Generalizing Graph ODE for Learning Complex System Dynamics across Environments

Zijie Huang
University of California, Los Angeles
Los Angeles, CA
zijiehuang@cs.ucla.edu

Yizhou Sun
University of California, Los Angeles
Los Angeles, CA
yzsun@cs.ucla.edu

Wei Wang
University of California, Los Angeles
Los Angeles, CA
weiwang@cs.ucla.edu

ABSTRACT

Learning multi-agent system dynamics has been extensively studied for various real-world applications, such as molecular dynamics in biology, multi-body system in physics, and particle dynamics in material science. Most of the existing models are built to learn single system dynamics, which learn the dynamics from observed historical data and predict the future trajectory. In practice, however, we might observe multiple systems that are generated across different environments, which differ in latent exogenous factors such as temperature and gravity. One simple solution is to learn multiple environment-specific models, but it fails to exploit the potential commonalities among the dynamics across environments and offers poor prediction results where per-environment data is sparse or limited. Here, we present GG-ODE (**G**eneralized **G**raph **O**rdinary **D**ifferential **E**quations), a machine learning framework for learning continuous multi-agent system dynamics across environments. Our model learns system dynamics using neural ordinary differential equations (ODE) parameterized by Graph Neural Networks (GNNs) to capture the continuous interaction among agents. We achieve the model generalization by assuming the dynamics across different environments are governed by common physics laws that can be captured via learning a shared ODE function. The distinct latent exogenous factors learned for each environment are incorporated into the ODE function to account for their differences. To improve model performance, we additionally design two regularization losses to (1) enforce the orthogonality between the learned initial states and exogenous factors via mutual information minimization; and (2) reduce the temporal variance of learned exogenous factors within the same system via contrastive learning. Experiments over various physical simulations show that our model can accurately predict system dynamics, especially in the long range, and can generalize well to new systems with few observations.

CCS CONCEPTS

• **Information systems** → **Physical data models**; • **Computing methodologies** → **Spatial and physical reasoning**.

KEYWORDS

Graph Neural Networks; Neural ODE; Dynamical Systems; Representation Learning

KDD '23, August 6–10, 2023, Long Beach, CA, USA
© 2023 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0103-0/23/08.
https://doi.org/10.1145/3580305.3599362

ACM Reference Format:
Zijie Huang, Yizhou Sun, and Wei Wang. 2023. Generalizing Graph ODE for Learning Complex System Dynamics across Environments . In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '23), August 6–10, 2023, Long Beach, CA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3580305.3599362

1 INTRODUCTION

Building a simulator that can understand and predict multi-agent system dynamics is a crucial research topic spanning over a variety of domains such as planning and control in robotics [26], where the goal is to generate future trajectories of agents based on what has been seen in the past. Traditional simulators can be very expensive to create and use [35] as it requires sufficient domain knowledge and tremendous computational resources to generate high-quality results[1]. Therefore, learning a neural-based simulator directly from data that can approximate the behavior of traditional simulators becomes an attractive alternative.

As the trajectories of agents are usually coupled with each other and co-evolve along with the time, existing studies on learning system dynamics from data usually view the system as a graph and employ Graph Neural Networks (GNNs) to approximate pair-wise node (agent) interaction to impose strong inductive bias [6]. As a pioneering work, Interaction Networks (IN) [6] decompose the system into distinct objects and relations, and learn to reason about the consequences of their interactions and dynamics. Later work incorporates domain knowledge [27], graph structure variances [31], and equivariant representation learning [11, 38] into learning from discrete GNNs, achieving state-of-the-art performance in various domains including mesh-based physical simulation [31] and molecular prediction [10]. However, these *discrete* models usually suffer from low accuracy in long-range predictions as (1) they approximate the system by discretizing observations into some fixed timestamps and are trained to make a single forward-step prediction and (2) their discrete nature fails to adequately capture systems that are continuous in nature such as the spread of COVID-19 [19] and the movements of an n-body system [18, 24].

Recently, researchers propose to combine ordinary differential equations (ODEs) - the principled way for modeling dynamical systems in a *continuous* manner in the past, with GNNs to learn continuous-time dynamics on complex networks in a data-driven way [18, 19, 47]. These Graph-ODE methods have demonstrated the power of capturing long-range dynamics, and are capable of learning from irregular-sampled partial observations [18]. They usually assume all the data are generated from one single system,

[1] To date, out of the 10 most powerful supercomputers in the world, 9 of them are used for simulations, spanning the fields of cosmology, geophysics and fluid dynamics [1]

and the goal is to learn the system dynamics from historical trajectories to predict the future. In practice, however, we might observe data that are generated from multiple systems, which can differ in their environments. For example, we may observe particle trajectories from systems that are with different temperatures, which we call exogenous factors. These exogenous factors can span over a wide range of settings such as particle mass, gravity, and temperature [3, 4, 35] across environments. One simple solution is to learn multiple environment-specific models, but it can fail to exploit the potential commonalities across environments and make accurate predictions for environments with sparse or zero observations. In many useful contexts, the dynamics in multiple environments share some similarities, yet being distinct reflected by the (substantial) differences in the observed trajectories. For example, considering the movements of water particles within multiple containers of varying shapes, the trajectories are driven by both the shared pairwise physical interaction among particles (i.e. fluid dynamics) and the different shapes of the containers where collisions can happen when particles hit the boundaries. Also, the computational cost for training multiple environment-specific models would be huge. More challengingly, the exogenous factors within each environment can be latent, such as we only know the water trajectories are from different containers, without knowing the exact shape for each of them. Therefore, how to learn a single efficient model that can generalize across environments by considering both their commonalities and the distinct effect of per-environment latent exogenous factors remains unsolved. This model, if developed, may help us predict dynamics for systems under new environments with very few observed trajectories.

Inspired by these observations, in this paper, we propose Generalized Graph ODE (GG-ODE), a general-purpose continuous neural simulator that learns multi-agent system dynamics across environments. Our key idea is to assume the dynamics across environments are governed by common physics laws that can be captured via learning a shared ODE function. We introduce in the ODE function a learnable vector representing the distinct latent exogenous factors for each environment to account for their differences. We learn the representations for the latent exogenous factors from systems' historical trajectories through an encoder by optimizing the prediction goal. In this way, different environments share the same ODE function framework while incorporating environment-specific factors in the ODE function to distinguish them.

However, there are two main challenges in learning such latent exogenous factor representations. Firstly, since both the latent initial states for agents and the latent exogenous factors are learned through the historical trajectory data, how can we differentiate them to guarantee they have different semantic meanings? Secondly, when inferring from different time windows from the same trajectory, how can we guarantee the learned exogenous factors are for the same environment?

Towards the first challenge, we enforce the orthogonality between the initial state encoder and the exogenous factor encoder via mutual information minimization. For the second challenge, we reduce the variance of learned exogenous factors within the same environment via a contrastive learning loss. We train our model in a multi-task learning paradigm where we mix the training data from multiple systems with different environments. In this way, the model is expected to fast adapt to other unseen systems with a few data points. We conduct extensive experiments over a wide range of physical systems, which show that our GG-ODE is able to accurately predict system dynamics, especially in the long range.

The main contributions of this paper are summarized as follows:

- We investigate the problem of learning continuous multi-agent system dynamics across environments. We propose a novel framework, known as GG-ODE, which describes the dynamics for each system with a shared ODE function and an environment-specific vector for the latent exogenous factors to capture the commonalities and discrepancies across environments respectively.
- We design two regularization losses to guide the learning process of the latent exogenous factors, which is crucial for making precise predictions in the future.
- Extensive experiments verify the effectiveness of GG-ODE to accurately predict system dynamics, especially in the long range prediction tasks. GG-ODE also generalizes well to unseen or low-resource systems that have very few training samples.

2 PROBLEM DEFINITION

We aim to build a neural simulator to learn continuous multi-agent system dynamics automatically from data that can be generalized across environments. Throughout this paper, we use boldface uppercase letters to denote matrices or vectors, and regular lowercase letters to represent the values of variables.

We consider a multi-agent dynamical system of N interacting agents as an evolving interaction graph $\mathcal{G}^t = \{\mathcal{V}, \mathcal{E}^t\}$, where nodes are agents and edges are interactions between agents that can change over time. For each dynamical system, we denote $e \in E$ as the environment from which the data is acquired. We denote $\boldsymbol{X}^{t,e} \in \mathcal{X}$ as the feature matrix for all N agents and $\boldsymbol{x}_i^{t,e}$ as the feature vector of agent i at time t under environment e. The edges between agents are assigned if two agents are within a connectivity radius R based on their current locations $\boldsymbol{p}_i^{t,e}$ which is part of the node feature vector, i.e. $\boldsymbol{p}_i^{t,e} \in \boldsymbol{x}_i^{t,e}$. They reflect the local interactions of agents and the radius is kept constant over time [35].

