Coupled Graph ODE for Learning Interacting System Dynamics

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

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

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

Abstract

Many real-world systems such as social networks and moving planets are dynamic in nature, where a set of coupled objects are connected via the interaction graph and exhibit complex behavior along the time. For example, the COVID-19 pandemic can be considered as a dynamical system, where objects represent geographical locations (e.g., states) whose daily confirmed cases of infection evolve over time. Outbreak at one location may influence another location as people travel between these locations, forming a graph. Thus, how to model and predict the complex dynamics for these systems becomes a critical research problem. Existing work on modeling graph-structured data mostly assumes a static setting. How to handle dynamic graphs remains to be further explored. On one hand, features of objects change over time, influenced by the linked objects in the interaction graph. On the other hand, the graph itself can also evolve, where new interactions (links) may form and existing links may drop, which may in turn be affected by the dynamic features of objects. In this paper, we propose *coupled graph ODE*: a novel latent ordinary differential equation (ODE) generative model that learns the coupled dynamics of nodes and edges with a graph neural network (GNN) based ODE in a continuous manner. Our model consists of two coupled ODE functions for modeling the dynamics of edges and nodes based on their latent representations respectively. It employs a novel encoder parameterized by a GNN for inferring the initial states from historical data, which serves as the starting point of the predicted latent trajectories. Experiment results on the COVID-19 dataset and the simulated social network dataset demonstrate the effectiveness of our proposed method.

CCS Concepts

• **Computing methodologies** → **Neural networks**; • **Networks** → **Network dynamics**.

Keywords

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

ACM Reference Format:
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 and Data Mining (KDD '21), August 14–18, 2021, Virtual Event, Singapore.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3447548.3467385

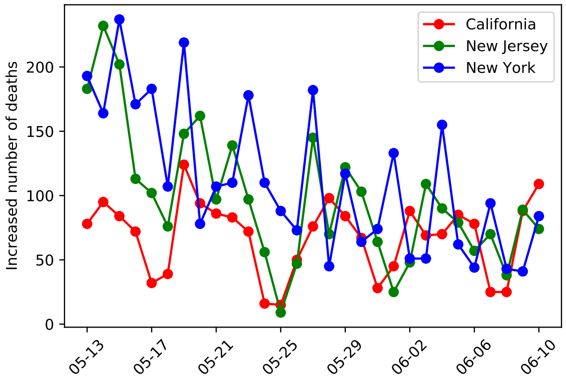


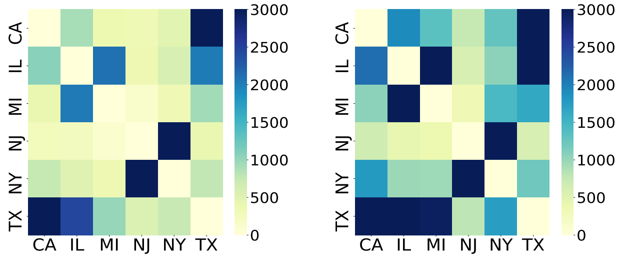
Figure 1: COVID-19 death count time series of three states in U.S. Correlation is higher between two states that have higher population flow.

1 Introduction

Real-world systems in various domains such as physics, biology, robotics can be viewed as dynamic interacting systems, where a set of objects interact with each other and demonstrate complex behavior longitudinally. Learning the underlying dynamics of an interacting system is essential in many real-world applications. For example, learning the movement of robotics can improve planning and control in future design [24]; studying the trajectories of moving planets can discover potential new physical laws [8]; understanding the spread of COVID-19 can help governments develop disease prevention and intervention plans [5], etc. With the recent advances in deep learning techniques, researchers have started building neural-based simulators, aiming to approximate complex system interactions with neural networks [2, 4, 18, 21, 24]. As interacting systems contain multiple objects and are thus graph structured data, existing work [2, 21] usually employs graph neural networks (GNN) to reason how objects interact and to predict object (node) features in the future. However, a fundamental assumption behind a vast majority of work is that the interaction graphs among objects are static [23], such as particles connected by springs where the spring structure remains unchanged. Nonetheless, the dynamic nature of many real-world systems does not only exhibit in the evolution of node features, but may also manifest as the dynamic changes in the graph structure. One example is the spread of COVID-19 within U.S., where nodes are 50 states and the interaction graph represents the population travel patterns between states. Both the daily outbreak statistics (such as the number of new cases of each state) and the mobility patterns between states (such as the number of people traveling from one state to another) evolve over time [5].

KDD '21, August 14–18, 2021, Virtual Event, Singapore

ACM ISBN 978-1-4503-8332-5/21/08.
https://doi.org/10.1145/3447548.3467385



(a) Population flow on May.11 **(b) Population flow on Aug.8**

Figure 2: Population flow in May and August with self-loop flow excluded (Diagonal entries). May has less population flow due to the "close border" policies in many states.

Even though the graph structure and node features are two distinct data representations, they are inherently correlated [23]. On one hand, node features are likely to be affected by other nodes whom they interact with in the graph. In the aforementioned COVID-19 example, New Jersey's daily confirmed cases are more likely to be affected by states with large population inflow (such as New York, Pennsylvania) than by others such as California. This is shown in Figure 1 where the daily death counts of New Jersey is more correlated with that of New York than California. Similar phenomena can also be observed in social networks where individuals are likely to be influenced by their friends [11, 19, 27, 33, 35]. On the other hand, the dynamics of node features may also affect the interaction. For example, the states' severity of the epidemic situation may, in short term, impact the population flow between them as shown in Figure 2. Inspired by these observations, we propose a novel ordinary differential equation (ODE) based generative model: coupled graph ODE, for predicting the dynamics of node features by jointly considering the evolution of nodes and edges.

In order to model the co-evolution of nodes and edges, we design two coupled ODE functions to model the continuous evolution of nodes and edges in the latent space respectively, considering the mutual influence between them. The continuous nature of our model allows it to track the evolution of the underlying system from irregular observations, and is expected to offer improved performance compared to using discrete methods to model a continuous dynamical system such as the spread of COVID-19 [6, 25]. For the edge ODE, the widely-used generative process assumes that the new edges are completely determined by the features of source and target nodes [11, 12]. However, we add an additional term to model the self-evolution of edges. Such self-evolution is widely observed in many real-world systems. For example, the population flow between two states will change naturally due to some seasonal factors (e.g. holidays), which is not necessarily related to the node features (severity of the epidemic situation). Likewise, for the node ODE, we consider the self-evolution of nodes, as well as the potential influence received from neighbors in the interaction graph.

Since we propose to learn continuous system dynamics using ODEs, a fundamental challenge lies in how to estimate the latent initial states for the whole system. We borrow a similar idea from [18, 26] where a VAE-based latent ODE model is proposed to estimate the latent initial states with uncertainty. As objects are highly-coupled in interacting systems, we propose a novel GNN as the encoder which infers the latent initial states for all objects simultaneously. Overall, our model consists of three parts that are jointly trained together: (1) An encoder that infers the latent initial states for all objects and edges simultaneously considering their interaction; (2) A generative model parameterized by two coupled ODEs that learns the evolution pattern for edges and nodes respectively. (3) Two decoders for nodes and edges respectively which project the latent states for nodes and edges to the original input spaces. We conduct extensive experiments on the COVID-19 dataset and one simulated social network dataset. Experiment results verify the effectiveness of our proposed method, especially for long-range predictions. We also conduct case studies on how travel-related policies could affect the number of confirmed cases in the future on the COVID-19 dataset, by adding intervention to the interaction graph, which has demonstrated that our model is a promising tool for policymakers.

