Node Encoder with $z$ masks in the output layer that generates a $4z$ dimensional latent vector for every node $i$. The directed colored arrows denote dependence.

### 3.1 LEARNED SIMULATOR OVERVIEW

The dynamics of a system of $N$ particles can be described by a coupled system of Ordinary Differential Equations (ODEs) that takes the following general form:

$$\frac{d}{dt}\mathbf{x}(t) = \mathbf{f}(\mathbf{x}(t), t, \boldsymbol{\omega}) \tag{1}$$

where, $\mathbf{x}(t) = [\mathbf{r}\ \dot{\mathbf{r}}]^T$ represents the state of the system at time $t$ with $\mathbf{r} \in \mathbb{R}^{Nxd}$ denoting the position vectors and $\dot{\mathbf{r}} \in \mathbb{R}^{Nxd}$ denoting the velocity vectors of $N$ particles in $d$ dimensions. The particles also possesses a physical property such as mass, charge, density, etc. and is denoted by $\boldsymbol{\omega} \in \mathbb{R}^N$. In general this problem requires an approximate numerical solution that is acquired by solving the ODE subject to the initial value condition $\mathbf{x}(0)$. In this paper, we are interested in learning the dynamics of the system ($\mathbf{f}$) which can typically be complex using a parameterized function approximator ($\mathbf{f}_{(\cdot)}$), where $(\cdot)$ denotes the parameters to be learned. Further, since we adopt a graph-based representation of the system of particles, we consider particles as nodes ($V$) in a graph ($\mathcal{G}$) such that every particle has a first-order neighborhood as well as hops of higher-order neighborhoods. Hence, every system state ($\mathbf{x}(t)$) has a corresponding adjacency matrix ($\mathbf{A}(t)$) which is a matrix representation of $\mathcal{G}(t) = (V, E)$.

To solve the forward problem, the learned simulator $\mathbf{f}_\theta : \mathbf{x}(t) \mapsto \dot{\mathbf{x}}(t)$ maps the current state of the system of particles at time $t$ to its first-order derivative which is the velocity and acceleration vectors of the system of particles. We note that the training is supervised on acceleration labels only. Then, using a choice of numerical integrator (i.e., Euler, Runge-Kuta) the next state $\mathbf{x}(t+1)$ can be obtained from $\dot{\mathbf{x}}(t) = [\dot{\mathbf{r}}\ \ddot{\mathbf{r}}]^T$.

### 3.2 LEARNING FORWARD DYNAMICS

#### 3.2.1 VECTOR AUTOREGRESSIVE (VAR) ENCODER

Following the established notations, we implement the learnable function $\mathbf{f}_\theta$ using Graph Networks (GN) (Battaglia et al. (2018)), arranged in the Encode-Process-Decode architecture, similar to previous work on GN-based learned simulators (Sanchez-Gonzalez et al. (2020); Pfaff et al. (2021)). Vector autoregression (VAR) is a statistical model used to capture the dependencies between quantities which are vectors and which change over time. In the simple case of a 2D problem, the motion of each particle in a trajectory can be described through a vector ($[\mathbf{r}_1, \mathbf{r}_2, \dot{\mathbf{r}}_1, \dot{\mathbf{r}}_2]^T$) in 4 dimensional state space. Each element of the vector is related to one another as well as to the next state's vector through some unknown coefficients (eg. parameterized as Neural Nets). This can compactly be captured using a linear model such as VAR. In the example below, we show a multivariate VAR model of order 2 with a noise term ($\boldsymbol{\epsilon}^t$).

$$\begin{bmatrix} r_1^t \\ r_2^t \\ \dot{r}_1^t \\ \dot{r}_2^t \\ r_1^{t-1} \\ r_2^{t-1} \\ \dot{r}_1^{t-1} \\ \dot{r}_2^{t-1} \end{bmatrix} = \begin{bmatrix} w'_{1,1} & w'_{1,2} & w'_{1,3} & w'_{1,4} & w'_{1,5} & w'_{1,6} & w'_{1,7} & w'_{1,8} \\ w'_{2,1} & w'_{2,2} & w'_{2,3} & w'_{2,4} & w'_{2,5} & w'_{1,6} & w'_{1,7} & w'_{1,8} \\ w'_{3,1} & w'_{3,2} & w'_{3,3} & w'_{3,4} & w'_{3,5} & w'_{1,6} & w'_{1,7} & w'_{1,8} \\ w'_{4,1} & w'_{4,2} & w'_{4,3} & w'_{4,4} & w'_{4,5} & w'_{1,6} & w'_{1,7} & w'_{1,8} \\ w'_{5,1} & w'_{5,2} & w'_{5,3} & w'_{5,4} & 0 & 0 & 0 & 0 \\ w'_{6,1} & w'_{6,2} & w'_{6,3} & w'_{6,4} & 0 & 0 & 0 & 0 \\ w'_{7,1} & w'_{7,2} & w'_{7,3} & w'_{7,4} & 0 & 0 & 0 & 0 \\ w'_{8,1} & w'_{8,2} & w'_{8,3} & w'_{8,4} & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} r_1^{t-1} \\ r_2^{t-1} \\ \dot{r}_1^{t-1} \\ \dot{r}_2^{t-1} \\ r_1^{t-2} \\ r_2^{t-2} \\ \dot{r}_1^{t-2} \\ \dot{r}_2^{t-2} \end{bmatrix} + \begin{bmatrix} \epsilon_1^t \\ \epsilon_2^t \\ \epsilon_3^t \\ \epsilon_4^t \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{2}$$

