Learning Compositional Dynamics Models for Model-based Control

by

Yunzhu Li

B.S., Peking University (2017)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2020

Author .....
Department of Electrical Engineering and Computer Science
May 15, 2020

Certified by .....
Antonio Torralba
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Certified by .....
Russ Tedrake
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by .....
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Learning Compositional Dynamics Models for Model-based Control

by

Yunzhu Li

Submitted to the Department of Electrical Engineering and Computer Science
on May 15, 2020, in partial fulfillment of the
requirements for the degree of
Master of Science

Abstract

Compared with off-the-shelf physics engines, a learnable simulator has a stronger ability to adapt to unseen objects, scenes, and tasks. However, existing models like Interaction Networks only work for fully observable systems; they also only consider pairwise interactions within a single time step, both restricting their use in practical systems. We introduce Propagation Networks (PropNets), a differentiable, learnable dynamics model that handles partially observable scenarios and enables instantaneous propagation of signals beyond pairwise interactions. In the second half of the thesis, I will discuss our attempt to extend PropNets to learn a particle-based simulator for handling matters of various substances—rigid or soft bodies, liquid, gas—each with distinct physical behaviors. Combining learning with particle-based systems brings in two major benefits: first, the learned simulator, just like other particle-based systems, acts widely on objects of different materials; second, the particle-based representation poses strong inductive bias for learning: particles of the same type have the same dynamics within. We demonstrate that our models not only outperform current learnable physics engines in forward simulation, but also achieve superior performance on various control tasks, such as manipulating a pile of boxes, a cup of water, and a deformable foam, with experiments both in simulation and in the real world. Compared with existing model-free deep reinforcement learning algorithms, model-based control with our models is also more accurate, efficient, and generalizable to new, partially observable scenes and tasks.

Thesis Supervisor: Antonio Torralba
Title: Professor of Electrical Engineering and Computer Science

Thesis Supervisor: Russ Tedrake
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

There are many people I want to thank for helping me to complete this work at MIT over the past couple of years.

First and foremost, I want to express my sincere gratitude to my advisors, Professor Antonio Torralba and Professor Russ Tedrake. I thank them for their continued inspiring and encouraging advisory, which has led me through my research with deep and profound insights. Antonio and Russ are world-renowned professors in different fields–computer vision and robotics. I have been very fortunate to be advised by both of them, which gives me a unique edge of learning the tools and drawing inspirations from both communities, allowing me to conduct research projects in an interdisciplinary field.

I would also like to share my sincere thankfulness to Professor Joshua Tenenbaum for the insightful discussions and the connection of our works to human cognition. I deeply appreciate the inspiration and support from my colleagues at MIT. Jiajun Wu and Jun-Yan Zhu are great mentors for guiding me through all aspects of a research cycle. Wei Gao and Hao He have been friends for many long conversations about research and life. Thanks to Peter Florence, Lucas Manuelli, and Greg Izatt for setting up and teaching me about the robot hardware system, as well as the discussion for helping me shape my opinions about robotics. Everyone in the vision group and the Robot Locomotion Group is impressively inspiring and it has been great to have the research conversations with many of them, especially Bolei Zhou, Hang Zhao, Wei-Chiu Ma, Xavier Puig Fernandez, Adrià Recasens, David Bau, Tao Pang, Tobia Marcucci, Shen Shen, Weiqiao Han, Hongkai Dai, Twan Koolen, Robin Deits, and many others.

I also deeply appreciate the guidance and support from my collaborators during my undergraduate period, Professor Yizhou Wang and Professor Tianfu Wu, who introduced me into the world of computer vision. I also thank Professor Sebastian Thrun and Professor Stefano Ermon for advising me during my time at Stanford, and Jiaming Song, Andre Esteva and Benyuan Sun for the memorable collaborations.

Finally, I would like to thank my Mom and Dad for their lifelong love and encouragement. Thanks to my partner, Sijia, for her support whenever I encounter any setbacks or hardships.

Contents

Preface

This work is organized in four chapters. Chapter 1 introduces the work, discusses the motivation, reviews related work, and states the contributions. Chapter 2 introduces Propagation Networks (PropNets), a learning-based dynamics model that handles partially observable scenarios and enables instantaneous propagation of signals beyond pairwise interactions. Chapter 3 describes Dynamic Particle Interaction Networks (DPI-Nets), an extension of PropNets, for learning particle dynamics of rigid bodies, deformable objects, and fluids. Chapter 4 concludes.

Chapter 2 and 3 were originally written as independent papers [26, 27]. Chapter 2 was published as

> Yunzhu Li, Jiajun Wu, Jun-Yan Zhu, Joshua B. Tenenbaum, Antonio Torralba, and Russ Tedrake. Propagation Networks for Model-Based Control Under Partial Observation. In *International Conference on Robotics and Automation* (ICRA), May 2019.

Chapter 3 was published as

> Yunzhu Li, Jiajun Wu, Russ Tedrake, Joshua B. Tenenbaum, and Antonio Torralba. Learning Particle Dynamics for Manipulating Rigid Bodies, Deformable Objects, and Fluids. In *International Conference on Learning Representations* (ICLR), May 2019.

Chapter 1

Introduction

1.1 Motivation

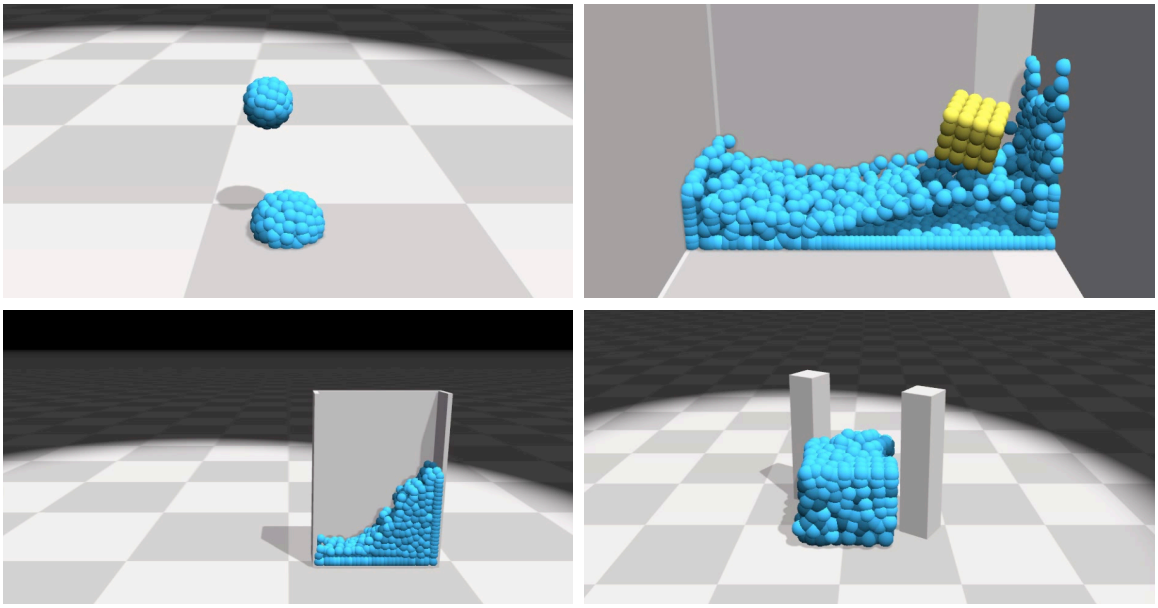
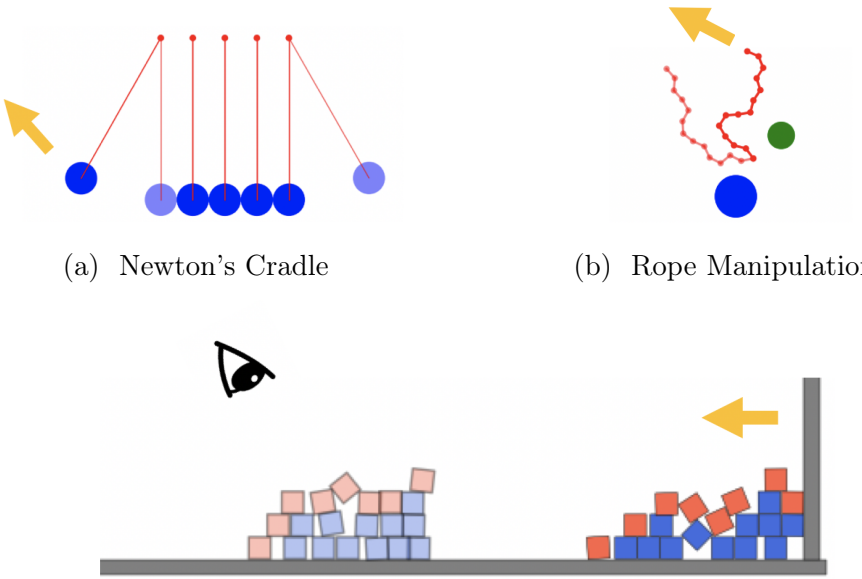


Figure 1-1: **Particle-based simulation.** The method represents the environment using particles and simulates the interactions between objects of different substances, including rigid bodies, deformable materials, and fluids.

Objects have distinct dynamics. Under the same push, a rigid box will slide, modeling clay will deform, and a cup full of water will fall with water spilling out. The diverse behavior of different objects poses challenges to traditional rigid-body simulators used in robotics [45, 44, 7]. Particle-based simulators aim to model the

dynamics of these complex scenes [29] (Figure 1-1); however, relying on approximation techniques for the sake of perceptual realism, their simulation often deviates from real-world physics, especially in the long term. Developing generalizable and accurate forward dynamics models is of critical importance for robot manipulation of distinct real-life objects.



(a) Newton's Cradle (b) Rope Manipulation

(c) Box Pushing

Figure 1-2: **Challenges for existing learning-based physics simulators:** Modeling the dynamics of (a) Newton's cradle or (b) a rope requires instantaneous propagation of multi-object interaction. (c) Pushing a group of boxes to a target configuration requires dynamics modeling under partial observations. Here, the camera is looking down and only red blocks are observable.

Recently, researchers have started building general-purpose neural physics simulators, aiming to approximate complex physical interactions with neural networks [3, 6]. They have succeeded in modeling the dynamics of both rigid bodies and deformable objects (e.g., ropes). More recent work has used Interaction Networks for discrete and continuous control [36, 16, 33, 37].

Interaction Networks, however, have two major limitations. First, Interaction Nets only consider pairwise interactions between objects, restricting its use in real-world scenarios, where simultaneous multi-body interactions often occur. Typical examples include Newton's cradle (Figure 1-2a) or rope manipulation (Figure 1-2b). Second,

they need to observe the full states of an environment; however, many real-world control tasks involve dealing with partial observable states. Figure 1-2c shows an example, where a robot wants to push a set of blocks into a target configuration; however, only the red blocks in the top layer are visible to the camera. It is, therefore, desired to develop methods that can address these two issues.

1.2 Contribution

1.2.1 Propagation Networks (PropNets)

We introduce Propagation Networks (PropNets), a learning-based physics simulator that simulates multi-body object interactions. PropNets handle partially observable situations by operating on a latent dynamics representation; it also enables instantaneous propagation of signals beyond pairwise interactions using multi-step effect propagation. Specifically, by representing a scene as a graph, where objects are the vertices and object interactions are the directed edges, we initialize and propagate the signals through the directed paths in the interaction graph at each time step.

Experiments demonstrate that PropNets consistently outperform Interaction Networks in forward simulation. PropNets' ability to handle partially observable environments brings significant benefits for control. Compared with Interaction Nets and state-of-the-art model-free deep reinforcement learning algorithms, model-based control using Propagation Networks is more sample-efficient, accurate, and generalizes better to new, partially observable scenarios.

1.2.2 Dynamic Particle Interaction Networks (DPI-Nets)

In the second half of this thesis, we propose to extend PropNets, learning a particle-based simulator for object of different materials using neural networks. We develop Dynamic Particle Interaction Networks (DPI-Nets) for learning particle dynamics, focusing on capturing the dynamic, hierarchical, and long-range interactions of particles. DPI-Nets can then be combined with classic perception and gradient-based control

algorithms for robot manipulation of deformable objects.