2 Problem Formulation

We consider a dynamical system with N interacting objects. Our input consists of the trajectories (features) of these objects and the directed weighted interaction graph among them which changes over time. We denote the snapshots of the interaction graph as $\mathcal{G} = \left\{G^1, G^2, \dots, G^T\right\}$, where $G^t = \left(\mathcal{V}, \mathcal{E}^t\right)$ is the interaction graph at timestamp t with $\mathcal{V}$ denoting the set of N interacting objects and $\mathcal{E}^t$ being the set of directed weighted edges, respectively. For every pair of connected nodes $i, j \in \mathcal{V}$ at timestamp t, $w^t_{i\to j} \in \mathbb{R}$ denotes the weight of the directed edge linking them. The edge weight can be asymmetric, i.e., $w^t_{i\to j} \in \mathbb{R}$ may not necessarily hold the same value as $w^t_{i\to j} \in \mathbb{R}$. We use $\mathcal{A} = \left\{A^1, A^2, \dots, A^T\right\}$ to denote the weighted adjacency matrix sequence.

We denote the node trajectory sequence as $\mathcal{X} = \{X^1, X^2, \dots, X^T\}$, where X^t is the feature matrix of all N objects at timestamp t. We use x^t_i to denote the feature vector of object i at timestamp t. Based on the observed coupled trajectories of a dynamical system, i.e. $\mathcal{X}, \mathcal{A}$, our goal is to learn the underlying dynamics which is built upon the latent representations for nodes $z^t_i \in \mathbb{R}^d$ and edges $z^t_{i\to j} \in \mathbb{R}^d$, and to utilize them to forecast trajectories $X^t (t > T)$ in the future.

3 Related Work and Preliminaries

3.1 Ordinary Differential Equations (ODE) for Multi-agent Dynamical Systems

The dynamic nature of a multi-agent dynamical system can be captured by a series of first-order ordinary differential equations (ODE), which describes the state evolution for a set of M latent dependent variables over continuous time $t \in \mathbb{R}$. Existing work [7, 26] usually associates each object with a latent state variable $z^t_i \in \mathbb{R}^d$, $i = 1, 2 \cdots N$, with the corresponding ODE: $\dot{z}^t_i := \frac{dz^t_i}{dt} = g\left(z^t_1, z^t_2 \cdots z^t_N\right)$, which describes how the trajectory of each object changes over time. The ODE function g is usually hand-crafted by domain experts in the past and some recent studies [18, 26] have proposed to parameterize it as a neural network which can be learned from data. To capture the continuous interaction among objects, graph neural network (GNN) is employed to parameterize the ODE function g in a recent study [18]. Given the latent initial states $z_1^0, \cdots z_N^0 \in \mathbb{R}^d$ for each object, z_i^t is the solution to an ODE initial-value problem (IVP), which can be evaluated at any desired time as shown in Eqn 1 using a numerical ODE solver such as Runge-Kuttais [29]. The latent state z_i^t is further decoded to generate the predicted trajectory at timestamp t: $x_i^t = f_{\text{dec}}(z_i^t)$. Given the ODE function, the latent initial states z_i^0 for each object determine the whole trajectory.

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

However, one major limitation of these methods is that, they assume the interaction graph among agents is static. Therefore, the set of M latent state variables z_i^t are only for nodes, i.e. $M = N$. In reality, the network structure may change over time, which requires the modeling of latent edge state $z_{i\to j}^t$ as well. Moreover, the evolution of latent node and edge states are highly-coupled. Taking the spread of COVID-19 as an example, the number of cases for each state x_i^t can be affected by other states x_j^t via the (past) population flow between them. On the other hand, the (future) population flow between two states may change in response to the varying severity of states' epidemic situation x_i^t, x_j^t.

3.2 Graph Neural Networks (GNN)

GNN is a class of neural networks that operate directly on graph-structured data by passing local messages[22, 32, 38]. It has been widely used for approximating pair-wise object interactions in multi-agent dynamical systems[4, 21].

3.2.1 GNN for Static Graphs The majority of work on GNN mainly focus on static graphs and is designed for tasks such as node classification [22, 32], graph clustering and matching [1], etc. While various architecture exists, the update procedure for a single GNN layer can be characterized by two major operations: (1) Extracting information. For example, graph convolution network (GCN) [22] utilizes the normalized Laplacian as the attention weight for attending each sender node with a linear transformation. It could be regarded as an approximation of spectral domain convolution of the graph signals. (2) Aggregating information from neighbors. Basic aggregation operators including mean, sum and max, while sophisticated pooling and normalization functions are also been proposed. In multi-agent dynamical systems where edges are static and only node attributes evolve, static GNNs are often employed as neural physical simulators to capture the complex interaction among objects, which reveals how system changes from timestamp t to timestamp $t+1$ [2]. However, discrete GNNs may have inferior performance compared with continuous graph ODE-based methods when the system is continuous by nature, such as the spread of COVID-19. They also fail to handle irregularity and partial observations in multi-agent dynamical systems, as opposed to the aforementioned ODE-based methods [18].

3.2.2 GNN for Dynamic Graphs In many real-world applications, both nodes and edges are dynamic such as the traffic network [25]. In order to learn hidden patterns from those dynamic graphs, spatial-temporal GNNs are proposed which is able to consider spatial and temporal dependency at the same time. To achieve this, existing approaches integrate static graph convolutions to capture spatial dependency with RNNs,CNNs or self-attention mechanism to model temporal dependency [12, 21, 28]. The learned node representations can be utilized for downstream tasks such as link prediction [11, 12, 17, 28]. However, they usually assume the new edges are solely determined by the end nodes, while in many real-life scenarios like the spread COVID-19, the self-evolution of edges also exists. Also, they are discrete models and may fail to model dynamical systems that are continuous by nature, compared to ODE-based methods.

3.3 Latent Graph ODE model for Dynamical Systems

Dynamical systems with static interaction graph is a special case in our setting. As mentioned in Sec 3.1, [18] employed GNN as the ODE function and it follows the framework of variational autoencoder (VAE) [20], where an approximate posterior distribution $q_\phi\left(z_i^0 \mid \mathcal{X}, A\right)$ is computed over each latent initial state for an object from the encoder. The prior distribution $p(z_i^0)$, which is a standard normal distribution, adds significant regularization over how latent distribution looks like via the Kullback–Leibler divergence term in the loss function, which differs VAE from other autoencoder frameworks. z_i^0 is then sampled from the posterior distribution and the entire trajectory is determined by z_i^0 and the generative model defined by the ODE function g for all objects. Finally, the decoder outputs the predicted trajectories by mapping z_i^t to the original feature space: $x_i^t = f_{\text{dec}}(z_i^t)$. We model dynamical systems with evolving interaction graph under the same framework, where in addition to modeling latent states for nodes, we also incorporate latent states for edges. Then the challenges lie in: (1) How can we infer the initial states for both edges and nodes considering their mutual influence? (2) How to specify the ODE functions for guiding the co-evolution for node and edge latent states respectively?

4 Model

In this section, we present Coupled Graph ODE (CG-ODE) for learning continuous multi-agent dynamical systems with evolving interaction graph. The overall framework is depicted in Figure 3. Following the framework of VAE, CG-ODE consists of three parts that are trained jointly: (1) An encoder that infers the latent initial states for nodes and edges considering the interaction among objects. (2) A generative model characterized by two coupled ODE functions for edges and nodes respectively, with the goal of learning the latent dynamics of the system. (3) Two decoders that generate the predicted nodes and edges based on the decoding likelihood determined by the latent states $p\left(x_i^t \mid z_i^t\right)$ and $p\left(w_{i\to j}^t \mid z_{i\to j}^t\right)$.

