Hamiltonian Graph Networks with ODE Integrators

Alvaro Sanchez-Gonzalez
DeepMind
London, UK
alvarosg@google.com

Victor Bapst
DeepMind
London, UK
vbapst@google.com

Kyle Cranmer
NYU
New York, USA
kc90@nyu.edu

Peter Battaglia
DeepMind
London, UK
peterbattaglia@google.com

Abstract

We introduce an approach for imposing physically informed inductive biases in learned simulation models. We combine graph networks with a differentiable ordinary differential equation integrator as a mechanism for predicting future states, and a Hamiltonian as an internal representation. We find that our approach outperforms baselines without these biases in terms of predictive accuracy, energy accuracy, and zero-shot generalization to time-step sizes and integrator orders not experienced during training. This advances the state-of-the-art of learned simulation, and in principle is applicable beyond physical domains.

1 Introduction

Learning to simulate complex physical processes has been shown to benefit from rich inductive biases about the structure of the system's state, such as modeling particles and their interactions via nodes and edges in a graph [1–5], and optimizing internal representations to predict known physical quantities [6]. Here we explore another class of physically informed inductive biases: that a system's dynamics can be modeled as an ordinary differential equation (ODE) and formulated as Hamiltonian mechanics. These have also been individually studied recently by Chen et al. [7], Rubanova et al. [8] (ODE bias), and Greydanus et al. [9] (Hamiltonian bias), but here we combine them and incorporate both into graph-network-based neural architectures [10, 1, 3] for learning simulation.

Our experiments show that our trained models have greater predictive accuracy than baselines, as well as better energy conservation (via the Hamiltonian inductive bias) and stronger generalization to novel time-steps and integrator orders (via the ODE integrator inductive bias). These inductive biases also carry trade-offs: with lower order integrators and coarse time-steps, both our learned Hamiltonian model as well as a model which uses the true Hamiltonian derived from physics struggle, because the Hamiltonian puts hard constraints on how the higher order structure of the dynamics must be modeled, whereas the less constrained models can learn ways of approximating such structure.

2 Background

Graph networks We represent a particle system as a graph whose nodes correspond to particles, and with edges connecting all nodes to each other. All of our models use a graph network (GN) [10], which operates on graphs $G = (\mathbf{u}, V, E)$ with global features, $\mathbf{u}$, and variable numbers of nodes, V, and edges, E. Here we focus on GNs which can compute per-node outputs, $V' = \mathrm{GN}_V(G)$,

Second Workshop on Machine Learning and the Physical Sciences (NeurIPS 2019), Vancouver, Canada.

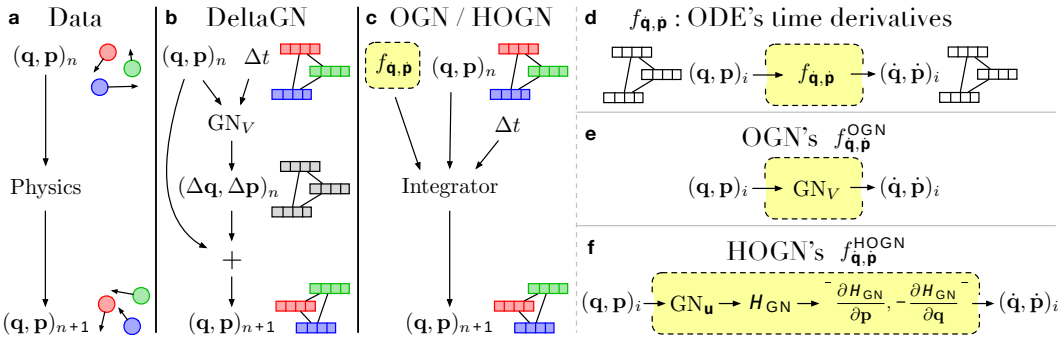

Figure 1: **(a)** The data reflects a system's temporal dynamics, which is governed by physics. **(b)** The baseline DeltaGN model takes as input a state represented by a graph and a time-step, and uses a GN_V to predict state changes. **(c)** Our OGN and HOGN models take as input an input state, time-step, and a function to evaluate the ODE's time derivatives, $f_{\mathbf{q},\mathbf{p}}$, and uses an integrator, which queries $f_{\mathbf{q},\mathbf{p}}$ at different points, to predict the state after the time-step. **(d)** The $f_{\mathbf{q},\mathbf{p}}$ takes as input any state and outputs its time derivatives. **(e)** The OGN model uses a GN_V as $f^{\mathrm{OGN}}_{\mathbf{q},\mathbf{p}}$. **(f)** The HOGN model's $f^{\mathrm{HOGN}}_{\mathbf{q},\mathbf{p}}$ uses a $\mathrm{GN}_{\mathbf{u}}$ to predict the Hamiltonian, which is then differentiated w.r.t. the input state.

and global outputs $\mathbf{u}' = \mathrm{GN}_{\mathbf{u}}(G)$. Preliminary experiments show our GN model outperformed an MLP-based approach by several orders of magnitude, consistently with previous work [1, 3].