Learning a particle-based simulator brings in two major benefits. First, the learned simulator, just like other particle-based systems, acts widely on objects of different states. DPI-Nets have successfully captured the complex behaviors of deformable objects, fluids, and rigid-bodies. With learned DPI-Nets, our robots have achieved success in manipulation tasks that involve deformable objects of complex physical properties, such as molding plasticine to a target shape.

Second, the particle-based representation poses strong inductive bias for learning: particles of the same type have the same dynamics within. This enables the model to adapt to new environments of unknown dynamics. Experiments suggest that DPI-Nets quickly learn to adapt to characterize a novel object of unknown physical parameters by performing online system identification. The adapted model also helps the robot to successfully manipulate object in the real world.

DPI-Nets combine three key features for effective particle-based simulation and control: multi-step spatial propagation, hierarchical particle structure, and dynamic interaction graphs. In particular, it employs dynamic interaction graphs, built on the fly throughout manipulation, to capture the meaningful interactions among particles of deformable objects and fluids. The use of dynamic graphs allows neural models to focus on learning meaningful interactions among particles, which is crucial for obtaining good simulation accuracy and high success rates in manipulation. As objects deform when robots interact with them, a fixed interaction graph over particles is insufficient for robot manipulating non-rigid objects.

Experiments demonstrate that DPI-Nets significantly outperform Interaction Networks [3], HRN [30], and a few other baselines. More importantly, unlike previous papers that focused purely on forward simulation, we have applied our model to downstream control tasks. Our DPI-Nets enable complex manipulation tasks for deformable objects and fluids, and adapts to scenarios with unknown physical parameters that need to be identified online. We have also performed real-world experiments to demonstrate our model's generalization ability.

1.3 Related Work

1.3.1 Physics Simulators with Analytical Gradients

Researchers have developed many physics simulators, where they have access to the analytical gradients for end-to-end learning and control [11, 9]. In particular, [3] and [6] have explored learning a simulator from data by approximating object interactions with neural networks. These methods mostly focus on modeling rigid body dynamics.

Recently, [38] proposed SPNets for extracting gradients from simulators of position-based fluids [28]. An inspiring concurrent work from [30] explored learning to approximate particle dynamics of deformable shapes with the Hierarchical Relation Network (HRN). Compared with these papers, we introduce state-specific modeling and dynamic graphs for accurate forward prediction for different states of matter (rigid bodies, deformable shapes, fluids). We also demonstrate how the learned dynamics model can be used for control in both the simulation and the real world.

Our approach is also complementary to some recent work on learning to discover the interaction graphs [47, 20]. Our model can also be naturally augmented with a perception module to handle raw visual input, as suggested by [48, 49, 14, 50, 25], for various dynamics reasoning and prediction tasks.

1.3.2 Control Using the Gradients from Physics Simulators

In robotics, the use of differentiable simulators, together with continuous and symbolic optimization algorithms, has enabled planning for increasingly complex whole-body motions with multi-contact and multi-object interactions [12, 35, 21, 46]. A few recent papers have employed the analytical gradients extracted from the simulators [10, 38] for control problems, such as tool manipulation and tool-use planning [46]. Yet these approaches often assume they have access to the full state of the system, make local approximations of the dynamics, and focus mainly on rigid-body systems.

Many recent papers have studied model-predictive control with deep networks [23, 15, 31, 13, 41]. They often learn an abstract state transition function, instead of an

explicit account of the environment [40, 32, 24], and then use the learned function to facilitate the training of a policy network. Some others have explored using Interaction Networks for planning and control. They often learn a policy based on Interaction Networks' rollouts [36, 16, 33, 37].

Our PropNets build on and extend these approaches to simultaneous multi-body interactions and deal with partially observable scenarios. Our DPI-Nets have further extended by learning a general physics simulator that takes raw object observations (e.g., positions, velocities) of each particle as input. We then integrate it into classic trajectory optimization algorithms for control, which can simulate and control deformable, particle-based objects, using dynamic graphs to tackle scenes with complex object interactions. Compared with physics-based simulators, our learned simulator can better generalize to novel testing scenarios where object and environment parameters are unknown.

Chapter 2

Propagation Networks for Partially Observable Environments

2.1 Dynamics Learning

2.1.1 Preliminaries

We assume that the interactions within a physical system can be represented as a directed graph, $G = \langle O, R \rangle$, where vertices O represent the objects, and edges R correspond to their relations (Figure 2-2). Graph G can be represented as

$$O = \{o_i\}_{i=1...|O|} \qquad R = \{r_k\}_{k=1...|R|} \tag{2.1}$$

Specifically, $o_i = \langle x_i, a_i^o, p_i \rangle$, where $x_i = \langle q_i, \dot{q}_i \rangle$ is the state of object i, containing its position q_i and velocity $\dot{q}_i$. a_i^o denote its attributes (e.g., mass, radius), and p_i is the external force on object i. For the relations, we have

$$r_k = \langle u_k, v_k, a_k^r \rangle, \quad 1 \leq u_k, v_k \leq |O|, \tag{2.2}$$

where u_k is the receiver, v_k is the sender, and both are integers. a_k^r is the type and attributes of relation k (e.g., collision, spring connection).

Our goal is to build a learnable physics engine to approximate the underlying

physical interactions. We can then use it to infer the system dynamics and predict the future from the observed interaction graph G:

$$G_{t+1} = \phi(G_t), \tag{2.3}$$

where G_t denotes the scene states at time t. We aim to learn $\phi(\cdot)$, a learnable dynamics model, to minimize $\|G_{t+1} - \phi(G_t)\|_2$.

Below we review our baseline model Interaction Networks (IN) [3]. IN is a general-purpose, learnable physics engine, performing object- and relation-centric reasoning about physics. IN defines an object function f_O and a relation function f_R to model objects and their relations in a compositional way. The future state at time $t+1$ is predicted as

$$\begin{aligned} e_{k,t} &= f_R(o_{u_k,t}, o_{v_k,t}, a^r_k), \quad k = 1 \dots |R|, \\ \hat{o}_{i,t+1} &= f_O(o_{i,t}, \sum_{k \in \mathcal{N}_i} e_{k,t}), \quad i = 1 \dots |O|, \end{aligned} \tag{2.4}$$

where $o_{i,t} = \langle x_{i,t}, a^o_i, p_{i,t} \rangle$ denotes object i at time t, u_k and v_k are the receiver and sender of relation r_k, and $\mathcal{N}_i$ denotes the relations where object i is the receiver.

2.1.2 Propagation Networks (PropNets)

IN defines a flexible and efficient model for explicit reasoning of objects and their relations in a complex system. It can handle a variable number of objects and relations and has performed well in domains like n-body systems, bouncing balls, and falling strings. However, one fundamental limitation of IN is that at every time step t, it only considers local information in the graph G and cannot handle instantaneous propagation of forces, such as Newton's cradle shown in Figure 2-1, where ball A's impact produces a compression wave that propagates through the balls immediately [42]. As force propagation is a common phenomenon in rigid-body dynamics, this shortcoming has limited IN's practical applicability.

To address the above issues, we propose Propagation Networks (PropNets) to

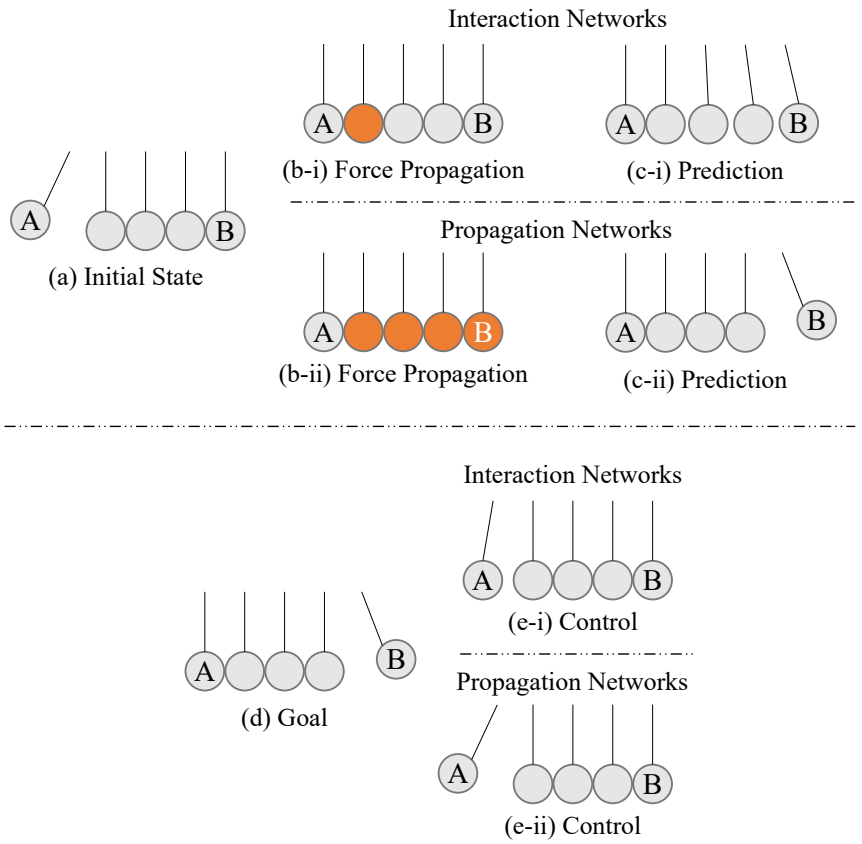


Figure 2-1: **Newton's Cradle.** (a) shows the initial states of Newton's cradle, based on which both the Interaction Networks and Propagation Networks try to predict future states; (b-i) The Interaction Networks can only propagate the force along with a single relation at a time step, thus results in a false prediction (c-i); (b-ii) Our proposed method can propagate the force correctly which leads to the correct prediction (c-ii); (d) A downstream task where we aim to achieve a specific goal using the learned model; (e-i) Model-based control methods fail to produce the correct control using Interaction Networks while (e-ii) our model can provide the desired control signal.

handle the instantaneous propagation of forces efficiently. Our method is inspired by message passing, a classic algorithm in graphical models [34].

Effect propagation

Effect propagation requires multi-step message passing along the directed edges in graph G. Forces ejected from ball A (Figure 2-1) should be propagated through the connected balls to ball B within a single time step. Force propagation is hard to analyze analytically for complex scenes. Therefore, we let PropNets learn to decide

whether an effect should be propagated further or withheld.

At time t, we denote the propagating effect from relation k at propagation step l as $e^l_{k,t}$, and the propagating effect from object i as $h^l_{i,t}$. Here, we have $1 \leq l \leq L$, where L is the maximum propagation steps within each step of the simulation. Propagation can be described as

$$
\begin{aligned}
&\text{Step 0:} && h^0_{i,t} = \mathbf{0}, \quad i = 1 \dots |O|, \\
&\text{Step } l = 1, \dots, L\text{:} && e^l_{k,t} = f^l_R(o_{u_k,t}, o_{v_k,t}, a^r_k, h^{l-1}_{u_k,t}, h^{l-1}_{v_k,t}), \quad k = 1 \dots |R|, \\
& && h^l_{i,t} = f^l_O(o_{i,t}, \sum_{k \in \mathcal{N}_i} e^l_{k,t}), \quad i = 1 \dots |O|, \\
&\text{Output:} && \hat{o}_{i,t+1} = h^L_{i,t}, \quad i = 1 \dots |O|,
\end{aligned} \tag{2.5}
$$

where $f^l_O(\cdot)$ denotes the object propagator at propagation step l, and $f^l_R(\cdot)$ denotes the relation propagator. Depending on the complexity of the task, the network weights can be shared among propagators at different propagation steps.

We name this model Vanilla PropNets. Experimental results show that the selection of L is task-specific, and usually a small L (e.g., $L = 3$) can achieve a good trade-off between the performance and efficiency.