Our model input consists of the trajectories of N agents over K timestamps $X^{t_{1:K},e} = \{\boldsymbol{X}^{t_1,e}, \boldsymbol{X}^{t_2,e}, \ldots, \boldsymbol{X}^{t_K,e}\}$, where the timestamps $t_1, t_2 \cdots t_K$ can have non-uniform intervals and be of any continuous values. Our goal is to learn a generalized simulator $s_\theta : X^{t_{1:K},e} \longrightarrow Y^{t_{K+1:T},e}$ that predicts node dynamics in the future for any environment e. Here $\boldsymbol{Y}^{t,e} \in \mathcal{Y}$ represents the targeted node dynamic information at time t, and can be a subset of the input features. We use $\boldsymbol{y}_i^{t,e}$ to denote the targeted node dynamic vector of agent i at time t under environment e.

3 PRELIMINARIES AND RELATED WORK

3.1 Dynamical System Simulations with Graph Neural Networks (GNNs)

Graph Neural Networks (GNNs) are a class of neural networks that operate on graph-structured data by passing local messages[17, 19, 25, 41, 46]. They have been extensively employed in various applications such as node classification [45, 48], link prediction [7, 32], and recommendation systems [14, 20, 42, 43]. By viewing each agent as a node and interaction among agents as edges, GNNs have shown to be efficient for approximating pair-wise node interactions and achieved accurate predictions for multi-agent dynamical systems [8, 24, 35]. The majority of existing studies propose discrete GNN-based simulators where they take the node features at time t as input to predict the node features at time t+1. To further capture the long-term temporal dependency for predicting future trajectories, some work utilizes recurrent neural networks such as RNN, LSTM or self-attention mechanism to make prediction at time t +1 based on the historical trajectory sequence within a time window [12, 13, 16, 36]. However, they all restrict themselves to learn a one-step state transition function. Therefore, when successively apply these one-step simulators to previous predictions in order to generate the rollout trajectories, error accumulates and impairs the prediction accuracy, especially for long-range prediction. Also, when applying most discrete GNNs to learn over multiple systems under different dynamical laws (environments), they usually retrain the GNNs individually for dealing with each specific system environment [24, 35], which yields a large computational cost.

3.2 Ordinary Differential Equations (ODEs) for Multi-agent Dynamical Systems

The dynamic nature of a multi-agent system can be captured by a series of nonlinear first-order ordinary differential equations (ODEs), which describe the co-evolution of states for a set of N dependent variables (agents) over continuous time $t \in \mathbb{R}$ as [5, 33]: $\dot{z}_i^t := \frac{dz_i^t}{dt} = g\left(z_1^t, z_2^t \cdots z_N^t\right)$. Here $z_i^t \in \mathbb{R}^d$ denotes the state variable for agent i at timestamp t and g denotes the ODE function that drives the system move forward. Given the initial states $z_1^0, \cdots z_N^0$ for all agents and the ODE function g, any black box numerical ODE solver such as Runge-Kuttais [29] can solve the ODE initial-value problem (IVP), of which the solution z_i^T can be evaluated at any desired time as shown in Eqn 1.

$$z_i^T = z_i^0 + \int_{t=0}^{T} g\left(z_1^t, z_2^t \cdots z_N^t\right) dt \tag{1}$$

Traditionally, the ODE function g is usually hand-crafted based on some domain knowledge such as in robot motion control [39] and fluid dynamics [28], which is hard to specify without knowing too much about the underlying principles. Even if the exact ODE functions are given, they are usually hard to scale as they require complicated numerical integration [30, 35]. Some recent studies [18, 19, 33] propose to parameterize it with a neural network and learn it in a data-driven way. They combine the expressive power of neural networks along with the principled modeling of ODEs for dynamical systems, which have achieved promising results in various applications [18, 19, 33].

3.3 GraphODE for Dynamical Systems

To model the complex interplay among agents in a dynamical system, researchers have recently proposed to combine ODE with GNNs, which has been shown to achieve superior performance in long-range predictions [18, 19, 47]. In [47], an encoder-processor-decoder architecture is proposed, where an encoder first computes the latent initial states for all agents individually based on their first observations. Then an ODE function parameterized by a GNN predicts the latent trajectories starting from the learned initial states. Finally, a decoder extracts the predicted dynamic features based on a decoding function that takes the predicted latent states as input. Later on, a Graph-ODE framework has been proposed [18, 19] which follows the structure of variational autoencoder [23]. They assume an approximated posterior distribution over the latent initial state for each agent, which is learned based on the whole historical trajectories instead of a single point as in [47]. The encoder computes the approximated posterior distributions for all agents simultaneously considering their mutual influence and then sample the initial states from them. Compared with [47], they are able to achieve better prediction performance, especially in the long range, and are also capable of handling the dynamic evolution of graph structures [19] which is assumed to be static in [47].

We follow a similar framework to this line but aim at generalizing GraphODE to model multiple systems across environments.

4 METHOD

In this section, we present Generalized Graph ODE (GG-ODE) for learning complex system dynamics across environments. As depicted in Figure 1, GG-ODE consists of four main components that are trained jointly: (1) an initial state encoder for inferring the latent initial states for all agents simultaneously; (2) an environment encoder which learns the latent representations for exogenous factors; (3) a generative model defined by a GNN-based ODE function that is shared across environments for modeling the continuous interaction among agents in the latent space. The distinct latent exogenous factors learned for each environment are incorporated into the ODE function to account for their discrepancies, and (4) a decoder that extracts the predicted dynamic features based on a decoding function. We now introduce each component in detail.

4.1 Initial State Encoder

Given the observed trajectories $X^{t_{1:K},e}$, the initial state encoder computes a posterior distribution of latent initial state $q_\phi\left(z_i^{0,e} \mid X^{t_{1:K},e}\right)$ for each agent, from which $z_i^{0,e}$ is sampled. The latent initial state $z_i^{0,e}$ for each agent determines the starting point for the predicted trajectory. We assume the prior distribution $p(z_i^{0,e})$ is a standard normal distribution, and use Kullback–Leibler divergence term in the loss function to add significant regularization towards how the learned distributions look like, which differs VAE from other autoencoder frameworks [19, 24, 33]. In multi-agent dynamical systems, agents are highly-coupled and influence each other. Instead of learning such distribution separately for each agent, such as using an RNN [33] to encode the temporal pattern for each individual trajectory, we compute the posterior distributions for all agents simultaneously (similar to [19]). Specifically, we fuse all trajectories as a whole into a temporal graph to consider both the temporal patterns of individual agents and the mutual interaction among them, where each node is an observation of an agent at a specific timestamp. Two types of edges are constructed, which are (1) spatial edges $\mathcal{V}^t$ that are among observations of interacting agents at each timestamp if the Euclidean distance between the agents' positions $r_{ij}^{t,e} = ||\boldsymbol{p}_i^{t,e} - \boldsymbol{p}_j^{t,e}||_2$ is within a (small) connectivity radius R; and

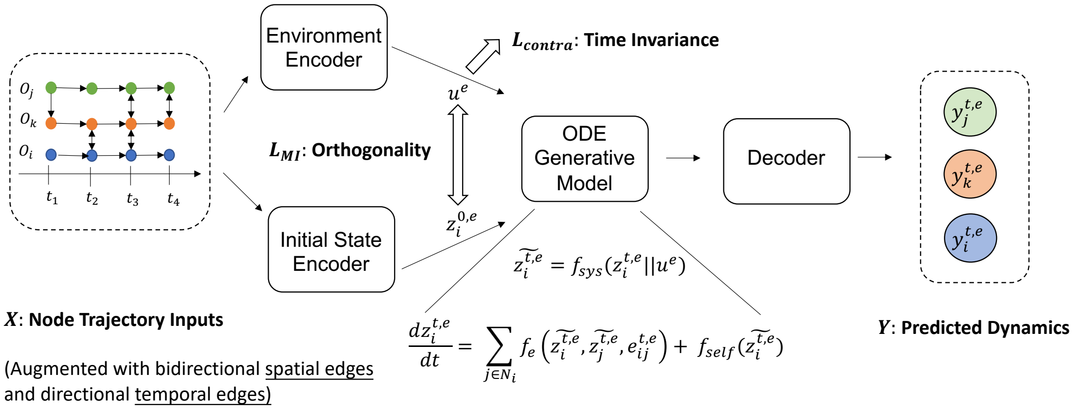

o_j
o_k
o_i
t_1
t_2
t_3
t_4
Environment Encoder
Initial State Encoder
L_{contra}: Time Invariance
u^e
L_{MI}: Orthogonality
$z_i^{0,e}$
ODE Generative Model
Decoder
$y_j^{t,e}$
$y_k^{t,e}$
$y_i^{t,e}$
$\widetilde{z_i^{t,e}} = f_{sys}(z_i^{t,e} || u^e)$
$\frac{dz_i^{t,e}}{dt} = \sum_{j \in N_i} f_e\left(\widetilde{z_i^{t,e}}, \widetilde{z_j^{t,e}}, e_{ij}^{t,e}\right) + f_{self}(\widetilde{z_i^{t,e}})$
X: Node Trajectory Inputs
(Augmented with bidirectional spatial edges and directional temporal edges)
Y: Predicted Dynamics

Figure 1: The overall framework of GG-ODE consists of four modules. First, an initial state encoder computes the latent initial states for all agents simultaneously by constructing a temporal graph from the input trajectories. Additionally, an environment encoder computes the latent representations for exogenous factors that are distinct for each environment. Then, the generative model defined by a GNN-based ODE function calls the solver to output the predicted latent states for agents in the future, where the learned exogenous factors are incorporated into the ODE function. Finally, a decoder generates the predicted dynamics for each agent based on the decoding likelihood determined by the latent states. Two regularization terms are added to preserve the orthogonality of two encoders and the time-invariant property of the environment encoder.

