Supplementary Material: Probabilistic Graph Networks for Learning Physics Simulations

0.1 Optimization and Hyper-parameters:

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 equipped with 4 Tesla V100s, each with a 32 GB GPU capacity.

We performed a coarse hyper-parameter optimization and report them in Table 1 and Table 2 for models operating on particle-based and mesh-based datasets respectively. We use Adam optimizer with an exponential learning rate decay from the initial learning rate to 10^{-6} and weight decay (L2 Norm) as $1e^{-8}$. We use Xavier initialization for all weight matrices. We note that the input state vectors were not normalized, in fact, we find input normalization to lead to slightly poorer performance than unnormalized input in the case of particle-based physics datasets. We note that each GN block consists of an edge MLP for neighborhood aggregation and a node MLP for node update, both having the same number of MLP layers and hidden neurons as the node and edge MLP encoders. Further, we note that we did not observe any significant improvement in performance by increasing the # of message passing blocks beyond 3 when training on particle-physics datasets. While different hyper-parameters were used to train Mesh Graph Net and Mesh Graph Net-NF due to differences in the number of nodes and edges in the dataset, Mesh Graph Net has the same architecture as GN while Mesh Graph Net-NF has the same architecture as GN-NF.

Table 1. Hyper-Parameters for Particle-Based Physics Simulations (** - refer to implementation details)

Models	# of GN Blocks	MLP Layers	Hidden Neurons	Initial Learning Rate	Epochs	# of Heads	# of MAF Layers	# of MAF MLP Layers	# of MAF MLP Hidden Neurons	Batch Size	# of Gradient Descent Steps
GN	3	3	280	$7e^{-5}$	400	-	-	-	-	128	$2.2e^{5}$
GLN	3	3	280	$7e^{-5}$	400	-	-	-	-	128	$2.2e^{5}$
GHN	3	3	280	$7e^{-5}$	400	-	-	-	-	128	$2.2e^{5}$
GN-NF	3	3	280	$7e^{-5}$	400	-	3	9	16	128	$2.2e^{5}$
GTN	3	3	280	$7e^{-5}$	400	5	-	-	-	128	$2.2e^{5}$
AGTN	3	3	**	$7e^{-5}$	400	5	-	-	-	128	$2.2e^{5}$
AGTN-NF	3	3	**	$7e^{-5}$	400	5	3	9	16	128	$2.2e^{5}$

Table 2. Hyper-Parameters for Mesh-Based Physics Simulation

Models	# of GN Blocks	MLP Layers	Hidden Neurons	Initial Learning Rate	Epochs	# of MAF Layers	# of MAF MLP Layers	# of MAF MLP Hidden Neurons	Batch Size	# of Gradient Descent Steps
Mesh Graph Net	10	2	128	$9e^{-5}$	1500	-	-	-	6	$2.1e^{6}$
Mesh Graph Net-NF	10	2	128	$9e^{-5}$	1500	2	8	8	6	$2.1e^{6}$

0.2 Model Implementation details:

In this section, we present more details on model implementations.

0.2.1 Graph Network (GN)

Following[1], we implement Graph Nets in Pytorch. Graph Nets comprise an Edge MLP, Node MLP that take as input node and edge features and outputs their 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 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 encoding step, we perform message-passing using an interaction network and finally decode the acceleration targets using an MLP decoder. The final layer of the MLP decoder does not perform a Layer Norm operation. We note that we used the same hyper-parameters as provided by Table 1.

0.2.2 Graph Lagrangian Network (GLN)

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 $\mathscr{L}$. The Lagrangian can be expanded as $\mathscr{L}(\mathbf{r}(t), \dot{\mathbf{r}}(t)) = T(\mathbf{r}, \dot{\mathbf{r}}, t) - V(\mathbf{r}, t)$, 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.

$$\frac{d}{dt}\left(\frac{\partial \mathscr{L}}{\partial \dot{\mathbf{r}}_i}\right) = \frac{\partial \mathscr{L}}{\partial \mathbf{r}_i} \tag{1}$$

By expanding the time derivative in the EL equation, we can express the acceleration of particles in the following general form: $\ddot{\mathbf{r}}_i = (\nabla_{\dot{\mathbf{r}}} \nabla_{\dot{\mathbf{r}}}^T \mathscr{L})^{-1}[\nabla_{\mathbf{r}} \mathscr{L} - (\nabla_{\mathbf{r}} \nabla_{\dot{\mathbf{r}}}^T \mathscr{L})\dot{\mathbf{r}}]$. The acceleration of the particles in the datasets considered in this paper takes on the following form: $\ddot{\mathbf{r}}_i = -\frac{1}{m_i} \sum_j \nabla_{\mathbf{r}_i} V_{ij}$, where $-\sum_j \nabla_{\mathbf{r}_i} V_{ij} = \nabla_{\mathbf{r}_i} \mathscr{L}$ 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 $\mathscr{L}$ with respect to the current position of each of the particles. The MLP encoders comprise hidden linear layers, each followed by a Softplus activation except for the final layer, which is followed by a Layer Norm operation. Following the encoding step, we perform message-passing using an interaction network to compute the Lagrangian. The GLN does not have a decoder and instead the accelerations are predicted as follows: $\hat{\ddot{\mathbf{r}}}_i(t) = \frac{\nabla_{\mathbf{r}_i^t} \mathscr{L}_\theta(\mathbf{x}_i(t))}{m_i}$. Finally, we minimize the target error as follows:

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

0.2.3 Graph Hamiltonian Network (GHN)

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 $\mathscr{H}$. The Hamiltonian $\mathscr{H}(\mathbf{r}(t), \mathbf{p}(t))$ is the Legendre transform of $\mathscr{L}(\mathbf{r}(t), \dot{\mathbf{r}}(t))$ such that $\mathscr{H}(\mathbf{r}(t), \mathbf{p}(t)) = T(\mathbf{r}, \dot{\mathbf{r}}, t) + V(\mathbf{r}, t)$. Moreover, the following partial derivative relations hold: $\frac{\partial \mathscr{L}(\mathbf{r}(t), \dot{\mathbf{r}}(t))}{\partial \mathbf{r}_i} = -\frac{\partial \mathscr{H}(\mathbf{r}(t), \mathbf{p}(t))}{\partial \mathbf{r}_i}$ and $\frac{\partial \mathscr{L}(\mathbf{r}(t), \dot{\mathbf{r}}(t))}{\partial t} = -\frac{\partial \mathscr{H}(\mathbf{r}(t), \mathbf{p}(t))}{\partial t}$

