SUPPLEMENTARY MATERIAL: AUTOREGRESSIVE GRAPH NETWORK FOR LEARNING MULTI-STEP PHYSICS

Anonymous authors
Paper under double-blind review

A APPENDIX

A.1 COMPLEXITY ANALYSIS AND PARAMETER COUNT

Table 1: Complexity Analysis [L-# of NN Layers; N-# of Nodes; $|\mathcal{E}|$-# of Edges; F-# of Hidden Neurons in a Single Layer; k-Look Back Length]

Model	Time Complexity	# of Model Parameters
Graph Nets	$\mathcal{O}(LNF^2 + L\|\mathcal{E}\|F^2 + L\|\mathcal{E}\|F)$	2875274
Autoregressive Graph Nets	$\mathcal{O}(LNF^4 + L\|\mathcal{E}\|F^4 + L\|\mathcal{E}\|F)$	2895248
Gated Graph Convolutional Recurrent Nets	$\mathcal{O}(k(LNF^2 + L\|\mathcal{E}\|F^2 + L\|\mathcal{E}\|F))$	7107122
Lagrangian Graph Nets	$\mathcal{O}(LNF^2 + L\|\mathcal{E}\|F^2 + +L\|\mathcal{E}\|F + LNF^2 + L\|\mathcal{E}\|F^2 + L\|\mathcal{E}\|F)$	2407824
Hamiltonian Graph Nets	$\mathcal{O}(LNF^2 + L\|\mathcal{E}\|F^2 + L\|\mathcal{E}\|F + LNF^2 + L\|\mathcal{E}\|F^2 + L\|\mathcal{E}\|F)$	2343057
Autoregressive Lagrangian Graph Nets	$\mathcal{O}(LNF^4 + L\|\mathcal{E}\|F^4 + L\|\mathcal{E}\|F + LNF^4 + L\|\mathcal{E}\|F^4 + L\|\mathcal{E}\|F)$	3263754
Autoregressive Hamiltonian Graph Nets	$\mathcal{O}(LNF^4 + L\|\mathcal{E}\|F^4 + L\|\mathcal{E}\|F + LNF^4 + L\|\mathcal{E}\|F^4 + L\|\mathcal{E}\|F)$	3198613
Graph Transformer Nets (Single Step)	$\mathcal{O}(LNF^2 + L\|\mathcal{E}\|F^2 + L\|\mathcal{E}\|F)$	37271634
Graph Transformer Nets (Multi Step)	$\mathcal{O}(k(LNF^2 + L\|\mathcal{E}\|F^2 + L\|\mathcal{E}\|F))$	45401482