Numerical integrators for solving ODEs Given a first-order ODE and initial conditions, numerical integration can be used to approximate solutions to the initial value problem,

$$\mathbf{y} \quad \mathbf{y}(t) \quad , \quad \dot{\mathbf{y}} \quad \frac{\mathrm{d}\mathbf{y}}{\mathrm{d}t} = f_{\mathbf{y}}(t, \mathbf{y}) \quad , \quad \mathbf{y}(t_0) = \mathbf{y}_0 \quad . \tag{1}$$

The family of Runge-Kutta (RK) integrators are fully differentiable and can generate trajectories via iterations of the form, $\mathbf{y}_{n+1} = \mathrm{RK}(t_n, \Delta t, \mathbf{y}_n, f_{\mathbf{y}})$, where $\Delta t = t_{n+1} \quad t_n$. The lowest order RK integrator (Euler integrator, RK1) iterates as, $\mathbf{y}_{n+1} = \mathbf{y}_n + \Delta t \quad f_{\mathbf{y}}(t_n, \mathbf{y}_n)$. Higher order RK integrators produce more accurate trajectories by composing multiple queries to the function $f_{\mathbf{y}}$. Here we explore first- through fourth-order RK integrators: RK1, RK2, RK3, and RK4 (See Supp. Sec. D.1 for results with symplectic integrators).

Hamiltonian mechanics In Hamiltonian mechanics, the Hamiltonian, $H(\mathbf{q}, \mathbf{p})$, is a function of the canonical position, $\mathbf{q}$, and momentum, $\mathbf{p}$, coordinates, and usually corresponds to the energy of the system [1]. The dynamics of the system follow Hamilton's equations, two first-order ODEs (analogous to Eq. 1 with $\mathbf{y} = (\mathbf{q}, \mathbf{p})$),

$$\dot{\mathbf{q}} \quad \frac{\mathrm{d}\mathbf{q}}{\mathrm{d}t} = \frac{\partial H}{\partial \mathbf{p}} \quad , \quad \dot{\mathbf{p}} \quad \frac{\mathrm{d}\mathbf{p}}{\mathrm{d}t} = \quad \frac{\partial H}{\partial \mathbf{q}} \quad) \quad (\dot{\mathbf{q}}, \dot{\mathbf{p}}) \quad \frac{\mathrm{d}(\mathbf{q}, \mathbf{p})}{\mathrm{d}t} = f_{\mathbf{q},\mathbf{p}}(\mathbf{q}, \mathbf{p}) \quad . \tag{2}$$

3 Models

Delta graph network (DeltaGN) Our main baseline, a "delta graph network" (DeltaGN), replicates previous approaches for learning to simulate [1, 3], and directly predicts changes to $\mathbf{q}$ and $\mathbf{p}$,

$$(\mathbf{q}, \mathbf{p})_{n+1} = (\mathbf{q}, \mathbf{p})_n + (\Delta\mathbf{q}, \Delta\mathbf{p})_n \quad \text{, where} \quad (\Delta\mathbf{q}, \Delta\mathbf{p})_n \quad \mathrm{GN}_V(\Delta t, \mathbf{q}_n, \mathbf{p}_n, \mathbf{c}; \phi) \quad . \tag{3}$$

The $\mathbf{c}$ are static parameters (masses, spring constants) of the system, and ϕ are the neural network parameters. The GN's signature matches the integrator's, and so is analogous to learning an integrator.

[1] Our methods are also compatible with time-dependent Hamiltonians, $\mathcal{H}(t, \mathbf{q}, \mathbf{p})$. However, since our dataset does not require time dependency, for simplicity we will omit t everywhere from here on.