4.1 Encoder for Initial States

Given the trajectory sequence $\mathcal{X}$ and the snapshots of the interaction graph among objects, the encoder firstly computes a posterior distribution of latent initial state for each object: $q_\phi\left(z_i^0 \mid \mathcal{X}, \mathcal{A}\right)$, from which z_i^0 is sampled. As in multi-agent dynamical systems, objects are highly-coupled and their mutual influence is propagated

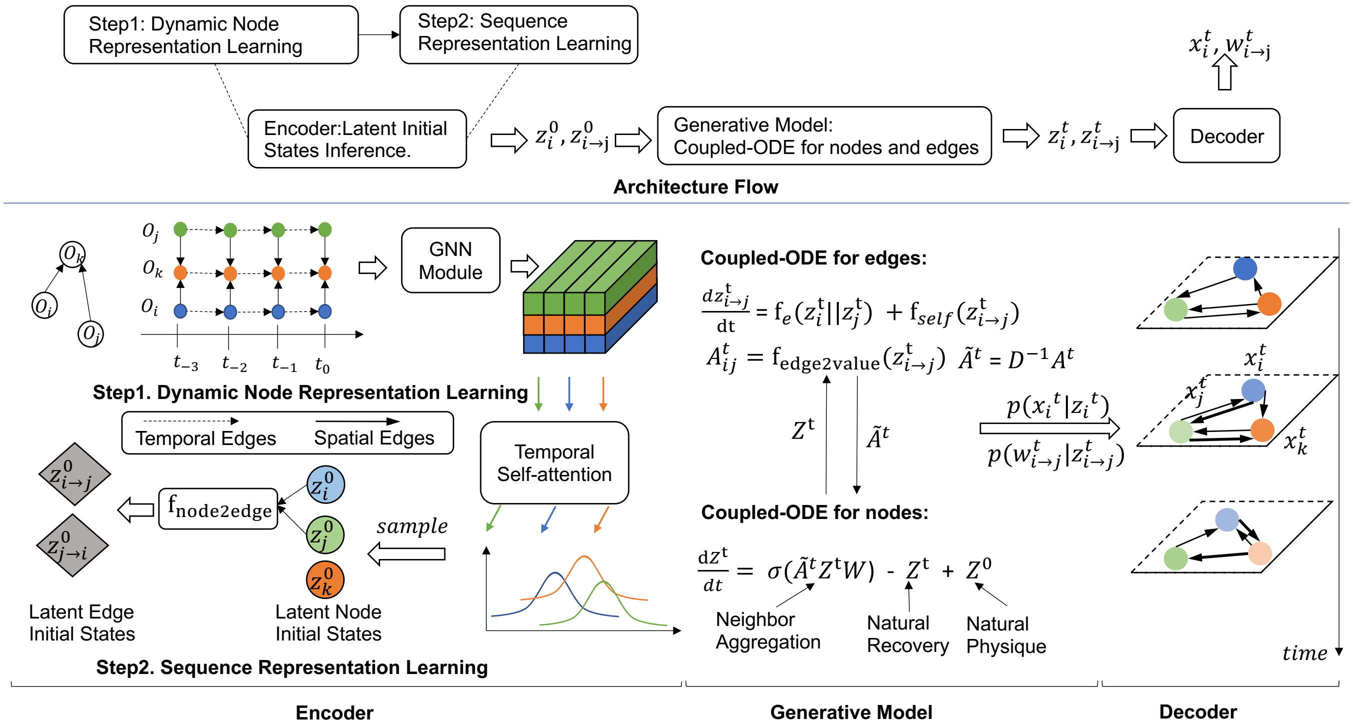

Step1: Dynamic Node Representation Learning
Step2: Sequence Representation Learning
Encoder:Latent Initial States Inference.
$z_i^0, z_{i\to j}^0$
Generative Model: Coupled-ODE for nodes and edges
$z_i^t, z_{i\to j}^t$
Decoder
$x_i^t, w_{i\to j}^t$
Architecture Flow
O_k
O_i
O_j
t_{-3} t_{-2} t_{-1} t_0
GNN Module
Step1. Dynamic Node Representation Learning
Temporal Edges
Spatial Edges
Temporal Self-attention
$z_{i\to j}^0$
$z_{j\to i}^0$
$f_{node2edge}$
z_i^0
z_j^0
z_k^0
sample
Latent Edge Initial States
Latent Node Initial States
Step2. Sequence Representation Learning
Coupled-ODE for edges:
$\frac{dz_{i\to j}^t}{dt} = f_e(z_i^t || z_j^t) + f_{self}(z_{i\to j}^t)$
$A_{ij}^t = f_{edge2value}(z_{i\to j}^t)$ $\tilde{A}^t = D^{-1}A^t$
Z^t
$\tilde{A}^t$
$p(x_i{}^t | z_i{}^t)$
$p(w_{i\to j}^t | z_{i\to j}^t)$
Coupled-ODE for nodes:
$\frac{dZ^t}{dt} = \sigma(\tilde{A}^t Z^t W) - Z^t + Z^0$
Neighbor Aggregation
Natural Recovery
Natural Physique
x_i^t
x_j^t
x_k^t
time
Encoder
Generative Model
Decoder

Figure 3: The Overall framework of Coupled Graph ODE: Firstly, the encoder computes the latent initial states for edges and nodes respectively based on the observed sequence of node attributes and adjacency matrix sequence so far with two steps: Step1: Dynamic node representation learning over the constructed temporal graph. Step2: Sequence representation learning for summarizing over each observation sequence. Then the generative model calls the ODE solver to solve the two coupled ODEs for nodes and edges, which outputs the predicted latent states for nodes and edges in the future. Finally, decoders generate the predicted nodes and edges based on their respective decoding likelihood determined by the latent states.

through the directed weighted edges, we compute the distributions for all objects simultaneously by considering both their trajectories and the dynamic interaction graph among them. After inferring the latent initial states for all nodes, we generate the latent initial states for edges based on the inferred node initial states.

4.1.1 Latent initial states for nodes We now present how to infer the latent initial states for each object. Instead of encoding the temporal pattern for each object independently using an RNN [26], we incorporate the structural pattern by constructing a temporal graph as shown in Figure 3 Step 1, where each node is an observation of an object at a specific timestamp. For edges, we firstly construct spatial edges between two objects at each timestamp t based on the corresponding weighted adjacency matrix A^t, where the edge weight is naturally given by $w_{i\to j}^t$. Then, to preserve the autoregressive nature of each trajectory, we only introduce directed temporal edges $w_{i(t)\to i(t')}$ where $t < t'$ are the timestamps of two consecutive observations of i. We use $w_{i(t)\to j(t')}$ as a uniform expression for both spatial and temporal edges, i.e. for spatial edges (when $t = t'$), $w_{i(t)\to j(t')}$ is equivalent to $w_{i\to j}^t$; for temporal edges (when $i = j$, $t < t'$), $w_{i(t)\to i(t')}$ becomes $w_{i\to i}^t$. By introducing temporal edges and stacking multiple layers of GNN, we can capture the influence from historical observations to the current observation.

Based on the constructed temporal graph, we infer the latent initial states for objects via a two-step process similar as in [26]: 1.) **Dynamic Node Representation Learning**, where we aim to learn a structural representation $h_{i(t)}$ for each observation x_i^t. 2.) **Sequence Representation Learning**, where we employ a self-attention mechanism to summarize each observation sequence into a fixed-dimensional vector u_i. The sequence representation u_i is then utilized to generate the mean and variance for the Gaussian posterior distribution for the latent initial state of object z_i^0.