The following calculations assume a sparse adjacency matrix. For a system with N nodes, a L layer MLP node encoder with element-wise activation requires a computation time of $\mathcal{O}(LNF^2 + LN)$. Similarly a L layer MLP edge encoder with element-wise activation requires a computation time of $\mathcal{O}(L|\mathcal{E}|F^2+LN)$. L blocks of message passing without aggregate and update functions using MLPs requires a computation time of $\mathcal{O}(L|\mathcal{E}|F)$ and with L layer MLP aggregate and update functions with element-wise activation require a computation time of $\mathcal{O}(L|\mathcal{E}|F + LNF^2 + L|\mathcal{E}|F^2 + LN)$. Hence the total computation time of a Graph Net with an edge and node encoder before and during message passing is then $\mathcal{O}(L|\mathcal{E}|F+LNF^2+L|\mathcal{E}|F^2+LN)$. Dropping small terms, we can see that the computation time becomes $\mathcal{O}(LNF^2+L|\mathcal{E}|F^2+L|\mathcal{E}|F)$. As for Autoregressive Graph Net, the MLP edge and node encoder before message passing undergoes an additional matrix multiplication with the masks in each layer, therefore bringing the total computation time to $\mathcal{O}(L|\mathcal{E}|F + LNF^4 + L|\mathcal{E}|F^4 + LN)$ and likewise dropping small terms bring the total computation time to $\mathcal{O}(LNF^4 + L|\mathcal{E}|F^4+L|\mathcal{E}|F)$. As for the Lagrangian and Hamiltonian Graph Nets, the computation of gradients during each forward pass requires an additional computation time of $\mathcal{O}(L|\mathcal{E}|F+LNF^2+L|\mathcal{E}|F^2)$, therefore bringing the total computation time to $\mathcal{O}(L|\mathcal{E}|F + LNF^2 + L|\mathcal{E}|F^2 + LN + L|\mathcal{E}|F + LNF^2 + L|\mathcal{E}|F^2)$. Likewise, dropping small terms reduces the big O to $\mathcal{O}(LNF^2 + L|\mathcal{E}|F^2 + L|\mathcal{E}|F + LNF^2 + L|\mathcal{E}|F^2 + L|\mathcal{E}|F)$. As for Autoregressive Lagrangian and Hamiltonian Nets, the total computation time becomes $\mathcal{O}(LNF^4+L|\mathcal{E}|F^4+L|\mathcal{E}|F+LNF^4+L|\mathcal{E}|F^4+L|\mathcal{E}|F)$. Finally, an attention step requires the computation of a query, key and value matrix, along with computation of attention weights and the final state update. Hence, the computation time of L message passing steps with self attention becomes $\mathcal{O}(LNF^2+L|\mathcal{E}|F)$. Unlike the other baselines, we do not consider an edge nor node MLP for the update and aggregation steps. Hence, the total computation time for a single step Graph Transformer Net (GTN) with an edge and node encoder and dropping small terms is $\mathcal{O}(LNF^2 + L|\mathcal{E}|F^2 + L|\mathcal{E}|F)$. As for multi-step Graph Transformer Net, the computation time is is $\mathcal{O}(T(LNF^2 + L|\mathcal{E}|F^2 + L|\mathcal{E}|F))$.

A.2 DATASETS:

The particle-based datasets (Cranmer et al. (2020)) 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. We further note that the particle systems under study are time-invariant since the dynamics of the systems do not vary with time.

A.3 Proposed Model Implementation details:

We implemented all our models along with the baselines using the Pytorch library. All our models were trained and tested on a NVIDIA DGX Station A100 workstation equiped with 4 Tesla V100s, each with a 32 GB GPU capacity. The proposed approaches as well as the baselines were trained with a batch size of 256.

Loss Function: Since we are interested in learning the forward problem, our optimization problem can be stated as follows:

$$\min_{\theta} \quad \|\ddot{\mathbf{r}}_i(t) - \mathbf{f}_\theta(\mathbf{x}_i(t), \boldsymbol{\omega})\|_2^2 \tag{1}$$

Optimization and Hyper-parameters: We note that while we did not perform a fine-grained hyper-parameter optimization, we did a coarse optimization and report them in Table 2. We use Adam optimizer with an exponential learning rate decay from 10^{-4} to 10^{-6} and weight decay (L2 Norm) as e^{-8}. We use Xavier initialization for all weight matrices. While our model can train in significantly less time-steps per dataset, we find that higher learning rates tend to destabilize the training. For the benchmark methods, we use the hyper-parameters reported in their papers but only change the number of GNs (message passing steps) and hidden layer sizes to be consistent with our approach in order to ensure an apples to apples comparison. We note that we set the number of hidden neurons of the Autoregressive encoders as well as the baseline linear encoders to 512. The hidden neuron size of 280 is only set for the MLP node and edge encoders. We note that the input state vectors were not normalized and in fact found input normalization to lead to slightly poorer performance than unnormalized input. Further, we note that we did not observe any significant improvement in performance by increasing the # of message passing blocks beyond 3.