In Eqn. 1, $\mathbf{W'} = \mathbf{W} \circ \mathbf{M}^{\mathbf{W}}$, where $\mathbf{W}$ represents a parameterizable autoregressive coefficient matrix such that elements $\{w'_{5:8,1:4}\} = 1$ and $\mathbf{M}^{\mathbf{W}}$ is a mask the size of $\mathbf{W}$ that can be used to induce the desired dependencies in a generalizable manner between input and output variables of interest. In the case above, the matrix $\mathbf{M}^{\mathbf{W}}$ creates the following temporal dependencies: $\mathbf{r}^t|(\mathbf{r}^{t-1}, \mathbf{r}^{t-2})$; $\mathbf{r}^{t-1}|\mathbf{r}^{t-1}$. Since we seek to parameterize such a VAR process using a neural network, we show a similar analogy in the context of a an autoencoder with a single hidden layer. We drop the noise term in our NN formulation. The output $\mathbf{H}(\mathbf{x}) = act(\mathbf{b}+(\mathbf{W}\circ\mathbf{M}^{\mathbf{W}})\mathbf{x})$ is used to estimate $\hat{\mathbf{x}} = act(c+(\mathbf{V}\circ\mathbf{M}^{\mathbf{V}})\mathbf{H}(\mathbf{x}))$, where, $\mathbf{M}^{\mathbf{W}}$ and $\mathbf{M}^{\mathbf{V}}$ are the masks that impose the autoregressive property while $\mathbf{V}$ and $\mathbf{W}$ are the weight matrices that parameterize the autoregressive coefficient matrix for a VAR based encoder of order $k$. The $k$, $d$-dimensional state vector of each node $i$ is mapped to an embedding $\mathbf{H}_i^{(t-k)}$ following the forward pass of the autoencoder. We begin by assigning each element of $\mathbf{x}(t)$ an integer $t$ and each unit in the hidden layer an integer $m$ which is assigned an integer $p$. The $s^{th}$ hidden unit's number $m(s,t) = s \mod 2n + (2n - (s \mod 2n)) + 2nt = p$ gives the maximum number of input units to which it can be connected. Further, we allow $m(s,t) = P$ while not allowing $m(s,t) = 0$ so as to ensure that the current state as well as the $k$th lag state are conditioned upon when generating high-dimensional temporal latent vectors as opposed to probabilisitic conditionals or prior parameterization in generative models.

The conditional temporal latent vectors can be generated using a deep autoencoder parameterized with the help of the following masking rules. In the single hidden layer case, the constraints on the maximum number of inputs to each hidden unit are encoded in the matrix masking the connections between the input and hidden units:

$$\mathbf{M}^{\mathbf{W}}_{s,p} = \begin{cases} 1_{m(s,t)\geq p}, & \text{if } m(s,t) \geq p \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

for $p \in \{2n, \ldots, P\}$, $s \in \{1, \ldots, S\}$, $n \in \{1, \ldots, N\}$ and $t \in \{0, \ldots, k\}$ (with abuse of notation). We also need to encode the constraint that the $p^{th}$ network output is connected to all $m(s,t) = p$. Hence the output weights can connect the $p^{th}$ network output to hidden units with $m(s,t) \leq p$. These constraints can be encoded in the output mask matrix as follows:

$$\mathbf{M}^{\mathbf{V}}_{p,s} = \begin{cases} 1_{p\geq m(s,t)}, & \text{if } p \geq m(s,t) \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

The proposed encoder architecture naturally generalizes to deep architectures. To impose the same constraint on the succeeding hidden layer, we have to ensure that each hidden unit $s'$ gets connected

to the preceeding layers such that $m^l(s,t) \leq m^{l+1}(s',t)$, where $l$ denotes the layer number. The rule given by Eqn. (3) can be generalized to any layer $l$. The last hidden layer ($l'$) should preserve the natural ordering (vector autoregressive graph) by enforcing Eqn. (4). Further, we are interested to generate latent temporal embeddings of varying size ($z$). To do so, we create $z$ copies of final layer masks such that each mask generates an embedding the size of the input vector (i.e. if the input of node $i$ has a size of $|\mathbf{x}_i|$, then the vector autoregressive output of the encoder will be of size $z \cdot |\mathbf{x}_i|$. We pick a $z$ value by matching the encoder output size with that of a typical autoencoder implemented in a GN.

