Learning Compositional Koopman Operators for Model-Based Control

Yunzhu Li[*] MIT CSAIL

Hao He[*] MIT CSAIL

Jiajun Wu MIT CSAIL

Dina Katabi MIT CSAIL

Antonio Torralba MIT CSAIL

Abstract

Finding an embedding space for a linear approximation of a nonlinear dynamical system enables efficient system identification and control synthesis. The Koopman operator theory lays the foundation for identifying the nonlinear-to-linear coordinate transformations with data-driven methods. Recently, researchers have proposed to use deep neural networks as a more expressive class of basis functions for calculating the Koopman operators. These approaches, however, assume a fixed dimensional state space; they are therefore not applicable to scenarios with a variable number of objects. In this paper, we propose to learn compositional Koopman operators, using graph neural networks to encode the state into object-centric embeddings and using a block-wise linear transition matrix to regularize the shared structure across objects. The learned dynamics can quickly adapt to new environments of unknown physical parameters and produce control signals to achieve a specified goal. Our experiments on manipulating ropes and controlling soft robots show that the proposed method has better efficiency and generalization ability than existing baselines.

1 Introduction

Simulating and controlling complex dynamical systems such as ropes or soft robots rely on the dynamics model's two key features: first, it needs to be *efficient* for system identification and motor control; second, it needs to be *generalizable* to a complex, constantly evolving environments.

In practice, computational models for complex, nonlinear dynamical systems are often not efficient enough for real-time control [21]. The Koopman operator theory identifies nonlinear-to-linear coordinate transformations allowing efficient linear approximation of nonlinear systems [31, 19]. Fast as they are; however, existing papers on Koopman operators focus on a single dynamical system, making it hard for them to generalize to cases where there are a variable number of components.

In contrast, recent advances in approximating dynamics models with deep nets have demonstrated its power in characterizing complex, generic environments. In particular, a few recent papers have explored the use of graph nets in dynamics modeling, taking into account the state of each object as well as their interactions. This allows their models to generalize to scenarios with a variable number of objects [4, 9]. Despite their strong generalization power, they are not as efficient in system identification and control, because deep nets are heavily over-parameterized, making optimization time-consuming and sample-inefficient.

In this paper, we propose compositional Koopman operators, integrating Koopman operators with graph networks for generalizable and efficient dynamics modeling. We build on the idea of encoding states into object-centric embeddings with graph neural networks, which ensures generalization power. But instead of using over-parameterized neural nets to model state transition, we identify the Koopman matrix and control matrix from data as a linear approximation of the nonlinear dynamical system. The linear approximation allows efficient system identification and control synthesis.

[*]indicates equal contributions. Video: https://youtu.be/idFH4K16cfQ

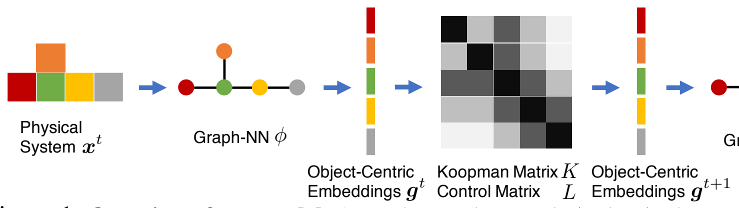
Physical System x^t
Graph-NN ϕ
Object-Centric Embeddings g^t
Koopman Matrix K
Control Matrix L
Object-Centric Embeddings g^{t+1}
Graph-NN ψ
Physical System x^{t+1}

Figure 1: **Overview of our model.** A graph neural network ϕ takes in the current state of the physical system $\boldsymbol{x}^t$, and generates object-centric representations in the Koopman space $\boldsymbol{g}^t$. We then use the block-wise Koopman matrix K and control matrix L identified from equation 4 to predict the Koopman embeddings in the next time step $\boldsymbol{g}^{t+1}$. Note that in K and L, object pairs of the same relation (e.g., adjacent blocks) share the same sub-matrix. Another graph neural network ψ maps $\boldsymbol{g}^{t+1}$ back to the original state space (i.e., $\boldsymbol{x}^{t+1}$). The mapping between $\boldsymbol{g}^t$ and $\boldsymbol{g}^{t+1}$ is linear and is shared across all time steps. We can iteratively apply K and L to the Koopman embeddings and roll multiple steps into the future.

The main challenge of extending Koopman theory to multi-object systems is scalability. The number of parameters in the Koopman matrix scales quadratically with the number of objects, which harms the learning efficiency and leads to overfitting. To tackle this issue, we exploit the structure of the underlying system and use the same block-wise Koopman sub-matrix for object pairs of the same relation. This significantly reduces the number of parameters that need to be identified by making it independent of the size of the system.

Our experiments include simulating and controlling ropes of variable lengths and soft robots of different shapes. The compositional Koopman operators are significantly more accurate than the state-of-the-art learned physics engines [4, 16], and is 20 times faster when adapting to new environments of unknown physical parameters. Our method also outperforms vanilla deep Koopman methods [17, 22] and Koopman models with manually-designed basis functions, which shows the advantages of using a structured Koopman matrix and graph neural networks. Please refer to our supplement video.

2 Method

2.1 Compositional Koopman Operators

The Koopman theory[†] [14, 15] says a non-linear dynamical system can be mapped to a higher dimension space where the dynamics becomes linear. Formally, we denote a non-linear dynamical system as $\boldsymbol{x}^{t+1} = F(\boldsymbol{x}^t, \boldsymbol{u}^t)$, where $\boldsymbol{x}^t$ and $\boldsymbol{u}^t$ are the state and the control signal at time step t. There is a function g that builds the correspondence between the original non-linear forwarding and the linear forwarding using the Koopman matrix K and the control matrix L, i.e., $g(\boldsymbol{x}^{t+1}) = Kg(\boldsymbol{x}^t) + L\boldsymbol{u}^t$.