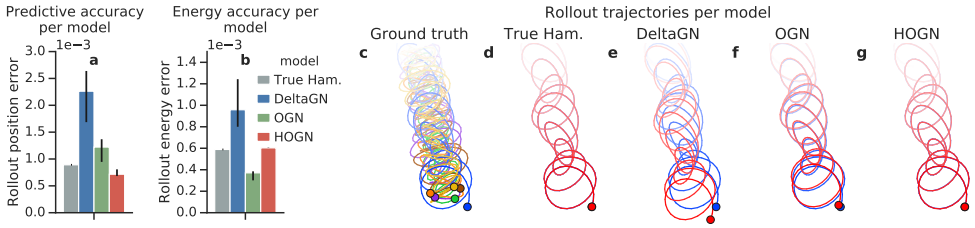

Figure 2: **(a)** Predictive accuracy (on 20-step trajectory) across models using RK4 for the ODE based models. The HOGN is most accurate. **(b)** Energy accuracy across the same models. **(c-g)** Last 300 steps of a 500-step trajectory of a 6-particle system, where dots indicate the final position of the particles (colors fade into the past). **(c)** Ground truth trajectory for all particles. **(d-g)** Trajectory for one of the particles (blue) superimposed by the trajectory obtained (red) when integrating the True Hamiltonian, or **(e-g)** using the best seed of the different learned models, at a time step of 0.1. While errors of all models are very small at the beginning of the trajectory, the HOGN is the only learned model still indistinguishable from ground truth at the end of the long 500-step trajectory (video link).

ODE graph network (OGN) Our "ODE graph network" (OGN) imposes an ODE integrator as an inductive bias in the GN, by assuming that the dynamics of $(\mathbf{q}, \mathbf{p})$ follow a first-order ODE (Eq. 1). We train a neural network that learns the ODE, that is, learns to produce the time derivatives $(\dot{\mathbf{q}}, \dot{\mathbf{p}})$ (which are independent from Δt). We again use the node output of a GN to model the per-particle time derivatives, and provide the GN, together with the initial conditions and Δt, to an RK integrator,

$$(\mathbf{q}, \mathbf{p})_{n+1} = \mathrm{RK}\left(\Delta t, (\mathbf{q}, \mathbf{p})_n, f^{\mathrm{OGN}}_{\mathbf{q},\mathbf{p}}\right) \tag{4}$$

$$f^{\mathrm{OGN}}_{\mathbf{q},\mathbf{p}}(\mathbf{q}, \mathbf{p}) \quad \mathrm{GN}_V(\mathbf{q}, \mathbf{p}, \mathbf{c}; \phi) = (\dot{\mathbf{q}}, \dot{\mathbf{p}}) \quad . \tag{5}$$

The $f_{\mathbf{q},\mathbf{p}}$ is a function that the integrator can use to operate on any $(\mathbf{q}, \mathbf{p})$ and query more than once. For fixed Δt and when using the RK1/Euler integrator (see Sec. 2) this is equivalent (up to a scale factor) to the DeltaGN.

Hamiltonian ODE graph network (HOGN) Our "Hamiltonian ODE graph network" (HOGN) imposes further constraints by using a $\mathrm{GN}_{\mathbf{u}}$ to compute a single scalar for the system—the Hamiltonian—via the global output, analytically differentiating it with respect to its inputs, $\mathbf{q}$ and $\mathbf{p}$, and, in accordance with Hamilton's equations (Eq. 2), treating these gradients as $\dot{\mathbf{p}}$ and $\dot{\mathbf{q}}$, respectively,

$$H_{\mathrm{GN}}(\mathbf{q}, \mathbf{p}) = \mathrm{GN}_{\mathbf{u}}(\mathbf{q}, \mathbf{p}, \mathbf{c}; \phi) \tag{6}$$

$$f^{\mathrm{HOGN}}_{\mathbf{q},\mathbf{p}}(\mathbf{q}, \mathbf{p}) \quad \frac{\partial H_{\mathrm{GN}}}{\partial \mathbf{p}}, \quad \frac{\partial H_{\mathrm{GN}}}{\partial \mathbf{q}} \quad = (\dot{\mathbf{q}}, \dot{\mathbf{p}}) \quad . \tag{7}$$

The resulting ODE defined by $f^{\mathrm{HOGN}}_{\mathbf{q},\mathbf{p}}$ can be integrated similarly to the OGN by passing those functions to the integrator (Eq. 4). Crucially, the H_{GN} is not supervised directly, but instead learned end-to-end through the integrator.

4 Results

We train and test our approach on datasets consisting of particle systems where particle j exerts a spring force on particle i, as defined by Hooke's law, $\mathbf{F}^{ij} = \quad k^{ij} \quad (\mathbf{q}^i \quad \mathbf{q}^j)$, where k^{ij} is the spring constant. All systems contained between 4 and 9 particles, and simulations were computed with time-steps of 0.005. To form the data from the raw simulations, we sub-sampled the trajectories at a fixed time-step of 0.1, except in the time-step generalization conditions described below. We trained all models to make next-step predictions of all particles' positions and momenta, with a loss function which penalized mean squared error between the true and predicted values.