Following[2], we train a Flattened Hamiltonian Graph Network subject to the encode-process-decode setup to approximate $\mathscr{H} = \sum_i \mathscr{H}_{self}(i) + \sum_{(i,j) \in \mathscr{E}} \mathscr{H}_{pair}(i, j)$, where $\mathscr{H}_{self}$ and $\mathscr{H}_{pair}$ are computed following the L^{th} message passing block's UPDATE and AGGREGATE steps. Then, we use auto-differentiation to compute the partial derivatives of $\mathscr{H}$ with respect to the current position of each of the particles. From the earlier sub-section, we know that $-\sum_j \nabla_{\mathbf{r}_i} V_{ij} = \nabla_{\mathbf{r}_i} \mathscr{L} = -\nabla_{\mathbf{r}_i} \mathscr{H}$. The MLP encoders comprise hidden linear layers, each followed by a Softplus activation except for the final layer, which is followed by a Layer Norm operation. Following the encoding step, we perform message-passing using an interaction network to compute the Hamiltonian. The GHN does not have a decoder and instead the accelerations are predicted as follows: $\hat{\ddot{\mathbf{r}}}_i(t) = -\frac{\nabla_{\mathbf{r}_i^t} \mathscr{H}_\theta(\mathbf{x}_i(t))}{m_i}$. Since we are interested to predict only $\hat{\ddot{\mathbf{r}}}(t)$, the following loss function is minimized:

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

0.2.4 Graph Transformer Network (GTN)

We implement a multi-step GTN that predicts the acceleration target ($\ddot{\mathbf{r}}^t$) corresponding to each state $\mathbf{x}^t$. We adapt the model proposed by[3] and implement a linear MLP node encoder and MLP edge encoder that maps the state vector across k time steps to a latent embedding of size 280. We impose translation invariance by providing the node encoder with the current velocities of the particles along with the mass of the particles. To the edge encoder, we pass the current relative position and velocity information along the edges as well as the norm of the relative features. The MLP node and edge encoders follow the same architecture that of GN. The output from the encoders are then passed as input to the Graph Transformer Unit. Further, we modify the GTN architecture to be similar to the base transformer used in[4], except our base Transformer is a Graph Transformer. Unlike the work by[4], we do not pre-compute the latent embeddings nor 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 MLP decoder to predict the acceleration target corresponding to that time step. The final layer of the MLP decoder does not perform a Layer Norm operation. Finally, we minimize the loss function in equation (4) while training the network end to end.

$$\min_{\theta} \quad \sum_{k=1}^{K} \|\ddot{\mathbf{r}}_i(k) - \mathbf{f}_\theta(\mathbf{x}_i(k), \boldsymbol{\omega})\|_2^2 \tag{4}$$

0.2.5 Auto-regressive Graph Transformer Network (AGTN)

The AGTN is similar to GTN in design but instead of MLP node and edge encoders, it uses the proposed auto-regressive (AR) node and edge encoders. In addition to the hidden neuron connections, the AR encoders map a particle's $k \times d$ dimensional state vector to k latent space embeddings. Since the AR encoders output a latent embedding of size $k \times d$, we tune a hyper-parameter ($u = Round(\frac{280}{kd})$) such that the encoders produce a $u \times k \times d$ dimensional latent space embedding. Each $u \times d$ dimensional latent embedding then corresponds to a k^{th} time step state vector input. We impose translation invariance by providing the AR node encoder with the current and the previous $(k-1)$ velocities of the particles. To the AR edge encoder, we pass the relative position and velocity information along the edges as well as the norm of the relative features. The AR encoders comprise hidden linear layers, each followed by a ReLU activation except for the final layer, which is followed by a Layer Norm operation. Further, unlike GTN, wherein the latent embeddings are computed for each time step, AR encoders compute all the embeddings in a parallel fashion. Since the output size of the AR encoders may have different dimensionality (smaller than 280) compared to that of the MLP encoders used in the other models above, we first concatenate the mass of the particles to each of the k latent space embedding and then upsample them using a linear layer to a size of 280 to ensure that the input size to the processor remains fair across all models. The output from the upsampling layer is then passed as input to the Graph Transformer Unit. The rest of the model architecture is exactly the same as GTN, including the loss function to be minimized.

0.2.6 Auto-regressive Graph Transformer Network with Normalizing Flows Decoder (AGTN-NF)

The AGTN-NF model shares the same encoding and processing steps as AGTN. The major difference between them comes from the difference in their decoders. While AGTN uses a MLP decoder, AGTN-NF uses a conditional normalizing flows decoder as explained in the Methods section. Specifically, we use the conditional Masked Autoregressive FLows model[5] as the decoder. The cMAF utilizes an auto-regressive MLP to autoregressively learn the conditional probability distribution of the acceleration targets. The hyper-parameters of the cMAF decoder are reported in Table 1. We expand equation (8) specified in the manuscript and minimize the resulting negative log-likelihood function given by equation (5) to train GN-NF end to end. Equation (6) is the negative log-likelihood function that needs to be minimized when training AGTN-NF.

$$\begin{aligned}
\underset{\phi}{\operatorname{argmin}} D_{KL}(P(\ddot{\mathbf{r}}|\mathbf{x})||Q_\phi(\ddot{\mathbf{r}}|\mathbf{x})) &= \mathbb{E}_{\ddot{\mathbf{r}}|\mathbf{x}\sim P_{data}}[\log P(\ddot{\mathbf{r}}|\mathbf{x}) - \log Q_\phi(\ddot{\mathbf{r}}|\mathbf{x})] \\
&= \mathbb{E}_{\ddot{\mathbf{r}}|\mathbf{x}\sim P_{data}}[\log P(\ddot{\mathbf{r}}|\mathbf{x})] \\
&+ \mathbb{E}_{\ddot{\mathbf{r}}|\mathbf{x}\sim P_{data}}[-\log Q_\phi(\ddot{\mathbf{r}}|\mathbf{x})] \\
&= -\mathscr{H}(\ddot{\mathbf{r}}|\mathbf{x}) + \mathbb{E}_{\ddot{\mathbf{r}}|\mathbf{x}\sim P_{data}}[-\log Q_\phi(\ddot{\mathbf{r}}|\mathbf{x})] \\
&\approx \frac{1}{A}\sum_{l} -\log Q_\phi(\ddot{\mathbf{r}}_a|\mathbf{x})
\end{aligned} \tag{5}$$