(2) temporal edges that preserve the autoregressive nature of each trajectory, defined between two consecutive observations of the same agent. Note that spatial edges are bidirectional while temporal edges are directional to preserve the autoregressive nature of each trajectory, as shown in Figure 1. Based on the constructed temporal graph, we learn the latent initial states for all agents through a two-step procedure: (1) dynamic node representation learning that learns the representation $\boldsymbol{h}_i^{t,e}$ for each observation node whose feature vector is $\boldsymbol{x}_i^{t,e}$. (2) sequence representation learning that summarizes each observation sequence (trajectory) into a fixed-dimensional vector through a self-attention mechanism.

4.1.1 Dynamic Node Representation Learning. We first conduct dynamic node representation learning on the temporal graph through an attention-based spatial-temporal GNN defined as follows:

$$\boldsymbol{h}_j^{l+1(t,e)} = \boldsymbol{h}_j^{l(t,e)} + \sigma\left(\sum_{i^{(t',e)} \in \mathcal{N}_j^{(t,e)}} \alpha_i^{l(t',e)\to j(t,e)} \times \boldsymbol{W}_v \widehat{\boldsymbol{h}}_i^{l(t',e)}\right)$$

$$\alpha_i^{l(t',e)\to j(t,e)} = \left(\boldsymbol{W}_k \widehat{\boldsymbol{h}}_i^{l(t',e)}\right)^T \left(\boldsymbol{W}_q \boldsymbol{h}_j^{l(t,e)}\right) \cdot \frac{1}{\sqrt{d}} \quad (2)$$

$$\widehat{\boldsymbol{h}}_i^{l(t',e)} = \boldsymbol{h}_i^{l(t',e)} + \mathrm{TE}(t'-t)$$

$$\mathrm{TE}(\Delta t)_{2i} = \sin\left(\frac{\Delta t}{10000^{2i/d}}\right), \ \mathrm{TE}(\Delta t)_{2i+1} = \cos\left(\frac{\Delta t}{10000^{2i/d}}\right)$$

where $\sigma(\cdot)$ is a non-linear activation function; d is the dimension of node embeddings. The node representation is computed as a weighted summation over its neighbors plus residual connection where the attention score is a transformer-based [40] dot-product of node representations by the use of value, key, query projection matrices $\boldsymbol{W}_v, \boldsymbol{W}_k, \boldsymbol{W}_q$. The learned attention scores are normalized via softmax across all neighbors. Here $\boldsymbol{h}_j^{l(t,e)}$ is the representation of agent j at time t in the l-th layer. $\boldsymbol{h}_i^{l(t',e)}$ is the general representation for a neighbor which is connected either by a temporal edge (where $t' < t$ and $i = j$) or a spatial edge (where $t = t'$ and $i \neq j$) to the observation $\boldsymbol{h}_j^{l(t,e)}$. We add temporal encoding [33, 40] to each neighborhood node representation in order to distinguish the message delivered via spatial and temporal edges respectively. Finally, we stack L layers to get the final representation for each observation node as : $\boldsymbol{h}_i^{t,e} = \boldsymbol{h}_i^{L(t,e)}$.

4.1.2 Sequence Representation Learning. We then employ a self-attention mechanism to generate the sequence representation $\boldsymbol{m}_i^e$ for each agent, which is used to compute the mean $\boldsymbol{\mu}_i^{0,e}$ and variance $\boldsymbol{\sigma}_i^{0,e}$ of the approximated posterior distribution of the agent's initial state. Compared with recurrent models such as RNN, LSTM [37], it offers better parallelization for accelerating training speed and in the meanwhile alleviates the vanishing/exploding gradient problem brought by long sequences [36].

We follow [19] and compute the sequence representation $\boldsymbol{m}_i^e$ as a weighted sum of observations for agent i:

$$\boldsymbol{m}_i^e = \frac{1}{K}\sum_t \sigma\left((\boldsymbol{a}_i^e)^T \widehat{\boldsymbol{h}}_i^{t,e} \widehat{\boldsymbol{h}}_i^{t,e}\right), \ \boldsymbol{a}_i^e = \tanh\left(\left(\frac{1}{K}\sum_t \widehat{\boldsymbol{h}}_i^{t,e}\right)\boldsymbol{W}_a\right), \quad (3)$$

where $\boldsymbol{a}_i^e$ is the average of observation representations with a nonlinear transformation $\boldsymbol{W}_a$ and $\widehat{\boldsymbol{h}}_i^{t,e} = \boldsymbol{h}_i^{t,e} + \mathrm{TE}(t)$. K is the number of observations for each trajectory. Then the initial state is drawn from the approximated posterior distribution as:

$$q_\phi\left(z_i^{0,e} \mid X^{t_{1:K},e}\right) = \mathcal{N}\left(\boldsymbol{\mu}_i^{0,e}, \boldsymbol{\sigma}_i^{0,e}\right), \ \boldsymbol{\mu}_i^{0,e}, \boldsymbol{\sigma}_i^{0,e} = f_{\mathrm{trans}}\left(\boldsymbol{m}_i^e\right)$$
$$z_i^{0,e} \sim p\left(z_i^{0,e}\right) \approx q_\phi\left(z_i^{0,e} \mid X^{t_{1:K},e}\right) \quad (4)$$

where f_{trans} is a simple Multilayer Perceptron (MLP) whose output vector is equally split into two halves to represent the mean and variance respectively.

4.2 Environment Encoder

The dynamic nature of a multi-agent system can be largely affected by some exogenous factors from its environment such as gravity, temperature, etc. These exogenous factors can span over a wide range of settings and are sometimes latent and not observable. To make our model generalize across environments, we design an environment encoder to learn the effect of the exogenous factors automatically from data to account for the discrepancies across environments. Specifically, we use the environment encoder to learn the representations of exogenous factors from observed trajectories and then incorporate the learned vector into the ODE function which is shared across environments and defines how the system evolves over time. In this way, we use a shared ODE function framework to capture the commonalities across environments while preserving the differences among them with the environment-specific latent representation, to improve model generalization performance. It also allows us to learn the exogenous factors of an unseen environment based on only its leading observations. We now introduce the environment encoder in detail.

The exogenous factors would pose influence on all agents within a system. On the one hand, they will influence the self-evolution of each individual agent. For example, temperatures would affect the velocities of agents. On the other hand, they will influence the pairwise interaction among agents. For example, temperatures would also change the energy when two particles collide with each other. The environment encoder $f_{\text{enc}}^{\text{env}}$ therefore learns the latent representation of exogenous factors $\boldsymbol{u}^e$ by jointly consider the trajectories from all agents, i.e. $f_{\text{enc}}^{\text{env}} : X^{t_{1:K},e} \longrightarrow \boldsymbol{u}^e$. Specifically, we learn an environment-specific latent vector from the aforementioned temporal graph in Sec 4.1 that is constructed from observed trajectories. The temporal graph contains both the information for each individual trajectory and the mutual interaction among agents through temporal and spatial edges. To summarize the whole temporal graph into a vector $\boldsymbol{u}^e$, we attend over the sequence representation $\boldsymbol{m}_i^e$ for each trajectory introduced in Sec 4.1 as:

$$\boldsymbol{u}^e = \frac{1}{N}\sum_i \sigma\left((\boldsymbol{b}^e)^T \boldsymbol{m}_i^e \boldsymbol{m}_i^e\right),\ \boldsymbol{b}^e = \tanh\left(\left(\frac{1}{N}\sum_i \boldsymbol{m}_i^e\right)\boldsymbol{W}_b\right), \quad (5)$$

where $\boldsymbol{W}_b$ is a transformation matrix and the attention weight is computed based on the average sequence representation with nonlinear transformation similar as in Eqn (3). Note that we use different parameters to compute the sequence representation $\boldsymbol{m}_i^e$ as opposed to the initial state encoder. The reason is that the semantic meanings of the two sequence representations are different: one is for the latent initial states and another is for the exogenous factors.