Fig. 2a shows that for models trained and tested on RK4, the OGN and HOGN have lower rollout error than the DeltaGN, and even lower than integrating the true Hamiltonian at the same time step. Fig. 2b shows that the average energy of the system also stays closer to the initial energy for the OGN and HOGN. The OGN energy seems to preserve energy better than the HOGN, however this is not

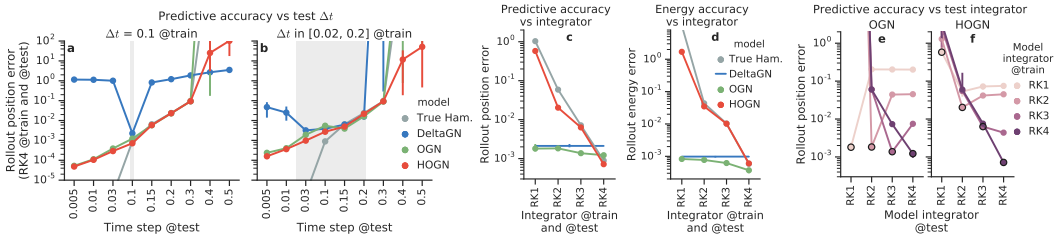

Figure 3: **(a-b)** Time-step size generalization error in 20-step-long trajectories when trained with **(a)** a fixed time-step of 0.1 or **(b)** variable time-steps (0.02-0.2). **(c-d)** Predictive accuracy and energy conservation across models and integrators. **(e-f)** Results when varying the integrator used at test time, where points that share the same train and test integrator are highlighted with black circles.

surprising: as shown below, the HOGN matches the true Hamiltonian well, and both have imperfect energy conservation when integrated with RK4, which is not symplectic[2]. Similar experiments with a third-order symplectic integrator[11] yielded similar rollout errors for the HOGN, with 3 times better energy conservation than the OGN (See Supp. Sec. D.1). Figs. 2c-g (and video link) illustrate the rollout accuracy of the different models for a long 500-step trajectory.

Generalizing to untrained time-steps Both the OGN and HOGN exhibit better zero-shot generalization than the DeltaGN on time-steps which were never experienced during training. Fig. 3a shows each model's performance when trained on a time-step of 0.1 and tested on time-steps between 0.005 and 0.5. We also trained models on multiple time-steps, from 0.02 to 0.2, to study whether that could improve generalization. We varied the time-steps of the ODE models via the integrator, while for the DeltaGN we provided the time-step as input to the model. Fig. 3b shows that the DeltaGN is on par with the other two models within the training range, but becomes worse outside this range. Note, predictions for long time-steps (> 0.3) become very inaccurate for all learned models, as well as the true Hamiltonian, which suggests that the models are determining the dynamics on the basis of features only appropriate for shorter time-steps. Results for other integrators in Supp. Sec. D.2.

Generalization across integrators We varied the integrators used by the model to examine how the quality of the integrator interacts with our different models. Figs. 3c-d show that for lower order integrators (RK1-3), the HOGN and true Hamiltonian have higher rollout and energy errors than the non-Hamiltonian models (DeltaGN and OGN). We speculate the non-Hamiltonian models are more accurate because they are not obligated to respect the constraint the Hamiltonian imposes between the $\mathbf{q}$'s and $\mathbf{p}$'s dynamics, and can instead, for instance, learn approximations to the time derivatives which are not consistent with the $\mathbf{q}$ and $\mathbf{p}$ gradient vector fields of any Hamiltonian.

However, Fig. 3e shows that when the OGN is tested on integrators different from those on which it was trained, its performance is always significantly worse. This indicates that the OGN is not learning accurate $(\dot{\mathbf{q}}, \dot{\mathbf{p}})$, but rather some other function, which interfaces with the training-time integrator to produce accurate trajectories, but is inappropriate for other integrators. By contrast, Fig. 3f shows that the HOGN model usually generalizes to greater accuracy when used with higher order integrators at test time, and specifically when trained with RK4 (dark purple line in Fig. 3f), perfectly matches the behavior of the true Hamiltonian (grey line in Fig. 3c).

Greydanus et al. [9], in their Hamiltonian Neural Networks, use finite differences to approximate the derivatives of $\mathbf{q}$ and $\mathbf{p}$ when not available analytically, use those approximated values during training to supervise the Hamiltonian gradients, and then generate test trajectories with an RK4 integrator. In our experiments this is analogous to training the HOGN using RK1 and testing on RK4, and indeed, our findings are generally consistent with theirs, but also indicate that directly learning through higher order integrators is generally better.