To learn a structural representation for each observation over the weighted, directed temporal graph, we propose an attention-based spatial-temporal GNN that attends over the immediate neighbors of a node as defined in Eqn 2. Here $h_{j(t)}^{l-1}$ are the representations of object j at timestamp t from layer $l-1$, $\sigma(\cdot)$ is a non-linear activation function and d is the dimension of the latent node representations. The attention score $e_{j(t)\to i(t')}^l$ for both spatial edges (where $t = t'$) and temporal edges (where $i = j$ and $t < t'$), is defined as the multiplication of the corresponding edge weight, and the computed affinity score based on representations of sender node and target node. We adopt the dot-product to compute the affinity score where W_v, W_k, W_q projects input node representations into values, keys and queries. The learned attention coefficient is normalized via softmax across all neighbors. As the temporal graph contains spatial and temporal edges, we add temporal encoding [26, 31] to the sender node representation in order to distinguish them. Finally, we stack L layers to get the final representation for each node: $h_{i(t')} = h^L_{i(t')}$

$$
\begin{gathered}
h^l_{i(t')} = h^l_{i(t')} + \sigma\left(\sum_{j(t)\in\mathcal{N}_{i(t')}} e^l_{j(t)\to i(t')} \times W_v \hat{h}^{l-1}_{j(t)}\right) \\
e^l_{j(t)\to i(t')} = w_{j(t)\to i(t')} \times \alpha^l_{j(t)\to i(t')} \\
\alpha^l_{j(t)\to i(t')} = \left(W_k \hat{h}^{l-1}_{j(t)}\right)^T \left(W_q h^{l-1}_{i(t')}\right) \cdot \frac{1}{\sqrt{d}} \qquad (2) \\
\hat{h}^{l-1}_{j(t)} = h^{l-1}_{j(t)} + \mathrm{TE}(t - t') \\
\mathrm{TE}(\Delta t)_{2i} = \sin\left(\frac{\Delta t}{10000^{2i/d}}\right),\ \mathrm{TE}(\Delta t)_{2i+1} = \cos\left(\frac{\Delta t}{10000^{2i/d}}\right)
\end{gathered}
$$

Next, we employ a self-attention mechanism to generate sequence representation for each object, which is then utilized to compute the posterior distribution for the latent node initial state. Compared with traditional recurrent models that encode temporal pattern within each sequence such as RNN, LSTM, self-attention mechanism can be better parallelized for speeding up the training process and alleviate the vanishing/exploding gradient problem in these models [28]. We introduce a global sequence vector a_i to calculate a weighted sum of observations as the sequence representation, where a_i is the average of node representations with a nonlinear transformation W_a. The process is shown in Figure 3 Step 2 and Eqn 3, where $\hat{h}_{i(t)} = h_{i(t)} + \mathrm{TE}(t)$.

$$u_i = \frac{1}{N}\sum_t \sigma\left(a_i^T \hat{h}_{i(t)} \hat{h}_{i(t)}\right),\ a_i = \tanh\left(\left(\frac{1}{N}\sum_t \hat{h}_{i(t)}\right) W_a\right) \qquad (3)$$

Finally, we compute the mean and variance of the approximated posterior distribution from the sequence representation u_i, and sample z_i^0 from it.

$$
\begin{gathered}
q_\phi\left(z_i^0 \mid \mathcal{X}, \mathcal{A}\right) = \mathcal{N}\left(\mu_{z_i^0}, \sigma_{z_i^0}\right),\ \mu_{z_i^0}, \sigma_{z_i^0} = f_{\mathrm{trans}}\left(u_i\right) \\
z_i^0 \sim p\left(z_i^0\right) \approx q_\phi\left(z_i^0 \mid \mathcal{X}, \mathcal{A}\right)
\end{gathered} \qquad (4)
$$

4.1.2 Latent initial states for edges Given the latent initial states for a pair of nodes z_i^0, z_j^0, the latent initial state for each edge is given by Eqn 5, where $||$ denotes the concatenation operation.

$$z^0_{i\to j} = f_{\mathrm{edge}}\left([z_i^0 || z_j^0]\right) \qquad (5)$$

4.2 ODE Generative Model and Decoder

After computing the latent initial states for nodes and edges, we now define the ODE function that drives the system to move forward. In multi-agent dynamical systems, the latent node and edge states are co-evolving along with time. We therefore propose the coupled ODE functions for edge and nodes respectively as shown in Eqn 6, where $Z^t \in \mathbb{R}^{N\times d}$ denotes the latent state matrix for all N objects, $W \in \mathbb{R}^{d\times d}$ is a linear feature transformation matrix. The node ODE function consists of three parts and can be understood from an epidemic modeling perspective [37]. If we view Z^t as the infection conditions for all states in the U.S at timestamp t, the first term accounts for the infection from neighbors; the second term $-Z^t$ can be viewed as natural recovery and the third term Z^0 is for natural physique [3]. Note that we use the normalized adjacency matrix $\tilde{A} = D^{-1}A$ to compute message passing from neighbors, where D is the degree matrix of A defined as $D_{ii} = \sum_j A_{ij}$. This is because when solving the ODE using a numerical solver, it is equivalent to stack multiple GNN layers as time progresses. Using an unnormalized adjacency matrix would therefore cause the potential gradient exploding problem. As our interaction graph is asymmetric, we normalize it to $D^{-1}A$ instead of $D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ for symmetric adjacency matrix. The edge ODE function consists of two parts. $f_e : \mathbb{R}^{2d} \to \mathbb{R}^d$ is a mapping function that transforms the concatenation of two nodes to the latent state of their corresponding edge. $f_{\mathrm{self}} : \mathbb{R}^d \to \mathbb{R}^d$ accounts for the self-evolution of edges. For example, the population flow between California and Washington may change over time due to factors like holidays and quarantine policies, which may not be driven by the severity of COVID-19 at these two locations, i.e. z_i^t and z_j^t. $f_{\mathrm{edge2value}} : \mathbb{R}^d \to \mathbb{R}$ transforms the latent edge states to a scalar, which is then utilized in the node ODE function.

$$
\begin{gathered}
\frac{dZ^t}{dt} = \sigma\left(\tilde{A}^t Z^t W\right) - Z^t + Z^0 \\
\frac{dz^t_{i\to j}}{dt} = f_e\left([z_i^t || z_j^t]\right) + f_{\mathrm{self}}\left(z^t_{i\to j}\right) \qquad (6) \\
A^t_{ij} = f_{\mathrm{edge2value}}\left(z^t_{i\to j}\right),\ \tilde{A}^t = D^{-1}A^t
\end{gathered}
$$

Given the coupled ODE functions and the initial states for nodes and edges, the trajectories for all objects are determined. We compute the predicted trajectories for each object and the interaction graph based on the decoding likelihood $p\left(\boldsymbol{x}_i^t \mid z_i^t\right)$ and $p\left(w^t_{i\to j} \mid z^t_{i\to j}\right)$ with two decoding functions $f_{\mathrm{decN}}, f_{\mathrm{decE}}$ respectively.