4.2.1 *Time Invariance.*

A desired property of the learned representation for exogenous factors $\boldsymbol{u}^e$ is that it should be time-invariant towards the input trajectory time window. In other words, for the same environment, if we chunk the whole trajectories into several pieces, the inferred representations should be similar to each other as they are describing the same environment.

To achieve this, we design a contrastive learning loss to guide the learning process of the exogenous factors. As shown in Figure 2, we force the learned exogenous factor representations to be similar if they are generated based on the trajectories from the same environment (positive pairs), and to be apart from each other if they are from different environments (negative pairs). Specifically, we define the contrastive leanring loss as follows:

$$\mathcal{L}_{\text{contra}} = -\log \frac{\exp\left(\text{sim}\left(f_{\text{enc}}^{\text{env}}\left(X^{t_1:t_2,e}\right), f_{\text{enc}}^{\text{env}}\left(X^{t_3:t_4,e}\right)\right)/\tau\right)}{\sum_{e'\neq e}\exp\big(\text{sim}\big(f_{\text{enc}}^{\text{env}}\big(X^{t_1:t_2,e}, f_{\text{enc}}^{\text{env}}\big(X^{t_5:t_6,e'}\big)/\tau\big)} \quad (6)$$

where τ is a temperature scalar and $\text{sim}(\cdot,\cdot)$ is cosine similarity between two vectors. Note that the lengths of the observation sequences can vary. The detailed generation process for positive and negative pairs can be found in Appendix A.3.2.

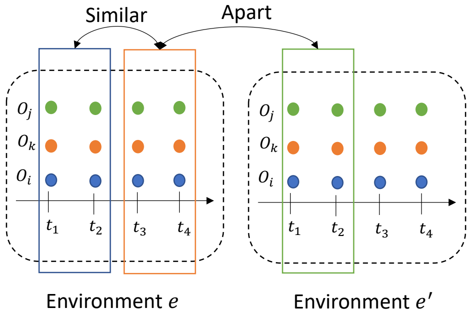


Figure 2: Temporal properties of the environment encoder. We use contrastive learning loss to force the latent exogenous factors learned from different windows within the same environment to be close to each other, and from different environments to be apart from each other.

4.2.2 *Orthogonality.*

GG-ODE features two encoders that take the input of observed trajectories $X^{t_{1:K},e}$ for learning the latent initial states and the latent exogenous factors respectively. As they are designed for different purposes but are both learned from the same input, we disentangle the learned representations from them via a regularization loss defined via mutual information minimization.

Mutual information measures the dependency between two random variables X, Z [49]. Since we are not interested in the exact value of the mutual information, a lower bound derived from Jensen Shannon Divergence [15] could be formulated as

$$I_{\text{JSD}}(X,Z) = E_{P_{XZ}}[-\text{sp}(-M(x,z))] - E_{P_X P_Z}[\text{sp}(M(x,z))], \quad (7)$$

where $P_X P_Z$ is the product of the marginal distributions and P_{XZ} is the joint distribution. $sp(w) = log(1+e^w)$ and M is a discriminator modeled by a neural network to compute the score for measuring their mutual information.

According to recent literature [15, 34, 49], the sample pair (positive pairs) (x,z) drawn from the joint distribution P_{XZ} are different representations of the same data sample, and the sample pair (negative pairs) drawn from $P_X P_Z$ are different representations from different data samples. We therefore attempt to minimize the mutual information from the two encoders as follows

$$\mathcal{L}_{\text{MI}} = \mathbb{E}_{e \in E, i}[-sp(-\Psi(\boldsymbol{z}_i^{0,e}, \boldsymbol{u}^e))] - \mathbb{E}_{e \in E \times e' \in E, i}[sp(\Psi(\boldsymbol{z}_i^{0,e}, \boldsymbol{u}^{e'}))] \tag{8}$$

where Ψ is a MLP-based discriminator. Specifically, we force the latent initial states $\boldsymbol{z}_i^{0,e}$ for all agents from environment e to be dissimilar to the learned exogenous factors $\boldsymbol{u}^e$. And construct negative pairs by replacing the learned exogenous factors from another environment as $\boldsymbol{u}^{e'}$. The generation process for positive and negative pairs can be found in Appendix A.3.2.

4.3 ODE Generative Model and Decoder

4.3.1 ODE Generative Model. After describing the initial state encoder and the environment encoder, we now define the ODE function that drives the system to move forward. The future trajectory of each agent can be determined by two important factors: the potential influence received from its neighbors in the interaction graph and the self-evolution of each agent. For example, in the n-body system, the position of each agent can be affected both by the force from its connected neighbors and its current velocity which can be inferred from its historical trajectories. Therefore, our ODE function consists of two parts: a GNN that captures the continuous interaction among agents and the self-evolution of the node itself. One issue here is how can we decide the neighbors for each agent in the ODE function as the interaction graph is evolving, the neighbors for each agent are dynamically changing based on their current positions, which are implicitly encoded in their latent state representations $\boldsymbol{z}_i^{t,e}, \boldsymbol{z}_j^{t,e}$. We propose to first decode the latent node representations $\boldsymbol{z}_i^{t,e}, \boldsymbol{z}_j^{t,e}$ with a decoding function f_{dec} to obtain their predicted positions $\boldsymbol{p}_i^{t,e}, \boldsymbol{p}_j^{t,e}$ at current timestamp. Then we determine their connectivity based on whether their Euclidean distance $r_{ij}^{t,e} = ||\boldsymbol{p}_i^{t,e} - p_j^{t,e}||_2$ is within the predefined radius R. This can be computed efficiently by using a multi-dimensional index structure such as the *k-d* tree. The decoding function f_{dec} is the same one that we will use in the decoder.

To incorporate the influence of exogenous factors, we further incorporate $\boldsymbol{u}^e$ into the general ODE function to improve model generalization ability as:

$$\begin{aligned} \frac{d\boldsymbol{z}_i^{t,e}}{dt} = g\left(\boldsymbol{z}_1^{t,e}, \boldsymbol{z}_2^{t,e} \cdots \boldsymbol{z}_N^{t,e}\right) &= \sum_{j \in \mathcal{N}_i} f_{\text{GNN}}(\widetilde{\boldsymbol{z}}_i^{t,e}, \widetilde{\boldsymbol{z}}_j^{t,e}) + f_{\text{self}}(\widetilde{\boldsymbol{z}}_i^{t,e}) \\ \widetilde{\boldsymbol{z}}_i^{t,e} &= f_{\text{env}}(\boldsymbol{z}_i^{t,e} || \boldsymbol{u}^e) \end{aligned} \tag{9}$$

where || denotes concatenation and f_{GNN} can be any GNN that conducts message passing among agents. $f_{\text{self}}, f_{\text{env}}$ are implemented as two MLPs respectively. In this way, we learn the effect of latent exogenous factors from data without supervision where the latent representation $\boldsymbol{u}^e$ is trained end-to-end by optimizing the prediction loss.

4.3.2 Decoder. Given the ODE function g and agents' initial states $\boldsymbol{z}_i^{0,e}$ for $i = 1, 2 \cdots N$, the latent trajectories for all agents are determined, which can be solved via any black-box ODE solver. Finally, a decoder generates the predicted dynamic features based on the decoding probability $p(\boldsymbol{y}_i^{t,e} | \boldsymbol{z}_i^{t,e})$ computed from the decoding function f_{dec} as shown in Eqn 10. We implement f_{dec} as a simple two-layer MLP with nonlinear activation. It outputs the mean of the normal distribution $p(\boldsymbol{y}_i^{t,e} | \boldsymbol{z}_i^{t,e})$, which we treat as the predicted value for each agent.

$$\begin{aligned} \boldsymbol{z}_i^{t_1,e} \cdots \boldsymbol{z}_i^{t_T,e} &= \text{ODESolve}(g, [\boldsymbol{z}_1^{0,e}, \boldsymbol{z}_2^{0,e} \cdots \boldsymbol{z}_N^{0,e}], (t_1 \cdots t_T)) \\ \boldsymbol{y}_i^{t,e} &\sim p(\boldsymbol{y}_i^{t,e} | \boldsymbol{z}_i^{t,e}) = f_{\text{dec}}(\boldsymbol{z}_i^{t,e}) \end{aligned} \tag{10}$$

4.4 Training

We now introduce the overall training procedure of GG-ODE . For each training sample, we split it into two halves along the time, where we condition on the first half $[t_1, t_K]$ in order to predict dynamics in the second half $[t_{K+1}, t_T]$. Given the observed trajectories $X^{t_{1:K},e}$, we first run the initial state encoder to compute the latent initial state $\boldsymbol{z}_i^{0,e}$ for each agent, which is sampled from the approximated posterior distribution $q_\phi\left(\boldsymbol{z}_i^{0,e} \mid X^{t_{1:K},e}\right)$. We then generate the latent representations of exogenous factors $\boldsymbol{u}^e$ from the environment e via the environment encoder. Next, we run the ODE generative model that incorporates the latent exogenous factors to compute the latent states for all agents in the future. Finally, the decoder outputs the predicted dynamics for each agent.