$$\min_{\phi} \quad \frac{1}{AK}\sum_{k}^{K}\sum_{a}^{A} -\log Q_\phi(\ddot{\mathbf{r}}_a|\mathbf{x}) \tag{6}$$

where, A denotes the number of training samples, $\log Q_\phi(\ddot{\mathbf{r}}|\mathbf{x}) = \log Q(\mathbf{z}_0) - \sum_{y=1}^{Y} \log \left| \det \left(\frac{\partial f_\phi(r_{y-1}, g(\mathbf{x}))}{\partial r_{y-1}} \right) \right|$, in which $r_0 = \mathbf{z}_0$, $r_y = f_\phi(\mathbf{r}_{y-1}, g(\mathbf{x}))$ and $r_Y = \ddot{\mathbf{r}}$. $\log Q_\phi(\ddot{\mathbf{r}}|\mathbf{x})$ is the log variant of equation (7) provided in the main manuscript.

0.3 Complexity Analysis and Parameter Count

Table 3. Comparison of Model Parameter Count. The # of Model Parameters for all models were computed when $k = 7$.

Model	# of Model Parameters
Graph Network/ Mesh Graph Net	2378043/ 1209474
Graph Network with NF/ Mesh Graph Net with NF	2231924/ 1194864
Graph Lagrangian Network	2231600
Graph Hamiltonian Network	1996960
Graph Transformer Network	8029563
Auto-regressive Graph Transformer Network	8057829
Auto-regressive Graph Transformer Network with NF	7911710

In this section, we compute the time complexity of a single forward pass through each of the models. Let N-# of Nodes; $|\mathscr{E}|$-# of Edges; F-# of Hidden Neurons in a Single Layer; F'-# of Hidden Neurons in a Single Layer of cNF decoder ($F' << F$); k-Look Back Length; d-dimensionality of the physics problem. The following calculations assume a sparse adjacency matrix. For simplicity, let's assume a single layer MLP for neighborhood aggregation and node update functions as well as a single attention head and a batch size of 1 throughout.

For a physics system with N nodes, a L layer MLP node encoder with element-wise activation requires a computation time of $\mathscr{O}(LNF^2 + LN)$. Similarly, a L layer MLP edge encoder with element-wise activation requires a computation time of $\mathscr{O}(L|\mathscr{E}|F^2 + L|\mathscr{E}|)$. L blocks of message passing without aggregate and update functions using MLPs requires a computation time of $\mathscr{O}(L|\mathscr{E}|F)$. L blocks of message passing with a single layer MLP aggregate and update functions and element-wise activation require a computation time of $\mathscr{O}(L|\mathscr{E}|F + LNF^2 + L|\mathscr{E}|F^2 + LN + L|\mathscr{E}|)$. Dropping small and non-dominant terms, the computation time becomes $\mathscr{O}(LNF^2 + L|\mathscr{E}|F^2)$.

As for the Lagrangian and Hamiltonian Graph Nets, the computation of gradients during each forward pass requires an additional computation time of $\mathscr{O}(L|\mathscr{E}|F + LNF^2 + L|\mathscr{E}|F^2)$, therefore bringing the total computation time to $\mathscr{O}(L|\mathscr{E}|F + LNF^2 + L|\mathscr{E}|F^2 + L|\mathscr{E}|F + LNF^2 + L|\mathscr{E}|F^2)$. Dropping the constants after grouping similar terms and non-dominant terms brings the total computation time to $\mathscr{O}(LNF^2 + L|\mathscr{E}|F^2)$.

Graph Transformer Networks and Auto-regressive Graph Transformer Networks include an attention step prior to message passing. An attention step requires the computation of a query, key and value matrix (each operation requiring $\mathscr{O}(NF^2)$), along with the attention weights ($\mathscr{O}(|\mathscr{E}|F)$ for computing α and $\mathscr{O}(|\mathscr{E}|)$ for performing softmax operation). Hence, the computation time of L message passing steps with self-attention and after dropping small and non-dominating terms is $\mathscr{O}(LNF^2 + L|\mathscr{E}|F^2)$. Hence, the total computation time for multi-step Graph Transformer Net is $\mathscr{O}(k(LNF^2 + L|\mathscr{E}|F^2) + LNF^2 + L|\mathscr{E}|F^2)$. Although we can drop the last two terms (non dominating), we are including them to compare against AGTN's complexity. For AGTN, the auto-regressive MLP edge and node encoder undergoes an additional matrix multiplication with the masks in each layer ($\mathscr{O}(NF^4 + |\mathscr{E}|F^4)$), therefore bringing the total computation time of AGTN to $\mathscr{O}(k(LNF^2 + L|\mathscr{E}|F^2) + LNF^4 + L|\mathscr{E}|F^4)$.

All the models above incur the same computation time during a single forward pass during training and inference. However, that is not the case for models with a Normalizing Flows decoder. The conditional Normalizing Flows (cNF) decoder is simply an auto-regressive MLP. The major cost of a cNF that isn't an auto-regressive flow usually comes from the determinant computation which is $\mathscr{O}(d^3)$. While the dimensionality of classical physics problems are typically 2 or 3, auto-regressive flows have a linear time complexity of $\mathscr{O}(d)$ when computing the determinant. However, while density estimation (single forward pass during training) is parallelizable and has a computation time of $\mathscr{O}(LNF'^4)$ (dropping determinant computation time since it is non-dominant), sampling (single forward pass during inference) is sequential and has a computation time of $\mathscr{O}(dLNF'^4)$. Hence, the total computation time of a single forward pass through AGTN-NF during training is $\mathscr{O}(k(LNF'^4 + L|\mathscr{E}|F^2) + LNF^4 + L|\mathscr{E}|F^4)$ while the total computation time of AGTN-NF's forward pass during inference is $\mathscr{O}(k(dLNF'^4 + L|\mathscr{E}|F^2) + LNF^4 + L|\mathscr{E}|F^4)$. The total computation time of GN-NF during inference is $\mathscr{O}(dLNF'^4 + L|\mathscr{E}|F^2)$ and during training is $\mathscr{O}(LNF'^4 + L|\mathscr{E}|F^2)$.