We have a vector autoregressive node encoder ($\mathcal{VAR}_V$) and an edge encoder ($\mathcal{VAR}_E$) whose corresponding output node and edge embeddings are denoted can be as $\mathbf{H}_i^{(0)}$ and $\mathbf{H}_{ij}^{(0)}$ respectively. The input to a VAR edge encoder is $[\mathbf{x}_i(t-k) - \mathbf{x}_j(t-k), \dot{\mathbf{x}}_i(t-k) - \dot{\mathbf{x}}_j(t-k), \ldots, \mathbf{x}_i(t) - \mathbf{x}_j(t), \dot{\mathbf{x}}_i(t) - \dot{\mathbf{x}}_j(t)]$ representing the relative position and velocity information between particles across $k$ (hyper-parameter) time-steps, while that of a VAR node encoder is $[\mathbf{r}_i(t-k), \dot{\mathbf{r}}_i(t-k), \ldots, \mathbf{r}_i(t), \dot{\mathbf{r}}_i(t)]$ representing the full state vector between particles across $k$ (hyper-parameter) time-steps.

### 3.2.2 Processor and Decoder

Having described the graph encoders, we now briefly describe the graph processor and decoder along with their respective inputs and outputs. The graph processor computes interactions among nodes through $L$ message passing blocks arranged in sequence using standard MLP edge encoders $\mathcal{E}_E$ and node encoders $\mathcal{E}_V$. Message-passing preserves constraints while allowing information to propagate between nodes and edges. The processor takes as input the latent temporal embeddings ($\mathbf{H}_i^{(0)}$ and $\mathbf{H}_{ij}^{(0)}$) generated by the autoregressive node and edge encoders. In the subsequent sections, we drop $t$ for brevity.

The edge encoder ($\mathcal{E}_E$) performs the following node-to-edge ($V \rightarrow E$) message passing operation,

$$V \rightarrow E : \mathbf{H}_{ij}^{(l)} = \mathcal{E}_E^{(l)}([\mathbf{H}_i^{(l)} - \mathbf{H}_j^{(l)}, |\mathbf{H}_i^{(l)} - \mathbf{H}_j^{(l)}|, \mathbf{H}_{ij}^{(l-1)}]) + \mathbf{H}_{ij}^{(l-1)} \tag{5}$$

where, $[.\,,.]$ denotes concatenation, $l$ denotes the $l$-th graph network block; Following the edge encoding operation, the node encoder ($\mathcal{E}_V$) takes as input the edge embedding and performs the following edge-to-node ($E \rightarrow V$) message passing operation,

$$E \rightarrow V : \mathbf{H}_i^{(l+1)} = \mathcal{E}_V^{(l)}([\sum_{i \in \mathcal{N}_j} \mathbf{H}_{ij}^{(l)}, \mathbf{x}_j]) + \mathbf{H}_i^{(l)} \tag{6}$$

The node encoder aggregates the edge features for a target node $j$ across its neighboring nodes ($\mathcal{N}_j$), concatenates the target node features and finally adds the previous block's latent graph features using residual connections. Finally, the output of the last message passing block gets passed to an MLP decoder that then predicts the acceleration ($\hat{\ddot{\mathbf{r}}}$). See Supplementary Materials for details on implementation.

## 4 Discussion

In this section, we discuss the key findings from our experiments. The particle-based datasets (Cranmer et al. (2020b)) are representative of common Newtonian dynamics that describes the dynamics of particles in 2 and 3 dimensions according to Newton's law of motion. These simulations were written using the JAX library. The variable parameters of these analytic simulation models include the number of simulations, number of particles, particle specific parameters, time-steps and step-size. Each simulation per analytic model is integrated over 1000 time-steps using an adaptive RK4 integrator. In order to evaluate the rigorousness of the proposed approach, we keep a number of hyper-parameter choices such as the number of hidden layers, hidden neurons, etc. constant. See Supplementary Materials for full details on choice of hyper-parameters. We benchmark our approach against the following work Graph Network (GN) (Sanchez-Gonzalez et al. (2020)), Graph Lagrangian Network (GLN) (Cranmer et al. (2020a)) and Graph Hamiltonian Network (GHN) (Greydanus et al. (2019); Cranmer et al. (2020b)). While Pfaff et al. (2020) propose a GN for learning physics simulations, we note that their approach when applied to particle based datasets simply assumes the architecture as proposed by Sanchez-Gonzalez et al. (2020).