We jointly train the encoders, ODE generative model, and decoder in an end-to-end manner. The loss function consists of three parts: (1) the evidence lower bound (ELBO) which is the addition of the reconstruction loss for node trajectories and the KL divergence term for adding regularization to the inferred latent initial states for all agents. We use $Z^{0,e}$ to denote the latent initial state matrix of all N agents. The standard VAE framework is trained to maximize ELBO so we take the negative as the ELBO loss; (2) the contrastive learning loss for preserving the time invariance properties of the learned exogenous factors; (3) the mutual information loss that disentangles the learned representations from the two encoders. λ_1, λ_2 are two hyperparameters for balancing the three terms. We summarize the whole procedure in Appendix A.4.

$$\mathcal{L} = \mathcal{L}_{\text{ELBO}} + \lambda_1 \mathcal{L}_{\text{contra}} + \lambda_2 \mathcal{L}_{MI} \tag{11}$$

$$\begin{aligned} \mathcal{L}_{\text{ELBO}(\theta,\phi)} = &-\mathbb{E}_{Z^{0,e} \sim \prod_{i=1}^N q_\phi\left(\boldsymbol{z}_i^{0,e} | X^{t_{1:K},e}\right)}[\log p_\theta(Y^{t_{K+1:T},e})] \\ &+\text{KL}[\prod_{i=1}^N q_\phi(\boldsymbol{z}_i^{0,e} | X^{t_{1:K},e}) \| p(Z^{0,e})] \end{aligned} \tag{12}$$

5 EXPERIMENTS

5.1 Experiment Setup

5.1.1 Datasets. We illustrate the performance of our model across two physical simulations that exhibit different system dynamics over time: (1) The Water dataset [35], which describes the fluid dynamics of water within a container. Containers can have different shapes and numbers of ramps with random positions inside them, which we view as different environments. The dataset is simulated using the material point method (MPM), which is suitable for simulating the behavior of interacting, deformable materials such as solids, liquids, gases [2]. For each data sample, the number of

[2] https://en.wikipedia.org/wiki/Material_point_method

Table 1: Mean Square Error (MSE) of rollout trajectories with varying prediction lengths. The transductive setting evaluates the testing sequences whose environments are seen during training. The inductive setting evaluates new systems with unseen environments during training. The best results are bold-faced.

Dataset	Lennard-Jones potential Transductive MSE (10^{-2})			Lennard-Jones potential Inductive MSE (10^{-1})			Water Transductive MSE (10^{-3})			Water Inductive MSE (10^{-2})		
Rollout Percentage	30%	60%	100%	30%	60%	100%	30%	60%	100%	30%	60%	100%
LSTM	6.73	20.69	31.88	1.64	8.82	18.01	4.87	23.09	30.44	1.01	6.72	14.79
NRI	5.83	17.99	28.18	1.33	4.34	13.97	3.87	19.64	26.34	0.83	3.84	10.59
NDCN	5.99	17.54	27.06	1.35	4.27	12.37	3.95	18.76	24.33	0.85	3.79	10.11
CG-ODE	5.43	17.01	26.01	1.32	4.25	12.03	3.41	18.13	23.62	0.80	3.64	9.91
SocialODE	5.62	17.23	26.89	1.34	4.26	12.44	3.68	18.42	23.77	0.84	3.70	10.01
GNS	**5.03**	16.28	25.44	1.28	4.23	11.88	**3.17**	17.88	23.14	0.76	3.45	9.78
GG-ODE	5.18	**16.03**	**24.97**	**1.10**	**3.98**	**10.77**	3.20	**16.94**	**22.58**	**0.63**	**3.11**	**8.02**
-w/o $\mathcal{L}_{\text{contra}}$	5.32	17.03	26.53	1.30	4.25	12.13	3.32	18.03	23.01	0.75	3.58	10.03
-w/o$\mathcal{L}_{MI}$	5.45	17.25	26.11	1.32	4.11	11.76	3.43	18.32	22.95	0.78	3.51	9.88
shared encoders	5.66	17.44	26.79	1.33	4.46	12.22	3.55	18.57	23.55	0.81	3.66	10.08

particles can vary but the trajectory lengths are kept the same as 600. The input node features are 2-D positions of particles, and we calculate the velocities and accelerations as additional node features using finite differences of these positions. The total number of data samples (trajectories) is 1200 and the number of environments is 68, where each environment can have multiple data samples with different particle initializations such as positions, velocities, and accelerations. (2) The Lennard-Jones potential dataset [21], which describes the soft repulsive and attractive interactions between simple atoms and molecules [3]. We generate data samples with different temperatures, which could affect the potential energy preserved within the whole system thus affecting the dynamics. We view temperatures as different environments. The total number of data samples (trajectories) is 6500 and the number of environments is 65. Under each environment, we generate 100 trajectories with different initializations. The trajectory lengths are kept the same as 100. The number of particles is 1000 for all data samples. More details about datasets can be found in Appendix A.1.

5.1.2 *Task Evaluation and Data Split.*

We predict trajectory rollouts across varying lengths and use Mean Square Error (MSE) as the evaluation metric.

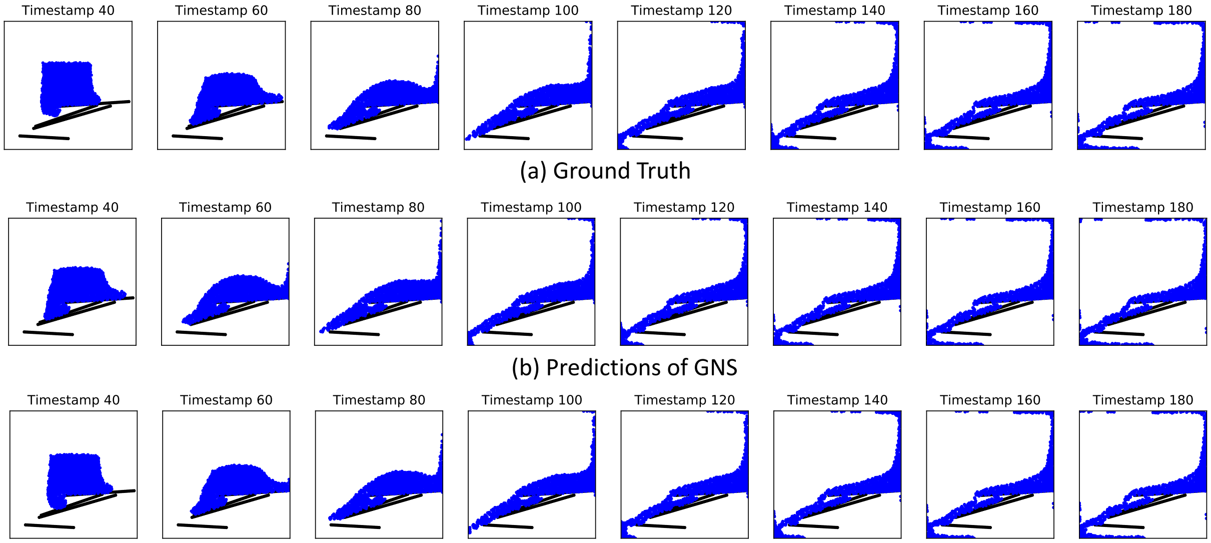
Task Evaluation. The trajectory prediction task is conducted under two settings: (1) Transductive setting, where we evaluate the test sequences whose environments are seen during training; (2) Inductive setting, where we evaluate the test sequences whose environments are not observed during training. It helps to test the model's generalization ability to brand-new systems.

Data Split. We train our model in a sequence-to-sequence setting where we split the trajectory of each training sample into two parts $[t_1, t_K]$ and $[t_{K+1}, t_T]$. We condition on the first part of observations to predict the second part. To conduct data split, we first randomly select 20% environments whose trajectories are all used to construct the testing set $X_{\text{test}}^{\text{Induct}}$ in the inductive setting. For the remaining trajectories that cover the 80% environments, we randomly split them into three partitions: 80% for the training set X_{train}, 10% for the validation set X_{val} and 10% for the testing set in the transductive setting $X_{\text{test}}^{\text{trans}}$. In other words, we have two test sets for the inductive and transductive settings respectively, one training set and one validation set. To fully utilize the data points within each trajectory, we generate training and validation samples by splitting each trajectory into several chunks that can overlap with each other, using a sliding window. The sliding window has three hyperparameters: the observation length and prediction length for each sample, and the interval between two consecutive chunks (samples). Specifically, for the Water dataset, we set the observation length as 50 and the prediction length as 150. We obtain samples from each trajectory by using a sliding window of size 200 and setting the sliding interval as 50. For the Lennard-Jones potential dataset, we set the observation length as 20, the prediction length as 50, and the interval as 10. The procedure is summarized in Appendix A.1.1. During evaluations for both settings, we ask the model to roll out over the whole trajectories without further splitting, whose prediction lengths are larger than the ones during training. The observation lengths during testing are set as 20 for the Lennard-Jones potential dataset and 50 for the Water dataset across the two settings.