Table 2: Hyper-Parameters

Datasets	# of GN Blocks	MLP Layers	Message Passing Hidden Neurons	Initial Learning Rate	Epochs
2D Spring	3	3	280	$3e^{-4}$	250
2D Damped	3	3	280	$3e^{-4}$	250
2D Gravity ($\frac{1}{r}$)	3	3	280	$3e^{-4}$	250
2D Gravity ($\frac{1}{r^2}$)	3	3	280	$3e^{-4}$	250
2D Charge	3	3	280	$3e^{-4}$	250
3D Spring	3	3	280	$3e^{-4}$	250
3D Damped	3	3	280	$3e^{-4}$	250
3D Gravity ($\frac{1}{r}$)	3	3	280	$3e^{-4}$	250
3D Gravity ($\frac{1}{r^2}$)	3	3	280	$3e^{-4}$	250
3D Charge	3	3	280	$3e^{-4}$	250

A.4 BASELINE MODEL IMPLEMENTATION DETAILS:

While we utilized similar hyper-parameters for the baselines, we present more details on how the baselines were implemented in this section.

A.4.1 GRAPH NETS

Following Sanchez-Gonzalez et al. (2020), we implement Graph Nets in Pytorch. Graph Nets comprise of an Edge MLP, Node MLP that take as input node and edge features and outputs thire corresponding embeddings. We impose translation invariance by providing the node encoder with the current and previous $(k-1)$ velocities along with the mass of the particles. To the edge encoder, we pass the relative position and velocity information along the edges as well as the norm of the relative features. The MLP encoders comprise of hidden linear layers, each followed by a ReLU activation except for the final layer, which is followed by a Layer Norm operation. During our experiments, we found that using a Layer Norm for GNs led to their best performance. Following the encode step, we perform message-passing using an interaction network and finally decode the acceleration targets using an MLP decoder. We note that we used the same hyperparameters as provided by Table 2.

A.4.2 LAGRANGIAN GRAPH NETWORKS

The Lagrangian formulation presents an elegant framework to predict the time evolution or dynamics of the state $(\mathbf{r}(t), \dot{\mathbf{r}}(t))$ of a physical system based on a single scalar function known as the Lagrangian $\mathcal{L}$. The Lagrangian can be expanded as $\mathcal{L}(\mathbf{r}(t), \dot{\mathbf{r}}(t)) = T(\mathbf{r}, \dot{\mathbf{r}}, t) - V(\mathbf{r}, t)$, where $T(\cdot)$ represents the total kinetic energy of the system, while $V(\cdot)$ represents the total potential energy of the system from which generalized forces can be derived. The standard form of Lagrange's equation for a system of particles subject to conservative forces is given by the Euler-Lagrange (EL) equation from which the dynamics of the system can be derived.

$$\frac{d}{dt}(\frac{\partial \mathcal{L}}{\partial \dot{\mathbf{r}}_i}) = \frac{\partial \mathcal{L}}{\partial \mathbf{r}_i} \tag{2}$$

By expanding the time derivative in the EL equation, we can express the acceleration of particles in the following general form: $\ddot{\mathbf{r}}_i = (\nabla_{\dot{\mathbf{r}}} \nabla_{\dot{\mathbf{r}}}^T \mathcal{L})^{-1}[\nabla_{\mathbf{r}} \mathcal{L} - (\nabla_{\mathbf{r}} \nabla_{\dot{\mathbf{r}}}^T \mathcal{L})\dot{\mathbf{r}}]$. The acceleration of the particles in the datasets considered in this paper take on the following form: $\ddot{\mathbf{r}}_i = -\frac{1}{m_i} \sum_j \nabla_{\mathbf{r}_i} V_{ij}$, where $-\sum_j \nabla_{\mathbf{r}_i} V_{ij} = \nabla_{\mathbf{r}_i} \mathcal{L}$ and m_i denotes the mass of particle i. Hence, given the initial position and velocities of particles, we can train a Graph Network described in the earlier sub-section to first compute the Lagrangian which is a scalar that can be obtained by summing the final output from the L^{th} message passing iteration along the row and column axes. Then, we use auto-differentiation to compute the partial derivatives of $\mathcal{L}$ with respect to the current position of each of the particles. Finally, we predict $\hat{\ddot{\mathbf{r}}}(t)$ by substituting the derivatives in the acceleration form of the EL equation and minimize the target error as follows:

$$\min_{\theta} \quad \|\ddot{\mathbf{r}}_i(t) - \frac{\nabla_{\mathbf{r}_i^t} \mathcal{L}_\theta(\mathbf{x}_i(t))}{m_i}\|_2^2 \tag{3}$$

In the case of Autoregressive Graph Lagrangian Nets, we simply replace the linear node and edge encoders with autoregressive node and edge encoders while following the hyper-parameters listed in Table 2.