0.4 Additional Results: Backward Simulation - RollOut RMSE

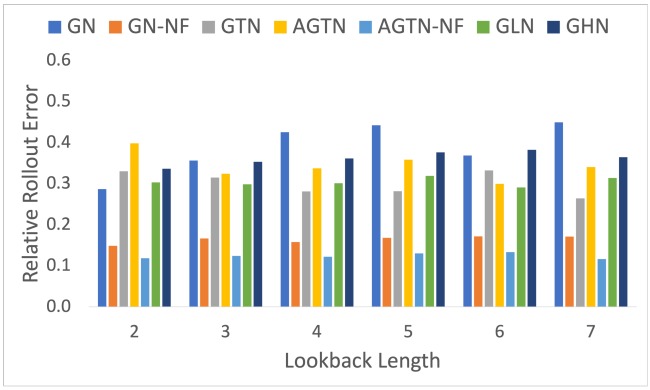
We subject the following models to make time-reversed simulation predictions by reversing the direction of time in our test trajectories. We report the mean rollout RMSE corresponding to the already identified k^* across select models. We further note that these models were not retrained and were only made to perform inference on the time-reversed test trajectories.

Table 4. Mean rollout RMSE comparison for an optimum k^* when time is reversed

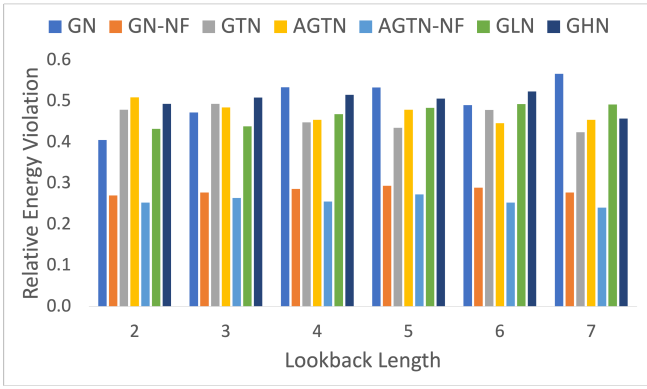
Datasets	GN	GN-NF	GTN	AGTN
2D Spring	**15.08**	15.74	15.46	15.45
2D Damped	**4.33**	4.58	4.58	4.82
2D Gravity ($\frac{1}{r1}$)	8.94	**7.85**	7.95	8.46
2D Gravity ($\frac{1}{r2}$)	7.19	2.83	16.34	**2.68**
2D Charge	3.89	**3.14**	3.72	3.96
3D Spring	18.85	19.19	18.61	**18.54**
3D Damped	5.16	5.12	5.13	**5.11**
3D Gravity ($\frac{1}{r1}$)	10.13	10.28	**10.10**	10.11
3D Gravity ($\frac{1}{r2}$)	3.48	**3.23**	3.32	3.51
3D Charge	7.54	**3.56**	8.93	4.02

0.5 Additional Results: Forward Simulation - Lookback Error Analysis

Figure 1a plots lookback length versus relative rollout error while Figure 1b plots lookback length versus relative energy violation. We note that the relative errors were averaged across datasets and then plotted against lookback length. We find that AGTN-NF and GN-NF have consistent error accumulation as lookback length varies. Although there is a slightly higher variance across lookback length for AGTN, it is still lower than that of GTN. Encoding a temporal causal bias on the input space of the state space enables GTN to become more robust to changes in lookback length.



(a) Lookback Analysis: Relative Rollout Error



(b) Lookback Analysis: Relative Energy Violation

Figure 1. Lookback Analysis

0.6 Additional Results: Forward Simulation - Rollout Error Analysis

Figure 2 plots the average RMSE as well as the 95% confidence intervals for Mesh Graph Net and Mesh Graph Net-NF. While the average RMSE of Mesh Graph Net increases almost linearly, that of Mesh Graph Net is not linear and the rate of error accumulation is much slower, leading to a 70% decrease in RMSE at the end of 600 time steps.

Figures 3-22 plot the average RMSE as well as the 95% confidence intervals for all particle-based physics simulation learning models (both one-step and multi-step) according to their respective datasets. For some datasets, we observe that the RMSE of all models are pretty similar till 200 time steps, after 200 time steps we observe the discrepancy in error accumulation. This further highlights the need for studying the performance of learned simulators over long time steps.

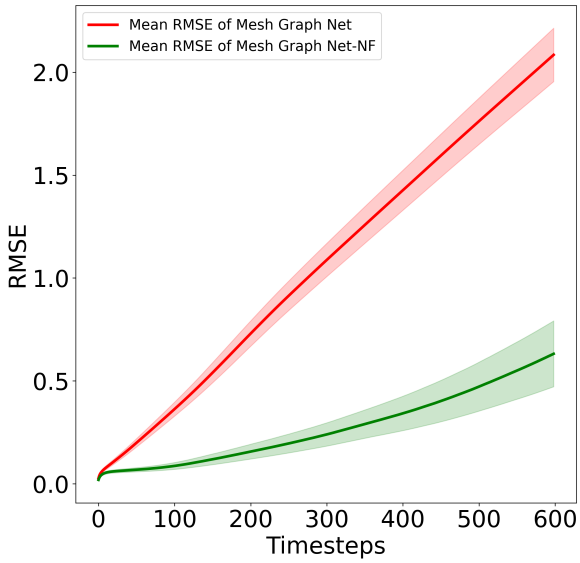


Figure 2. Rollout RMSE Comparison between Mesh Graph Net and Mesh Graph Net-NF on the Cylinder Flow dataset