The dynamics of a physical system are governed by physical rules, which are usually shared across the many objects in the system. We propose *Compositional Koopman Operators* to incorporate such compositionality to the Koopman theory. In section C in the appendix, we analyze a simple multi-object system - a linear spring system. Motivated by the analysis, we draw several principles to inject the right inductive bias to the Compositional Koopman Operators.

Consider a system with N objects, we denote $\boldsymbol{x}^t$ as the system state at time t and $\boldsymbol{x}_i^t$ is the state of the i'th object. Denoting $\boldsymbol{g}^t$ as the Koopman embedding, we propose following assumptions on the compositional structure of the Koopman embedding and the Koopman matrix.

- **The Koopman embedding of the system is composed of the Koopman embedding of every objects.** Similar to the decomposition in the state space, we assume the Koopman embedding can be divided into object-centric sub-embeddings, i.e. $\boldsymbol{g}^t = [\boldsymbol{g}_1^{t\top}, \cdots, \boldsymbol{g}_N^{t\top}]^\top \in \mathbb{R}^{Nm}$, where we overload the notation and use $\boldsymbol{g}_i^t = g_i(\boldsymbol{x}^t) \in \mathbb{R}^m$ as the Koopman embedding for the i'th object.
- **The Koopman matrix has a block-wise structure.** It is natural to think the Koopman matrix is composed of block matrices after assuming an object-centric Koopman embeddings. In equation 1, $K_{ij} \in \mathbb{R}^{m\times m}$ and $L_{ij} \in \mathbb{R}^{m\times|\boldsymbol{u}_i|}$ are blocks of the Koopman matrix and the control matrix, while $\boldsymbol{u}^t = [\boldsymbol{u}_1^{t\top}, \cdots, \boldsymbol{u}_N^{t\top}]^\top$ is the control signal at time t:

$$\begin{bmatrix} \boldsymbol{g}_1^{t+1} \\ \vdots \\ \boldsymbol{g}_N^{t+1} \end{bmatrix} = \begin{bmatrix} K_{11} & \cdots & K_{1N} \\ \vdots & \ddots & \vdots \\ K_{N1} & \cdots & K_{NN} \end{bmatrix} \begin{bmatrix} \boldsymbol{g}_1^{t} \\ \vdots \\ \boldsymbol{g}_N^{t} \end{bmatrix} + \begin{bmatrix} L_{11} & \cdots & L_{1N} \\ \vdots & \ddots & \vdots \\ L_{N1} & \cdots & L_{NN} \end{bmatrix} \begin{bmatrix} \boldsymbol{u}_1^{t} \\ \vdots \\ \boldsymbol{u}_N^{t} \end{bmatrix}. \quad (1)$$

[†]Please see section *B* in the supplementary material for a more detailed introduction to the Koopman theory.

Note that those matrix blocks are not independent but some of them share the same set of values.

- **The same physical interactions shall share the same transition block.** The equivalence between the blocks should reflect the equivalence of the objects, where we use the same transition sub-matrix for object pairs of the same relation. For example, if the system the composed of N identical objects, then, by symmetry, all the diagonal blocks should be the same while all the off-diagonal blocks should also be the same. The repetitive structure allows us to efficiently identify the values using least square regression.

2.2 Learning the Koopman Embeddings Using Graph Neural Networks

For a physical system that contains N objects, we can represent the system using a graph $G^t = \langle O^t, R \rangle$, where vertices $O^t = \{\boldsymbol{o}_i^t\}$ represent objects and edges $R = \{\boldsymbol{r}_k\}$ represent pair-wise relations. Specifically, $\boldsymbol{o}_i^t = \langle \boldsymbol{x}_i^t, \boldsymbol{a}_i^o \rangle$, where $\boldsymbol{x}_i^t$ is the state of object i and $\boldsymbol{a}_i^o$ is a one-hot vector indicating the type of this object. Note the operator $\langle \cdot, \cdot \rangle$ denotes vector concatenation. For relation, we have $\boldsymbol{r}_k = \langle u_k, v_k, \boldsymbol{a}_k^r \rangle, 1 \leq u_k, v_k \leq N$, where u_k and v_k are integers denoting the receiver and the sender, respectively, and $\boldsymbol{a}_k^r$ is a one-hot vector denoting the type of the relation k.

We use a graph neural network similar to Interaction Networks (IN) [4] to generate object-centric Koopman embeddings. IN is a general-purpose, learnable physics engine that performs 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. Specifically, we iteratively calculate the edge effect $\boldsymbol{e}_k^t = f_R(\boldsymbol{o}_{u_k}^t, \boldsymbol{o}_{v_k}^t, \boldsymbol{a}_k^r)_{k=1\ldots N^2}$, and node effect $\boldsymbol{g}_i^t = f_O(\boldsymbol{o}_i^t, \sum_{k \in \mathcal{N}_i} \boldsymbol{e}_k^t)_{i=1\ldots N}$, where $\boldsymbol{o}_i^t = \langle \boldsymbol{x}_i^t, \boldsymbol{a}_i^o \rangle$ denotes object i at time t, u_k and v_k are the receiver and sender of relation $\boldsymbol{r}_k$ respectively, and $\mathcal{N}_i$ denotes the relations where object i is the receiver. In total, we denote the graph encoder as ϕ. To train the model, we also define a graph decoder ψ to map the Koopman embeddings back to the original state space. Figure 1 shows an overview of our model. Please refer to Section D in the supplementary material for the loss functions and training protocols.