Object- and relation-encoding with residual connections

We notice that Vanilla PropNets is not efficient for fast online control. As information such as states $o_{i,t}$ and attributes a^r_k are fixed at a specific time step, they can be shared without re-computation between each sequential propagation step. Hence, inspired by the ideas on fast RNNs training [22, 4], we propose to encode the shared information beforehand and reuse them along the propagation steps. We denote the encoder for objects as $f^{\text{enc}}_O(\cdot)$ and the encoder for relations as $f^{\text{enc}}_R(\cdot)$. Then,

$$
c^o_{i,t} = f^{\text{enc}}_O(o_{i,t}), \qquad c^r_{k,t} = f^{\text{enc}}_R(o_{u_k,t}, o_{v_k,t}, a^r_k). \tag{2.6}
$$

In practice, we add residual links [17] between adjacent propagation steps that connect $h_{i,t}^l$ and $h_{i,t}^{l-1}$. This helps address gradient vanishing and exploding problem, and provides access to historical effects. The update rules become

$$\begin{aligned} e_{k,t}^l &= f_R^l(c_{k,t}^r, h_{u_k,t}^{l-1}, h_{v_k,t}^{l-1}), \\ h_{i,t}^l &= f_O^l(c_{i,t}^o, \sum_{k \in \mathcal{N}_i} e_{k,t}^l, h_{i,t}^{l-1}), \end{aligned} \tag{2.7}$$

where propagators $f_O^l(\cdot)$ and $f_R^l(\cdot)$ now take a new sets of inputs, which is different from Vanilla PropNets.

Based on the assumption that the effects between propagation steps can be represented as simple transformations (e.g., identity-mapping in Newton's cradle), we can use small networks as function approximators for the propagators $f_O^l(\cdot)$ and $f_R^l(\cdot)$ for better efficiency. We name this updated model Propagation Networks (PropNets).

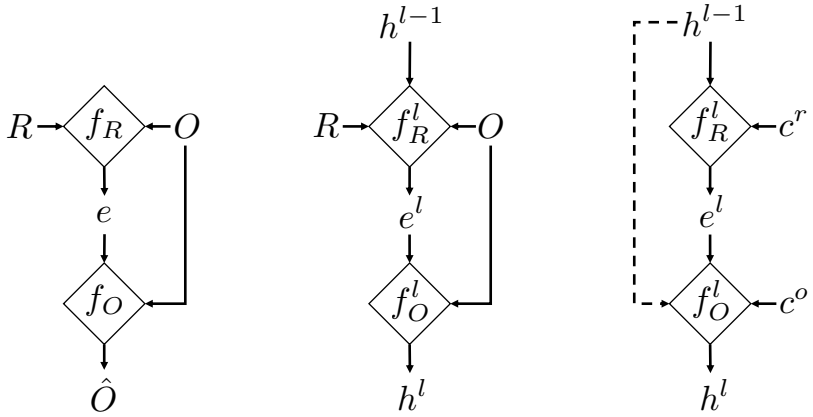


Figure 2-2: **Graphical illustration of Propagation Networks.** (a) The structure of Interaction Networks as detailed in Equation 2.4; (b) The internal structure of Vanilla PropNets is described in Equation 2.5, where the effects e^l and h^l are propagated through the propagators f_O^l and f_R^l along the directed relations in the graph G; (c) The shared object encoding c^o and relation encoding c^r are inputs to the internal modules, where there are also residual connections for better effect propagation as described in Equation 2.6 and 2.7.

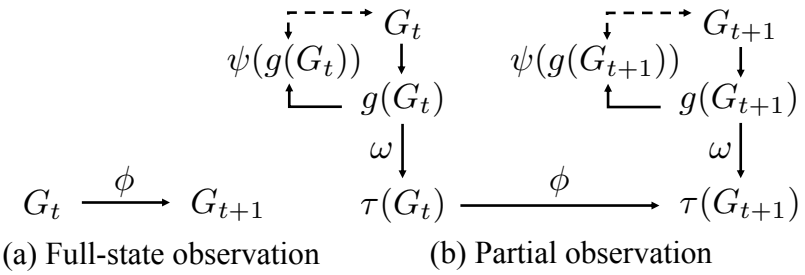


Figure 2-3: **Comparison between fully- and partially-observable scenarios.** (a) Forward model for fully observable environments (Equation 2.3). (b) For partially observable scenarios, we first map the observation to a latent space using function $\tau(\cdot)$, and then specify the forward dynamics over the latent space using $\phi(\cdot)$ as described in Equation 2.8. $\tau(\cdot)$ consists of $g(\cdot)$ and $\omega(\cdot)$, where $g(\cdot)$ maps the observation to object-based representations, which are then aggregated to a global representation using $\omega(\cdot)$. A decoding function $\psi(\cdot)$ maps the encoding back to the original observation space to ensure a nontrivial encoding.

2.1.3 Extending to Partially Observable Scenarios

For many real-world situations, however, it is often hard or impossible to estimate the full state of environments. We extend Equation 2.3 using PropNets to handle such partially observable cases by operating on a latent dynamics model:

$$\tau(G_{t+1}) = \phi(\tau(G_t)), \tag{2.8}$$

where $\tau(\cdot)$ is an encoding function that maps the current observation to a latent representation. As shown in Figure 2-3b, $\tau(\cdot)$ consists of two parts: first, PropNets $g(\cdot)$ that map the current observation to object-centric representations; second, $\omega(\cdot)$ that aggregates the object-centric representations into a fixed-dimensional global representation. We use a global representation for partially observable cases, because the number and set of observable objects vary over time, making it hard to define object-centric dynamics. In fully observable environments, $\tau(\cdot)$ reduces to an identity mapping and the dynamics is defined on the object level over the state space (Equation 2.3 and Figure 2-3a). To train such a latent dynamics model, we seek to minimize the loss function: $\mathcal{L}_{\text{forward}} = \|\tau(G_{t+1}) - \phi(\tau(G_t))\|_2$.

In practice, we use a small history window of length T_{history} for the state representation, i.e., the input to $\phi(\cdot)$ is the concatenation of $\tau(G_t), \tau(G_{t-1}), ..., \tau(G_{t-T_{\text{history}}+1})$.

Using the above loss alone leads to trivial solutions such as $\phi(x) = \tau(x) = 0$ for any valid x. We tackle this based on an intuitive idea: an ideal encoding function $\tau(\cdot)$ should be able to reserve information about the scene observation. Hence, we use an aggregation function $\omega(\cdot)$ that has no learnable parameters like summation or average and introduce a decoding function $\psi(\cdot)$ to ensure a nontrivial $\tau(\cdot)$ by minimizing an additional auto-encoder reconstruction loss [18]: $\mathcal{L}_{\text{encode}} = \|G - \psi(g(G))\|_2$, where $\psi(\cdot)$ is realized as PropNets. The full model is shown in Figure 2-3b.

2.2 Control Using the Learned Dynamics

Compared to model-free approaches, model-based methods offer many advantages, such as generalization and sample efficiency, as it can approximate the policy gradient or value estimation without exhausted trials and errors.

However, an accurate model of the environment is often hard to specify and brings significant computational costs for even a single-step forward simulation. It would be desirable to learn to approximate the underlying dynamics from data.

A learned dynamics model is naturally differentiable. Given the model and a desired goal, we can perform forward simulation, optimizing the control inputs by minimizing a loss between simulated results and a goal. The model can also estimate the uncertain attributes online by minimizing the difference between predicted future and actual outcome. Algorithm 1 outlines our control algorithm, which provides a natural testbed for evaluating the dynamics models.

2.2.1 Model-predictive Control Using Shooting Methods

Let $\mathcal{G}_g$ be our goal and $\hat{u}_{1:T}$ be the control inputs (decision variables), where T is the time horizon. These task-specific control inputs are part of the dynamics graph. Typical choices include observable objects' initial velocity/position and external forces/attributes on objects/relations. We denote the graph encoding as $G^\tau = \tau(G)$, and the resulting trajectory after applying the control inputs as $\mathcal{G} = \{G_i^\tau\}_{i=1:T}$. The

Algorithm 1 Control on Learned Dynamics at Time Step t

Input: Learned forward dynamics model $\phi(\cdot)$
Predicted dynamics graph encoding $\hat{G}_t^\tau$
Current dynamics graph encoding G_t^τ
Goal $\mathcal{G}_g$, current estimation of the attributes A
Current control inputs $\hat{u}_{t:T}$
States history $\bar{\mathcal{G}} = \{G_i^\tau\}_{i=1...t}$
Time horizon T
Output: Controls $\hat{u}_{t:T}$, predicted next time step $\hat{G}_{t+1}^\tau$

Update A by descending with the gradients
$\nabla_A \mathcal{L}_{\text{state}}(\hat{G}_t^\tau, G_t^\tau)$
Forward simulation using the current graph encoding
$\hat{G}_{t+1}^\tau \leftarrow \phi(G_t^\tau)$
Make a buffer for storing the simulation results
$\mathcal{G} \leftarrow \bar{\mathcal{G}} \cup \hat{G}_{t+1}^\tau$
for $i = t+1, ..., T-1$ **do**
Forward simulation
$\hat{G}_{i+1}^\tau \leftarrow \phi(\hat{G}_i^\tau)$; $\mathcal{G} \leftarrow \mathcal{G} \cup \hat{G}_{i+1}^\tau$
end for
Update $\hat{u}_{t:T}$ by descending with the gradients
$\nabla_{\hat{u}_{t:T}} \mathcal{L}_{\text{goal}}(\mathcal{G}, \mathcal{G}_g)$

Return $\hat{u}_{t:T}$ and $\hat{G}_{t+1}^\tau \leftarrow \phi(G_t^\tau)$

task here is to determine the control inputs by minimizing the gap between the actual outcome and the specified goal $\mathcal{L}_{\text{goal}}(\mathcal{G}, \mathcal{G}_g)$.

Our Propagation Networks can do forward simulation by taking the dynamics graph at time t as input, and produce the graph at next time step, $\hat{G}_{t+1}^\tau = \phi(G_t^\tau)$. Let's denote the forward simulation from time step t as $\hat{\mathcal{G}} = \{\hat{G}_i^\tau\}_{i=t+1...T}$ and the history until time t as $\bar{\mathcal{G}} = \{G_i^\tau\}_{i=1...t}$. We can back-propagate from the loss $\mathcal{L}_g(\bar{\mathcal{G}} \cup \hat{\mathcal{G}}, \mathcal{G}_g)$ and use stochastic gradient descent (SGD) to update the control inputs. This is known as the shooting method in trajectory optimization [43].

If the time horizon T is too long, the learned model might deviate from the ground truth due to accumulated prediction errors. Hence, we use Model-Predictive Control (MPC) [5] to stabilize the trajectory by doing forward simulation at every time step as a way to compensate the simulation error.

2.2.2 Online Adaptation

In many situations, inherent attributes such as masses, friction, and damping are not directly observable. Instead, we can interact with the objects and use PropNets to estimate these attributes online (denoted as A) with SGD updates by minimizing the difference between the predicted future states and the actual future states $\mathcal{L}_{\text{state}}(\hat{G}_t^\tau, G_t^\tau)$.

2.3 Experiments

In this section, we evaluate the performance of our model on both simulation and control in three scenarios: Newton's Cradle, Rope Manipulation, and Box Pushing. We also test how the model generalizes to new scenarios and how it learns to adapt online.

2.3.1 Physics Simulation

We aim to predict the future states of physical systems. We first describe the network used across tasks and then present the setup of each task as well as the experimental results.

Model architecture

For the IN baseline, we use the same network as described in the original work [3]. For Vanilla PropNets, we adopt similar network structure where the relation propagator $f_R^l(\cdot)(1 \leq l \leq L)$ is an MLP with four 150-dim hidden layers and the object propagator $f_O^l(\cdot)(1 \leq l \leq L-1)$ has one 100-dim hidden layer. Both output a 100-dim propagation vector. For fully observable scenarios, $f_O^L(\cdot)$ has one 100-dim hidden layer and outputs a 2-dim vector representing the velocity at the next time step. For partially observable cases, $f_O^L(\cdot)$ outputs one 100-dim vector as the latent representation.

For PropNets, we use an MLP with three 150-dim hidden layers as the relation encoder $f_R^{\text{enc}}(\cdot)$ and one 100-dim hidden layer MLP as the object encoder $f_O^{\text{enc}}(\cdot)$.

Light-weight neural networks are used for the propagators $f_O^l(\cdot)$ and $f_R^l(\cdot)$, both of which only contain one 100-dim hidden layer.

Newton's cradle

A typical Newton's cradle consists of a series of identically sized rigid balls suspended from a frame. When one ball at the end is lifted and released, it strikes the stationary balls. Forces will transmit through the stationary balls and push the last ball upward immediately. In our fully observable setup, the graph G of n balls has $2n$ objects representing the balls and the corresponding fixed pinpoints above the balls, as shown in Figure 2-1a, where $n = 5$. There will be $2n$ directed relations describing the rigid connections between the fixed points and the balls. Collisions between adjacent balls introduce another $2(n-1)$ relations.