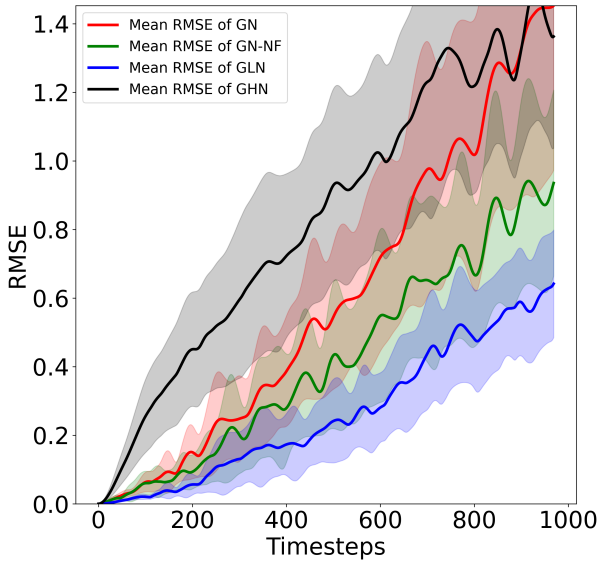


Figure 3. Rollout RMSE (2D Spring Dataset)

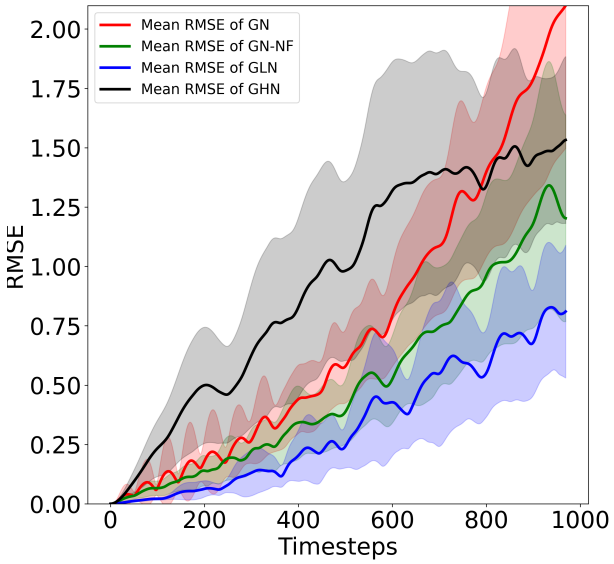


Figure 4. Rollout RMSE (3D Spring Dataset)

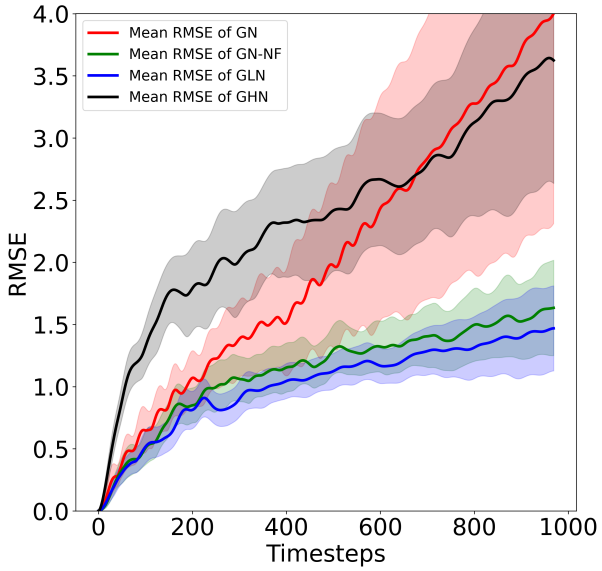


Figure 5. Rollout RMSE (2D Damped Spring Dataset)

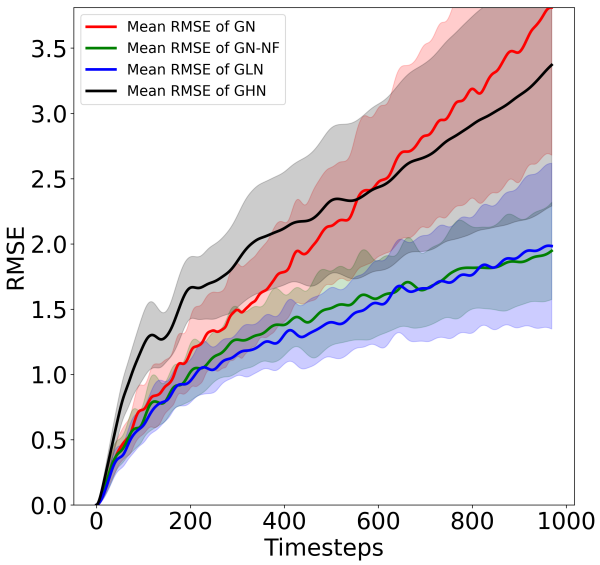


Figure 6. Rollout RMSE (3D Damped Spring Dataset)

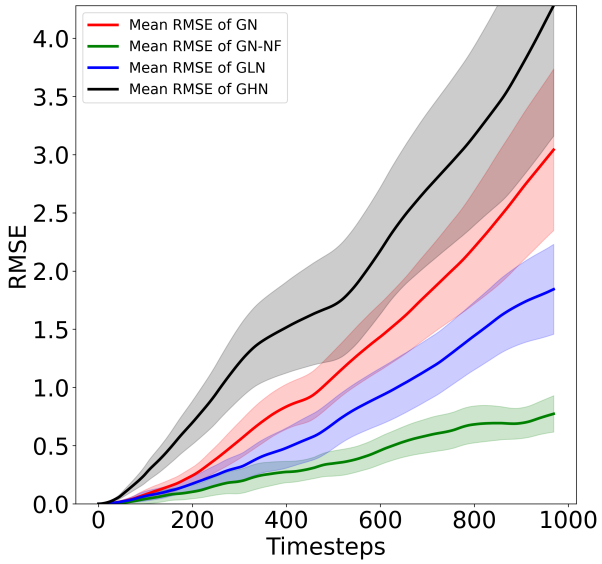


Figure 7. Rollout RMSE (2D r1 Dataset)