### 4.1 PERFORMANCE OF FORWARD DYNAMICS PREDICTION

We train all baselines as well as our models on one-step target acceleration predictions (i.e., Given $\mathbf{x}(t : t+k)$, predict $\ddot{\mathbf{r}}(t+k)$) using a 70:30 train, test ratio. Once trained, we sample 30 initial conditions that were not a part of the train/test distribution to generate 30 simulations over a 1000 time-step roll-out using an RK4 integrator. In this section, we consider the following case studies:

1. Translation invariance in AGN.
2. Autoregressive encoders in physics-constrained GNs.
3. The influence of lookback length on the forward prediction performance across GN and physics constrained GNs.

The optimum lookback length ($k^*$) is a function of the model as well as the dataset. While we discuss the impact of $k$ on the stability and forward prediction performance later, tables 1 and 2 reports the Mean Squared Error (MSE) on multi-time step predictions when optimum $k^*$ is chosen for each model/dataset. In the category of Graph Nets, our AGN outperforms Graph Nets in 9 out of 10 datasets while in the physics constrained Graph Nets category, our models outperform the baselines in 7 out of 10 datasets.

Table 1: Mean multi-Step Forward Simulation Prediction GNs (Optimum $k^*$)

| Datasets | GN | AGN (Ours) |
|---|---|---|
| **2D Spring** | 5.38 | **1.77** |
| **2D Damped** | 9.37 | **8.17** |
| **2D Gravity ($\frac{1}{r1}$)** | 10.55 | **9.41** |
| **2D Gravity ($\frac{1}{r2}$)** | 5.12 | **3.58** |
| **2D Charge** | 3.85 | **1.26** |
| **3D Spring** | 10.57 | **7.54** |
| **3D Damped** | 16.13 | **13.46** |
| **3D Gravity ($\frac{1}{r1}$)** | **2.19** | 7.38 |
| **3D Gravity ($\frac{1}{r2}$)** | 0.88 | **0.79** |
| **3D Charge** | 0.94 | **0.47** |

Table 2: Mean multi-Step Forward Simulation Prediction Physics Constrained GNs (Optimum $k^*$)

| Datasets | GLN | GHN | AGLN (Ours) | AGHN (Ours) |
|---|---|---|---|---|
| **2D Spring** | 0.84 | 2.05 | **0.61** | 1.49 |
| **2D Damped** | 5.28 | 426.62 | **2.67** | 19.49 |
| **2D Gravity ($\frac{1}{r1}$)** | **2.94** | 8.12 | 9.14 | 11.08 |
| **2D Gravity ($\frac{1}{r2}$)** | 5.70 | 52.78 | **1.69** | 6.46 |
| **2D Charge** | 20.68 | 42.74 | **1.48** | 6.44 |
| **3D Spring** | **0.64** | 1.74 | 1.21 | 3.37 |
| **3D Damped** | **2.53** | 20.69 | 10.38 | 85.62 |
| **3D Gravity ($\frac{1}{r1}$)** | 2.43 | 5.52 | **1.68** | 3.92 |
| **3D Gravity ($\frac{1}{r2}$)** | 9.24 | 1.41 | 0.87 | **0.78** |
| **3D Charge** | 37.11 | 0.92 | 1.05 | **0.75** |

#### 4.1.1 TRANSLATION INVARIANCE

Typically, translation invariance is enforced by construction (Sanchez-Gonzalez et al. (2020); Pfaff et al. (2020); Rubanova et al. (2022)) by never providing the absolute position as inputs to the network. Prior work have shown that such a form of translation invariance leads to lower one-step and multi-step prediction errors. Contrary to such a form of translation invariance, in our approach, we find that the autoregressive encoder enforces translation invariance by taking in as input the absolute positions. We hypothesize that the autoregressive encoder generates embeddings that capture spatio-temporal information regardless of the specific absolute location of the particles. In other words, by providing absolute positions as input, the autoregressive encoder learns translation invariant features.

Table 3: Mean multi-Step Forward Simulation Prediction when $k = 5$ (GN)

| Datasets | GN w. position | GN w/o. position | AGN w. position (Ours) | AGN w/o. position (Ours) |
|---|---|---|---|---|
| **2D Spring** | 7.81 | 6.72 | **2.68** | 14.78 |
| **2D Damped** | 25.12 | 19.39 | **12.67** | 284.55 |
| **2D Gravity ($\frac{1}{r1}$)** | **5.43** | 14.96 | 27.58 | 45.64 |
| **2D Gravity ($\frac{1}{r2}$)** | 801.37 | 299.76 | **3.58** | 97.09 |
| **2D Charge** | 318.36 | 38.06 | **8.08** | 580.14 |
| **3D Spring** | 23.04 | 18.09 | **10.75** | 31.75 |
| **3D Damped** | 43.49 | 202.71 | **15.73** | 57.47 |
| **3D Gravity ($\frac{1}{r1}$)** | 3.50 | 2.69 | 7.38 | **2.32** |
| **3D Gravity ($\frac{1}{r2}$)** | 564.43 | 1.42 | **1.16** | 2.33 |
| **3D Charge** | 1.15 | 3.82 | **1.05** | 3.38 |