4.3 Training

Now that we have described all the elements, the overall training process goes as follows: Each training sample is separated into two halves along the time, where we condition on the first half $[T_0, T_1]$ in order to predict/reconstruct the second half $[T_1, T_2]$. Given the trajectory sequence $\mathcal{X}$ and weighted adjacency matrix sequence $\mathcal{A}$, we firstly run the encoder to compute the posterior distribution $q_\phi\left(z_i^0 \mid \mathcal{X}, \mathcal{A}\right)$ for each object, based on the first half. Then we sample the latent node initial states z_i^0 from it for all objects, and compute the latent initial states for edges as $z^0_{i\to j} = f_{\mathrm{edge}}\left(\left[z_i^0 || z_j^0\right]\right)$. We then run the generative model defined by two coupled ODE functions to compute latent states for predicted nodes and edges in the future. Next, we run the decoder to compute the mean of each decoding distribution as :$\mu_i^t = f_{\mathrm{decN}}(z_i^t)$, $\mu^t_{i\to j} = f_{\mathrm{decE}}(z^t_{i\to j})$, which is treated as the predicted value for edges and nodes. Finally, we jointly train the encoder, generative model and decoder by maximizing the evidence lower bound (ELBO) as shown below, where the first term is the reconstruction loss for nodes and edges, and the second term is the KL divergence. We additionally introduce a hyperparameter λ_{edge} for balancing the reconstruction loss of

Algorithm 1: Coupled Graph ODE training procedure.

```
Input: Adjacency matrix sequence $\mathcal{A} = \{A^1, A^2, \ldots, A^T\}$;
Node feature sequences $\mathcal{X} = \{X^{(1)}, X^{(2)}, \ldots, X^{(T)}\}$.
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 encoder:
        Construct the temporal graph as shown in Figure 3
          Step 1 based on the observed data in the first half;
        Conduct dynamic node representation learning on
          the temporal graph according to Eqn 2;
        Generate sequence representation for each object
          according to Eqn 3, then sample latent initial states
          $z_i^0$ for each object according to Eqn 4;
        Generate latent initial state $z_{i\to j}^0$ for each edge
          according to Eqn 5;
        //For the generative model:
        Given initial nodes, edges state, and timestamps to
          predict $[T_1, T_2]$, solve the coupled ODE in Eqn 6;
        //For the decoder:
        Compute predicted nodes and edges based on the
          decoding likelihood $p\left(x_i^t \mid z_i^t\right)$ and
          $p\left(w_{i\to j}^t \mid z_{i\to j}^t\right)$ respectively;
    end
    Update the parameters $\phi$ and $\theta$ by optimizing ELBO loss
      in Eq. 7;
end
```

edges and nodes.

$$
\begin{aligned}
&\mathrm{ELBO}\,(\theta,\phi) = \mathbb{E}_{Z^0 \sim \prod_{i=1}^{N} q_\phi(z_i^0 | \mathcal{X},\mathcal{A})}\left[\log p_\theta\left(\mathcal{X},\mathcal{A}\right)\right] \\
&- \mathrm{KL}[\prod_{i=1}^{N} q_\phi\left(z_i^0 \mid \mathcal{X},\mathcal{A}\right) \| p(Z^0)] \\
&= \left(1-\lambda_{edge}\right)\mathcal{L}_{node} + \lambda_{edge}\mathcal{L}_{edge} - \mathrm{KL}[\prod_{i=1}^{N} q_\phi\left(z_i^0 \mid \mathcal{X},\mathcal{A}\right) \| p(Z^0)]
\end{aligned} \tag{7}
$$

The reconstruction loss is estimated as below where the constant σ is the standard derivation of each prior distribution. The overall pipeline is illustrated in Algo 1.

$$
\begin{aligned}
\mathcal{L}_{node} &= -\sum_i \sum_t \frac{\left\|\mathbf{x}_i^t - \mu_i^t\right\|^2}{2\sigma^2} \\
\mathcal{L}_{edge} &= -\sum_i \sum_j \sum_t \frac{\left\|\mathbf{w}_{i\to j}^t - \mu_{i\to j}^t\right\|^2}{2\sigma^2}
\end{aligned} \tag{8}
$$

5 Experiments

In this section, we present the evaluation results over our model. We first introduce the dataset we used, followed by our experimental results and analysis.

5.1 Experiment Setup

5.1.1 Dataset We conduct experiments on the COVID-19 data as well as the simulated social network data from [11]. For the COVID-19 dataset, we utilize the daily trendency data [9] from the Johns Hopkins University (JHU) Center for Systems Science and Engineering[1] to train our model for the United States. More specifically, we focus on predicting the state-level daily cumulative deaths. For node features, we choose five out of ten dynamic features provided by JHU , which are: #Confirmed, #Deaths, #Recovered, Mortality-Rate and Testing-Rate. Details about their semantic meaning and preprocessing can be found in Appendix A. Additionally, we utilize the population for each state as one static feature, which has been widely used in many existing disease prediction models [16, 34, 39, 40]. We use the mobility data provided by Safegraph[2] to construct the interaction graph. SafeGraph is a company that aggregates anonymized location data from numerous mobile applications [5]. The mobility data captures the movement of people between census block groups (CBGs) and we group them by states to form the daily population flow including in-state flow (See Appendix A for details). We additionally use the in-state flow as the sixth node dynamic feature, thus each node has seven features in total. The social network data simulates the opinion migration of individuals in a social network over time [11]. We set the number of nodes as 80 and generate 399 timestamps. The initial positions (opinions) of individuals follow uniform distribution in a 2-d space. We set the noise parameter as 0.2, the sparsity parameter as $e^{-0.4}$.

5.1.2 Data Split and Task We train our model in a sequence to sequence setting where we split the time of each training sample into two parts $[T_0, T_1]$ and $[T_1, T_2]$. We condition on the first half of observations and reconstruct the second half. To achieve this, we generate training samples by setting three hyperparameters: prediction length, condition length and interval, where prediction length is the size of the second half; condition length is the size of the first half, and interval is the overlap between two consecutive training samples. We generate different training samples to train our model when predicting at different horizons. For the COVID-19 dataset, we utilize data from April.12.2020 to Nov.30.2020 to train our model and test the performance on data from Dec.01.2020 to Dec.31.2020. For the social network dataset, we utilize data from the first 320 timestamps to make predictions in timestamps 321-399.

5.2 Baselines

For both datasets, we compare with the following three discrete neural network-based methods.

- **LSTM** [30]: A classic recurrent neural network (RNN) that learns the dynamics of each node independently.
- **NRI** [21]: A VAE-based relation inference model. The encoder infers the static graph structure among nodes and the GNN-based decoder uses the inferred graph to generate the node features in the future.
- **VGRNN** [12]: A VAE-based graph recurrent neural network that jointly learns the evolution of network topology and node attribute changes.

[1] https://github.com/CSSEGISandData/COVID-19

[2] https://www.safegraph.com/covid-19-data-consortium

Table 1: Mean Absolute Percentage Error (MAPE) for Cumulative Deaths

Step Length	Pred Date	UCLA-SuEIR	UT-Mobility	Columbia	IHME	LSTM	NRI	VGRNN	CG-ODE
1-week -ahead	Nov.29-Dec.05	0.03297	0.02707	**0.02001**	-	0.08094	0.07784	0.06807	0.02144
	Dec.07-Dec.12	0.02283	0.03736	0.02455	0.02458	0.08363	0.07448	0.06086	0.02653
	Dec.14-Dec.19	0.01946	0.04178	**0.01443**	-	0.07144	0.06462	0.06102	0.01997
	Dec.21-Dec.26	0.01851	0.05460	0.02595	-	0.04912	0.04616	0.04297	**0.01849**
	Average	0.02344	0.04020	**0.02124**	0.02458	0.07128	0.06578	0.05823	0.02161
2-weeks -ahead	Nov.29-Dec.12	0.11036	0.07119	0.08194	-	0.15922	0.15004	0.13791	**0.04341**
	Dec.07-Dec.19	0.07951	0.05830	0.09248	0.06252	0.14873	0.13782	0.12812	**0.04702**
	Dec.14-Dec.26	0.06356	0.04112	0.05174	-	0.13012	0.11423	0.10712	**0.03709**
	Average	0.08448	0.05687	0.07539	0.06252	0.14602	0.13403	0.12438	**0.04251**
3-weeks -ahead	Nov.29-Dec.19	0.17361	0.13255	0.13721	-	0.11793	0.10752	0.10624	**0.04513**
	Dec.06-Dec.26	0.13116	0.09570	0.14445	0.10671	0.19561	0.18088	0.17322	**0.09832**
	Average	0.15239	0.11413	0.14083	0.10671	0.15677	0.14420	0.13973	**0.07173**

Table 2: Mean Absolute Percentage Error (MAPE) for Social Data.