We generated 2,000 rollouts over 1,000 time steps, of which 85% of the rollouts are randomly chosen as the training set, while the rest are held as the validation set. The model was trained with a mini-batch of 32 using Adam optimizer [19] with an initial learning rate of 1e-3. We reduce the learning rate by 0.8 each time the validation error stops decreasing for over 20 epochs.

Figure 2-1a-c show some qualitative results, where we compare IN and PropNets. IN cannot propagate the forces properly: the rightmost ball starts to swing up before the first collision happens. Quantitative results also show that our method significantly outperforms IN in tracking object positions. For 1,000 forward steps, IN results in an MSE of 336.46, whereas PropNets achieves an MSE of 7.85.

Rope manipulation

We then manipulate a particle-based rope in a 2D plane using a spring-mass model, where one end of the rope is fixed to a random point near the center and the rest of the rope is free to move. Two circular obstacles are placed at random positions near the rope and are fixed to the ground. Random forces are applied to the masses on the rope and the rope is moving in compliant with the forces. More specifically, for a rope containing n particles, there will be a total of $n + 2$ objects. Each pair of adjacent

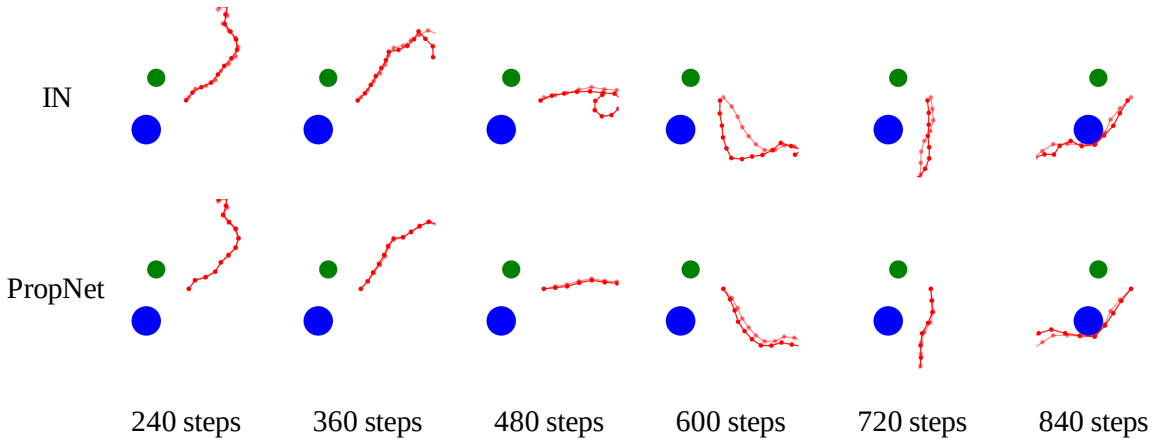


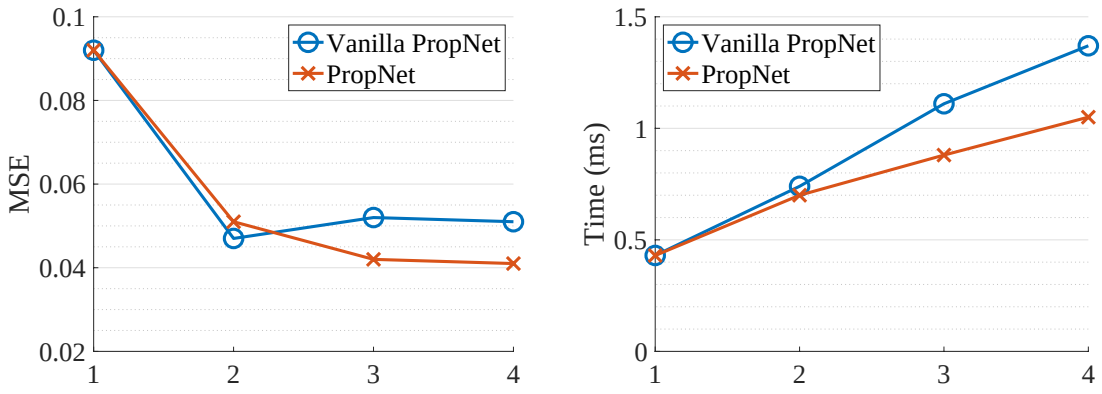
Figure 2-4: **Qualitative results on rope simulation.** Results on the planar rope simulation, where every mass on the rope has been applied a random force and the rope is moving in the planar in compliant with the forces. Our model better matches the ground truth and suffers less from the drifting problem as time horizon becomes longer. Here the transparent trajectories indicate the ground truth.

masses will have spring relations connecting each other, resulting in $2(n-1)$ directed edges in the dynamics graph G. Each mass will have a collision relation with each fixed obstacle, which adds to the graph another $4n$ edges. Frictional force applied to each mass is modeled as a directed edge connecting the mass itself.

We use the same network as described above and generate 5,000 rollouts over 100 time steps. Figure 2-4 and Figure 2-5a show qualitative and quantitative results, respectively. We train the models with a 15-dim rope and evaluated in situations where the rope length can vary between 10 and 20. As can be seen from the figures, although the length of the underlying force propagation is fewer than Newton's Cradle's, our proposed method can still track the ground truth much more accurately and outperform IN by a large margin.

Box pushing

In this case, we are pushing a pile of boxes forward (Figure 2-6b). We place a camera at the top of the scene, and only red boxes are observable. More challengingly, the observable boxes are not tracked. Therefore, the visibility of a specific box might change over time. The vertices in the graph are then defined as the state of the



(a) Rope: Velocity

(b) Rope: Forward Time

Figure 2-5: **Quantitative results on rope simulation.** We vary the propagation steps L between 2 to 4 for Vanilla PropNets and PropNets, which shows a trade-off between accuracy and efficiency. When $L = 1$, both models reduce to Interaction Networks (IN).

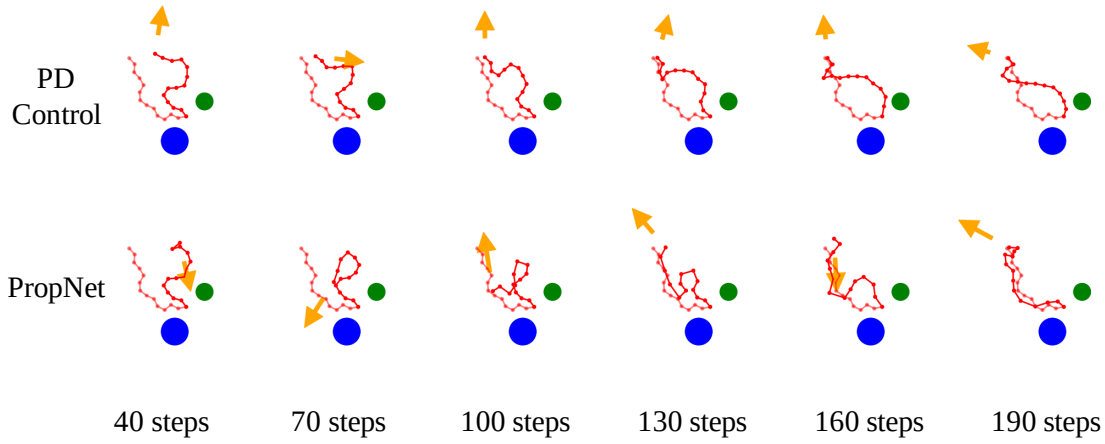
observable boxes and edges are defined as directional relations connecting every pair of observable boxes. Specifically, if there are n observable boxes, $n(n-1)$ edges are automatically generated. The dynamics function $\phi(\cdot)$ then takes both the scene representation and the action (i.e., position and velocity of the pusher) as input to perform an implicit forward simulation. As it is hard to explicitly evaluate a latent dynamics model, we evaluate the downstream control tasks instead.

Ablation studies

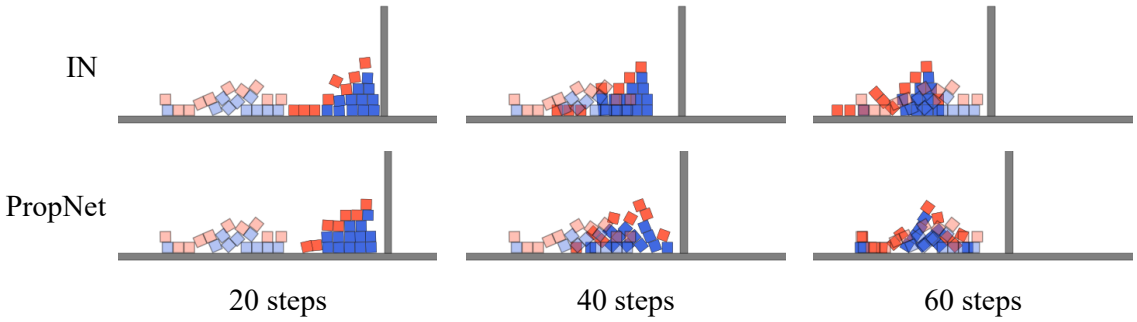
We also provide ablation studies on how the number of propagation steps L influences the final performance. Empirically, a larger L can model a longer propagation path. They are however harder to train and more likely to overfit the training set, often leading to poor generalization. Figure 2-5a and 2-5b show the ablation studies regarding the choice of L. PropNets achieves a high accuracy at $L = 3$, with a good trade-off between speed and accuracy. Vanilla PropNets achieves its best accuracy at $L = 2$ but generalizes less well as L increases further. This shows the benefits of using the shared encoding and residual connections used in PropNet, as described in Section 2.1.2.

2.3.2 Control

We now evaluate the applicability of the learned model on control tasks. We first describe the three tasks: Newton's Cradle, Rope Manipulation, and Box Pushing, which include both open-loop and feedback continuous control tasks, as well as fully and partially observable environments. We evaluate the performance against various baselines and test its ability on generalization and online adaptation.

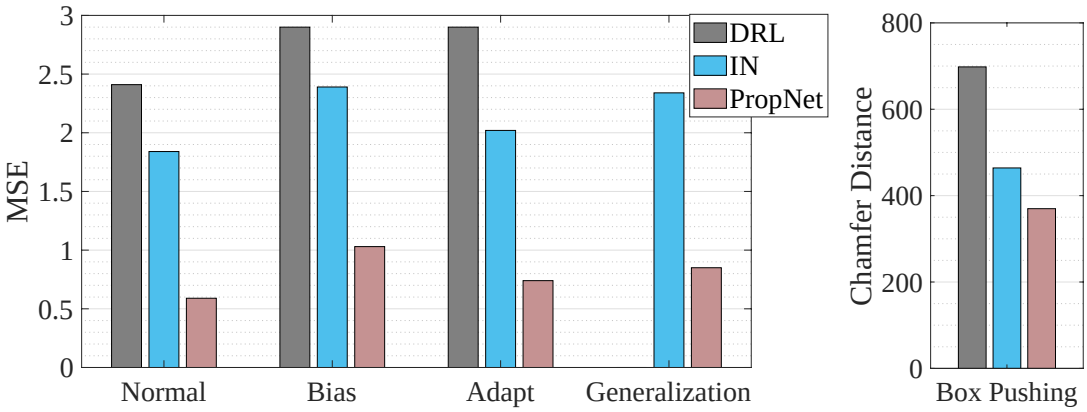


(a) Rope Manipulation



(b) Box Pushing

Figure 2-6: **Qualitative results on control.** (a) The rope manipulation task defines a continuous control problem which is to achieve a specified goal configuration by applying forces to the top two masses on the free end of the rope. The applied forces are visualized as yellow arrows and the goal configuration is shown as transparent. Note that instead of naively trying to match the top two masses (PD control), our control method based on PropNets can achieve the goal configuration by exploring the rich dynamics of the rope. (b) The box pushing task requires solving a control problem under partial observation (only red blocks are observable). The goal configuration is shown as transparent. Doing control with our Propagation Networks achieves more accurate outcome than with an IN.