[2]Symplectic integrators are designed to have more accurate energy conservation.

5 Discussion and future work

We incorporated two physically informed inductive biases—ODE integrators and Hamiltonian mechanics—into graph networks for learning simulation, and found they could improve performance, energy, and zero-shot time-step generalization. We also analyzed the impact of varying the ODE integrator independently between training and test, and found that the Hamiltonian approaches benefit most from training with an RK4 integrator, while non-Hamiltonian ones can learn more accurate predictions when trained on lower order integrators. However the HOGN generalizes better across integrators, and approximates the true Hamiltonian best in terms of performance and energy accuracy.

Similar to the graph network's inductive bias for representing complex systems as nodes interacting via edges, the ODE integrator and Hamiltonian inductive biases are informed by physics, but are not specific to physics. Our approach may be applicable to many complex multi-entity systems which can be modeled with ODEs or have some notion of canonical position and momentum coordinates. Moreover, there are other physically informed inductive biases to explore in future work, such as imposing separable potential and kinetic energy terms, Lagrangian mechanics, time-reversibility, or entropic constraints.

Acknowledgments

We acknowledge Irina Higgins, Danilo Rezende, Razvan Pascanu, Shirley Ho and Miles Cranmer for helpful discussions and comments on the manuscript. KC is partially supported by NSF awards OAC-1836650 and the Moore-Sloan Data Science Environment at NYU.

References

[1] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Rezende, and Koray Kavukcuoglu. Interaction networks for learning about objects, relations and physics. In *Proceedings of the 30th Conference on Neural Information Processing Systems (NeurIPS 2016)*, 2016.

[2] Michael B. Chang, Tomer Ullman, Antonio Torralba, and Joshua B. Tenenbaum. A compositional object-based approach to learning physical dynamics. In *Proceedings of the 5th International Conference on Learning Representations (ICLR 2017)*, 2017.

[3] 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. In *Proceedings of the 35th International Conference on Machine Learning (ICML 2018)*, 2018.

[4] Damian Mrowca, Chengxu Zhuang, Elias Wang, Nick Haber, Li Fei-Fei, Joshua B. Tenenbaum, and Daniel L. K. Yamins. Flexible neural representation for physics prediction. In *Proceedings of the 32nd Conference on Neural Information Processing Systems (NeurIPS 2018)*, 2018.

[5] 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.

[6] Sungyong Seo and Yan Liu. Differentiable physics-informed graph networks. *arXiv preprint arXiv:1902.02950*, 2019.

[7] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in neural information processing systems*, pages 6571–6583, 2018.

[8] Yulia Rubanova, Ricky T. Q. Chen, and David Duvenaud. Latent odes for irregularly-sampled time series. *CoRR*, abs/1907.03907, 2019. URL `http://arxiv.org/abs/1907.03907`.

[9] Sam Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian neural networks. *CoRR*, abs/1906.01563, 2019. URL `http://arxiv.org/abs/1906.01563`.

[10] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, pages 1–38, 2018.

[11] J Candy and W Rozmus. A symplectic integration algorithm for separable hamiltonian functions. *Journal of Computational Physics*, 92(1):230–256, 1991.

[12] Test of 3rd-order vs 4th-order symplectic integrator with strange result. Computational Science Stack Exchange, 2015. URL `https://scicomp.stackexchange.com/q/20535`.

A Data generation

A.1 Initial conditions

Each particle i had initial conditions (all in figures in international system of units) drawn from independent uniform distributions:

Mass m^i *2* $[0.1, 1]$

Spring constant k^i *2* $[0.5, 1]$

Initial position $\mathbf{q}_0^i$ *2* $[\ \ 1, 1]^2$

Initial velocity $\mathbf{v}_0^i$ *2* $[\ \ 3, 3]^2$, $\mathbf{p}_0^i = m^i \mathbf{v}_0^i$

A.2 Physics simulator

Our dataset consists of simulations for particle-spring systems where particle j exerts a force on particle i (Hooke's force) $\mathbf{F}^{ij} = \quad k^{ij} \quad (\mathbf{q}^i \quad \mathbf{q}^j)$ where k^{ij} is the spring constant, calculated as $k^{ij} = k^i \quad k^j$.

Trajectories were generated using RK4 with a generation time step of 0.005s, with trajectories of up to 4s in length.

A.3 Dataset

The datasets were obtained by subsampling the trajectories generated with the physics simulator at different time intervals.

We generated data for systems with between 2 and 15 particles, although we only used data with 4, 5, 6, 8 and 9 particles for training.