Pred Length	10	20	40
LSTM	0.12419	0.37031	0.69579
NRI	0.28879	0.41980	0.68417
VGRNN	**0.11312**	0.27789	0.56763
CG-ODE	0.12359	**0.26340**	**0.45434**

For the COVID-19 dataset, we additionally considers traditional statistical models which learn the dynamic for each state (node) independently. We choose the following four baselines developed by different institutions and obtain their predictions from the forecast hub[3] which is officially used by the centers for disease control and prevention (CDC)[4].

- **UCLA-SuEIR** [40]: A SuEIR model which is a variant of the SEIR [16] model considering both untested and unreported cases. The model considers reopening and assumes susceptible population will increase after the reopening. Parameters are learned via machine learning algorithms.
- **IHME** [13]: A non-linear curve-fitting method with the assumption that current interventions remain unchanged.
- **UT-Mobility** [36]: A non-linear curve-fitting method where mobility data within each state is utilized to quantify the changing impact of social-distancing.
- **Columbia** [34]: A survival-convolution model with piecewise transmission rates that incorporates incubation period and provides a time-varying effective reproductive number.

5.3 Performance Evaluation

We evaluate the performance of our model based on Mean Absolute Percentage Error (MAPE) as shown in Table 1 and Table 2. For the COVID-19 dataset, as the prediction results for statistical baselines are obtained from their weekly official submissions to CDC, we compare the performance across all models using the same weekly prediction periods. Specifically, Pred Date denotes the targeted prediction period while Step Length denotes the number of days before the prediction is made. For example, for Pred Date of Nov.29 - Dec.05, the prediction is made for Dec.05 by using the data up to Nov.29. Therefore it is a 1-week-ahead prediction.

We first observe that CG-ODE is able to outperform all baselines in long-term predictions by a big margin while achieves similar short-term prediction performance. In both datasets, there is a wider performance gap between CG-ODE and other baselines (e.g. LSTM) that do not consider the interaction among objects, when predicting longer-range node attributes. This indicates that the interaction graph plays a more important role in facilitating long-term predictions. Similar observation can be found in some dynamic physical systems such as particles connected by springs [18]: to predict the location for each object in a spring system, usually the object's own velocity can be a good approximation for predicting the location at the next timestamp, while it fails to predict locations in the longer-range without considering its interaction among objects. Secondly, neural network-based baselines fail to produce accurate predictions for the COVID19-dataset, which is expected as they are discrete models and may fail to capture the underlying dynamics for a continuous interacting system. Among three neural network-based models, by comparing LSTM with NRI and VGRNN, where the latter two consider underlying interaction among objects and LSTM only models the trajectory for each state independently, we found that by jointly modeling the evolution of graph and node attributes, models can achieve better prediction results. However, NRI performs bad on the social network dataset. This is because the topology change in the social network dataset is more sharp than that of the COVID-19 dataset, and the static network topology assumption in NRI would no longer holds. Among four statistical methods, we observe that UT-Mobility shows better performance in long-term predictions than others. This indicates that the mobility data can serve as a useful signal for predicting the spread of COVID-19. However, UT-Mobility only utilizes the mobility data for each state independently, instead of utilizing it to model the interaction among states as in our model, thus it achieves worse prediction results compared to CG-ODE.

Hyperparameter Study. We then study two important hyperparameters in CG-ODE in the COVID-19 dataset, which are λ_{edge} in Eqn 7 for balancing the reconstruction loss for nodes and edges, and the condition length for different prediction horizons. Figure 4 shows the MAPE changes as a function of λ_{edge} for three prediction

[3] https://github.com/reichlab/covid19-forecast-hub/tree/master/data-processed
[4] https://www.cdc.gov/coronavirus/2019-ncov/covid-data/forecasting-us.html

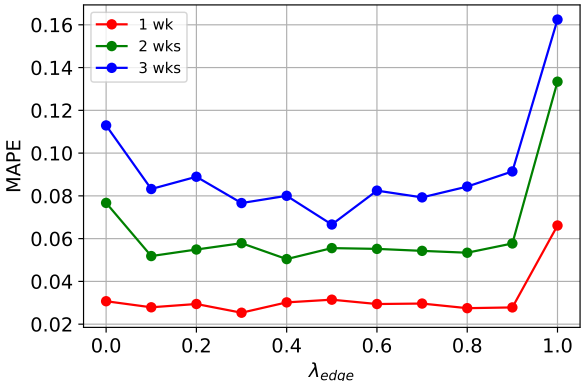


Figure 4: MAPE as a function of λ_{edge} on the COVID-19 dataset

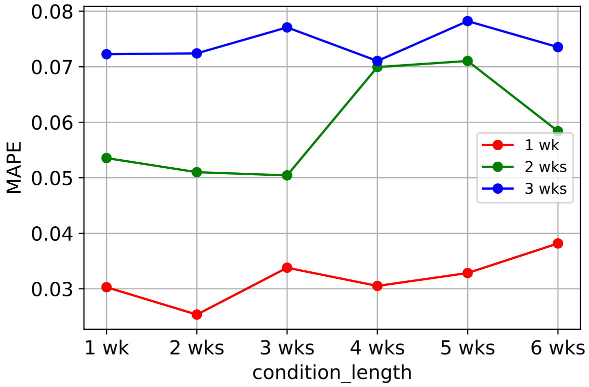


Figure 5: MAPE as a function of condition length on the COVID-19 dataset

horizons respectively. First, we can see that the optimal λ_{edge} for 1-week-, 2-week- and 3-week-ahead predictions are 0.3, 0.4, 0.5 respectively, which increases when predicting node attributes in the longer range. This is consistent with the prediction errors illustrated in Table 1, where the performance gap between our models and other baselines that do not consider graph interaction increases, when predicting longer-range node attributes. They indicate that the interaction graph plays a more important role in facilitating long-term predictions. Second, for all of the three prediction horizons, when $\lambda_{edge} = 0$, the MAPE increases sharply as the model is only trained to recover the node attributes, without supervision from the dynamic interaction graph. In this case, our model has degenerated to a relation inference model where the ground truth graph is not known during training and is learned in an unsupervised way. Notably, CG-ODE is able to achieve comparable results for the 3-week-ahead predictions compared with statistical baselines even when $\lambda_{edge} = 0$, which verifies the effectiveness of our co-evolution model and the importance of introducing interaction graph for long-term predictions. Last, when $\lambda_{edge} = 1$, our model is only trained for recovering the dynamic interaction graph, and learns the dynamic node attributes in an unsupervised way. Therefore, the MAPE increases as expected. However, the prediction error is still comparable with some statistical baselines especially in the long-term prediction. For example in the 3-week-ahead prediction, UCLA-SuEIR has MAPE of 0.15239 while CG-ODE has MAPE of 0.16245 when $\lambda_{edge} = 1$. This shows the capability of CG-ODE of learning on semi-supervised data or sparse data.