5.2 Baselines

We compare both discrete neural models as well as continuous neural models where they do not have special treatment for modeling the influence from different environments. For discrete ones we choose: NRI [24] which is a discrete GNN model that uses VAE to infer the interaction type among pairs of agents and is trained via one-step predictions; GNS [35], a discrete GNN model that uses multiple rounds of message passing to predict every single step; LSTM [37], a classic recurrent neural network (RNN) that learns the dynamics of each agent independently. For the continuous models, we compare with NDCN [47] and Social ODE [44], two ODE-based methods that follow the encoder-processor-decoder structure with GNN as the ODE function. The initial state for each agent is drawn from a single data point instead of a leading sequence. CG-ODE [19] which has the same architecture as our model, but with two coupled ODE functions to guide the evolution of systems.

[3] https://en.wikipedia.org/wiki/Lennard-Jones_potential



(a) Ground Truth

(b) Predictions of GNS

(c) Predictions of GG-ODE

Figure 3: Visualization of the transductive prediction results for the Water dataset. Black lines are ramps within the container. The length of the observation sequence is set as 20. GNS makes less accurate predictions compared with GG-ODE.

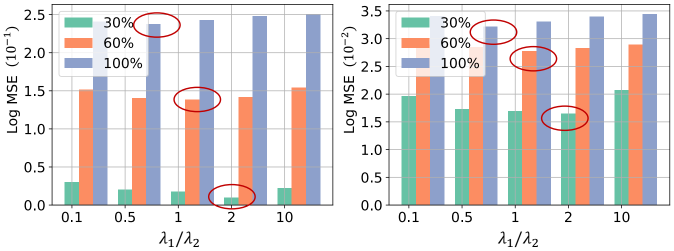
5.3 Performance Evaluation

We evaluate the performance of our model based on Mean Square Error (MSE) as shown in Table 1. As data samples have varying trajectory lengths, we report the MSEs over three rollout percentages regarding different prediction horizons: 30%, 60%, 100% where 100% means the model conditions on the observation sequence and predicts all the remaining timestamps.

Firstly, we can observe that GG-ODE consistently outperforms all baselines across different settings when making long-range predictions, while achieving competitive results when making short-range predictions. This demonstrates the effectiveness of GG-ODE in learning continuous multi-agent system dynamics across environments. By comparing the performance of LSTM with other methods, we can see that modeling the latent interaction among agents can indeed improve the prediction performance compared with predicting trajectories for each agent independently. Also, we can observe the performance gap between GG-ODE and other baselines increase when we generate longer rollouts, showing its expressive power when making long-term predictions. This may be due to the fact that GG-ODE is a continuous model trained in a sequence-to-sequence paradigm whereas discrete GNN methods are only trained to make a fixed-step prediction. Another continuous model NDCN only conditions a single data point to make predictions for the whole trajectory in the future, resulting in suboptimal performance. Finally, we can see that GG-ODE has a larger performance gain over existing methods in the inductive setting than in the transductive setting, which shows its generalization ability to fast adapt to other unseen systems with a few data points. Figure 3 visualizes the prediction results under the transductive setting for the Water dataset.

5.3.1 Ablation Studies. To further analyze the rationality behind our model design, we conduct an ablation study by considering three model variants: (1) We remove the contrastive learning loss which forces the learned exogenous factors to satisfy the time invariance property, denoted as $-w/o\mathcal{L}_{\text{contra}}$; (2) We remove the mutual information minimization loss which reduces the variance of the learned exogenous factors from the same environment, denoted as $-w/o\mathcal{L}_{MI}$. (3) We share the parameters of the two encoders for computing the latent representation $\boldsymbol{m}_i^e$ for each observation sequence in the temporal graph, denoted as shared encoders. As shown in Table 1, all three variants have inferior performance compared to GG-ODE , verifying the rationality of the three key designs. Notably, when making long-range predictions, removing $\mathcal{L}_{MI}$ would cause more harm to the model than removing $\mathcal{L}_{\text{contra}}$. This can be understood as the latent initial states are more important for making short-term predictions, while the disentangled latent initial states and exogenous factors are both important for making long-range predictions.

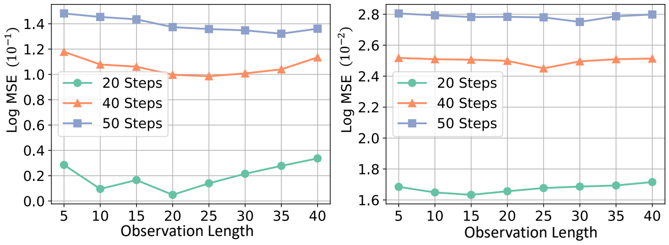
5.3.2 Hyperparameter Study. We study the effect of λ_1/λ_2, which are the hyperparameters for balancing the two regularization terms that guide the learning of the two encoders, towards making predictions under different horizons. As illustrated in Figure 4, the optimal ratio for making 30%, 60%, 100% rollout predictions are 2, 1,0.5 respectively, under both the transductive and inductive settings. They indicate that the exogenous factors modeling plays a more important role in facilitating long-term predictions, which is consistent with the prediction errors illustrated in Table 1 when comparing $-w/o\mathcal{L}_{MI}$ with $-w/o\mathcal{L}_{\text{contra}}$. However, overly elevating $\mathcal{L}_{MI}$ would also harm the model performance, as the time invariance property achieved by $\mathcal{L}_{\text{contra}}$ is also important to guarantee the correctness of the learned latent initial states, which determines the starting point of the predicted trajectories in the future.



(a) Inductive Setting (b) Transductive Setting

Figure 4: Effect of λ_1/λ_2 on the Lennard-Jones potential dataset. Best results are circled in red for each setting.

5.3.3 Sensitivity Analysis. GG-ODE can take arbitrary observation lengths to make trajectory predictions, as opposed to existing baselines that only condition on observations with fixed lengths. It allows the model to fully utilize all the information in the past. We then study the effect of observation lengths on making predictions in different horizons. As shown in Figure 5, the optimal observation lengths for predicting the rollouts with 20, 40, and 50 steps are 20, 25, 35 in the inductive setting, and 15, 25, 30 in the transductive setting. When predicting long-range trajectories, our model typically requires a longer observation sequence to get more accurate results. Also, for making predictions at the same lengths, the inductive setting requires a longer observation length compared with the transductive setting.

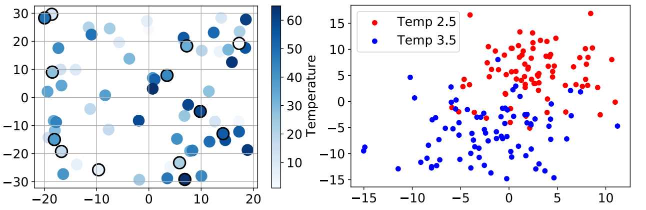


(a) Inductive Setting (b) Transductive Setting

Figure 5: Effect of observation length on the Lennard-Jones potential dataset.

5.4 Case Study

We conduct a case study to examine the learned representations of the latent exogenous factors on the Lennard-Jones potential dataset. We first randomly choose one data sample for each of the 65 temperatures and visualize the learned representations of exogenous factors. As shown in Figure 6 (a), the representations of higher temperatures are closer to each other on the right half of the figure, whereas the lower temperatures are mostly distributed on the left half. Among the 65 temperatures, 20% of them are not seen during training which we circled in black. We can see those unseen temperatures are also properly distributed, indicating the great generalization ability of our model. We next plot the representations for all data samples under temperatures 2.5 and 3.5 respectively as shown in Figure 6 (b). We can see that the learned representations are clustered within the two temperatures, indicating our contrastive learning loss is indeed beneficial to guide the learning process of exogenous factors.



(a) Exogenous Factors Across Environments (b) Exogenous Factors from two Environments

Figure 6: T-SNE visualization of the learned exogenous factors on the Lennard-Jones potential dataset. (a) We randomly pick one data sample per temperature, where temperatures tested in the inductive setting are circled in black. (b) Visualization of data samples from two temperatures.

6 CONCLUSION

In this paper, we investigate the problem of learning the dynamics of continuous interacting systems across environments. We model system dynamics in a continuous fashion through graph neural ordinary differential equations. To achieve model generalization, we learn a shared ODE function that captures the commonalities of the dynamics among environments while design an environment encoder that learns environment-specific representations for exogenous factors automatically from observed trajectories. To disentangle the representations from the initial state encoder and the environment encoder, we propose a regularization loss via mutual information minimization to guide the learning process. We additionally design a contrastive learning loss to reduce the variance of learned exogenous factors across time windows under the same environment. The proposed model is able to achieve accurate predictions for varying physical systems under different environments, especially for long-term predictions. There are some limitations though. Our current model only learns one static environment-specific variable to achieve model generalization. However, the environment can change over time such as temperatures. How to capture the dynamic influence of those evolving environments remain challenging.