A.4.3 HAMILTONIAN GRAPH NETWORKS

The Hamiltonian formulation, like its Lagrangian counter-part presents an elegant framework to predict the dynamics of the state given by position and momentum $(\mathbf{r}(t), \mathbf{p}(t))$ of a physical system based on a single scalar function known as the Hamiltonian $\mathcal{H}$. The Hamiltonian $\mathcal{H}(\mathbf{r}(t), \mathbf{p}(t))$ is the Legendre transform of $\mathcal{L}(\mathbf{r}(t), \dot{\mathbf{r}}(t))$ such that $\mathcal{H}(\mathbf{r}(t), \mathbf{p}(t)) = T(\mathbf{r}, \dot{\mathbf{r}}, t) + V(\mathbf{r}, t)$. Moreover, the following partial derivative relations hold: $\frac{\partial \mathcal{L}(\mathbf{r}(t), \dot{\mathbf{r}}(t))}{\partial \mathbf{r}_i} = -\frac{\partial \mathcal{H}(\mathbf{r}(t), \mathbf{p}(t))}{\partial \mathbf{r}_i}$ and $\frac{\partial \mathcal{L}(\mathbf{r}(t), \dot{\mathbf{r}}(t))}{\partial t} = -\frac{\partial \mathcal{H}(\mathbf{r}(t), \mathbf{p}(t))}{\partial t}$

Following Cranmer et al. (2020), we train a Flattened Hamiltonian Graph Network subject to the encode-process-decode setup to approximate $\mathcal{H} = \sum_i \mathcal{H}_{self}(i) + \sum_{(i,j)\in\mathcal{E}} \mathcal{H}_{pair}(i,j)$, where $\mathcal{H}_{self}$

and $\mathcal{H}_{pair}$ are computed following the L^{th} message passing block's UPDATE and AGGREGATE steps. Like the original paper, we also regularize $\mathcal{H}_{pair}$. From the earlier sub-section, we know that $-\sum_j \nabla_{\mathbf{r}_i} V_{ij} = \nabla_{\mathbf{r}_i}\mathcal{L} = -\nabla_{\mathbf{r}_i}\mathcal{H}$. Finally, we predict $\hat{\ddot{\mathbf{r}}}(t)$ by minimizing the target error as follows:

$$\min_{\theta} \quad \|\ddot{\mathbf{r}}_i(t) + \frac{\nabla_{\mathbf{r}_i^t}\mathcal{H}_\theta(\mathbf{x}_i(t))}{m_i}\|_2^2 + \lambda\|\mathcal{H}_{self,\theta}(i) + \mathcal{H}_{pair,\theta}(i,j)\|_2^2 + \lambda\|\nabla_{\boldsymbol{\omega}}\mathcal{H}_\theta(\mathbf{x}_i(t))\|_2^2 \tag{4}$$

Similarly, like Autoregressive Graph Lagrangian Nets, we simply replace the linear node and edge encoders of the Flattened Hamiltonian Graph Network with autoregressive node and edge encoders while following the hyper-parameters listed in Table 2.