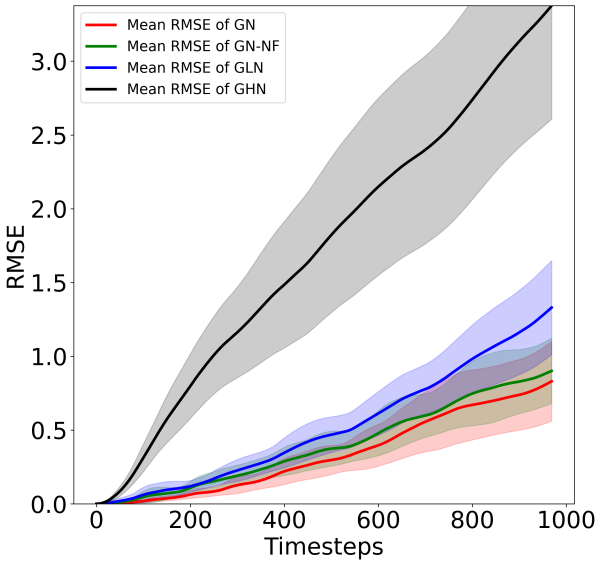


Figure 8. Rollout RMSE (3D r1 Dataset)

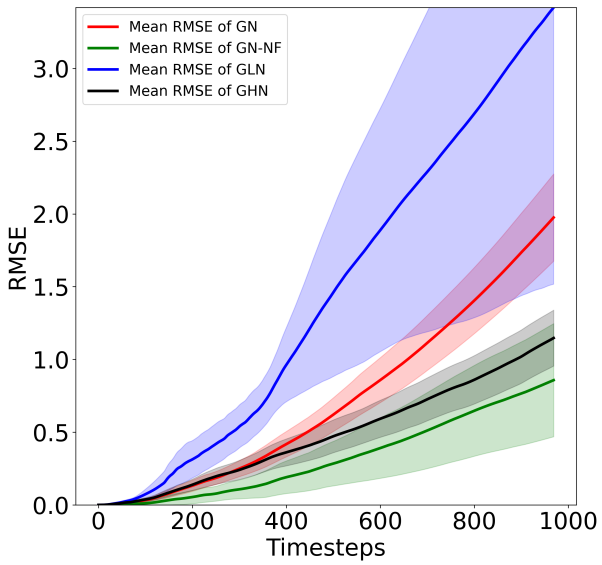


Figure 9. Rollout RMSE (2D r2 Dataset)

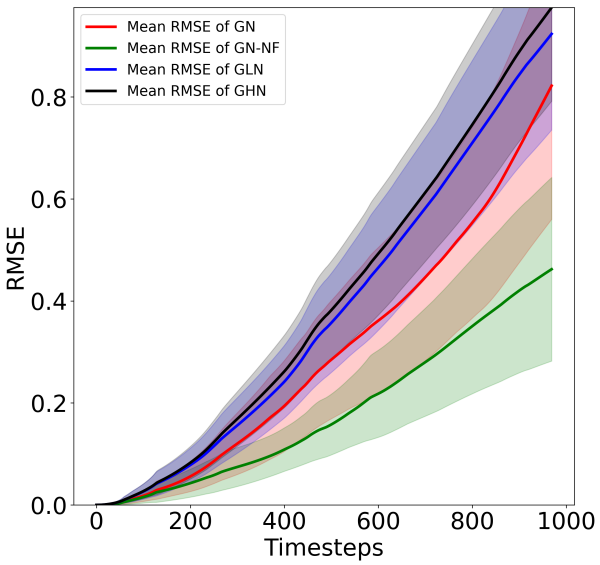


Figure 10. Rollout RMSE (3D r2 Dataset)

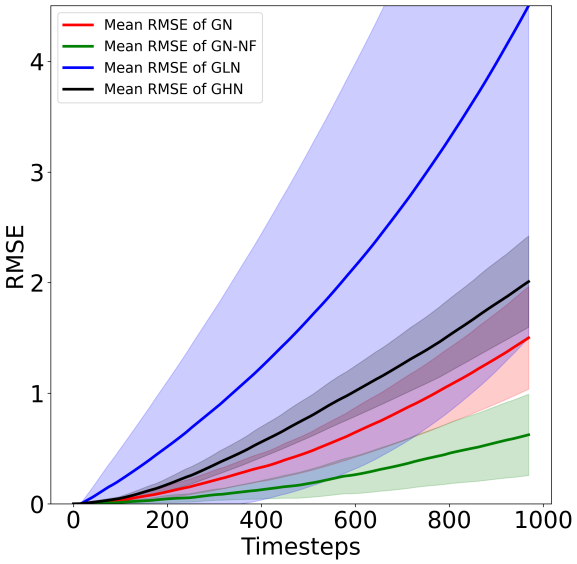


Figure 11. Rollout RMSE (2D Charge Dataset)

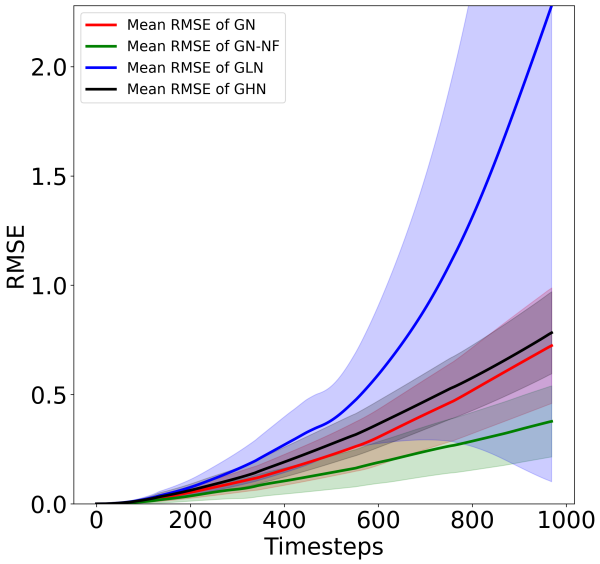


Figure 12. Rollout RMSE (3D Charge Dataset)

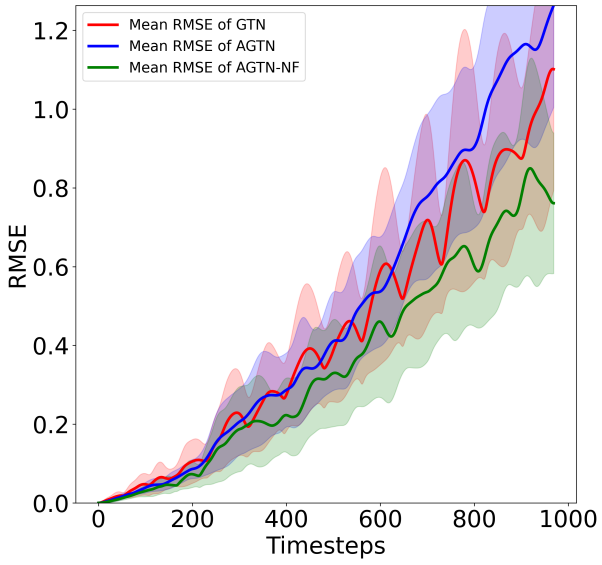


Figure 13. Rollout RMSE (2D Spring Dataset)