(a) Rope Manipulation (b) Box Pushing

Figure 2-7: **Quantitative results on control tasks.** (a) For rope manipulation, the algorithms attempt to match a specific configuration under situations where the ground-truth attributes are known ("Normal"), where the value of the attributes are unknown ("Bias"), where algorithms actively estimate these attributes online ("Adapt"), and where ropes are of varied length between 10 to 20 when the model is only trained on ropes of length 15 ("Genearlize"). DRL has the same performance for "Bias" and "Adapt" as it is model-free; it requires a fixed length input, and thus cannot generalize to ropes of a different length. (b) For box pushing, Propagation Networks again outperforms the other methods.

Newton's Cradle

In this scenario, we assume full-state observation and a control task would be to determine the initial angle of the left-most ball, so as to let the right-most ball achieve a specific height, which can be solved with an accurate forward simulation model.

This is an open-loop control task where we only have control over the initial condition. We thus use a simplified version of Algorithm 1. Given the initial physics graph and a learned dynamics model, we iteratively do forward simulation and update the control inputs by minimizing the loss function $\mathcal{L}_{\text{goal}}(\mathcal{G}, \mathcal{G}_g)$. In this specific task, the loss $\mathcal{L}_{\text{goal}}$ is the $\mathcal{L}_2$ distance between the target height of the right-most ball and the highest height that has been achieved in $\mathcal{G}$.

We initialize the swing up angle as 45° and then optimize the angle with a learning rate of 0.1 for 50 iterations using Adam optimizer. We compare our model with IN. Qualitative results are shown in Figure 2-1e. Quantitatively, PropNets's output angle has an MSE of 3.08 from the ground truth initial angle, while the MSE for interaction

nets is 296.66.

Rope Manipulation

Here we define the task as to move the rope to a target configuration, where the only controls are the top two masses at the moving end of the rope (Figure 2-6a). The controller tries to match the target configuration by "swinging" the rope, which requires to leverage the dynamics of the rope. The loss $\mathcal{L}_{\text{goal}}$ here is the $\mathcal{L}_2$ distance between the resulting configuration and the goal configuration.

We first assume the attributes of the physics graph is known (e.g., mass, friction, damping) and compare the performance between Proportional-Derivative controller (PD) [1], Model-free Deep Reinforcement Learning (Actor-Critic method optimized with PPO [39] - DRL), as well as Interaction Networks (IN) and Propagation Networks (PropNets) with Algorithm 1. Figure 2-7 shows quantitative results, where bars marked as "Normal" are the results in this task (a hand-tuned PD controller has an MSE of 2.50). PropNets outperforms the competing baselines. Figure 2-6a shows a qualitative sample. Compared with the PD controller, our method leverages the dynamics and manages to match the target, instead of naively matching the free end of the rope.

We then consider situations where some of the attributes are unknown and can only be guessed before actually interacting with the objects. We randomly add noise of 15% of the original scale to the attributes as the initial guesses. The "Bias" bars in Figure 2-7 show that models trained with ground-truth attributes will encounter performance drop when the supplied attributes are not accurate. However, model-based methods can do online adaptation using the actual output from the environment as feedback to correct the attribute estimation. By updating the estimated attributes over the first 20 steps of the time horizon with standard SGD, we can improve the manipulation performance so as to catch up with the situations where attributes are accurate (bars marked as "Adapt" in Figure 2-7).

We further test whether our model generalizes to new scenarios, where the length of the rope is varied between 10 to 20. As can be seen in Figure 2-7, our proposed

method can still achieve a good performance, even though the original PropNets is only trained in situations with a fixed length 15 (PD has an MSE of 2.72 for generalization).

Box Pushing

In this case, we aim to push a pile of boxes to a target configuration within a predefined time horizon (Figure 2-6b). We assume partial observation where a camera is placed at the top of the scene, and we can only observe the states of the boxes marked in red. The model trained with partial observation is compared with two baselines: DRL and IN. The loss function $\mathcal{L}_{\text{goal}}$ used for MPC is the $\mathcal{L}_2$ distance between the resulting scene encoding and the target scene encoding.

We evaluate the performance by the Chamfer Distance (CD) [2] between the observable boxes at the end of the episode and the target configurations, where for each box in each set, CD finds the nearest box in the other set, and sums the distance up. The negative of the distance is used as the reward for DRL. Figure 2-6b and Figure 2-7b show qualitative and quantitative results, respectively. Our method outperforms the baselines due to its explicit modeling of the dynamics and its ability to handle multi-object interactions.

Chapter 3

Learning Particle Dynamics for Manipulating Rigid Bodies, Deformable Objects, and Fluids

3.1 Approach

3.1.1 Dynamic Particle Interaction Networks (DPI-Nets)

Particle-based system is widely used in physical simulation due to its flexibility in modeling various types of objects [29]. We extend existing systems that model object-level interactions to allow particle-level deformation. Consider object set $\{\mathbf{o}_i\}$, where each object $\mathbf{o}_i = \{o_i^k\}_{k=1...|\mathbf{o}_i|}$ is represented as a set of particles. We now define the graph on the particles and the rules for influence propagation.

Dynamic graph building

The vertices of the graph are the union of particles for all objects $O = \{o_i^k\}_{i=1...|O|,k=1...|\mathbf{o}_i|}$. The edges R between these vertices are dynamically generated over time to ensure efficiency and effectiveness. The construction of the relations is specific to environment and task, which we'll elaborate in Section 3.2. A common choice is to consider the neighbors within a predefined distance.

An alternative is to build a static, complete interaction graph, but it has two major drawbacks. First, it is not efficient. In many common physical systems, each particle is only interacting with a limited set of other particles (e.g., those within its neighborhood). Second, a static interaction graph implies a universal, continuous neural function approximator; however, many physical interactions involve discontinuous functions (e.g. contact). In contrast, using dynamic graphs empowers the model to tackle such discontinuity.

Hierarchical modeling for long-range dependence

Propagation Networks [27] require a large L to handle long-range dependence, which is both inefficient and hard to train. Hence, we add one level of hierarchy to efficiently propagate the long-range influence among particles [30]. For each object that requires modeling of the long-range dependence (e.g. rigid-body), we cluster the particles into several non-overlapping clusters. For each cluster, we add a new particle as the cluster's root. Specifically, for each object $\mathbf{o}_i$ that requires hierarchical modeling, the corresponding roots are denoted as $\tilde{\mathbf{o}}_i = \{\tilde{o}_i^k\}_{k=1...|\tilde{\mathbf{o}}_i|}$, and the particle set containing all the roots is denoted as $\tilde{O} = \{\tilde{o}_i^k\}_{i=1...|O|,k=1...|\tilde{\mathbf{o}}_i|}$. We then construct an edge set $R_{\text{LeafToRoot}}$ that contains directed edges from each particle to its root, and an edge set $R_{\text{RootToLeaf}}$ containing directed edges from each root to its leaf particles. For each object that need hierarchical modeling, we add pairwise directed edges between all its roots, and denote this edge set as $R_{\text{RootToRoot}}$.

We employ a multi-stage propagation paradigm: (1) propagation among leaf nodes, $\phi_{\text{LeafToLeaf}}(\langle O, R\rangle)$; (2) propagation from leaf nodes to root nodes, $\phi_{\text{LeafToRoot}}(\langle O \cup \tilde{O}, R_{\text{LeafToRoot}}\rangle)$; (3) propagation between roots, $\phi_{\text{RootToRoot}}(\langle \tilde{O}, R_{\text{RootToRoot}}\rangle)$; (4) propagation from root to leaf, $\phi_{\text{RootToLeaf}}(\langle O \cup \tilde{O}, R_{\text{RootToLeaf}}\rangle)$. The signals on the leaves are used to do the final prediction.

Applying to objects of various materials

We define the interaction graph and the propagation rules on particles for different types of objects as follows:

- *Rigid bodies.* All the particles in a rigid body are globally coupled; hence for each rigid object, we define a hierarchical model to propagate the effects. After the multi-stage propagation, we average the signals on the particles to predict a rigid transformation (rotation and translation) for the object. The motion of each particle is calculated accordingly. For each particle, we also include its offset to the center-of-mass to help determine the torque.

- *Elastic/Plastic objects.* For elastically deforming particles, only using the current position and velocity as the state is not sufficient, as it is not clear where the particle will be restored after the deformation. Hence, we include the particle state with the resting position to indicate the place where the particle should be restored. When coupled with plastic deformation, the resting position might change during an interaction. Thus, we also infer the motion of the resting position as a part of the state prediction. We use hierarchical modeling for this category but predict the next state for each particle individually.

- *Fluids.* For fluid simulation, one has to enforce density and incompressibility, which can be effectively achieved by only considering a small neighborhood for each particle [28]. Therefore, we do not need hierarchical modeling for fluids. We build edges dynamically, connecting a fluid particle to its neighboring particles. The strong inductive bias leveraged in the fluid particles allows good performance even when tested on data outside training distributions.

For the interaction between different materials, two directed edges are generated for any pairs of particles that are closer than a certain distance.

3.1.2 Control on the Learned Dynamics

Model-based methods offer many advantages when compared with their model-free counterparts, such as generalization and sample efficiency. However, for cases where an accurate model is hard to specify or computationally prohibitive, a data-driven approach that learns to approximate the underlying dynamics becomes useful.

Function approximators such as neural networks are naturally differentiable. We can rollout using the learned dynamics and optimize the control inputs by minimizing a loss between the simulated results and a target configuration. In cases where certain physical parameters are unknown, we can perform online system identification by minimizing the difference between the model's prediction and the reality. The control algorithm is very similar to Algorithm 1 and a detailed version can be found in Appendix A.1.

Model predictive control using shooting methods

Let's denote $\mathcal{G}_g$ as the goal and $\hat{u}_{1:T}$ be the control inputs, where T is the time horizon. The control inputs are part of the interaction graph, such as the velocities or the initial positions of a particular set of particles. We denote the resulting trajectory after applying $\hat{u}$ as $\mathcal{G} = \{G_i\}_{i=1:T}$. The task here is to determine the control inputs as to minimize the distance between the actual outcome and the specified goal $\mathcal{L}_{\text{goal}}(\mathcal{G}, \mathcal{G}_g)$.

Our Dynamic Particle Interaction Networks does forward simulation by taking the dynamics graph at time t as input, and produces the graph at next time step, $\hat{G}_{t+1} = \Phi(G_t)$, where Φ is implemented as DPI-Nets as described in the previous section. Let's denote the the history until time t as $\bar{\mathcal{G}} = \{G_i\}_{i=1\ldots t}$, and the forward simulation from time step t as $\hat{\mathcal{G}} = \{\hat{G}_i\}_{i=t+1\ldots T}$. The loss $\mathcal{L}_{\text{goal}}(\bar{\mathcal{G}} \cup \hat{\mathcal{G}}, \mathcal{G}_g)$ can be used to update the control inputs by doing stochastic gradient descent (SGD). This is known as the shooting method in trajectory optimization [43].

The learned model might deviate from the reality due to accumulated prediction errors. We use Model-Predictive Control (MPC) [5] to stabilize the trajectory by doing forward simulation and updating the control inputs at every time step to compensate the simulation error.

Online adaptation

In many real-world cases, without actually interacting with the environment, inherent attributes such as mass, stiffness, and viscosity are not directly observable. DPI-Nets can estimate these attributes on the fly with SGD updates by minimizing the distance

between the predicted future states and the actual future states $\mathcal{L}_{\text{state}}(\hat{G}_t, G_t)$.

3.2 Experiments

We evaluate our method on four different environments containing different types of objects and interactions. We will first describe the environments and show simulation results. We then present how the learned dynamics help to complete control tasks in both simulation and the real world.

3.2.1 Environments

- **FluidFall** (Figure 3-1a). Two drops of fluids are falling down, colliding, and merging. We vary the initial position and viscosity for training and evaluation.
- **BoxBath** (Figure 3-1b). A block of fluids is flushing a rigid cube. In this environment, we have to model two different materials and the interactions between them. We randomize the initial position of the fluids and the cube to test the model's generalization ability.
- **FluidShake** (Figure 3-1c). We have a box of fluids and the box is moving horizontally, The speed of the box is randomly selected at each time step. We vary the size of the box and volume of the fluids to test generalization.
- **RiceGrip** (Figure 3-1d). We manipulate an object with both elastic and plastic deformation (e.g., sticky rice). We use two cuboids to mimic the fingers of a parallel gripper, where the gripper is initialized at a random position and orientation. During the simulation of one grip, the fingers will move closer to each other and then restore to its original positions. The model has to learn the interactions between the gripper and the "sticky rice", as well as the interactions within the "rice" itself.