A.4.4 Gated Graph Recurrent Neural Network

We implement the Gated Graph Recurrent Neural Network (GGRNN) model proposed by Seo et al. (2018). In addition to using their base GGRNN cell, we implement a linear MLP encoder within the time recursive call to the GGRNN cell that maps each state at time t to a latent embedding of size 280, which is then passed as input to the GGRNN cell. Further, since we adopt the many-to-many training setting, GGRNN predicts the acceleration target ($\ddot{\mathbf{x}}^t$) corresponding to each state $\mathbf{x}^t$. The final hidden layer output from each time recursive call to the GGRNN cell is passed to a decoder to predict the acceleration target corresponding to that time step. Finally, we minimize the following optimization function while training the network end to end. We used the same hyperparameters as provided by Table 2.

$$\min_{\theta} \quad \sum_{t=1}^{k} \|\ddot{\mathbf{r}}_i(t) - \mathbf{f}_\theta(\mathbf{x}_i(t), \boldsymbol{\omega})\|_2^2 \tag{5}$$

A.4.5 Graph Transformer Network

We implement two variants of the Graph Transformer Network (GTN), a single step GTN that predict's the acceleration target given the entire state vector and a multi-step GTN that predicts the acceleration target ($\ddot{\mathbf{x}}^t$) corresponding to each state $\mathbf{x}^t$. To that end, for the single step GTN, we adapt the model proposed by Shi et al. (2020) and like the case of GGRNN, implement a linear MLP encoder that maps the state vector across k time-steps to a latent embedding of size 280, which is then passed as input to the GTN cell. In the case of multi-step GTN, we slightly modify GTN to be similar to the base transformer used in Han et al. (2022). The modifications we made purely reflect the use of previous state's latent embeddings in computing the attention over latent embeddings seen until time t. Unlike the work by Han et al. (2022), we do not pre-compute the latent embeddings, we do not regularize the learning of the latent embeddings for state reconstruction nor do we train the model to minimize latent predictions output by GTN. Since we adopt the many-to-many training setting for training the multi-step GTN, the final hidden layer output from each time recursive call to the GTN is passed to a decoder to predict the acceleration target corresponding to that time step. Finally, we minimize the optimization function present in Eqn.(5) while training the network end to end.

A.5 Additional Analysis

A.5.1 Failure Cases:

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.

A.5.2 PARTIAL TIME TRANSLATION INVARIANCE

Typically, spatial 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 spatial translation invariance leads to lower one-step and multi-step prediction errors across multiple domains. We observe the same phenomenon in our study when we compare the results obtained in Table 3 from the supplementary material for $k = 5$, when we train GNs with/without absolute position. On the contrary, when AGN is trained with/without position, we find that the one-step MSE as well as the multi-step MSE is significantly worse when positional information is not provided to the network. This difference in results could likely stem from the treatment of the data by the models since GNs encodes an undirected feature graph of the data while an AGN encodes a directed temporal graph. In essence, AGN treats the data sequentially while GN does not. Hence, instead of requiring spatial translation invariance, AGNs need to posses time translation invariance (i.e., acceleration shifts when position and velocity are shifted). Work by Van Den Oord et al. (Van Den Oord et al. (2016)) shows that spatial translation invariance in CNNs can be achieved using an autorergessive strategy with weight sharing and also through data augmentation (Biscione & Bowers (2020)). Hence, we hypothesize that AGNs possess partial time-translation invariance as a result of causal convolution between a time shifted trajectory and fixed AGN encoder weights that are shared across shifted trajectory samples. Further, prior work in NLP have shown improvement in machine translation tasks when absolute and relative positional features are encoded Wang & Chen (2020); Huang et al. (2020). As a result, in addition to relative edge information, we find that providing AGN with absolute positions generally leads to better roll-out prediction performance. Figures 1 - 4 plot the mean MSE of a few example datasets for which the GN and AGN models are provided with/without absolute position.