Figure 5 shows the MAPE changes as a function of condition length for three different prediction horizons. The optimal condition length for 1-week-, 2-week- and 3-week-ahead predictions are 2 weeks, 3 weeks, 4 weeks respectively, which increases when predicting node attributes in the longer range. This is expected as long-term prediction would usually depend more on the dynamic pattern in the historical data, i.e. require longer range dependency from the past. This is similar to a spring system: the location of an object at next timestamp can be well-approximated by its current location and velocity, while the locations in the longer future should depend on its historical trajectories, instead of a single point.

5.4 Case Studies

We conduct a case study by adding three different interventions to the interaction graph. Table 3 shows the summation of the number of deaths reduced for all states on Dec.26 compared to the ground truth, when adding different interventions. We set the duration of each intervention as 2 weeks and study the effect of adding the same intervention at different times. For example, 1-wk-ahead means the intervention period is one week ago, i.e. the intervention ends at Dec.20 and starts at Dec.7. The first intervention adds 20% reduction to all in-state population flows. The second intervention adds 20% reduction to all between-state population flows. The third one removes the same amount of population flow as in the second intervention, but reduces the population flow in descending order of states' original outflow. Specifically, we rank states by their daily population outflow in descending order and set the outflow for each state to zero starting from the state with the largest population outflow, until the total amount of outflow reduction equals to that of the second intervention.

We firstly observe that reducing the in-state flow will decrease the number of deaths the most. This is expected as the value of in-state population is much larger than that of the between-state population. Secondly, compared with evenly reducing between-state flow for all states, reducing the population outflow from core states will result in a larger drop in the number of deaths. This can be due to the fact that states with larger population outflow are likely to have larger in-state flow as well, due to the loose control over traveling. Thus the severity of these states are likely to be higher. Finally, by comparing the number of reduced deaths for the same intervention happened at different times, we notice that the effect of all interventions tends to decrease day by day.

6 Conclusion

In this paper, we investigate the problem of learning the dynamics of interacting systems by jointly modeling the evolution of nodes and edges. We model system dynamics in a continuous fashion through two coupled neural ordinary differential equations. Specifically, the evolution of a node would depend on its self-evolution and influence received from the interaction graph; the evolution of

Table 3: Number of Deaths Reduced on Dec.26

	1-wk-ahead	2-wk-ahead	3-wk-ahead
In-state flow deduction (20%)	-8973	- 7084	-6824
Between-state flow deduction (20%)	-2465	-2215	-2197
Flow deducted from core states	-3854	-3625	-3517

an edge would depend on its end node's attributes and the edge's self-evolution. We infer the latent initial states for the two ODEs through a novel encoder, which is a VAE-based graph neural network (GNN) that infers the initial states for all objects simultaneously with uncertainty. The proposed model, coupled graph ODE (CG-ODE) is able to achieve accurate prediction for the cumulative deaths of COVID-19 in the United States as well as the simulated social network dataset, especially for long-term predictions. We also conduct an ablation study where we add intervention to the interaction graph and provide insights on how to make efficient intervention policies to control the population flow between the 50 states within the United States. There are some limitations though. Our current model tries to learn latent edge representations by assuming a fully-connected graph, which is time-consuming especially for large dataset. In the future, we plan to design efficient sampling methods for the edge ODEs to balance model efficiency and performance.

Acknowledgments

This work was partially supported by NSF IIS- 2031187, NSF DGE-1829071, NSF III-1705169, NSF CAREER Award 1741634, NSF 1937599, NIH R35-HL135772, NIH/NIBIB R01 EB027650, DARPA HR00112090027, Okawa Foundation Grant, and Amazon Research Awards.

References

[1] Yunsheng Bai, Hao Ding, Song Bian, Ting Chen, Yizhou Sun, and Wei Wang. 2019. SimGNN: A Neural Network Approach to Fast Graph Similarity Computation. In *WSDM'19*.

[2] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, and koray kavukcuoglu. 2016. Interaction Networks for Learning about Objects, Relations and Physics. In *NIPS'16*.

[3] Maximilian Behr, Peter Benner, and Jan Heiland. 2019. Solution formulas for differential Sylvester and Lyapunov equations. In *Calcolo*.

[4] Michael B Chang, Tomer Ullman, Antonio Torralba, and Joshua B Tenenbaum. 2016. A Compositional Object-Based Approach to Learning Physical Dynamics. *ICLR'16* (2016).

[5] Serina Chang, Emma Pierson, Pang Wei Koh, Jaline Gerardin, Beth Redbird, David Grusky, and Jure Leskovec. 2021. Mobility network models of COVID-19 explain inequities and inform reopening. In *Nature*.

[6] Zhengping Che, Sanjay Purushotham, Kyunghyun Cho, David Sontag, and Yan. Liu. 2018. Recurrent Neural Networks for Multivariate Time Series with Missing Values. In *Scientific Reports*.

[7] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. 2018. Neural Ordinary Differential Equations. In *NIPS'18*.

[8] Miles Cranmer, Alvaro Sanchez-Gonzalez, Peter Battaglia, Rui Xu, Kyle Cranme, David Spergel, and Shirley Ho. 2020. Discovering Symbolic Models from Deep Learning with Inductive Biases. In *Neurips'20*.

[9] Ensheng Dong, Hongru Du, and Lauren Gardner. 2020. An interactive web-based dashboard to track COVID-19 in real time. In *The Lancet Infectious Diseases*.

[10] J.R. Dormand and P.J. Prince. 1980. A family of embedded Runge-Kutta formulae. In *Journal of Computational and Applied Mathematics*.

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

[12] Ehsan Hajiramezanali, Arman Hasanzadeh, Krishna Narayanan, Nick Duffield, Mingyuan Zhou, and Xiaoning Qian. 2019. Variational graph recurrent neural networks. In *Neurips'19*.

[13] IHME COVID-19 health service utilization forecasting team and Christopher JL Murray. 2020. Forecasting COVID-19 impact on hospital bed-days, ICU-days, ventilator-days and deaths by US state in the next 4 months. In *medRxiv preprint :2020.03.27.20043752*.

[14] Dan Hendrycks and Kevin Gimpel. 2019. Decoupled weighted decay regularization. *ICLR'19* (2019).

[15] Dan Hendrycks and Kevin Gimpel. 2020. Gaussian Error Linear Units (GELUs). *arXiv* (2020).

[16] H. W Hethcote. 2000. The mathematics of infectious diseases. In *SIAM review*.

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

[18] Zijie Huang, Yizhou Sun, and Wei Wang. 2020. Learning Continuous System Dynamics from Irregularly-Sampled Partial Observations. In *Neurips'20*.

[19] 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)*.

[20] Diederik P. Kingma and Max Welling. 2014. Auto-Encoding Variational Bayes.. In *ICLR'14*.

[21] 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).

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

[23] Jundong li, Harsh Dani, Xia Hu, Jiliang Tang, Yi Chang, and Huan Liu. 2017. Attributed Network Embedding for Learning in a Dynamic Environment. In *CIKM'17*.

[24] 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*.

[25] Michael Poli, Stefano Massaroli, Junyoung Park, Atsushi Yamashita, Hajime Asama, and Jinkyoo Park. 2019. Graph Neural Ordinary Differential Equations. *arXiv* (2019).

[26] Yulia Rubanova, Ricky T. Q. Chen, and David K Duvenaud. 2019. Latent Ordinary Differential Equations for Irregularly-Sampled Time Series. In *Neurips'19*.