We use all four environments in evaluating our model's performance in simulation. We use the rollout MSE as our metric. We further use the latter two for control, because

they involve fully actuated external shapes that can be used for object manipulation. In FluidShake, the control task requires determining the speed of the box at each time step, in order to make the fluid match a target configuration within a time window; in RiceGrip, the control task corresponds to select a sequence of grip configurations (position, orientation, closing distance) to manipulate the deformable object as to match a target shape. The metric for performance in control is the Chamfer distance between the manipulation results and the target configuration.

3.2.2 Physics Simulation

We present implementation details for dynamics learning in the four environment and perform ablation studies to evaluate the effectiveness of the introduced techniques.

Implementation details

For FluidFall, we dynamically build the interaction graph by connecting each particle to its neighbors within a certain distance d. No hierarchical modeling is used.

For BoxBath, we model the rigid cube as in Section 3.1.1, using multi-stage hierarchical propagation. Two directed edges will be constructed between two fluid particles if the distance between them is smaller than d. Similarly, we also add two directed edge between rigid particles and fluid particles when their distance is smaller than d.

For FluidShake, we model fluid as in Section 3.1.1. We also add five external particles to represent the walls of the box. We add a directed edge from the wall particle to the fluid particle when they are closer than d. The model is a single propagation network, where the edges are dynamically constructed over time.

For RiceGrip, we build a hierarchical model for rice and use four propagation networks for multi-stage effect propagation (Section 3.1.1). The edges between the "rice" particles are dynamically generated if two particles are closer than d. Similar to FluidShake, we add two external particles to represent the two "fingers" and add an edge from the "finger" to the "rice" particle if they are closer than the distance d. As

Methods	FuildFall	BoxBath	FluidShake	RiceGrip
IN [3]	2.74 ± 0.56	N/A	N/A	N/A
HRN [30]	0.21 ± 0.04	3.62 ± 0.40	3.58 ± 0.77	0.17 ± 0.11
DPI-Nets w/o hierarchy	**0.15 ± 0.03**	2.64 ± 0.69	**1.89 ± 0.36**	0.29 ± 0.13
DPI-Nets	**0.15 ± 0.03**	**2.03 ± 0.41**	**1.89 ± 0.36**	**0.13 ± 0.07**

Table 3.1: **Quantitative results on forward simulation.** MSE ($\times 10^{-2}$) between the ground truth and model rollouts. The hyperparameters used in our model are fixed for all four environments. FluidFall and FluidShake involve no hierarchy, so DPI-Nets performs the same as the variant without hierarchy. DPI-Nets significantly outperforms HRN [30] in modeling fluids (BoxBath and FluidShake) due to the use of dynamic graphs.

"rice" can deform both elastically and plastically, we maintain a resting position that helps the model restore a deformed particle. The output for each particle is a 6-dim vector for the velocity of the current observed position and the resting position. More training details for each environment can be found in Appendix A.4. Details for data generation are in Appendix A.3.

Results

Qualitative and quantitative results are in Figure 3-1 and Table 3.1. We compare our method (DPI-Nets) with three baselines, Interaction Networks [3], HRN [30], and DPI-Nets without hierarchy. Note that we use the same set of hyperparameters in our model for all four testing environments.

Specifically, Interaction Networks (IN) consider a complete graph of the particle system. Thus, it can only operate on small environments such as FluidFall; it runs out of memory (12GB) for the other three environments. IN does not perform well, because its use of a complete graph makes training difficult and inefficient, and because it ignores influence propagation and long-range dependence.

Without a dynamic graph, modeling fluids becomes hard, because the neighbors of a fluid particle changes constantly. Table 3.1 shows that for environments that involve fluids (BoxBath and FluidShake), DPI-Nets performs better than those with a static interaction graph. Our model also performs better in scenarios that involve objects of multiple states (BoxBath, Figure 3-1b), because it uses state-specific modeling.

Models such as HRN [30] aim to learn a universal dynamics model for all states of matter. It is therefore solving a harder problem and, for this particular scenario, expected to perform not as well. When augmented with state-specific modeling, HRN's performance is likely to improve, too. Without hierarchy, it is hard to capture long-range dependence, leading to performance drop in environments that involve hierarchical object modeling (BoxBath and RiceGrip).

Appendix A.2 includes results on scenarios outside the training distribution (e.g., more particles). DPI-Nets performs well on these out-of-sample cases, successfully leveraging the inductive bias.

Ablation studies

We also test our model's sensitivity to hyperparameters. We consider three of them: the number of roots for building hierarchy, the number of propagation steps L, and the size of the neighborhood d. We test them in RiceGrip. As can be seen from the results shown in Figure 3-2a, DPI-Nets can better capture the motion of the "rice" by using fewer roots, on which the information might be easier to propagate. Longer propagation steps do not necessarily lead to better performance, as they increases training difficulty. Using larger neighborhood achieves better results, but makes computation slower. Using one TITAN Xp, each forward step in RiceGrip takes 30ms for $d = 0.04$, 33ms for $d = 0.08$, and 40ms for $d = 0.12$.

We also perform experiments to justify our use of different motion predictors for objects of different states. Figure 3-2b shows the results of our model *vs*. a unified dynamics predictor for all objects in BoxBath. As there are only a few states of interest (solids, liquids, and soft bodies), and their physical behaviors are drastically different, it is not surprising that DPI-Nets, with state-specific motion predictors, perform better, and are equally efficient as the unified model (time difference smaller than 3ms per forward step).

3.2.3 Control

We leverage Dynamic Particle Interaction Networks for control tasks in both simulation and real world. Because trajectory optimization using shooting method can easily stuck to a local minimum, we first randomly sample N_{sample} control sequences, and select the best performing one according to the rollouts of our learned model. We then optimize it via shooting method using our model's gradients. We also use online system identification to further improve the model's performance. Figure 3-3 and Figure 3-4 show qualitative and quantitative results, respectively. More details of the control algorithm can be found in Appendix A.5.

- **FluidShake.** We aim to control the speed of the box to match the fluid particles to a target configuration. We compare our method (RS+TO) with random search over the learned model (without trajectory optimization - RS) and Model-free Deep Reinforcement Learning (Actor-Critic method optimized with PPO [39] (RL). Figure 3-4a suggests that our model-based control algorithm outperforms both baselines with a large margin. Also RL is not sample-efficient, requiring more than 10 million time steps to converge while ours requires 600K time steps.

- **RiceGrip.** We aim to select a sequence of gripping configurations (position, orientation, and closing distance) to mold the "sticky rice" to a target shape. We also consider cases where the stiffness of the rice is unknown and need to be identified. Figure 3-4b shows that our Dynamic Particle Interaction Networks with system identification performs the best, and is much more efficient than RL (150K *vs*. 10M time steps).

- **RiceGrip in the real world.** We generalize the learned model and control algorithm for RiceGrip to the real world. We first reconstruct object geometry using a depth camera mounted on our Kuka robot using TSDF volumetric fusion [8]. We then randomly sampled N_{fill} particles within the object mesh as the initial configuration for manipulation. Figure 3-3c and Figure 3-4b shows that, using DPI-Nets, the robot successfully adapts to the real world environment

of unknown physical parameters and manipulates a deformable foam into various target shapes. The learned policy in RiceGrip does not generalize to the real world due to domain discrepancy, and outputs invalid gripping configurations.

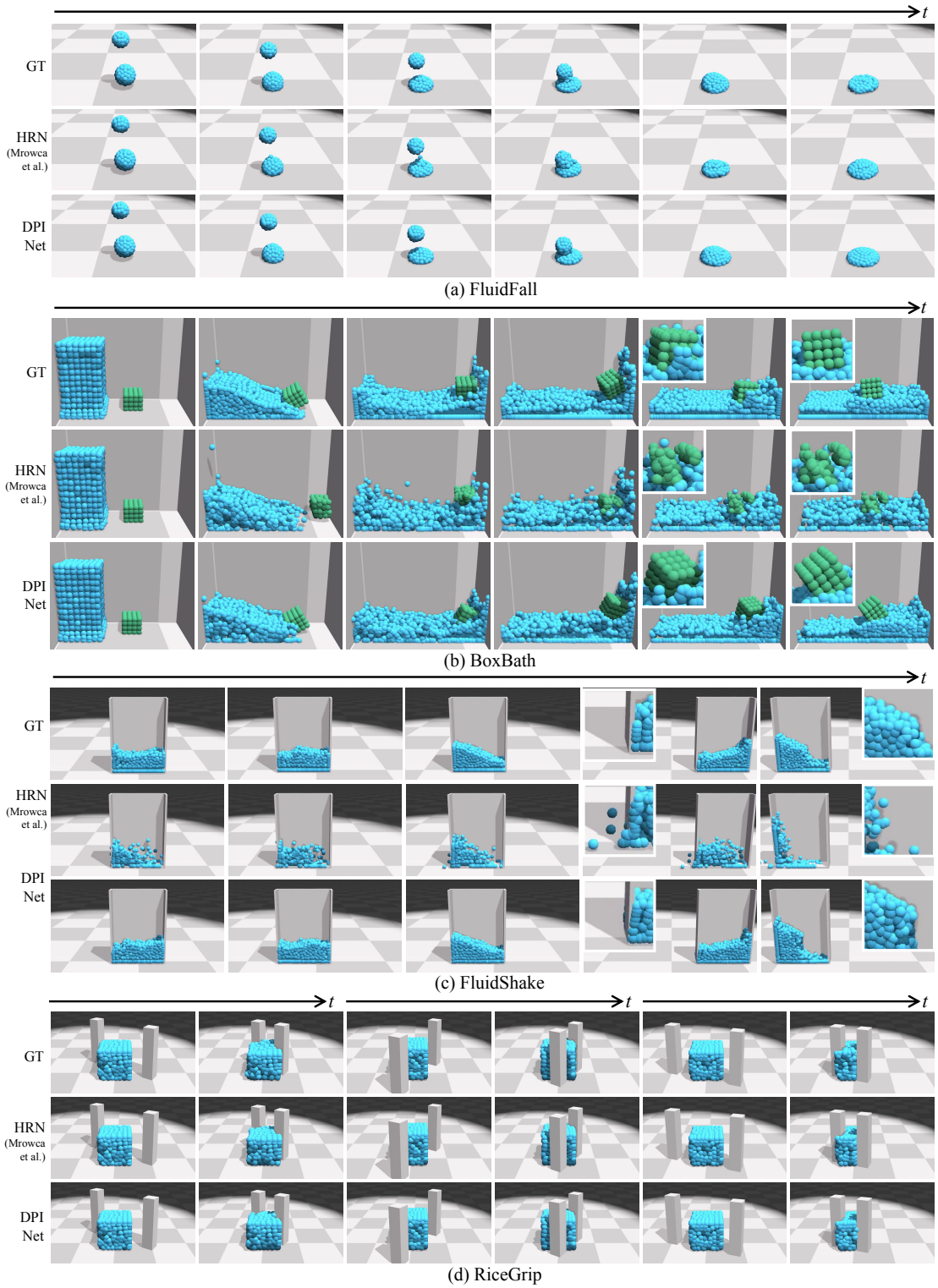
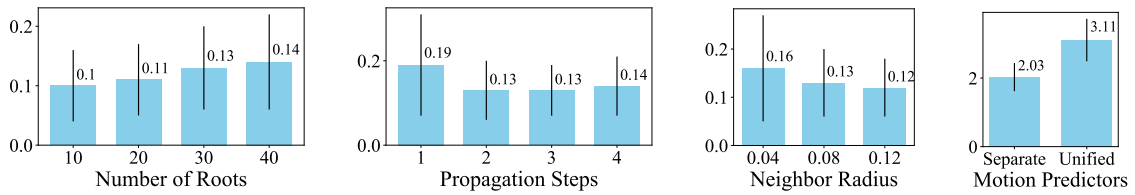


Figure 3-1: **Qualitative results on forward simulation.** We compare the ground truth (GT) and the rollouts from HRN [30] and our model (DPI-Nets) in four environments (FluidFall, BoxBath, FluidShake, and RiceGrip). The simulations from our DPI-Nets are significantly better. We provide zoom-in views for a few frames to show details. Please see our video for more empirical results.