The learned dynamics in the Koopman space is linear. We can perform efficient **system identification using least-square regression**, and **control synthesis using quadratic programming.** Please see Section E and F for the exact algorithms.

3 Experiments

Environments. We evaluate our method by assessing how well our method can simulate and control ropes and soft robots. Specifically, we consider three environments. (1) **Rope** (Figure 2a): the top mass of a rope is fixed to a specific height. We apply force to the top mass to move it in a horizontal line. The rest of the masses are free to move according to spring force and gravity. (2) **Soft** (Figure 2b): we aim to control a soft robot that is consist of soft blocks. Blocks in dark grey as rigid and those in light blue are soft blocks. Each one of the dark blue blocks are soft but have an actuator inside that can contract or expand the block. One of the block is pinned to the ground as shown using the red dots. (3) **Swim** (Figure 2c): instead of pinning the soft robot to the ground, we let the robot swim in fluids. The colors shown in this environment have the same meaning as in **Soft**.

Baselines. We compare our model to following baselines: Interaction Networks [4] (**IN**), Propagation Networks [16] (**PN**) and Koopman method with hand-crafted Koopman base functions (**KPM**). Section I explains more on how we use them to perform simulation and control tasks.

3.1 Simulation

Please refer to Section G in the supplementary material for details on data generation, and Section H for evaluation protocols. Figure 2 shows qualitative results on simulation. Our model accurately predicts system dynamics for more than 100 steps. For Rope, the small prediction error comes from the slight delay of the force propagation inside the rope; hence, the tail of the rope usually has a larger error. For Soft, our model captures the interaction between the body parts and generates accurate prediction over the global movements of the robot. The error mainly comes from the mis-alignment of some local components. Figure 3 shows quantitative results. IN and PN do not work well due to inefficient system identification ability. The KPM baseline with simple hand-crafted polynomial Koopman basis works reasonably well in the Rope and Swim environment, partly due to the fact that KPM uses the same system identification algorithm as our model, leveraging the prior knowledge on the structure of the system. Our model significantly outperforms all the baselines except in the Swim environment, where we are on par with KPM.

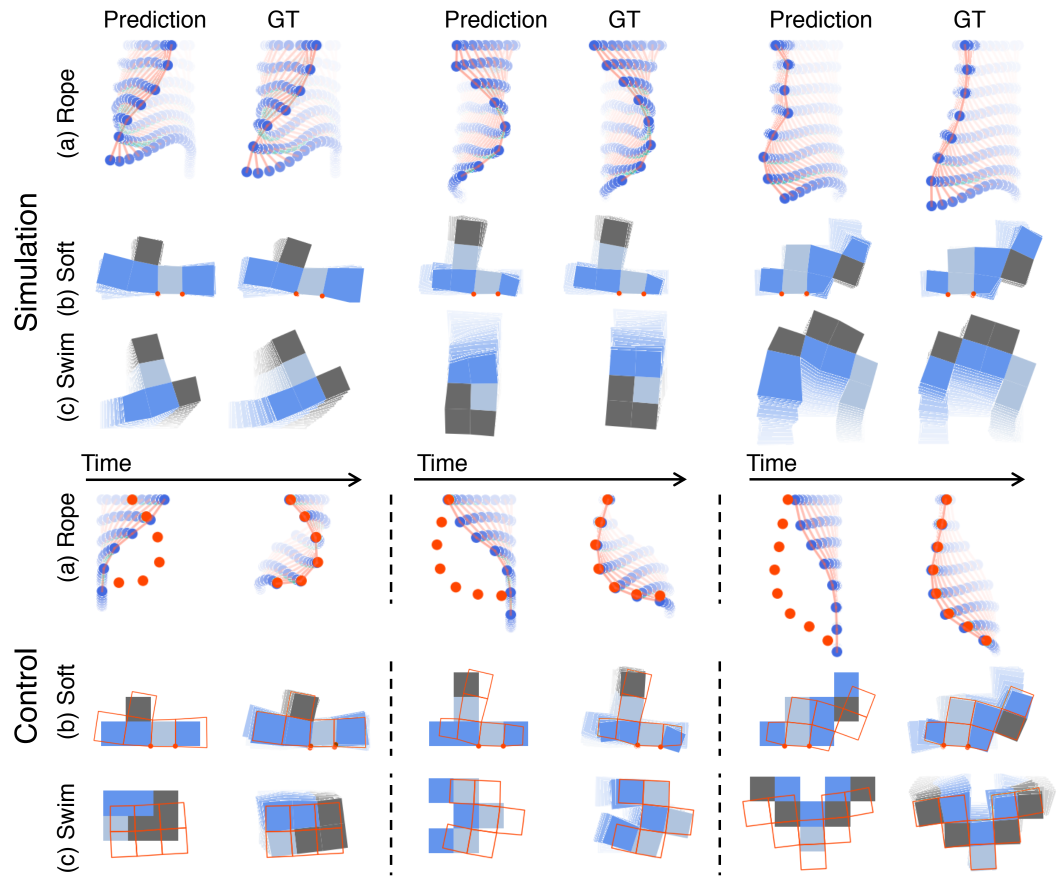

Figure 2: **Qualitative results.** Top: our model prediction matches the ground truth over a long period of time. Bottom: for control, we use red dots or frames to indicate the goal. We apply the control signals generated from our identified model to the original simulator, which allows the agent to accurately achieve the goal. Please refer to our supplementary video for more results.