The training dataset consisted of 10000 one-step pairs of states (for each number of particles) separated by either a fixed time step of 0.1 s, or by a random time step drawn uniformly from the $[0.02, 0.2]$ s interval and rounded to the nearest exact multiple of the generation time-step. In the latter case, to avoid having discrete values of the time step, instead of using a fixed generation time-step of 0.005 s, we used random values form the interval 0.005 s $\quad$ 10%. Each pair was sampled from a different randomly generated trajectory. Additionally we generated 1000 validation and 1000 test pairs for each number of particles.

We also stored validation and test trajectories of length 20 sampled at 0.005, 0.01, 0.03, 0.1, 0.15, 0.2, 0.3, 0.4, 0.5 seconds (1000 trajectories for each number of particles and time step). All results presented in the paper correspond to the full test dataset of 20-step test trajectories for systems with 4, 5, 6, 8 and 9 particles.

B Graph Network

Each of the MLPs used on the edge model, node model and global model of the graph network, have output sizes of [64, 64], and *softplus* activations after each layer including the last one. The graph network always uses *sum* as the aggregation function for edges and nodes.

The output of the network is obtained by applying an additional non-activated linear layer with the desired output size to the output globals (HOGN, output size of 1 to represent a single scalar value per graph) or to the output nodes (OGN and DeltaGN, output size of 4 to represent values associated to 4 canonical coordinates for each node/particle).

We chose *softplus* because *relu* activation did not work well with the Hamiltonian model. We hypothesize this is a consequence of Eq. 7: a function approximator with *relu* activation is a piecewise linear function (linear almost everywhere), so after taking the derivatives of the output of the network with respect to its inputs, it becomes a function that is constant almost everywhere, and the gradient-based optimization process struggles to optimize it. This is consistent with [9] where they find better results for *tanh* activation than for *relu* activation. In our case *softplus* seemed to produce better results than *tanh*.

We always remove the mean position of the objects from the inputs to the GraphNetwork. This does not prevent the models from predicting correct absolute positions since all dynamics are translation invariant and models (Including RK integrators) only predict relative differences with respect to the previous state. This allows the model to produce correct predictions even when particles have travelled far away from their initial positions.

Also, because the ground truth Hamiltonian in spring systems is time-independent (it is a conservative system), we omit feeding absolute time to the GraphNetworks to avoid unnecessary dependencies.

C Training details

We used TensorFlow and the DeepMind Graph Nets library to build our neural networks, calculate the analytic gradients of the Hamiltonian, and train our models. We trained with a batch size of 100, for a million training steps. We used AdamOptimizer to minimize the loss.

Due to the different nature of the models (which may require different learning rates) we ran each of the models with 13 initial learning rates uniformly spanning an interval between 10^{-1} and 10^{-4} in log scale. Learning rate was decayed exponentially at a rate of 0.1 every $2 \quad 10^5$, with a lower limit of 10^{-7}.

For each model, we report median values and min-max range for the 4 learning rates with the smallest rollout error. In practice the variance across seeds given the same learning rate was negligible (except on generalization results).

Rollout error is defined as the RMS position error averaged across all examples, dimensions, particles, and sequence axis. Energy error is calculated as the RMS of the deviation between the mean energy of a trajectory and the initial energy (normalized by the initial energy, to yield relative errors), averaged across all examples.

D Additional results

D.1 Symplectic integrators

We attempted using symplectic integrators of first order (Symplectic Euler, S1), second order (Verlet Integrator, S2) and third order (S3) with coefficients as described in [11][3]. It is worth noting that these integrators are only symplectic for separable Hamiltonians $H(\mathbf{q}, \mathbf{p}) = T(\mathbf{p}) + V(\mathbf{q})$, or equivalently, for systems for which $\dot{\mathbf{p}} = f_{\mathbf{p}}(\mathbf{q})$ and $\dot{\mathbf{q}} = f_{\mathbf{q}}(\mathbf{p})$. However, we do not impose this constraint, neither to the HOGN (by adding the scalar output of two independent networks, with $\mathbf{p}$ and $\mathbf{q}$ inputs respectively) nor the OGN (by only feeding $\mathbf{q}$ to the network that outputs $\dot{\mathbf{p}}$ and vice versa), so there are not any guarantees that the learned models will correctly preserve energy for any Hamiltonian.