ACKNOWLEDGMENTS

This work was partially supported by NSF 1829071, 2031187, 2106859, 2119643, 2200274, 2211557, 1937599, 2303037, NASA, research awards from Amazon, Cisco, NEC, and DARPA #HR00112290103, DARPA #HR0011260656. We would like to thank Mathieu Bauchy, Han Liu and Abhijeet Gangan for their help to the dataset generation procedure and valuable discussion throughout the project.

REFERENCES

[1] 2019. A Sneak Peek at 19 Science Simulations for the Summit Supercomputer in 2019.

[2] Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer Normalization. *CoRR* (2016).

[3] Fabien Baradel, Natalia Neverova, Julien Mille, Greg Mori, and Christian Wolf. 2020. CoPhy: Counterfactual Learning of Physical Dynamics. In *ICLR*.

[4] Fabien Baradel, Natalia Neverova, Julien Mille, Greg Mori, and Christian Wolf. 2020. CoPhy: Counterfactual Learning of Physical Dynamics. In *ICLR*.

[5] Cx K Batchelor and GK Batchelor. 2000. *An introduction to fluid dynamics*. Cambridge university press.

[6] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, and koray kavukcuoglu. 2016. Interaction Networks for Learning about Objects, Relations and Physics. In *Advances in Neural Information Processing Systems 29*. 4502–4510.

[7] Austin Benson and Jon Kleinberg. 2019. Link prediction in networks with core-fringe data. 94–104.

[8] Michael B Chang, Tomer Ullman, Antonio Torralba, and Joshua B Tenenbaum. 2016. A Compositional Object-Based Approach to Learning Physical Dynamics. *ICLR* (2016).

[9] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. 2018. Neural Ordinary Differential Equations. In *Advances in Neural Information Processing Systems 31*. 6571–6583.

[10] Nicholas Gao and Stephan Günnemann. 2022. Ab-Initio Potential Energy Surfaces by Pairing GNNs with Neural Wave Functions. In *International Conference on Learning Representations*. https://openreview.net/forum?id=apv504XsysP

[11] Johannes Gasteiger, Florian Becker, and Stephan Günnemann. 2021. Gemnet: Universal directional graph neural networks for molecules. *Advances in Neural Information Processing Systems* 34 (2021), 6790–6802.

[12] Yupeng Gu, Yizhou Sun, and Jianxi Gao. 2017. The Co-Evolution Model for Social Network Evolving and Opinion Migration. In *KDD'17*.

[13] Ehsan Hajiramezanali, Arman Hasanzadeh, Krishna Narayanan, Nick Duffield, Mingyuan Zhou, and Xiaoning Qian. 2019. Variational Graph Recurrent Neural Networks. In *Advances in Neural Information Processing Systems 32*. 10701–10711.

[14] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. 2020. Lightgcn: Simplifying and powering graph convolution network for recommendation. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*. 639–648.

[15] R Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Phil Bachman, Adam Trischler, and Yoshua Bengio. 2018. Learning deep representations by mutual information estimation and maximization. *arXiv preprint arXiv:1808.06670* (2018).

[16] Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. 2020. Heterogeneous Graph Transformer. In *Proceedings of the 2020 World Wide Web Conference*.

[17] Zijie Huang, Zheng Li, Haoming Jiang, Tianyu Cao, Hanqing Lu, Bing Yin, Karthik Subbian, Yizhou Sun, and Wei Wang. 2022. Multilingual Knowledge Graph Completion with Self-Supervised Adaptive Graph Alignment. In *Annual Meeting of the Association for Computational Linguistics (ACL)*.

[18] Zijie Huang, Yizhou Sun, and Wei Wang. 2020. Learning Continuous System Dynamics from Irregularly-Sampled Partial Observations. In *Advances in Neural Information Processing Systems*.

[19] Zijie Huang, Yizhou Sun, and Wei Wang. 2021. Coupled Graph ODE for Learning Interacting System Dynamics. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*.

[20] Amin Javari, Zhankui He, Zijie Huang, Raj Jeetu, and Kevin Chen-Chuan Chang. 2020. Weakly Supervised Attention for Hashtag Recommendation Using Graph Data. In *Proceedings of The Web Conference 2020 (WWW '20)*. 1038–1048.

[21] John Edward Jones. 1924. On the determination of molecular fields.—I. From the variation of the viscosity of a gas with temperature. *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character* 106, 738 (1924), 441–462.

[22] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. *ICLR* (2015).

[23] Diederik P. Kingma and Max Welling. 2014. Auto-Encoding Variational Bayes.. In *ICLR*.

[24] Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. 2018. Neural Relational Inference for Interacting Systems. *arXiv preprint arXiv:1802.04687* (2018).

[25] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR'17*.

[26] Yunzhu Li, Jiajun Wu, Jun-Yan Zhu, Joshua B Tenenbaum, Antonio Torralba, and Russ Tedrake. 2019. Propagation Networks for Model-Based Control Under Partial Observation. In *ICRA*.

[27] Yunzhu Li, Jiajun Wu, Jun-Yan Zhu, Joshua B Tenenbaum, Antonio Torralba, and Russ Tedrake. 2019. Propagation networks for model-based control under partial observation. In *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 1205–1211.

[28] Harvard Lomax, Thomas H Pulliam, David W Zingg, and TA Kowalewski. 2002. Fundamentals of computational fluid dynamics. *Appl. Mech. Rev.* 55, 4 (2002), B61–B61.

[29] Schober Michael, Särkkä Simo, and Philipp Hennig. 2019. A probabilistic model for the numerical solution of initial value problems. In *Statistics and Computing*. 99–122.

[30] F Mokalled, L Mangani, and M Darwish. 2016. The Finite Volume Method on Computational Fluid Dynamics.

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

[32] Liang Qu, Huaisheng Zhu, Qiqi Duan, and Yuhui Shi. 2020. Continuous-time link prediction via temporal dependent graph neural network. 3026–3032.

[33] Yulia Rubanova, Ricky T. Q. Chen, and David K Duvenaud. 2019. Latent Ordinary Differential Equations for Irregularly-Sampled Time Series. In *Advances in Neural Information Processing Systems 32*. 5320–5330.

[34] Eduardo Hugo Sanchez, Mathieu Serrurier, and Mathias Ortner. 2020. Learning disentangled representations via mutual information estimation. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXII 16*. Springer, 205–221.

[35] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter W. Battaglia. 2020. Learning to Simulate Complex Physics with Graph Networks. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*. 8459–8468.

[36] Aravind Sankar, Yanhong Wu, Liang Gou, Wei Zhang, and Hao Yang. 2020. DySAT: Deep Neural Representation Learning on Dynamic Graphs via Self-Attention Networks. In *WSDM'20*.

[37] Hochreiter Sepp and Schmidhuber Jürgen. 1997. Long Short-term Memory. *Neural computation* (1997).

[38] Chence Shi, Shitong Luo, Minkai Xu, and Jian Tang. 2021. Learning gradient fields for molecular conformation generation. In *International Conference on Machine Learning*. PMLR, 9558–9568.

[39] R.J. Spiteri, D.K. Pai, and U.M. Ascher. 2000. Programming and control of robots by means of differential algebraic inequalities. *IEEE Transactions on Robotics and Automation* 16, 2 (2000), 135–145. https://doi.org/10.1109/70.843168

[40] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems 30*. 5998–6008.

[41] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. *ICLR'18* (2018).

[42] Ruijie Wang, Zijie Huang, Shengzhong Liu, Huajie Shao, Dongxin Liu, Jinyang Li, Tianshi Wang, Dachun Sun, Shuochao Yao, and Tarek Abdelzaher. 2021. DyDiff-VAE: A Dynamic Variational Framework for Information Diffusion Prediction. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '21)*. 163–172.

[43] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. 2019. Neural graph collaborative filtering. In *Proceedings of the international ACM SIGIR conference on Research and development in Information Retrieval*. 165–174.

[44] Song Wen, Hao Wang, and Dimitris Metaxas. 2022. Social ODE: Multi-agent Trajectory Forecasting with Neural Ordinary Differential Equations. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXII*. Springer, 217–233.

[45] Man Wu, Shirui Pan, Chuan Zhou, Xiaojun Chang, and Xingquan Zhu. 2020. Unsupervised domain adaptive graph convolutional networks. 1457–1467.

[46] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks?. In *ICLR'19*.

[47] Chengxi Zang and Fei Wang. 2020. Neural dynamics on complex networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 892–902.

[48] Tianxiang Zhao, Xiang Zhang, and Suhang Wang. 2021. Graphsmote: Imbalanced node classification on graphs with graph neural networks. In *Proceedings of the ACM International Conference on Web Search and Data Mining*. 833–841.