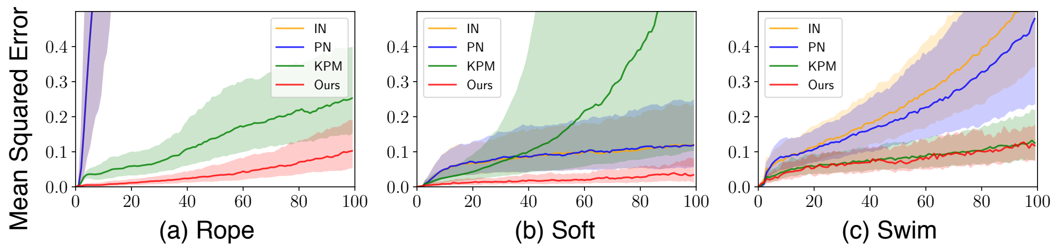

Figure 3: **Quantitative results on simulation.** The x axis shows time steps. The solid lines indicate medians and the transparent regions are the interquartile ranges of simulation errors. Our method significantly outperforms baselines in both Rope and Soft.

3.2 Control

As the simulation errors of IN and PN are too large for control, we compare our model with KPM. In Rope, we ask the models to perform open-loop control where it only solves the Quadratic Programming (QP) once at the beginning. The length of the control sequence is 40. When it comes to Soft/Swim, each model is asked to generate control signals of 64 steps, and we allow the model to receive feedback after 32 steps. Thus every model have a second chance to correct its control sequence by solving the QP again at the time step 32. As shown in Figure 2, our model can accurately manipulate the rope to reach a target state. It learns to leverage inertia to reach target shapes. As for controlling a soft body swinging on the ground or swimming in the water, our model can move each parts (the boxes) of the body to the exact target positions. The small control error comes from the slight misalignment of the orientation and the size of the body parts. Figure 4 in the supplementary material shows that quantitatively our model outperforms KPM, too.

References

[1] Ian Abraham, Gerardo De La Torre, and Todd D Murphey. Model-based control using koopman operators. *arXiv preprint arXiv:1709.01568*, 2017.

[2] Hassan Arbabi, Milan Korda, and Igor Mezic. A data-driven koopman model predictive control framework for nonlinear flows. *arXiv preprint arXiv:1804.05291*, 2018.

[3] 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*, 2018.

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

[5] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[6] Daniel Bruder, Brent Gillespie, C David Remy, and Ram Vasudevan. Modeling and control of soft robots using the koopman operator and model predictive control. In *RSS*, 2019.

[7] Daniel Bruder, C David Remy, and Ram Vasudevan. Nonlinear system identification of soft robot dynamics using koopman operator theory. *arXiv preprint arXiv:1810.06637*, 2018.

[8] Steven L Brunton, Bingni W Brunton, Joshua L Proctor, and J Nathan Kutz. Koopman invariant subspaces and finite linear representations of nonlinear dynamical systems for control. *PloS one*, 11(2):e0150171, 2016.

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

[10] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *ICML*, 2019.

[11] 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.

[12] Michael Janner, Sergey Levine, William T Freeman, Joshua B Tenenbaum, Chelsea Finn, and Jiajun Wu. Reasoning about physical interactions with object-oriented prediction and planning. In *ICLR*, 2019.

[13] Eurika Kaiser, J Nathan Kutz, and Steven L Brunton. Data-driven discovery of koopman eigenfunctions for control. *arXiv preprint arXiv:1707.01146*, 2017.

[14] Bernard O Koopman. Hamiltonian systems and transformation in hilbert space. *Proceedings of the National Academy of Sciences of the United States of America*, 17(5):315, 1931.

[15] BO Koopman and J v Neumann. Dynamical systems of continuous spectra. *Proceedings of the National Academy of Sciences of the United States of America*, 18(3):255, 1932.

[16] 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.

[17] Bethany Lusch, J Nathan Kutz, and Steven L Brunton. Deep learning for universal linear embeddings of nonlinear dynamics. *Nature communications*, 9(1):4950, 2018.

[18] Giorgos Mamakoukas, Maria Castano, Xiaobo Tan, and Todd Murphey. Local koopman operators for data-driven control of robotic systems. In *RSS*, 2019.

[19] Alexandre Mauroy and Jorge Goncalves. Linear identification of nonlinear systems: A lifting technique based on the koopman operator. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pages 6500–6505. IEEE, 2016.

[20] Alexandre Mauroy and Jorge Goncalves. Koopman-based lifting techniques for nonlinear systems identification. *arXiv preprint arXiv:1709.02003*, 2017.

[21] David Mayne. Nonlinear model predictive control: Challenges and opportunities. In *Nonlinear model predictive control*, pages 23–44. Springer, 2000.

[22] Jeremy Morton, Freddie D Witherden, Antony Jameson, and Mykel J Kochenderfer. Deep dynamical modeling and control of unsteady fluid flows. *arXiv preprint arXiv:1805.07472*, 2018.

[23] Jeremy Morton, Freddie D Witherden, and Mykel J Kochenderfer. Deep variational koopman models: Inferring koopman observations for uncertainty-aware dynamics modeling and control. In *IJCAI*, 2019.

[24] 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.

[25] 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.

[26] Joshua L Proctor, Steven L Brunton, and J Nathan Kutz. Generalizing koopman theory to allow for inputs and control. *SIAM Journal on Applied Dynamical Systems*, 17(1):909–930, 2018.