Fig. D.1a shows the predictive accuracy including results for models trained with symplectic integrators. Contrary to the RK integrators, in this case the HOGN seems to be able to produce lower errors than the true Hamiltonian even for low order integrators (S1 and S2), in a similar way to how the OGN was able to produce very low errors for low order RK integrators. While this seems like an advantage of using symplectic integrators it also indicates that the HOGN model is not really learning something close to the true Hamiltonian in these cases, and, as we will see in the next section, it will cause worse generalization to unseen test time-steps. Similarly, Fig. D.2d shows that the models trained with the S1, S2 integrators do not generalize at all to higher order symplectic integrators, and only the model trained with S3 shows good generalization across all integrators that is comparable to the generalization across integrators of the model trained with RK4 (Fig. D.2b) and the behavior of the true Hamiltonian (Fig. D.1a).

We speculate these differences in accuracy between training the HOGN with the RK integrators and the symplectic integrators are related to fundamental differences between the algorithms behind the two types of integrators. Firstly while the output of RK integrators is usually a weighted average of the results obtained across several internal iterations for intermediate time-steps, the symplectic integrators feed each internal iteration with just the output of the previous internal iteration, and

[3] We also tried a fourth order symplectic integrator, as described in [11], however it did not improve results, possibly due to the linear nature of the Hooke's force[12].

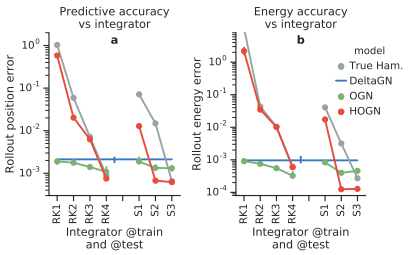

Figure D.1: **(a-b)** Predictive accuracy and energy conservation across models and integrators including symplectic integrators (analogous to Fig. 3c-d).

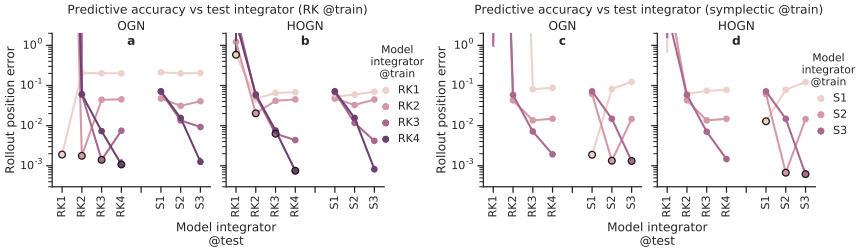

Figure D.2: **(a-d)** Predictive accuracy when varying the integrator used at test time, where points that share the same train and test integrator are highlighted with black circles (analogous to Fig. 3e-f).

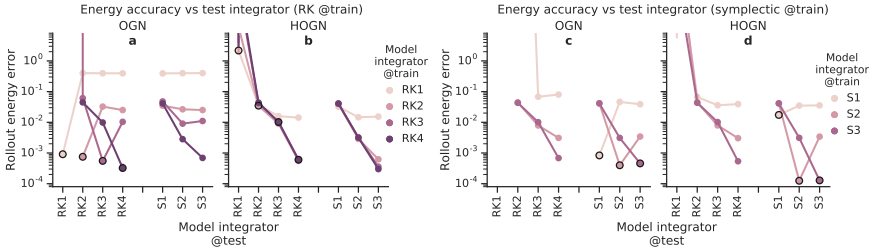

Figure D.3: **(a-d)** Energy accuracy when varying the integrator used at test time, where points that share the same train and test integrator are highlighted with black circles (analogous to Fig. D.2).

output just the result of the very last internal iteration. This means that the trained models can learn to encode additional information in the outputs of the intermediate internal iterations, which may allow the neural networks to distinguish whether the gradients are being evaluated for the first, the second, etc. or the final internal iteration. Secondly, in RK integrators, the derivatives of the position and momentum are always evaluated at the same point in each internal iteration, but in the symplectic integrators the derivatives of the position and the momentum are evaluated at different points, in alternating fashion. This has two consequences, the first consequence is that an n-th order symplectic integrator performs $2n$ network evaluations, while a n-th order RK integrator performs only n network evaluations, as it shares the evaluations of the position and momentum gradients at the same point. The additional network evaluations may explain why apparently lower order symplectic integrators, when used with learned models, can produce lower errors (e.g. S2 vs RK2). The second consequence is that, if as speculated before, the model can identify which calls corresponds to which internal iterations, it may also identify when it is being evaluated to produce gradients for the momentum or for the position. This would allow the model to get around the Hamiltonian constraint (that used to force it to produce a position and momentum vector field of gradients consistent with a common Hamiltonian), and allow it to start behaving more similarly to the OGN, where the derivatives of the position and momentum can be fully independent.