[49] Wei Zhu, Haitian Zheng, Haofu Liao, Weijian Li, and Jiebo Luo. 2021. Learning bias-invariant representation by cross-sample mutual information minimization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 15002–15012.

A APPENDIX

A.1 Datasets

We conduct experiments over two datasets: The Water dataset and the Lennard-Jones potential dataset. As introduced in Sec 2, the edges between agents are assigned if the Euclidean distance between the agents' positions $r_{ij}^{t,e} = ||\boldsymbol{p}_i^{t,e} - \boldsymbol{p}_j^{t,e}||_2$ is within a (small) connectivity radius R. The connectivity radius for the two datasets is set as 0.015 and 2.5 respectively. The number of particles is kept the same as 1000 for all trajectories in the Lennard-Jones potential dataset, while in the Water dataset, each data sample can have a varying number of particles, and the maximum number of particles is 1000.

A.1.1 Data Split. Our model is trained in a sequence-to-sequence mode, where we split the trajectory of each training sample into two parts $[t_1, t_K]$ and $[t_{K+1}, t_T]$. We condition on the first part of observations to predict the second part. To fully utilize the data points within each training sample, we split each trajectory into several chunks with three hyperparameters: the observation length and prediction length for each sample, and the interval between two consecutive chunks (samples). We summarize the procedure in Algorithm 1, where K is the number of trajectories and d is the input feature dimension.

Algorithm 1: Data Splitting Procedure.

```
Input: Original Training trajectories $X_{\text{input}} \in \mathbb{R}^{K \times N \times T \times d}$;
  Observation length $O$; Prediction length $M$; Interval $I$;
  Trajectory length $T$.
Output: Training samples after splitting $X_{\text{train}}$.
sample_length = $O + M$;
num_chunk = ($T$ - sample_length )//interval + 1;
for i in range (0,K) do
    for j in range(0,num_chunk,I) do
        Generate the split training sample as
          $X_{\text{input}}[i, :, j : j + \text{sample\_length}, :]$
        Add the training sample to the training set $X_{\text{train}}$.
    end
end
```

A.1.2 Input Features and Prediction Target. For the Water dataset, the input node features are 2-D positions $p_i^{t,e}$, and we additionally calculate the 2-D velocities and accelerations using finite differences of these positions as $v_i^{t,e} = p_i^{t,e} - p_i^{t-1,e}$, $a_i^t = v_i^{t,e} - v_i^{t-1,e} = p_i^{t,e} - 2p_i^{t-1,e} + p_i^{t-2,e}$. For positions, velocities, and accelerations, we precompute their mean and variance across all samples and normalize them with z-score. For the Lennard-Jones potential dataset, the input node features are 3-D positions, velocities, and accelerations. We train the model to predict the future positions for each agent along the time for both datasets.

A.2 Software and Experiment Environment

We implement our model in PyTorch. All experiments are conducted on a GPU powered by an NVIDIA A100. For all datasets, we train over 100 epochs and select the one with the lowest validation loss as the reported model. We report the average results over 10 runs. Encoders, the generative model, and the decoder are jointly optimized using Adam optimizer [22] with a learning rate 0.005. The batch size for the Water dataset is set as 128, and for the Lennard-Jones potential dataset. is set as 256. Note that the batch size denotes the number of data samples generated as in Alg 1.

A.3 Implementation Details

We now introduce the implementation details of our model.

A.3.1 Initial State Encoder. The initial state encoder aims to infer latent initial states for all agents simultaneously via a two-step procedure: Firstly, the encoder computes the structural representation for each observation node by the use of a spatial-temporal GNN. We set the number of GNN layers l as 2 and the hidden dimension as 64 across all datasets. LayerNorm [2] is employed to provide training stability in our experiment. Next, a self-attention-based sequence representation learning procedure computes the sequence representation for each agent and samples the initial state from it. We use a 2-layer MLP as f_{trans} in Eqn 4 with latent dimensions as 128 and activation function as Tanh.

A.3.2 Environment Encoder. The environment encoder learns the latent representations of exogenous factors based on the observed trajectories. The architecture is the same as the initial state encoder but are using two sets are parameters with the same hyperparameter settings introduced in Sec A.3.1.

Contrastive Learning Loss Sampling. The contrastive learning loss $\mathcal{L}_{\text{contra}}$ shown in Eqn 6 is designed to achieve the time invariance properties of the learned exogenous factors. Specifically, we sample the positive pairs $X^{t_1:t_2,e}, X^{t_3:t_4,e}$ using two strategies: (1) The intra-sample generation, where $^{t_1:t_2,e}, X^{t_3:t_4,e}$ are from the same training sample but representing two different time windows. We achieve this by randomly selecting two timestamps within each training sample to serve as t_1, t_3 respectively, and then set the window size as the observation length L to get $t_2 = t_1 + L, t_4 = t_3 + L$. (2) The cross-sample generation, where $^{t_1:t_2,e}, X^{t_3:t_4,e}$ are from two different samples within the same environment e. Specifically, for each training sample, we first randomly choose another sample under the same environment. Then we generate t_1, t_3 by randomly selecting one timestamp for each of them. Finally, we calculate t_2, t_4 by adding the observation length. To generate negative pair $X^{t_5:t_6,e'}$ for each $X^{t_1:t_2,e}$, we first randomly select one another environment e', from which we randomly pick one data sample. Similarly, we then randomly select one timestamp within that data sample to serve as t_5 and then obtain t_6 as $t_6 = t_5 + L$. The temperature scalar τ in Eqn 6 is set as 0.05.

Mutual Information Minimization Loss Sampling. To disentangle the representations of the latent initial states and the exogenous factors, we design the mutual information minimization loss in Eqn 8 as a regularization term during training. We conduct the sampling procedure for positive and negative pairs as follows: For each training sample, we pair the latent initial states $z_i^{0,e}$ of all the N agents with the learned exogenous factors $\boldsymbol{u}^e$, thus constructing N positive pairs. To generate negative pairs, we randomly select another environment e' and pair it with the latent initial states of all agents within one training sample. Thus we obtain the same number of positive and negative pairs during training. The discriminator Ψ is implemented as a two-layer MLP with hidden dimension and out dimension as 128 and 64 respectively.

A.3.3 ODE Function and Solver. The ODE function introduced in Eqn 1 consists of two parts: the GNN f_{GNN} that captures the mutual interaction among agents and f_{self} that captures the self-evolution of agents. We use the following two-layer message passing GNN function as f_{GNN}:

$$\begin{aligned} v \longrightarrow e : \mathbf{e}_{(i,j)}^{l_1(t,e)} &= f_e^1\left(\left[\mathbf{z}_i^{t,e} || \mathbf{z}_j^{t,e}\right]\right) \\ e \longrightarrow v : \mathbf{z}_j^{l_1(t,e)} &= f_v^1\left(\sum_{i \neq j} \mathbf{e}_{(i,j)}^{l_1(t,e)}\right) \\ v \longrightarrow e : \mathbf{z}_j^{l_2(t,e)} &= f_e^2\left(\left[\mathbf{z}_i^{l_1(t,e)} || \mathbf{z}_j^{l_1(t,e)}\right]\right) \end{aligned} \tag{13}$$

where || denotes concatenation, f_e^1, f_v^1, f_e^2 are two-layer MLPs with hidden dimension size of 64. We use $\mathbf{z}_j^{l_2(t,e)}$ as output representation for agent j at timestamp t from f_{GNN}. The self-evolution function f_{self} and the transformation function f_{env} are also implemented as two-layer MLPs with hidden dimension of 64. We use the fourth-order Runge-Kutta method from torchdiffeq python package [9] as the ODE solver, which solves the ODE systems on a time grid that is five times denser than the observed time points. We also utilize the Adjoint method described in [9] to reduce memory usage.

A.4 Pseudo-Code of GG-ODE Training

Algorithm 2: Generalized Graph ODE training procedure.

```
Input: Observed trajectories $X^{t_{1:K},e}$.
Output: Model parameters $\phi$ and $\theta$.
while model not converged do
    for Each training sample do
        Separate the sequence into observed half $[T_0, T_1]$
          and predicted half $[T_1, T_2]$;
        //For the initial state encoder:
        Generate the latent initial states $z_i^{0,e}$ for each agent
          according to Eqn 4;
        //For the environment encoder:
        Compute the latent representation for exogenous
          factors as in Eqn 5;
        //For the generative model:
        Given the latent initial states, the latent exogenous
          factors, and timestamps to predict $[T_1, T_2]$, solve
          the ODE function in Eqn 9;
        //For the decoder:
        Compute predicted node dynamics based on the
          decoding likelihood $p(\boldsymbol{y}_i^{t,e}|z_i^{t,e})$;
    end
    Update the parameters $\phi$ and $\theta$ by optimizing loss term
      defined in Eqn. 11;
end
```