[27] 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.

[28] 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.

[29] Naoya Takeishi, Yoshinobu Kawahara, and Takehisa Yairi. Learning koopman invariant subspaces for dynamic mode decomposition. In *Advances in Neural Information Processing Systems*, pages 1130–1140, 2017.

[30] Matthew O Williams, Maziar S Hemati, Scott TM Dawson, Ioannis G Kevrekidis, and Clarence W Rowley. Extending data-driven koopman analysis to actuated systems. *IFAC-PapersOnLine*, 49(18):704–709, 2016.

[31] Matthew O Williams, Ioannis G Kevrekidis, and Clarence W Rowley. A data–driven approximation of the koopman operator: Extending dynamic mode decomposition. *Journal of Nonlinear Science*, 25(6):1307–1346, 2015.

Appendix

A Related Work

Koopman operators. The Koopman operator formalism of dynamical systems is rooted in the seminal works of Koopman and Von Neumann in the early 1930s [14, 15]. The core idea is to map the state of a nonlinear dynamical system to an embedding space, over which we can linearly propagate into the future. The linear representation will enable efficient prediction, estimation, and control using tools from linear dynamical systems [30, 26, 20]. People have been using hand-designed Koopman observables for various modeling and control tasks [8, 13, 1, 7, 2]. Some recent works have applied the method to the real world and successfully control soft robots with great precision [6, 18].

However, hand-crafted basis functions sometimes fail to generalize to more complex environments. Learning these functions from data using neural nets turns out to generate a more expressive invariant subspace [17, 29] and has achieved successes in fluid control [22]. Morton et al. [23] has also extended the framework to account for uncertainty in the system by inferring a distribution over observations. Our model differs by explicitly modeling the compositionality of the underlying system with graph networks. It generalizes better to environments of a variable number of objects or soft robots of different shapes.

Learning-based physical simulators. Battaglia et al. [4] and Chang et al. [9] first explored learning a simulator from data by approximating object interactions with neural networks. These models are no longer bounded to hard-coded physical rules, and can quickly adapt to scenarios where the underlying physics is unknown. Please refer to Battaglia et al. [3] for a full review. Recently, Mrowca et al. [24] extended these models to approximate particle dynamics of deformable shapes and fluids. Flexible as they are, these models become less efficient during model adaptation in complex scenarios, because the optimization of neural networks usually needs a lot of samples and compute, which limits its use in an online setting. The use of Koopman operators in our model enables efficient system identification, because we only have to identify the transition matrices, which is essentially a least-square problem and can be solved very efficiently.

Learned physics engines have also been used for planning and control. Many prior papers in this direction learn a latent dynamics model together with a policy in a model-based reinforcement learning setup [27, 11, 25, 10]; a few alternatives uses the learned model in model-predictive control (MPC) [28, 16, 12]. In this paper, we leverage the fact that the embeddings in the Koopman space is propagating linearly through time, which allows us to formulate the control problem as quadratic programming and optimize the control signals much more efficiently.

B The Koopman Operators

Let $\boldsymbol{x}^t \in \mathcal{X} \subset \mathbb{R}^n$ be the state vector for the system at time step t. We consider a non-linear discrete-time dynamical system described by $\boldsymbol{x}^{t+1} = F(\boldsymbol{x}^t)$. The Koopman operator $\mathcal{K}$ is an infinite-dimensional linear operator that acts on all observable functions $g : \mathcal{X} \to \mathbb{R}$. The Koopman theory says that any non-linear discrete-time system can be mapped to a linear discrete-time system through a certain Koopman operator [14]. The non-linear system forwarding in the original state space corresponds to the forwarding of its observations described by the Koopman operator, i.e. $\mathcal{K}g(\boldsymbol{x}^t) = g(F(\boldsymbol{x}^t)) = g(\boldsymbol{x}^{t+1})$.

Although the theory guarantees the existence of the Koopman operator, its use in practice is limited by its infinite dimensionality. Most often, we assume there is an invariant subspace $\mathcal{G}$, spanned by a set of base observation functions $\{g_1, \cdots, g_m\}$, such that $Kg \in \mathcal{G}$ for any $g \in \mathcal{G}$, where K is a $\mathbb{R}^{m \times m}$ matrix. With a slightly abuse of the notation, we now use $g(\boldsymbol{x}^t) : \mathbb{R}^n \to \mathbb{R}^m$ to represent $[g_1(\boldsymbol{x}^t), \cdots, g_m(\boldsymbol{x}^t)]^T$. By constraining the Koopman operator on this invariant subspace, we get a finite dimensional linear operator K that we refer as the *Koopman matrix*.

Traditionally, people hand-craft base observation functions from the knowledge of underling physics. The system identification problem is then reduced to finding the Koopman operator $\mathcal{K}$, which can be solved by linear regression given historical data of the system. Recently, researchers have also explored data-driven methods that automatically find the Koopman invariant subspace via representing the observation functions $g(\boldsymbol{x})$ via deep neural networks.

Above is the Koopman theory on modeling unforced dynamics. Now consider a system with an external control input $\boldsymbol{u}^t$ and a dynamics model $\boldsymbol{x}^{t+1} = F(\boldsymbol{x}^t, \boldsymbol{u}^t)$. We aim to find the observation functions and the linear dynamics model in the form of $g(\boldsymbol{x}^{t+1}) = Kg(\boldsymbol{x}^t) + L\boldsymbol{u}^t$, where we assume the control signal has linear effects in the observation domain. Here the coefficient matrix L is referred to as the *control matrix*.