Table 4: Mean multi-Step Forward Simulation Prediction when $k = 5$ (Physics Constrained GNs)

| Datasets | GLN | GHN | AGLN (Ours) | AGHN (Ours) |
|---|---|---|---|---|
| **2D Spring** | 1.06 | 2.52 | **0.72** | **2.22** |
| **2D Damped** | 8.88 | 16492.84 | **3.28** | **19.49** |
| **2D Gravity ($\frac{1}{r1}$)** | **3.77** | 301.52 | 18.21 | **11.08** |
| **2D Gravity ($\frac{1}{r2}$)** | 18.94 | 52.78 | **3.85** | **6.46** |
| **2D Charge** | 78.97 | 42.74 | **11.59** | **6.44** |
| **3D Spring** | **1.12** | **2.01** | 1.37 | 58.18 |
| **3D Damped** | **5.99** | 250.32 | 36.24 | **85.62** |
| **3D Gravity ($\frac{1}{r1}$)** | 2.43 | 54.22 | **1.67** | **3.92** |
| **3D Gravity ($\frac{1}{r2}$)** | 240.38 | 1.41 | **1.32** | **0.78** |
| **3D Charge** | 37.12 | 3.72 | **9.25** | **0.75** |

To test the hypothesis, we present our AGN model with and without absolute positions as input. Similarly, in order to ensure that the dataset is not a confounding variable, we also present the GN

model with and without absolute positions when $k = 5$. Table 3 reflects the mean multi-step MSE loss with and without absolute position of particles as input in 30 held-out simulations across the various particle simulation datasets. We notice that when absolute position is not provided as input to our model, its predictive performance deteriorates and also tends to become unstable across few datasets. Meanwhile, providing absolute position as input to a GN deteriorates its performance and causes instability. While the instability could also be caused by the choice of $k$, we find deteriorating forward prediction performance to be a common theme. By providing absolute position as input to our model, we find that it outperforms the state of the art GN in 8 out of 10 datasets. Figure 2 visualizes the rollout MSE error on the 2D Spring dataset when GN and AGN are provided with/without absolute positional information.

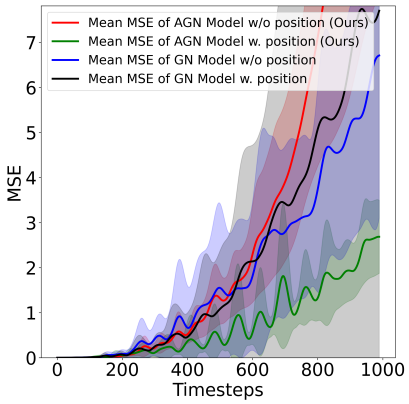


Figure 2: Rollout MSE - GN (2D Spring)

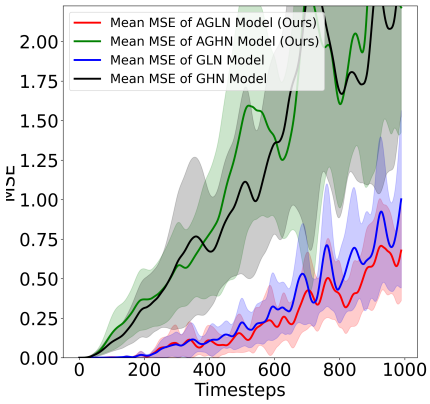


Figure 3: Rollout MSE - Physics Constrained (2D Spring)

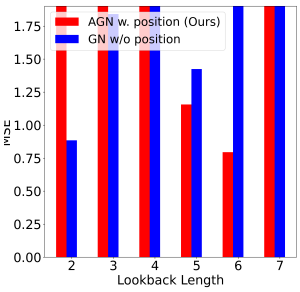


Figure 4: Lookback Length Comparison - GN (3D Gravity ($r2$))

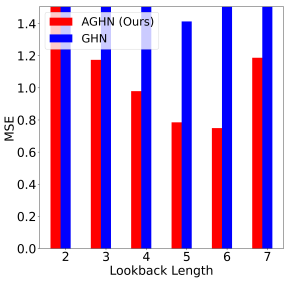


Figure 5: Lookback Length Comparison - GHN (3D Gravity ($r2$))

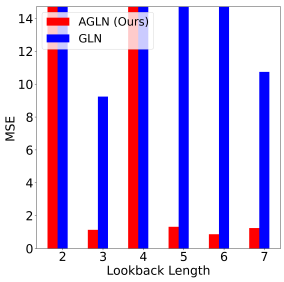


Figure 6: Lookback Length Comparison - GLN (3D Gravity ($r2$))