(a) RiceGrip (b) BoxBath

Figure 3-2: **Ablation studies.** We perform ablation studies to test our model's robustness to hyperparameters. The performance is evaluated by the mean squared distance ($\times 10^{-2}$) between the ground truth and model rollouts. (a) We vary the number of roots when building hierarchy, the propagation step L during message passing, and the size of the neighborhood d. (b) In BoxBath, DPI-Nets use separate motion predictors for fluids and rigid bodies. Here we compared with a unified motion predictor.

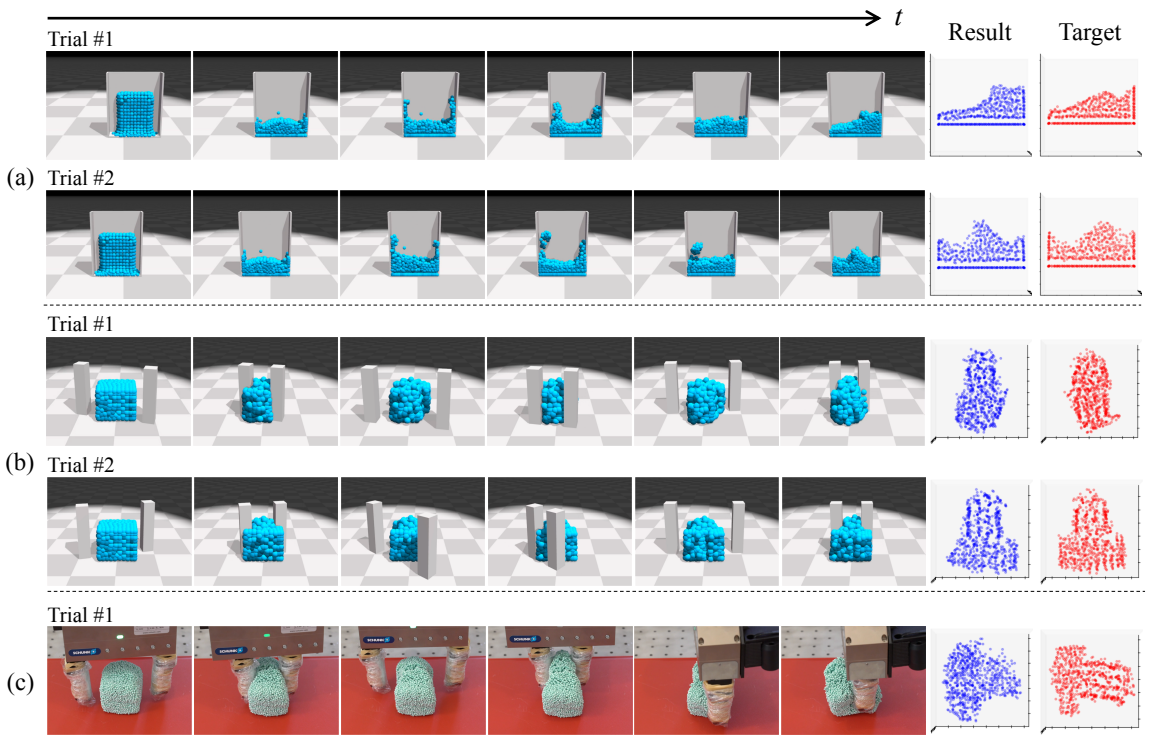
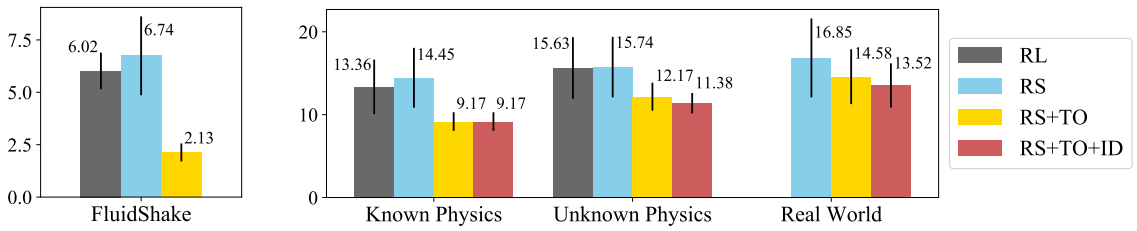


Figure 3-3: **Qualitative results on control.** (a) FluidShake - Manipulating a box of fluids to match a target shape. The Result and Target indicate the fluid shape when viewed from the cutaway view. (b) RiceGrip - Gripping a deformable object and molding it to a target shape. (c) RiceGrip in Real World - Generalize the learned dynamics and the control algorithms to the real world by doing online adaptation. The last two columns indicate the final shape viewed from the top.



(a) FluidShake (b) RiceGrip

Figure 3-4: **Quantitative results on control.** We show the results on control (as evaluated by the Chamfer distance ($\times 10^{-2}$) between the manipulated result and the target) for (a) FluidShake and (b) RiceGrip by comparing with four baselines. **RL**: Model-free deep reinforcement learning optimized with PPO; **RS**: Random search the actions from the learned model and select the best one to execute; RS + **TO**: Trajectory optimization augmented with model predictive control; RS + TO + **ID**: Online system identification by estimating uncertain physical parameters during run time.

Chapter 4

Conclusion

We have presented Propagation Networks (PropNets), a general learnable physics engine that outperforms the previous state-of-the-art with a large margin. We have also demonstrated PropNets's applicability in model-based control under both fully and partially observable environments. With propagation steps, PropNets can propagate the effects along relations and model the dynamics of long-range interactions within a single time step. We have also proposed to improve PropNets's efficiency by adding residual connections and shared encoding.

In the second half of the thesis, we have extended the model and shown a learned particle dynamics model can approximate the interaction of diverse objects, and can help to solve complex manipulation tasks of deformable objects. Our system requires standard open-source robotics and deep learning toolkits, and can be potentially deployed in household and manufacturing environment. Robot learning of dynamic scenes with particle-based representations shows profound potentials due to the generalizability and expressiveness of the representation. Our study helps lay the foundation for it.

Appendix A

Supplementary Materials for DPI-Nets

A.1 Control Algorithm

Algorithm 2 Control on Learned Dynamics at Time Step t

Input: Learned forward dynamics model Φ
Predicted dynamics graph $\hat{G}_t$
Current dynamics graph G_t
Goal $\mathcal{G}_g$, current estimation of the attributes A
Current control inputs $\hat{u}_{t:T}$
States history $\bar{\mathcal{G}} = \{G_i\}_{i=1...t}$
Time horizon T
Output: Controls $\hat{u}_{t:T}$, predicted next time step $\hat{G}_{t+1}$

Update A by descending with the gradients $\nabla_A \mathcal{L}_{\text{state}}(\hat{G}_t, G_t)$
Forward simulation using the current graph $\hat{G}_{t+1} \leftarrow \Phi(G_t)$
Make a buffer for storing the simulation results $\mathcal{G} \leftarrow \bar{\mathcal{G}} \cup \hat{G}_{t+1}$
for $i = t+1, ..., T-1$ **do**
Forward simulation: $\hat{G}_{j+1} \leftarrow \Phi(\hat{G}_j)$; $\mathcal{G} \leftarrow \mathcal{G} \cup \hat{G}_{j+1}$
end for
Update $\hat{u}_{t:T}$ by descending with the gradients $\nabla_{\hat{u}_{t:T}} \mathcal{L}_{\text{goal}}(\mathcal{G}, \mathcal{G}_g)$
Return $\hat{u}_{t:T}$ and $\hat{G}_{t+1} \leftarrow \Phi(G_t)$

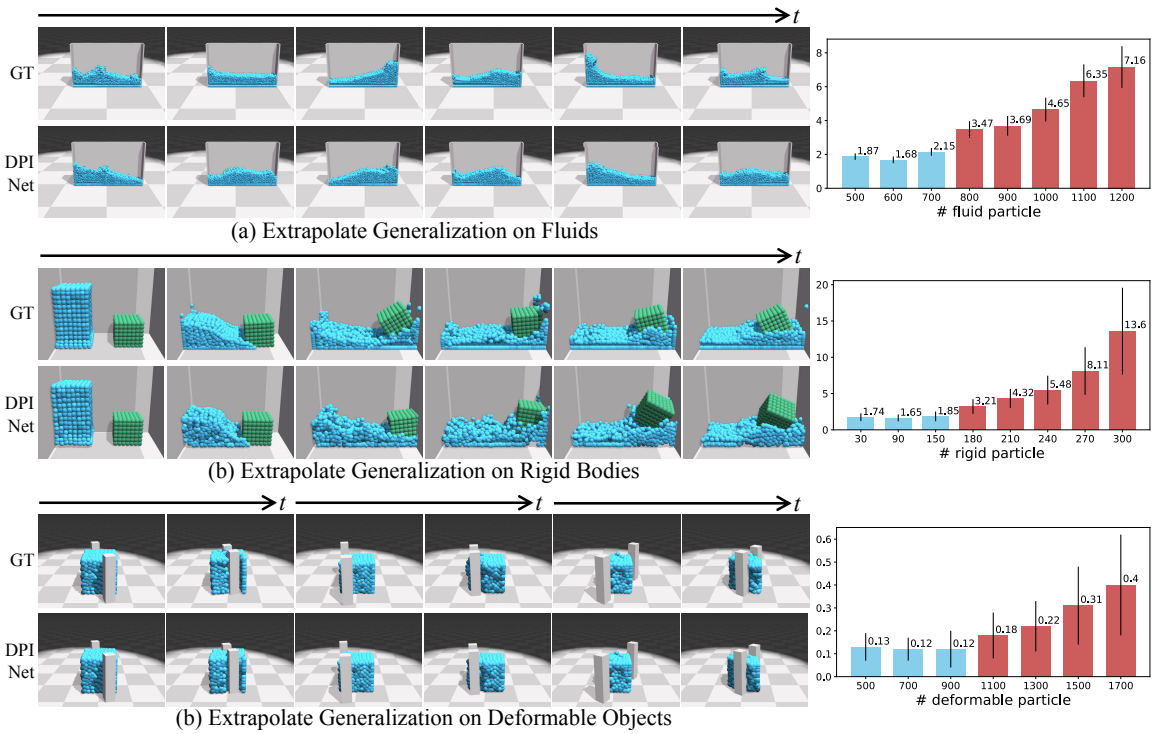


Figure A-1: **Extrapolate generalization on fluids, rigid bodies, and deformable objects.** The performance is evaluated by the MSE ($\times 10^{-2}$) between the ground truth and rollouts from DPI-Nets. The blue bars denote the range of particle numbers that have been seen during training, which indicate interpolation performance. The red bars indicate extrapolation performance that our model can generalize to cases containing two times more particles than cases it has been trained on.

A.2 Generalization on Extrapolation

We show our model's performance on fluids, rigid bodies, and deformable objects with a larger number of particles than they have in the training set. Figure A-1 shows qualitative and quantitative results. Our model scales up well to larger objects.

A.3 Data Generation

The data is generated using NVIDIA FleX. We have developed a Python interface for the ease of generating and interacting with different environments. We will release the code upon publication.

- **FluidFall.** We generated 3,000 rollouts over 120 time steps. The two drops of fluids contain 64 and 125 particles individually, where the initial position of one of the drop in the 3 dimensional coordinates is uniformly sampled between (0.15, 0.55, 0.05) and (0.25, 0.7, 0.15), while the other drop is uniformly sampled between (0.15, 0.1, 0.05) and (0.25, 0.25, 0.15).

- **BoxBath.** We generated 3,000 rollouts over 150 time steps. There are 960 fluid particles and the rigid cube consist particles ranging between 27 and 150. The fluid particle block is initialized at (0, 0, 0), and the initial position of the rigid cube is randomly initialized between (0.45, -0.0155, 0.02) to (1.2, -0.0155, 0.4).

- **FluidShake.** We generated 2,000 rollouts over 300 time steps. The height of the box is 1.0 and the thickness of the wall is 0.025. For the initial fluid cuboid, the number of fluid particles is uniformly sampled between 10 and 12 in the x direction, between 15 and 20 in the y direction, 3 in the z direction. The box is fixed in the y and z direction, and is moving freely in the x direction. We randomly place the initial x position between -0.2 to 0.2. The sampling of the speed is implemented as $\mathrm{v} = \mathrm{v} + \mathrm{rand}(-0.15, 0.15) - 0.1\mathrm{x}$, in order to encourage motion smoothness and moving back to origin, where speed v is initialized as 0.