C Motivating Example

Consider a system with n balls moving in a 2D plane, each pair connected by a linear spring. Assume all balls have mass 1 and all springs share the same stiffness coefficient k. We denote the i's ball's position as (x_i, y_i) and its velocity as $(\dot{x}_i, \dot{y}_i)$. For ball i, equation 2 describes its dynamics, where $\boldsymbol{x}_i \triangleq [x_i, y_i, \dot{x}_i, \dot{y}_i]^T$ denotes ball i's state:

$$\dot{\boldsymbol{x}}_i = \begin{bmatrix} \dot{x}_i \\ \dot{y}_i \\ \ddot{x}_i \\ \ddot{y}_i \end{bmatrix} = \begin{bmatrix} \dot{x}_i \\ \dot{y}_i \\ \sum_{j=1}^n k(x_j - x_i) \\ \sum_{j=1}^n k(y_j - y_i) \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ k - nk & 0 & 0 & 0 \\ 0 & k - nk & 0 & 0 \end{bmatrix}}_{\triangleq A} \begin{bmatrix} x_i \\ y_i \\ \dot{x}_i \\ \dot{y}_i \end{bmatrix} + \sum_{j \neq i} \underbrace{\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ k & 0 & 0 & 0 \\ 0 & k & 0 & 0 \end{bmatrix}}_{\triangleq B} \begin{bmatrix} x_i \\ y_i \\ \dot{x}_i \\ \dot{y}_i \end{bmatrix}. \tag{2}$$

We can represent the state of the whole system using the union of every ball's state, where $\boldsymbol{x} = [\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n]^T$. Then the transition matrix is essentially a block matrix, where the matrix parameters are shared among the diagonal or off-diagonal blocks as shown in equation 3:

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{\boldsymbol{x}}_1 \\ \dot{\boldsymbol{x}}_2 \\ \vdots \\ \dot{\boldsymbol{x}}_n \end{bmatrix} = \begin{bmatrix} A & B & \cdots & B \\ B & A & \cdots & B \\ \vdots & \vdots & \ddots & \vdots \\ B & B & \cdots & A \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_n \end{bmatrix}. \tag{3}$$

Based on the linear spring system, we make three observations for multi-object systems.

- **The system state is composed of the state of each individual object.** The dimension of the whole system scales linearly with the number of objects. We formulate the system state by concatenating the state of every object, corresponding to an object-centric state representation.
- **The transition matrix has a block-wise substructure.** After assuming an object-centric state representation, the transition matrix naturally has a block-wise structure as shown in equation 3.
- **The same physical interactions share the same transition block.** The blocks in the transition matrix encode actual interactions and generalize across systems. A and B govern the dynamics of the linear spring system, and are shared by systems with a different number of objects.

These observations motivate us to exploit the structure of multi-object systems, instead of assuming that a 3-ball system and a 4-ball system have different dynamics and need to be learned separately.

D Training GNN models

To make predictions on the states, we use a graph decoder ψ to map the Koopman embeddings back to the original state space. In total, we have three losses to train the graph encoder and decoder. The first term is the auto-encoding loss $\mathcal{L}_{\text{ae}} = \frac{1}{T}\sum_i \|\psi(\phi(\boldsymbol{x}^i)) - \boldsymbol{x}^i\|_2$. The second term is the prediction loss. To calculate it, we rollout in the Koopman space and denote the embeddings as $\hat{\boldsymbol{g}}^1 = \boldsymbol{g}^1$, and $\hat{\boldsymbol{g}}^{t+1} = K\hat{\boldsymbol{g}}^t + L\boldsymbol{u}^t$, for $t = 1, \cdots, T-1$. The prediction loss is defined as the difference between the decoded states and the actual states, i.e., $\mathcal{L}_{\text{pred}} = \frac{1}{T}\sum_{i=1}^T \|\psi(\hat{\boldsymbol{g}}^i) - \boldsymbol{x}^i\|_2$. Third, we employ a metric loss to encourage the Koopman embeddings preserving the distance in the original state space. The loss is defined as the absolute error between the distances measured in the Koopman space and that in the original space, i.e., $\mathcal{L}_{\text{metric}} = \sum_{ij} \big| \|\boldsymbol{g}^i - \boldsymbol{g}^j\|_2 - \|\boldsymbol{x}^i - \boldsymbol{x}^j\|_2 \big|$. Having Koopman embeddings that can reflect the distance in the state space is important as we are using the distance between the Koopman embeddings to define the cost function for downstream control tasks.

The ultimate training loss is simply the combination of all the terms above: $\mathcal{L} = \mathcal{L}_{\text{ae}} + \mathcal{L}_{\text{pred}} + \mathcal{L}_{\text{metric}}$. We then minimize the loss $\mathcal{L}$ by optimizing the parameters in the graph encoder ϕ and graph decoder ψ using stochastic gradient descent. Once the model is trained, it can be used for system identification, future prediction, and control synthesis.