### 4.1.2 Physics Constraints

Next, we perform experiments by changing the GN to have physics biases (Lagrangian and Hamiltonian). Since these constraints require computing the gradients with respect to $r^t$ and momentum at time t, they require absolute positions of the particles as input. Like AGN, we employ an autoregressive node and edge encoder to capture spatio-temporal latent embeddings that are then used as input to approximate the Lagrangian and Hamiltonian of the parameteric learnable function ($f_{(.)}$). For $k = 5$, we report the mean multi-step forward prediction MSE in table 4. Comparing GLN and AGLN, we find AGLN to have superior forward prediction performance in 7 out of 10 datasets. Similarly, comparing GHN and AGHN, we find AGHN to have superior forward prediction performance in 9 out of 10 datasets. While training, we notice that both GLN and GHN are sensitive

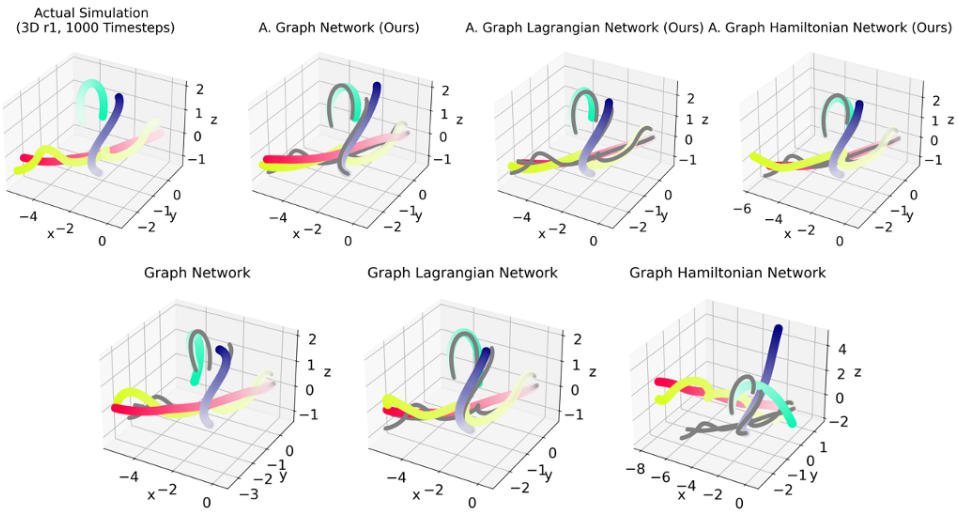


Figure 7: Rollout prediction Comparison 3D Gravity ($r1$)

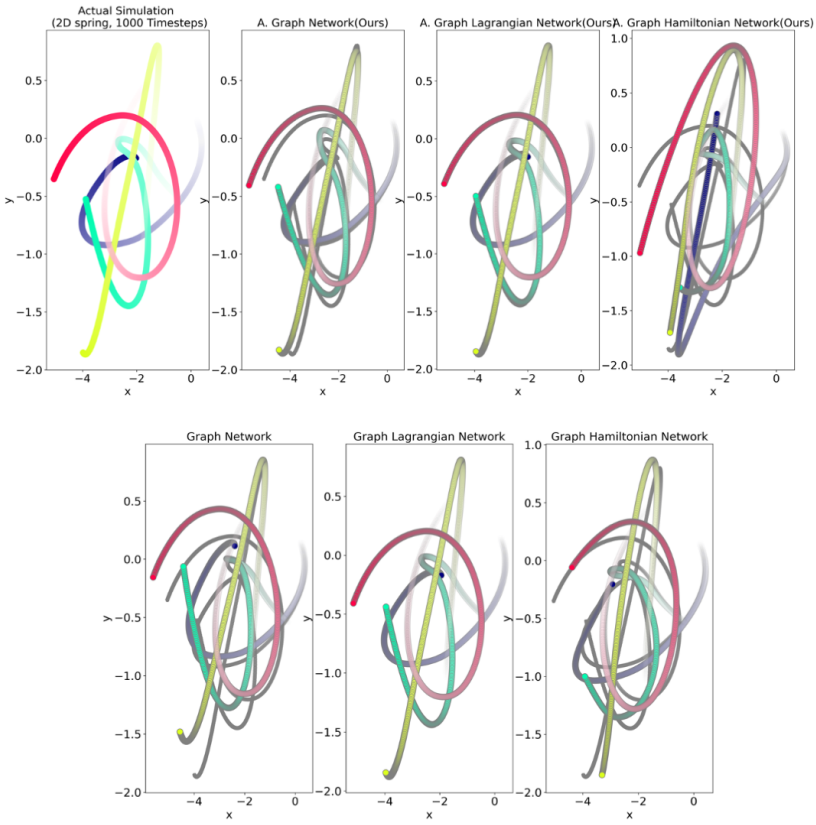


Figure 8: Rollout prediction Comparison 2D Spring

to choice of lookback length and sub-optimal lookback length choices affect the prediction performance as well as causes a blow-up of error (instability). On the contrary, we find that our AGLN and AGHN like AGN are quite stable even when using a sub-optimal lookback length. Further, the