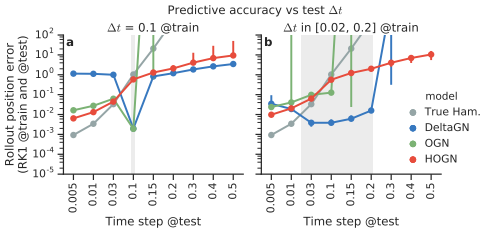

Figure D.4: **(a-b)** Time generalization for RK1 integrator (analogous to Fig. 3a-b).

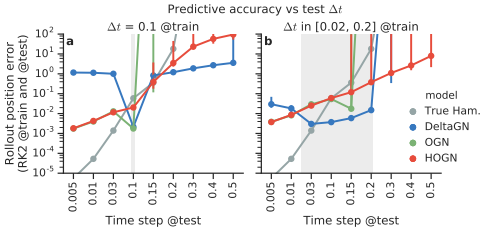

Figure D.5: **(a-b)** Time generalization for RK2 integrator (analogous to Fig. 3a-b).

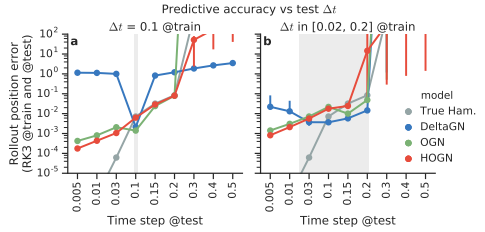

Figure D.6: **(a-b)** Time generalization for RK3 integrator (analogous to Fig. 3a-b).

With respect to energy conservation, Fig. D.1b, shows that models trained with higher order symplectic integrators (S1 and S2) preserve energy much better than the RK integrators, and, as mentioned in the main text, allow the HOGN to preserve energy much better than the OGN. It is also worth noting, that the HOGN trained with RK2, RK3 and RK4 integrators also improves in energy conservation when integrated with symplectic integrators (Fig. D.3b), confirming that the models trained with high order RK integrators learn something very similar to to the ground truth Hamiltonian that even generalizes to a different family of integrators.

D.2 Time generalization for other integrators

Figs. D.4, D.5, and D.6 show that for RK integrators of lower order (RK1-RK3) the OGN overfits heavily to the time-step used during training, however, the HOGN presents consistent generalization to time-steps not seen during training, yielding approximately similar errors. This again is evidence that the HOGN learns something more consistent with a Hamiltonian, and that in fact follows more closely the true Hamiltonian than the OGN.

In the case of symplectic integrators, we observe that even the HOGN overfits to the time-step used during training when using first order (S1) and second order (S2) integrators (Figs. D.7 and D.8), and starts generalizing well only with the third order (S3) integrator (Fig. D.9), although even for S3 the generalization to longer (> 0.2) time-steps is comparatively worse than with RK4 (Fig. 3a-b). This is consistent with the previous discussion (Section D.1) in that the HOGN is not really learning an accurate Hamiltonian when trained with a symplectic integrator.

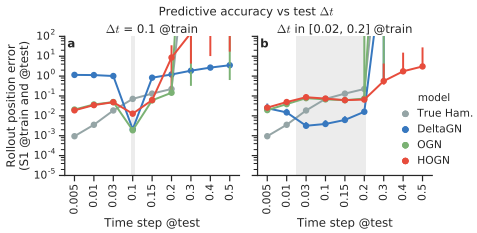

Figure D.7: **(a-b)** Time generalization for S1 integrator (analogous to Fig. 3a-b).

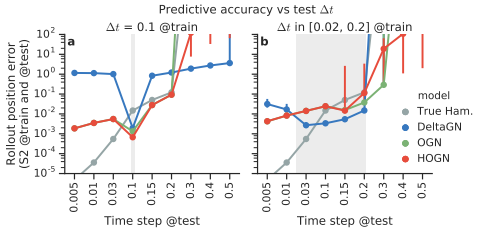

Figure D.8: **(a-b)** Time generalization for S2 integrator (analogous to Fig. 3a-b).

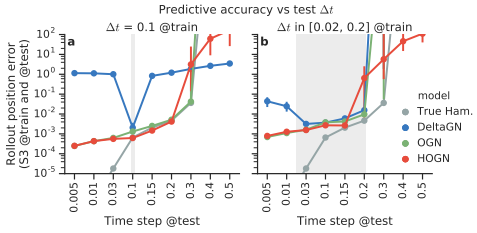

Figure D.9: **(a-b)** Time generalization for S3 integrator (analogous to Fig. 3a-b).