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

Datasets	GN w. position	GN w/o. position	AGN (vector) w. position (Ours)	AGN (vector) w/o. position (Ours)	AGN (scalar) w. position (Ours)	AGN (scalar) w/o. position (Ours)
2D Spring	7.81	6.72	2.68	14.78	**1.36**	13.75
2D Damped	25.12	19.39	12.67	284.55	**7.25**	192.70
2D Gravity ($\frac{1}{r1}$)	**5.43**	14.96	27.58	45.64	17.59	24.82
2D Gravity ($\frac{1}{r2}$)	801.37	299.76	3.58	97.09	3.24	**1.59**
2D Charge	318.36	38.06	8.08	580.14	**1.77**	2.83
3D Spring	23.04	18.09	10.75	31.75	16.17	**5.86**
3D Damped	43.49	202.71	**15.73**	57.47	20.26	451.74
3D Gravity ($\frac{1}{r1}$)	3.50	2.69	7.38	**2.32**	19.58	4.30
3D Gravity ($\frac{1}{r2}$)	564.43	1.42	1.16	2.33	1.59	**1.06**
3D Charge	1.15	3.82	1.05	3.38	**0.47**	2.49

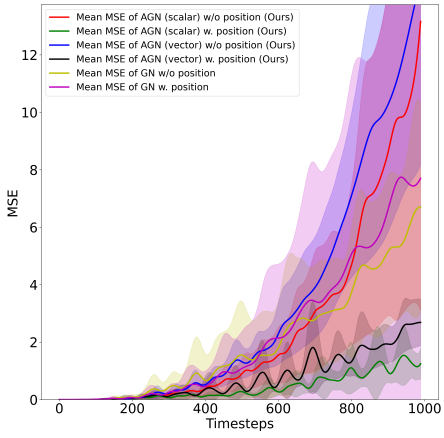

Figure 1: Rollout MSE (2D Spring Dataset)

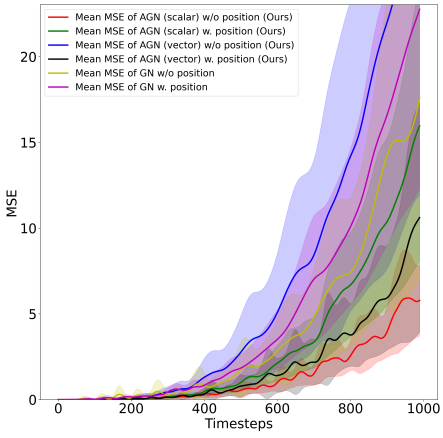

Figure 2: Rollout MSE (3D Spring Dataset)

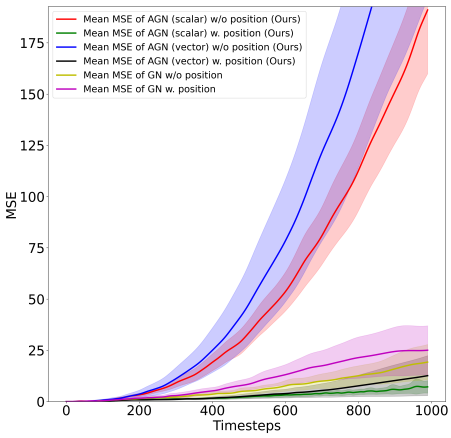

Figure 3: Rollout MSE (2D Damped Spring Dataset)

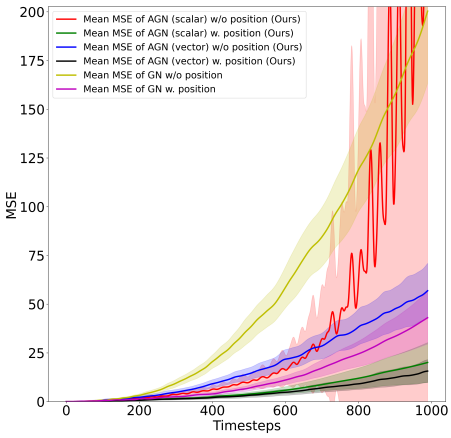

Figure 4: Rollout MSE (3D Damped Spring Dataset)

A.5.3 LOOKBACK LENGTH (GN) - ANALYSIS

The process of choosing the maximum lag or lookback length k in a AR/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 for non-physics constrained models and 3-6 for physics constrained models. We observe that in comparison to the baselines, the AGN performs better across almost all datasets when k 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. Unlike AGN and GN, we notice AGHN and AGLN to have superior performance across 8 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. Below, we include additional look back length comparison graphs for all datasets and for physics-induced and non-physics induced models.