physics constrained GNs outperform GN and AGN across all datasets (Table 1 and Table 2). This shows that enforcing physics constraints such as energy conservation and arrow of time enables GNs to approximate the true underlying dynamics of the system. Figures 7 and 8 visualize rollout predictions of a 2D and 3D dataset for all models when $k = 5$. We can observe that the autoregressive models are able to make robust multi-step predictions with less error.

#### 4.1.3 LOOKBACK LENGTH

The process of choosing the maximum lag or lookback length $k$ in a VAR model requires special attention because inference is dependent on the selected $k$. We find that this process is similarly important in the context of learning algorithms that are conditioned on the previous states of the system. The choice of $k$ is influenced by the dynamics of the system (i.e., how much temporal information is present in the history of previous states) as well as the form a parametric model assumes. Further, the choice of $k$ may cause some models to become unstable when predicting multi-time step ahead predictions. To that end, we perform an ablation study to understand the prediction performance of all the models when $k$ varies. We choose $k$ to vary between 2-7 and retrain each of the models. We observe that in comparison to GN, the AGN performs better across almost all datasets when $k$ is varies between 3-7 and tends to perform poorly when $k = 2$. This is likely due to insufficient temporal information when conditioning on just the current and previous states. Further, we observe that even when $k$ is not optimal for a specific dataset, the prediction is quite stable over time. Figure 4 depicts the MSE for the 3D Gravity ($r2$) dataset as $k$ varies. The optimal prediction performance of AGN is attained when $k = 6$ and that of GN is attained when $k = 2$. Please see supplementary file for lookback length comparisons across datasets. Unlike AGN and GN, we notice AGHN and AGLN to have superior performance across 7 datasets regardless of choice of $k$. Physics constrained GNs are sensitive to the choice of $k$ and tend to perform reliable predictions only when optimal $k$ is known. However, physics constrained AGNs while sensitive to choice of $k$, do not blow-up with error and tend to produce reliable state estimations. The autoregressive constraint regularizes the physics constrained GNs to respect the conservation of energy over time when conditioned on appropriate historical states. Figures 5 and 6 depict the MSE for the 3D Gravity ($r2$) dataset as $k$ varies. Please see supplementary file for lookback length comparisons across datasets.

#### 4.1.4 COMPLEXITY AND FAILURE CASES

The computation of the proposed AGN, AGLN and AGHN model is similar to that of GN, GLN and GHN. The graph $\mathcal{G}$ contains $N$ nodes and because of the assumption of a fully-connected graph due to the nature of the datasets has $N \cdot (N-1)$ edges. A GN with $L$ message passing blocks hence has a computation time of $\mathcal{O}(N^2 L c^2)$, with $c$ being the number of hidden units. For a sparse graph, the computation time becomes $\mathcal{O}(N L c^2)$.

While AGN and its other variants have superior prediction performance, it fails in the 3D Gravity ($r1$) dataset when compared with GN across all $k$. Similarly, when comparing against GLN and GHN, AGLN and AGHN fail in 3D Spring and Damped datasets across all $k$. While the difference in prediction error is not substantial due to the autoregressive regularization, we speculate that the reason for such failure cases could be due to the equal temporal importance enforced upon the conditioning states when different time-steps may actually be contributing differently to the current state acceleration prediction.

## 5 CONCLUSION

In this paper, we introduce a Graph Network with a temporal inductive bias to learn forward physics simulations. The temporal bias is enforced by an Autoregressive encoder that attends to the previous state inputs of a system of particles. The proposed model circumvents sequential computations required to compute latent vectors and therefore does not sacrifice speed for accuracy and instead allows a GN to enjoy the best of both worlds. This bias enables our model to perform robust multi-step predictions when compared against state of the art GNs. Future work can explore varying temporal dependencies (velocity leads to change in position .etc). Further, we speculate that the noise term we dropped from our VAR formulation could be combined with noise and noise-contrastive strategies to further improve the robustness of multi-step forward predictions.

## REFERENCES


Brandon Amos and J Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning*, pp. 136–145. PMLR, 2017. ISBN 2640-3498.

Shaojie Bai, J Zico Kolter, and Vladlen Koltun. Deep equilibrium models. *Advances in Neural Information Processing Systems*, 32, 2019.

Shaojie Bai, Vladlen Koltun, and J Zico Kolter. Multiscale deep equilibrium models. *Advances in Neural Information Processing Systems*, 33:5238–5250, 2020.

Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Rezende, and Koray Kavukcuoglu. Interaction networks for learning about objects, relations and physics. *Advances in Neural Information Processing Systems*, pp. 4509–4517, 2016. ISSN 10495258.

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andrew Ballard, Justin Gilmer, George Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261v3*, 2018.

Ricky T Q Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural Ordinary Differential Equations. In S Bengio, H Wallach, H Larochelle, K Grauman, N Cesa-Bianchi, and R Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/69386f6bb1dfed68692a24c8686939b9-Paper.pdf.

Miles Cranmer, Sam Greydanus, Stephan Hoyer, Peter Battaglia, David Spergel, and Shirley Ho. Lagrangian neural networks. *arXiv*, pp. 1–9, 2020a. ISSN 23318422.

Miles D Cranmer, Alvaro Sanchez-Gonzalez, Peter W Battaglia, Rui Xu, Kyle Cranmer, David N Spergel, and Shirley Ho. Discovering Symbolic Models from Deep Learning with Inductive Biases. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020b. URL https://proceedings.neurips.cc/paper/2020/hash/c9f2f917078bd2db12f23c3b413d9cba-Abstract.html.

Katerina Fragkiadaki, Pulkit Agrawal, Sergey Levine, and Jitendra Malik. Learning Visual Predictive Models of Physics for Playing Billiards. In Yoshua Bengio and Yann LeCun (eds.), *4th International Conference on Learning Representations, {ICLR} 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1511.07404.

Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. Made: Masked autoencoder for distribution estimation. In *International conference on machine learning*, pp. 881–889. PMLR, 2015.

Samuel Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian Neural Networks. In H Wallach, H Larochelle, A Beygelzimer, F d\textquotesingle Alché-Buc, E Fox, and R Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/26cd8ecadce0d4efd6cc8a8725cbd1f8-Paper.pdf.

Eric J. Hall, Søren Taverniers, Markos A. Katsoulakis, and Daniel M. Tartakovsky. GINNs: Graph-Informed Neural Networks for multiscale physics. *Journal of Computational Physics*, 433:1–20, 2021. ISSN 10902716. doi: 10.1016/j.jcp.2021.110192.

Xu Han, Han Gao, Tobias Pffaf, Jian-Xun Wang, and Li-Ping Liu. Predicting Physics in Mesh-reduced Space with Temporal Attention. *arXiv preprint arXiv:2201.09113*, 2022.

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. ISSN 0027-8424.

Yunzhu Li, Jiajun Wu, Russ Tedrake, Joshua B Tenenbaum, and Antonio Torralba. Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. *arXiv preprint arXiv:1810.01566*, 2018.

Michael Lutter, Christian Ritter, and Jan Peters. Deep Lagrangian Networks: Using Physics as Model Prior for Deep Learning. BT - 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019, 2019. URL `https://openreview.net/forum?id=BklHpjCqKm`.

Nima Mohajerin. Modeling dynamic systems for multi-step prediction with recurrent neural networks. 2017.

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

Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W Battaglia. Learning Mesh-Based Simulation with Graph Networks. In *9th International Conference on Learning Representations, {ICLR} 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL `https://openreview.net/forum?id=roNqYL0{_}XP`.

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. ISSN 0021-9991.

Yulia Rubanova, Alvaro Sanchez-Gonzalez, Tobias Pfaff, and Peter Battaglia. Constraint-based graph network simulator. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 18844–18870. PMLR, 2022. URL `https://proceedings.mlr.press/v162/rubanova22a.html`.

Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin Riedmiller, Raia Hadsell, and Peter Battaglia. Graph networks as learnable physics engines for inference and control. *35th International Conference on Machine Learning, ICML 2018*, 10:7097–7117, 2018.

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*, pp. 8459–8468. PMLR, 2020. ISBN 2640-3498.

Zhenjia Xu, Jiajun Wu, Andy Zeng, Joshua B Tenenbaum, and Shuran Song. DensePhysNet: Learning Dense Physical Object Representations Via Multi-Step Dynamic Interactions. In Antonio Bicchi, Hadas Kress-Gazit, and Seth Hutchinson (eds.), *Robotics: Science and Systems XV, University of Freiburg, Freiburg im Breisgau, Germany, June 22-26, 2019*, 2019. doi: 10.15607/RSS.2019.XV.046. URL `https://doi.org/10.15607/RSS.2019.XV.046`.

Shuqi Yang, Xingzhe He, and Bo Zhu. Learning physical constraints with neural projections. *Advances in Neural Information Processing Systems*, 33:5178–5189, 2020.

Ruiyang Zhang, Yang Liu, and Hao Sun. Physics-informed multi-LSTM networks for metamodeling of nonlinear structures. *Computer Methods in Applied Mechanics and Engineering*, 369:113226, 2020. ISSN 0045-7825.