- **RiceGrip.** We generated 5,000 rollouts over 30 time steps. We randomize the size of the initial "rice" cuboid, where the length of the three sides is uniformly sampled between 8.0 and 10.0. The material property parameters `Stiffness` is uniformly sampled between 0.3 and 0.7, `PlasticThreshold` is uniformly sampled between 1e-5 and 5e-4, and `PlasticCreep` is uniformly sampled between 0.1 and 0.3. The position of the gripper is randomly sampled within a circle of radius 0.5. The orientation of the gripper is always perpendicular to the line connecting the origin to the center of the gripper and the close distance is uniformly sampled between 0.7 to 1.0.

Of all the generated data, 90% of the rollouts are used for training, and the rest 10% are used for validation.

A.4 Training Details

The models are implemented in PyTorch, and are trained using Adam optimizer ([19]) with a learning rate of 0.0001. The number of particles and relations might be different at each time step, hence we use a batch size of 1, and we update the weights of the networks once every 2 forward rounds.

The neighborhood d is set as 0.08, and the propagation step L is set as 2 for all four environments. For hierarchical modeling, it does not make sense to propagate more than one time between leaves and roots as they are disjoint particle sets, and each propagation stage between them only involves one-way edges; hence $\phi_{\text{LeafToLeaf}}$ uses $L = 2$. $\phi_{\text{LeafToRoot}}$ uses $L = 1$. $\phi_{\text{RootToRoot}}$ uses $L = 2$, and $\phi_{\text{RootToLeaf}}$ uses $L = 1$.

For all propagation networks used below, the object encoder f_O^{enc} is an MLP with two hidden layers of size 200, and outputs a feature map of size 200. The relation encoder f_R^{enc} is an MLP with three hidden layers of size 300, and outputs a feature map of size 200. The propagator f_O and f_R are both MLP with one hidden layer of size 200, in which a residual connection is used to better propagate the effects, and outputs a feature map of size 200. The propagators are shared within each stage of propagation. The motion predictor f_O^{output} is an MLP with two hidden layers of size 200, and output the state of required dimension. ReLU is used as the activation function.

- **FluidFall.** The model is trained for 13 epochs. The output of the model is the 3 dimensional velocity, which is multiplied by Δt and added to the current position to do rollouts.

- **BoxBath.** In this environment, four propagation networks are used due to the hierarchical modeling and the number of roots for the rigid cube is set as 8. We have two separate motion predictor for fluids and rigid body, where the fluid predictor output velocity for each fluid particle, while the rigid predictor takes the mean of the signals over all its rigid particles as input, and output a rigid transformation (rotation and translation). The model is trained for 5 epochs.

- **FluidShake.** Only one propagation network is used in this environment, and the model is trained for 5 epochs.

- **RiceGrip.** Four propagation networks are used due to the hierarchical modeling, and the number of roots for the “rice" is set as 30. The model is trained for 20 epochs.

A.5 Control Details

N_{sample} is chosen as 20 for all three cases, where we sample 20 random control sequences, and choose the best performing one as evaluated using our learned model. The evaluation is based on the Chamfer distance between the controlling result and the target configuration.

- **FluidShake.** In this environment, the control sequence is the speed of the box along the x axis. The method to sample the candidate control sequence is the same as when generating training data of this environment. After selected the best performing control sequence, we first use RMSprop optimizer to optimize the control inputs for 10 iterations using a learning rate of 0.003. We then use model-predictive control to apply the control sequence to the FleX physics engine using Algorithm 2.

- **RiceGrip.** In this environment, we need to come up with a sequence of grip configurations, where each grip contains positions, orientation, and closing distance. The method to sample the candidate control sequence is the same as when generating training data of this environment. After selected the best performing control sequence, we first use RMSprop optimizer to optimize the control inputs for 20 iterations using a learning rate of 0.003. We then use model-predictive control to apply the control sequence to the FleX physics engine using Algorithm 2.

- **RiceGrip in Real World.**

In this environment, we need to come up with a sequence of grip configurations, where each grip contains positions, orientation, and closing distance. The method to sample the candidate control sequence is the same as when generating training data of **RiceGrip**, and N_{fill} is chosen as 768. Different from the previous case, the physical parameters are always unknown and has to be estimated online. After selected the best performing control sequence, we first use RMSprop optimizer to optimize the control inputs for 20 iterations using a learning rate of 0.003. We then use model-predictive control to apply the control sequence to the real world using Algorithm 2.

Bibliography

[1] Karl Johan Åström and Tore Hägglund. *PID controllers: theory, design, and tuning*, volume 2. Instrument society of America Research Triangle Park, NC, 1995.

[2] Harry G Barrow, Jay M Tenenbaum, Robert C Bolles, and Helen C Wolf. Parametric correspondence and chamfer matching: Two new techniques for image matching. In *IJCAI*, 1977.

[3] Peter W. Battaglia, Razvan Pascanu, Matthew Lai, Danilo Rezende, and Koray Kavukcuoglu. Interaction networks for learning about objects, relations and physics. In *NIPS*, 2016.

[4] James Bradbury, Stephen Merity, Caiming Xiong, and Richard Socher. Quasi-recurrent neural networks. In *ICLR*, 2017.

[5] Eduardo F Camacho and Carlos Bordons Alba. *Model predictive control*. Springer Science & Business Media, 2013.

[6] Michael B Chang, Tomer Ullman, Antonio Torralba, and Joshua B Tenenbaum. A compositional object-based approach to learning physical dynamics. In *ICLR*, 2017.

[7] Erwin Coumans. Bullet physics engine. *Open Source Software: http://bulletphysics. org*, 2010.

[8] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *SIGGRAPH*, 1996.

[9] Filipe de Avila Belbute-Peres, Kevin Smith, Kelsey Allen, Josh Tenenbaum, and J Zico Kolter. End-to-end differentiable physics for learning and control. In *NIPS*, 2018.

[10] Filipe de Avila Belbute-Peres, Kevin A Smith, Kelsey Allen, Joshua B Tenenbaum, and J Zico Kolter. End-to-end differentiable physics for learning and control. In *NIPS*, 2018.

[11] Jonas Degrave, Michiel Hermans, and Joni Dambre. A differentiable physics engine for deep learning in robotics. In *ICLR Workshop*, 2016.

[12] Robin Deits and Russ Tedrake. Footstep planning on uneven terrain with mixed-integer convex optimization. In *2014 IEEE-RAS international conference on humanoid robots*, pages 279–286. IEEE, 2014.

[13] Gregory Farquhar, Tim Rocktäschel, Maximilian Igl, and Shimon Whiteson. Treeqn and atreec: Differentiable tree planning for deep reinforcement learning. In *ICLR*, 2018.

[14] Katerina Fragkiadaki, Pulkit Agrawal, Sergey Levine, and Jitendra Malik. Learning visual predictive models of physics for playing billiards. In *ICLR*, 2016.

[15] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep q-learning with model-based acceleration. In *ICML*, 2016.

[16] Jessica B Hamrick, Andrew J Ballard, Razvan Pascanu, Oriol Vinyals, Nicolas Heess, and Peter W Battaglia. Metacontrol for adaptive imagination-based optimization. In *ICLR*, 2017.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[18] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Sci.*, 313(5786):504–507, 2006.

[19] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[20] Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. Neural relational inference for interacting systems. In *ICML*, 2018.

[21] Scott Kuindersma, Robin Deits, Maurice Fallon, Andrés Valenzuela, Hongkai Dai, Frank Permenter, Twan Koolen, Pat Marion, and Russ Tedrake. Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot. *Autonomous robots*, 40(3):429–455, 2016.

[22] Tao Lei, Yu Zhang, Sida I. Wang, Hui Dai, and Yoav Artzi. Simple recurrent units for highly parallelizable recurrence. In *EMNLP*, 2018.

[23] Ian Lenz, Ross A Knepper, and Ashutosh Saxena. Deepmpc: Learning deep latent features for model predictive control. In *RSS*, 2015.

[24] Yunzhu Li, Hao He, Jiajun Wu, Dina Katabi, and Antonio Torralba. Learning compositional koopman operators for model-based control. In *ICLR*, 2020.

[25] Yunzhu Li, Toru Lin, Kexin Yi, Daniel Bear, Daniel LK Yamins, Jiajun Wu, Joshua B Tenenbaum, and Antonio Torralba. Visual grounding of learned physical models. *arXiv preprint arXiv:2004.13664*, 2020.

[26] Yunzhu Li, Jiajun Wu, Russ Tedrake, Joshua B Tenenbaum, and Antonio Torralba. Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. In *ICLR*, 2019.

[27] Yunzhu Li, Jiajun Wu, Jun-Yan Zhu, Joshua B Tenenbaum, Antonio Torralba, and Russ Tedrake. Propagation networks for model-based control under partial observation. In *ICRA*, 2019.

[28] Miles Macklin and Matthias Müller. Position based fluids. *ACM TOG*, 32(4):104, 2013.

[29] Miles Macklin, Matthias Müller, Nuttapong Chentanez, and Tae-Yong Kim. Unified particle physics for real-time applications. *ACM TOG*, 33(4):153, 2014.

[30] Damian Mrowca, Chengxu Zhuang, Elias Wang, Nick Haber, Li Fei-Fei, Joshua B Tenenbaum, and Daniel LK Yamins. Flexible neural representation for physics prediction. In *NIPS*, 2018.

[31] Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *ICRA*, 2018.

[32] Junhyuk Oh, Satinder Singh, and Honglak Lee. Value prediction network. In *NIPS*, 2017.

[33] Razvan Pascanu, Yujia Li, Oriol Vinyals, Nicolas Heess, Lars Buesing, Sebastien Racanière, David Reichert, Théophane Weber, Daan Wierstra, and Peter Battaglia. Learning model-based planning from scratch. *arXiv:1707.06170*, 2017.

[34] Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Elsevier, 2014.

[35] Michael Posa, Cecilia Cantu, and Russ Tedrake. A direct method for trajectory optimization of rigid bodies through contact. *The International Journal of Robotics Research*, 33(1):69–81, 2014.

[36] Sébastien Racanière, Théophane Weber, David Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adrià Puigdomènech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, Razvan Pascanu, Peter Battaglia, David Silver, and Daan Wierstra. Imagination-augmented agents for deep reinforcement learning. In *NIPS*, 2017.

[37] 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 *ICML*, 2018.

[38] Connor Schenck and Dieter Fox. Spnets: Differentiable fluid dynamics for deep neural networks. In *CoRL*, 2018.

[39] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv:1707.06347*, 2017.

[40] David Silver, Hado van Hasselt, Matteo Hessel, Tom Schaul, Arthur Guez, Tim Harley, Gabriel Dulac-Arnold, David Reichert, Neil Rabinowitz, Andre Barreto, and Thomas Degris. The predictron: End-to-end learning and planning. In *ICML*, 2017.

[41] Aravind Srinivas, Allan Jabri, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Universal planning networks. In *ICML*, 2018.

[42] David E Stewart. Rigid-body dynamics with friction and impact. *SIAM review*, 42(1):3–39, 2000.

[43] Russ Tedrake. Underactuated robotics: Learning, planning, and control for efficient and agile machines course notes for mit 6.832, 2009.

[44] Russ Tedrake and the Drake Development Team. Drake: Model-based design and verification for robotics, 2019.

[45] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *IROS*, pages 5026–5033. IEEE, 2012.

[46] Marc Toussaint, K Allen, K Smith, and J Tenenbaum. Differentiable physics and stable modes for tool-use and manipulation planning. In *RSS*, 2018.

[47] Sjoerd van Steenkiste, Michael Chang, Klaus Greff, and Jürgen Schmidhuber. Relational neural expectation maximization: Unsupervised discovery of objects and their interactions. In *ICLR*, 2018.

[48] Nicholas Watters, Andrea Tacchetti, Theophane Weber, Razvan Pascanu, Peter Battaglia, and Daniel Zoran. Visual interaction networks. In *NIPS*, 2017.

[49] Jiajun Wu, Erika Lu, Pushmeet Kohli, Bill Freeman, and Josh Tenenbaum. Learning to see physics via visual de-animation. In *NIPS*, 2017.

[50] Kexin Yi, Chuang Gan, Yunzhu Li, Pushmeet Kohli, Jiajun Wu, Antonio Torralba, and Joshua B. Tenenbaum. {CLEVRER}: Collision events for video representation and reasoning. In *ICLR*, 2020.