E System identification

For a sequence of observations $\widetilde{\boldsymbol{x}} = [\boldsymbol{x}^1, \cdots, \boldsymbol{x}^T]$ from time 1 to time T, we first map them to the Koopman space as $\widetilde{\boldsymbol{g}} = [\boldsymbol{g}^1, \cdots, \boldsymbol{g}^T]$ using the graph encoder ϕ, where $\boldsymbol{g}^t = \phi(\boldsymbol{x}^t)$. We use $\boldsymbol{g}^{i:j}$ to denote the sub-sequence $[\boldsymbol{g}^i, \cdots, \boldsymbol{g}^j]$. To identify the Koopman matrix, we solve the linear equation $\min_K \|K\boldsymbol{g}^{1:T-1} - \boldsymbol{g}^{2:T}\|_2$. As a result, $K = \boldsymbol{g}^{2:T}(\boldsymbol{g}^{1:T-1})^\dagger$ will asymptotically approach the Koopman operator $\mathcal{K}$ with an increasing T. For cases where there are control inputs $\widetilde{\boldsymbol{u}} = [\boldsymbol{u}^1, \cdots, \boldsymbol{u}^{T-1}]$, the calculation of the Koopman matrix and the control matrix is essentially solving a least square problem w.r.t. the objective

$$\min_{K,L} \|K\boldsymbol{g}^{1:T-1} + L\widetilde{\boldsymbol{u}} - \boldsymbol{g}^{2:T}\|_2. \tag{4}$$

As we mentioned in the Section 2.1, the dimension of the Koopman space is linear to the number of objects in the system, i.e., $\widetilde{\boldsymbol{g}} \in \mathbb{R}^{Nm\times T}$ and $K \in \mathbb{R}^{Nm\times Nm}$. If we do not enforce any structure on the Koopman matrix K, we will have to identify N^2m^2 parameters. Instead, we can significantly reduce the number by leveraging the assumption on the structure of K. Assume we know some blocks ($\{K_{ij}\}$) of the matrix K are shared and in total there are h different kinds of blocks, which we denote as $\hat{K} \in \mathbb{R}^{h\times m\times m}$. Then, the number of parameter to be identified reduce to hm^2. Usually, h is much smaller than N^2. Now, for each block K_{ij}, we have a one-hot vector $\sigma_{ij} \in \mathbb{R}^h$ indicating its type, i.e. $K_{ij} = \sigma_{ij}\hat{K}$. Finally, as shown in equation 5, we represent the Koopman matrix as the tensor product of the index tensor σ and the parameter tensor $\hat{K}$:

$$K = \sigma \otimes \hat{K} = \begin{bmatrix} \sigma_{11}\hat{K} & \cdots & \sigma_{1N}\hat{K} \\ \vdots & \ddots & \vdots \\ \sigma_{N1}\hat{K} & \cdots & \sigma_{NN}\hat{K} \end{bmatrix}, \text{where} \quad \sigma = \begin{bmatrix} \sigma_{11} & \cdots & \sigma_{1N} \\ \vdots & \ddots & \vdots \\ \sigma_{N1} & \cdots & \sigma_{NN} \end{bmatrix} \in \mathbb{R}^{N\times N\times h}. \tag{5}$$

Similarly, we assume that the control matrix L has the same block structure as the Koopman matrix and denote its parameter as $\hat{L} \in \mathbb{R}^{h\times N\times|\boldsymbol{u}_i|}$. The least square problem of identifying $\hat{K}$ and $\hat{L}$ becomes

$$\min_{\hat{K},\hat{L}} \|(\sigma \otimes \hat{K})\boldsymbol{g}^{1:T-1} + (\sigma \otimes \hat{L})\widetilde{\boldsymbol{u}} - \boldsymbol{g}^{2:T}\|_2. \tag{6}$$

Since the linear least square problems described in equation 4 and equation 6 have analytical solutions, we have a very efficient system identification method.

F Control Synthesis

During inference, a small amount of the historical data will be used for system identification. We first use the graph encoder to map the system states to the Koopman space. We then identify the Koopman matrix and control matrix by solving the least square problem (equation 4 or equation 6).

For a control task, the goal is to synthesize a sequence of control inputs $\boldsymbol{u}^{1:T}$ that minimize $C = \sum_{t=1}^T c_t(\boldsymbol{x}^t, \boldsymbol{u}^t)$, the total incurred cost, where $c_t(\boldsymbol{x}^t, \boldsymbol{u}^t)$ is the instantaneous cost. For example, considering the control task of reaching a desired state $\boldsymbol{x}^*$ at time T, we can design the following instantaneous cost, $c_t(\boldsymbol{x}^t, \boldsymbol{u}^t) = \mathbb{1}_{[t=T]}(\|\boldsymbol{x}^t - \boldsymbol{x}^*\|_2^2 + \lambda\|\boldsymbol{u}^t\|_2^2)$. The first cost term promotes the control sequence that makes the system state close to the goal, while the second term regularizes the value of the control signals.

Open-loop control via quadratic programming (QP). Our model maps the original nonlinear dynamics to a linear dynamical system [5]. Thus, we can solve the control task by solving a linear control problem. With the assumption that the Koopman embeddings preserve the distance measure, we define the control cost as $c_t(\boldsymbol{g}^t, \boldsymbol{u}^t) = \mathbb{1}_{[t=T]}(\|\boldsymbol{g}^t - \boldsymbol{g}^*\|_2^2 + \lambda\|\boldsymbol{u}^t\|_2^2)$. Basically, we reduce the problem to minimizing a quadratic cost function $C = \sum_{t=1}^T c_t(\boldsymbol{g}^t, \boldsymbol{u}^t)$ over variables $\{\boldsymbol{g}^t, \boldsymbol{u}^t\}_{t=1}^T$ under linear constrains $\boldsymbol{g}^{t+1} = K\boldsymbol{g}^t + L\boldsymbol{u}^t$, where $\boldsymbol{g}^1 = \phi(\boldsymbol{x}^1)$ and $\boldsymbol{g}^* = \phi(\boldsymbol{x}^*)$. It is exactly a Quadratic Programming problem, which can be solved very efficiently.