Non-Physics Induced Models: Figures 5 and 6 plot the roll-out energy MSE and roll-out positional MSE across different k and baselines for 2D Spring dataset, while Figures 7 and 8 plot the roll-out MSE and energy MSE for 3D Damped dataset.

Physics Induced Models: Figures 9 and 10 plot the roll-out energy MSE and roll-out positional MSE across different k and baselines for 2D Gravity(R2) dataset, while Figures 11 and 12 plot the roll-out MSE and energy MSE for 3D Charge dataset.

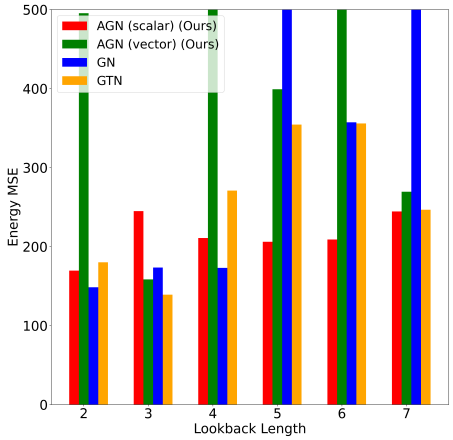

Figure 5: Roll-out Energy MSE Lookback Length Comparison (2D Spring Dataset)

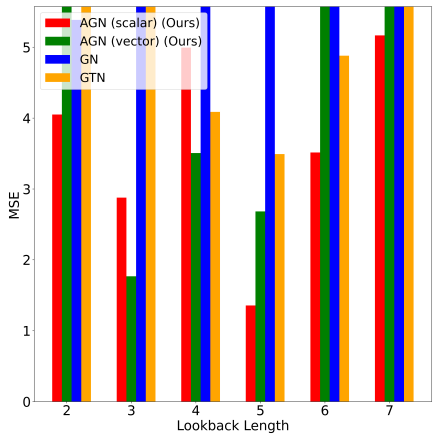

Figure 6: Roll-out MSE Lookback Length Comparison (2D Spring Dataset)

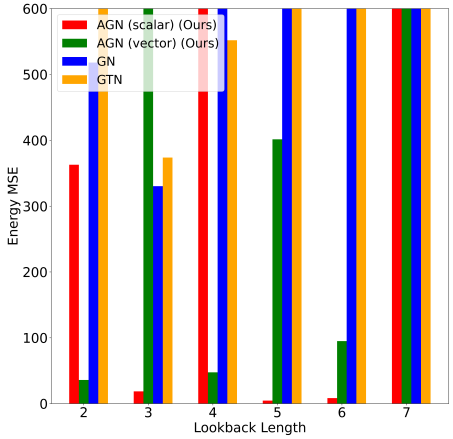

Figure 7: Roll-out Energy MSE Lookback Length Comparison (3D Damped Dataset)

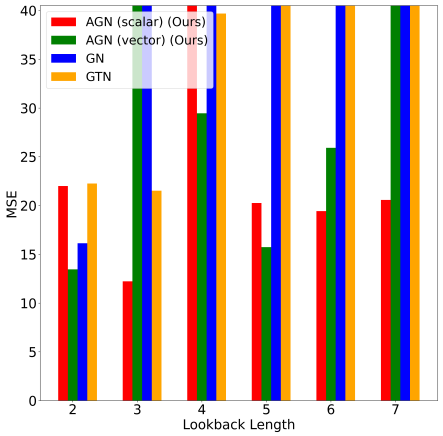

Figure 8: Roll-out MSE Lookback Length Comparison (3D Damped Dataset)