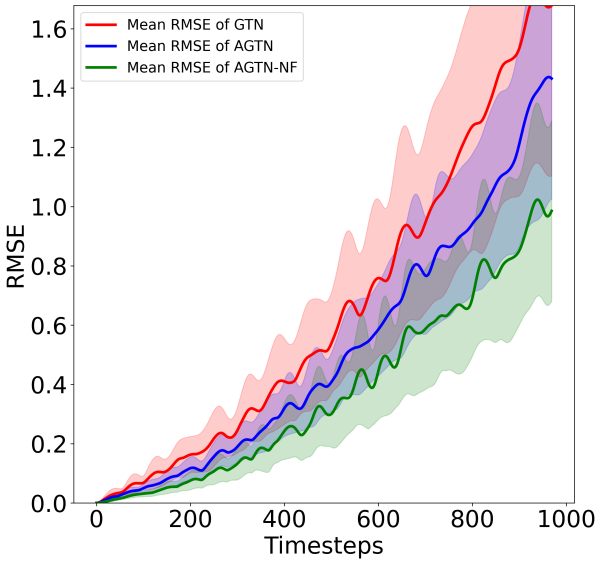


Figure 14. Rollout RMSE (3D Spring Dataset)

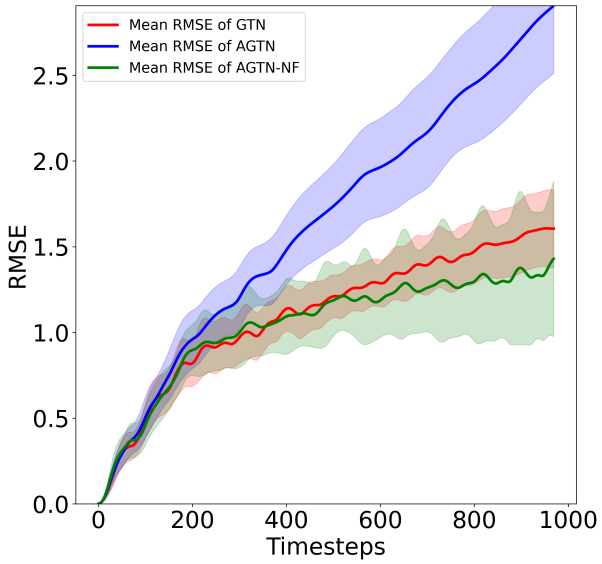


Figure 15. Rollout RMSE (2D Damped Spring Dataset)

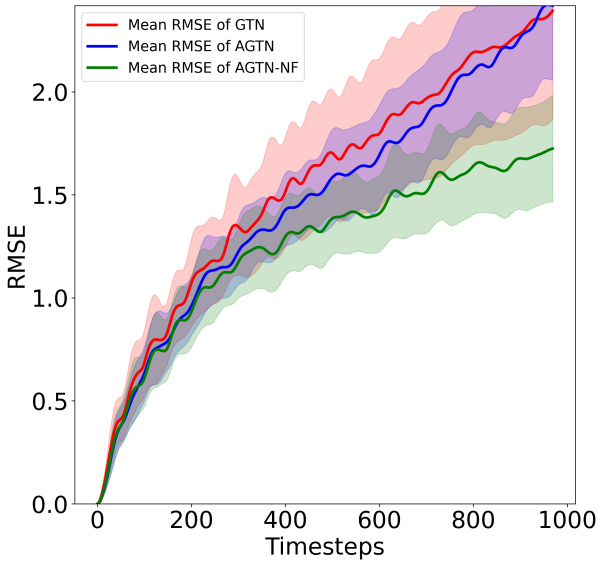


Figure 16. Rollout RMSE (3D Damped Spring Dataset)

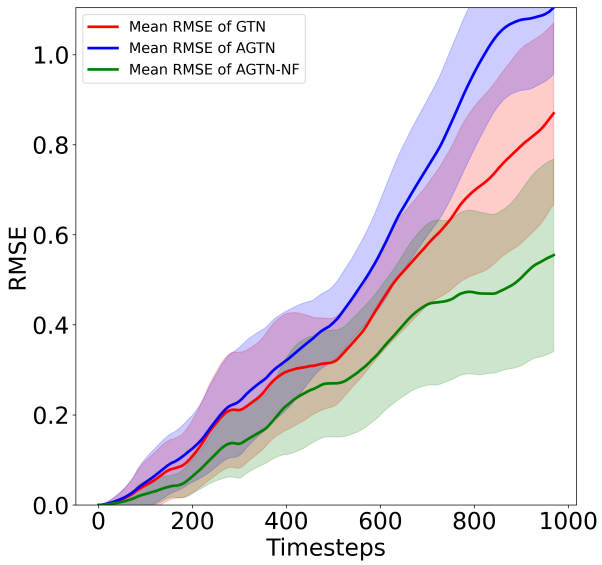


Figure 17. Rollout RMSE (2D r1 Dataset)

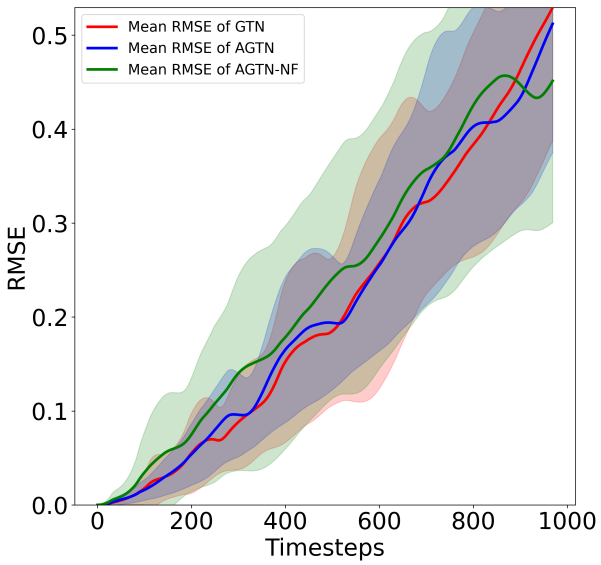


Figure 18. Rollout RMSE (3D r1 Dataset)