Model predictive control (MPC). Solving the QP gives us control signals that are open-loop, which might not be good enough for long-term control as the prediction error accumulates. We can combine it with Model Predictive Control, assuming feedback from the environment every τ steps.

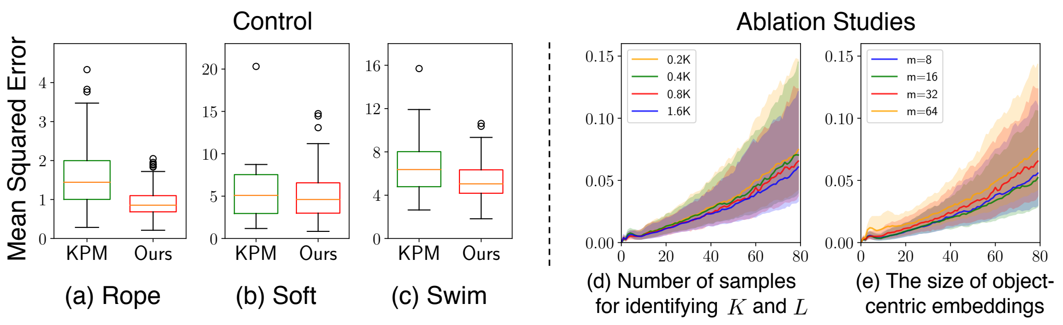

Figure 4: **Quantitative results on control and ablation studies on model hyperparameters.** Left: box-plots show the distributions of control errors. The yellow line in the box indicates the median. Our model consistently achieves smaller errors in all environments against KPM. Right: our model's simulation errors with different amount of data for system identification (d) and different dimensions of the Koopman space (e).

G Data Generation

We generate 10,000 samples for Rope and 50,000 samples for Soft and Swim. Amount them, 90% are used for training and the rest for testing. Each data sample has 100 time steps.

H Training and Evaluation Protocols

Our model is trained on the sub-sequence of length 64 from the training set. For evaluation, we use two metrics: **simulation error** and **control error**. For a given data sample, the simulation error at time step t is defined as the mean squared error between the model prediction $\hat{\boldsymbol{x}}^{t+1}$ and the ground truth $\boldsymbol{x}^{t+1}$. For control, we pick the t_0'th frame $\boldsymbol{x}^{t_0}$ and the t_0+t'th frame $\boldsymbol{x}^{t_0+t}$ from a data sample. Then we ask the model to generate a control sequence of length t to transfer the system from the initial state $\boldsymbol{x}^{t_0}$ to the target state $\boldsymbol{x}^{t_0+t}$. The control error at time step t is defined as the mean squared error between the target state and the state of the system after executing t steps of the generated control signals. For our experiments we have $t = 64$.

I Baselines

We compare our model to following baselines: Interaction Networks [4] (**IN**), Propagation Networks [16] (**PN**) and Koopman method with hand-crafted Koopman base functions (**KPM**). IN and PN are the state-of-the-art learning-based physical simulators. For control, we first finetune the parameters in IN and PN to adapt to the test environment of unknown physical parameters. We then apply gradient descent to the control signals by minimizing the distance between the prediction from IN/PN and the target. The generated control sequence is fed to the original simulator to evaluate the performance. Similar to our method, KPM fits a linear dynamics in the Koopman space. Instead of learning Koopman observations from data, KPM uses polynomials of the original states as the basis functions. In our setting, we set the maximum order of the polynomials to be three to make the dimension of the hand-crafted Koopman embeddings match our model's.

J Ablation Study

Structure of the Koopman matrix. We explore three different structures of the Koopman matrix, *Block*, *Diag* and *None*, to understand its effect on the learned dynamics. *None* assumes no structure in the Koopman matrix. *Diag* assumes a diagonal block structure of K: all off-diagonal blocks (K_{ij} where $i \neq j$) are zeros and all diagonal blocks share the same values. *Block* predefines a block-wise structure, decided by the relation between the objects as introduced in Section 2.2.

Table 1 includes our model's simulation error and control error with different Koopman matrix structures in Rope. Besides the result on the validation set, we also report models' extrapolation ability in parentheses, where the model is evaluated on systems with more objects than those in training: each rope has 5 to 9 objects

Table 1: Ablation study results on the Koopman matrix structure (Rope environment).

	SIMULATION	CONTROL
DIAG	0.052 (0.075)	2.337 (2.809)
NONE	0.056 (0.043)	1.522 (1.288)
BLOCK	**0.046 (0.041)**	**0.854 (1.101)**

in validation, while for extrapolation, each rope has 10
to 14 objects.

Our model with *Block* structure consistently achieves a smaller error in all settings. *Diag* assumes an oversimplified structure, leading to significantly larger errors and failing to make sensible controls. *None* has comparable simulation errors but large control errors. Without structure in the Koopman matrix, it overfits the data and makes the resulting linear dynamics less amiable to the control.

Hyperparameters. In our main experiments, the dimension of the Koopman embedding is set to $m = 32$ per object. Online system identification requires 800 data samples for each training/test case. To understand our model's robustness under different hyperparameters, we vary the dimension of the Koopman embedding from 8 to 64 and the number of data samples used for system identification from 200 to $1{,}600$. Figure 4d shows that the model gives better simulation results when identifying the system with more data samples. Figure 4f shows that dimension 16 gives the best results on simulation. It may suggest that the intrinsic dimension of the Koopman invariant space of the Rope system is around 16 per object.