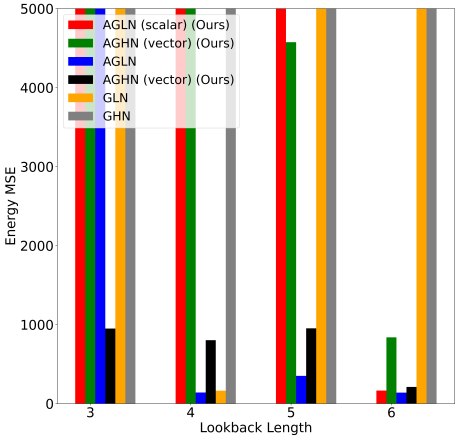

Figure 9: Roll-out Energy MSE Lookback Length Comparison (2D R2 Dataset)

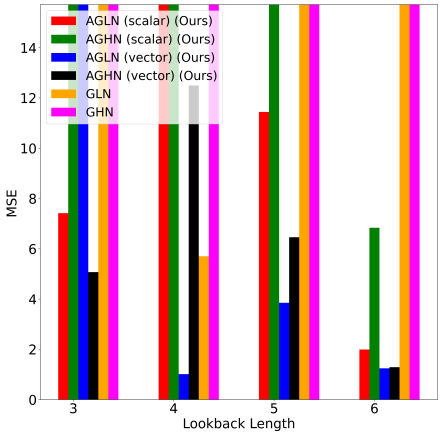

Figure 10: Roll-out MSE Lookback Length Comparison (2D R2 Dataset)

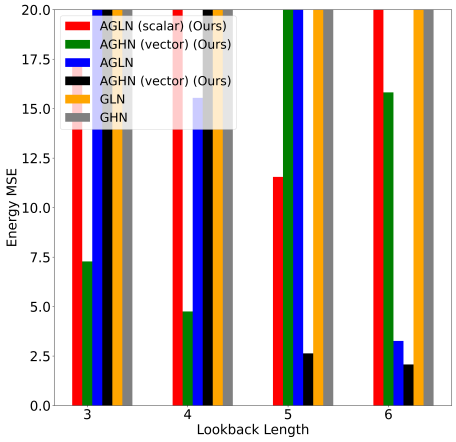

Figure 11: Roll-out Energy MSE Lookback Length Comparison (3D Charge Dataset)

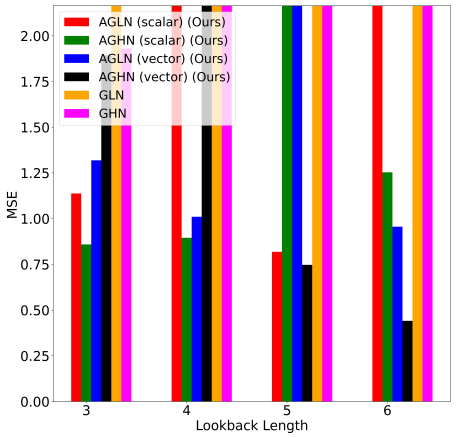

Figure 12: Roll-out MSE Lookback Length Comparison (3D Charge Dataset)

REFERENCES

Valerio Biscione and Jeffrey Bowers. Learning translation invariance in CNNs. *arXiv preprint arXiv:2011.11757*, 2020.

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*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/c9f2f917078bd2db12f23c3b413d9cba-Abstract.html.

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.

Zhiheng Huang, Davis Liang, Peng Xu, and Bing Xiang. Improve transformer models with better relative position embeddings. *arXiv preprint arXiv:2009.13658*, 2020.

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

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, 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.

Youngjoo Seo, Michaël Defferrard, Pierre Vandergheynst, and Xavier Bresson. Structured sequence modeling with graph convolutional recurrent networks. In *International conference on neural information processing*, pp. 362–373. Springer, 2018.

Yunsheng Shi, Zhengjie Huang, Shikun Feng, Hui Zhong, Wenjin Wang, and Yu Sun. Masked label prediction: Unified message passing model for semi-supervised classification. *arXiv preprint arXiv:2009.03509*, 2020.

Aäron Van Den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. In *International conference on machine learning*, pp. 1747–1756. PMLR, 2016.

Yu-An Wang and Yun-Nung Chen. What do position embeddings learn? an empirical study of pre-trained language model positional encoding. *arXiv preprint arXiv:2010.04903*, 2020.