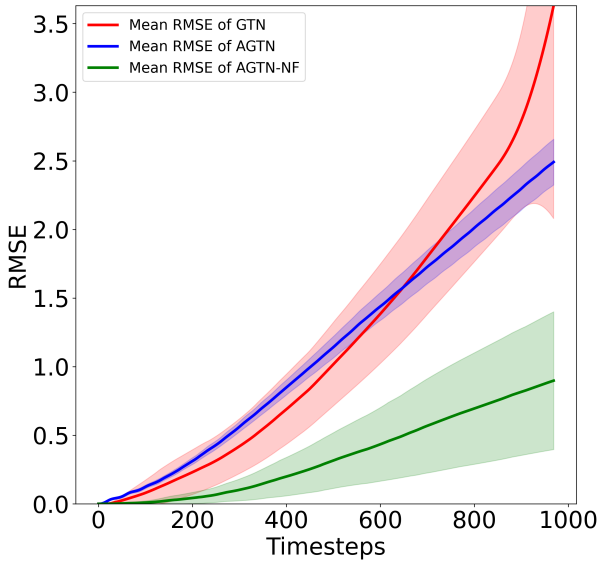


Figure 19. Rollout RMSE (2D r2 Dataset)

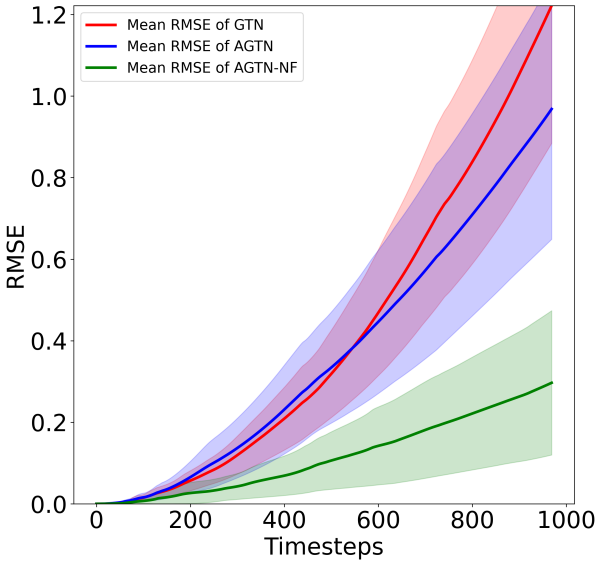


Figure 20. Rollout RMSE (3D r2 Dataset)

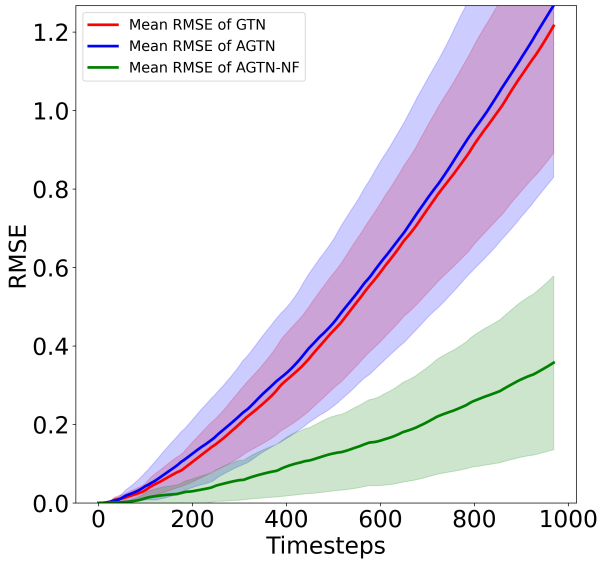


Figure 21. Rollout RMSE (2D Charge Dataset)

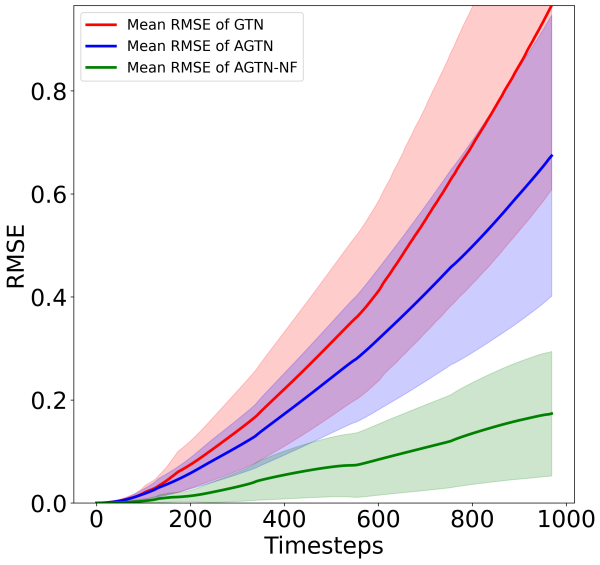


Figure 22. Rollout RMSE (3D Charge Dataset)

References

1. Sanchez-Gonzalez, A. *et al.* Learning to simulate complex physics with graph networks. In *International Conference on Machine Learning*, 8459–8468 (PMLR, 2020).

2. Cranmer, M. D. *et al.* Discovering Symbolic Models from Deep Learning with Inductive Biases. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M.-F. & Lin, H.-T. (eds.) *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual* (2020).

3. Shi, Y. *et al.* Masked label prediction: Unified message passing model for semi-supervised classification. *arXiv preprint arXiv:2009.03509* (2020).

4. Han, X., Gao, H., Pffaf, T., Wang, J.-X. & Liu, L.-P. Predicting Physics in Mesh-reduced Space with Temporal Attention. *arXiv preprint arXiv:2201.09113* (2022).

5. Papamakarios, G., Pavlakou, T. & Murray, I. Masked Autoregressive Flow for Density Estimation. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, 2335–2344 (Curran Associates Inc., Red Hook, NY, USA, 2017).