[27] Shengzhong Liu Huajie Shao Dongxin Liu Jinyang Li Tianshi Wang Dachun Sun Shuochao Yao Tarek Abdelzaher Ruijie Wang, Zijie Huang. 2021. DyDiff-VAE: A Dynamic Variational Framework for Information Diffusion Prediction. In *SIGIR'21*.

[28] 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*.

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

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

[31] 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 *NIPS'17*.

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

[33] Haiwen Wang, Ruijie Wang, Chuan Wen, Shuhao Li, Yuting Jia, Weinan Zhang, and Xinbing Wang. 2020. Author Name Disambiguation on Heterogeneous Information Network with Adversarial Representation Learning. In *AAAI '20*.

[34] Qinxia Wang, Shanghong Xie, Yuanjia Wang, and Zeng Donglin. 2020. Survival-Convolution Models for Predicting COVID-19 Cases and Assessing Effects of Mitigation Strategies. In *Frontiers in Public Health*.

[35] Ruijie Wang, Yuchen Yan, Jialu Wang, Yuting Jia, Ye Zhang, Weinan Zhang, and Xinbing Wang. 2018. AceKG: A Large-Scale Knowledge Graph for Academic Data Mining. In *CIKM '18*.

[36] Spencer Woody, Mauricio Tec, Maytal Dahan, Kelly Gaither, Michael Lachmann, Spencer J. Fox, Lauren Ancel Meyers, and James Scott. 2020. Projections for first-wave COVID-19 deaths across the U.S. using social-distancing measures derived from mobile phones.

[37] Louis-Pascal Xhonneux, Meng Qu, and Jian Tang. 2020. Continuous Graph Neural Networks. In *ICML'20*.

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

[39] Chaoqi Yang, Ruijie Wang, Fangwei Gao, Dachun Sun, Jiawei Tang, and Tarek Abdelzaher. 2020. Analyzing the Design Space of Re-opening Policies and COVID-19 Outcomes in the US. arXiv:2005.00112

[40] Difan Zou, Lingxiao Wang, Pan Xu, Jinghui Chen, Weitong Zhang, and Quanquan Gu. 2020. Epidemic Model Guided Machine Learning for COVID-19 Forecasts in the United States. In *medRxiv preprint :2020.05.24.20111989*.

A Data Preparation

We utilize the COVID-19 data [9] from the Johns Hopkins University (JHU) Center for Systems Science and Engineering to train our model for the United States (See Section 5). Additionally, we incorporate mobility data from SafeGraph to construct the interaction graph and use the in-state flow as one node feature. SafeGraph data captures the movement of people between census block groups (CBGs), which are geographical units that typically contain a population of between 600 and 3,000 people, and points of interest (POIs) like restaurants, grocery stores, or religious establishments [5]. Specifically, we utilize the Social Distancing Metrics dataset [5] from SafeGraph, which contains daily estimates of the proportion of people staying home in each CBG. As the scale for different node features and mobility data are different, we now introduce the preprocessing process for each of them below.

- # **Confirmed**. The number of daily increased confirmed cases for each state. We divide each value by 10 for normalization.
- # **Deaths**. The number of daily increased deaths for each state. We keep the original value for this feature without normalization.
- # **Recovered**. The number of daily increased recovered cases for each state. We divide each value by 10 for normalization.
- **Mortality-Rate**. The number of daily cumulative deaths * 100/ the number of daily cumulative confirmed cases for each state. The value is within range [0, 10] for the provided data, and we keep the original value without normalization.
- **Testing-Rate**. The number of daily cumulative test results per 100,000 persons for each state. The cumulative test results are equal to the summation of total positive cases and total negative cases, obtained from the COVID Tracking Project [6].
- **Population**. The number of population for each state. We divide each value by 1000000 for normalization.
- **Mobility data**. The number of daily population flow between and within each state. We divide each value by 1000000 for normalization. Note that the population flows between two states are asymmetric.

We concatenate #Confirmed, #Deaths, #Recovered, Mortality-Rate, Testing-Rate, Population and In-State Mobility data to generate each node attribute of size seven. The In-State Mobility data is the number of people moving between POIS within each state, which does not contain the inflow population from other states.

We generate training samples from April.12 to Nov.31 with three hyperparameters: condition length, prediction length and interval. The code snippet is shown below, where we split the time of each training sample into two halves $[T_0, T_1]$ and $[T_1, T_2]$. We condition on the first half with size equals to the condition length, to predict/reconstruct the second half with size equals to prediction length. The interval is the forwarding steps along the time between two consecutive training samples. We generate different training samples for predicting different horizons. For all settings, we set the interval as 3. For the simulated social network dataset, we set the interval as 5. To generate testing sequences for the social network dataset which does not have any test points as in the COVID-19 dataset, we utilize the feature sequence from timestamps 321-399 as the input testing features, and generate testing sequences following the pseudo code below.

```
condition_length = 14 # number of days to condition on
prediction length = 7 # number of days to predict
interval = 3 # number of steps forward for next sample

sample_length = condition_length + prediction_length
total_timestamps = feature_train.shape[1]
#feature_train of shape [num_states,num_times,num_features]
feature_sequences = []
for i in range(0, total_timestamps - sample_length + 1,
     interval):
   feature_sequences.append(features[:, i:i +
        sample_length, :])
```

B Implementation Details

We now introduce the implementation details in our experiment.
ODE Solver. For the COVID-19 dataset, we use the fourth-order Runge-Kutta method [29] from the torchdiffeq python package [7] as the ODE solver. It solves the ODE system on a time grid that is five times denser than the observed time points. For the simulated social dataset, we use the Euler method [10] which is another type of fixed step solver. We also utilize the Adjoint method described in [7] which reduces the memory cost for backpropagation to a constant.
Encoder. The encoder consists of two modules and aims to infer the latent initial states for nodes and edges simultaneously considering the interaction among objects. It firstly constructs a temporal graph and then performs dynamic node representation learning and sequence representation learning to generate latent initial states for nodes. Then the edge initial state is generated by the latent states of its end nodes. We set the latent representation dimension in the GNN of the first module as 64 and the number of layers as 1 for both dataset. We use Gelu [15] as the activation function. For sequence representation learning, we set the output dimension as 20 and 30 for the COVID-19 dataset and the social network dataset respectively, which is the hidden dimension in the ODE function.
ODE functions. The generative model consists of two coupled ODE functions for nodes and edges respectively, in order to capture their co-evolution along with the time. We set the dimension for both latent states for nodes and edges as 20 and 30 respectively for the COVID-19 dataset and the social network dataset. For edge ODE, we implement the self-evolution function f_{self} as a simple two-layer Multi-Layer Perception (MLP). The transfer function $f_{\text{edge2value}}$ is also a simple MLP, which transforms latent edge states to a scalar and is then utilized in the node ODE.
Training details. We implement our model in pytorch. Encoder, generative model and the decoder parameters are jointly optimized with AdamW optimizer [14] with learning rate 0.0005. For both datasets, we train our model over 50 epochs and report the results by the best validation value.

[5] https://docs.safegraph.com/docs/social-distancing-metrics
[6] https://covidtracking.com/
[7] https://github.com/rtqichen/torchdiffeq

Table 4: Root Mean Square Error (RMSE) for Social Data.

Pred Length	10	20	40
LSTM	0.15594	0.51567	0.76808
NRI	0.58779	0.68384	0.71086
VGRNN	0.14609	0.28104	**0.50778**
CG-ODE	**0.10914**	**0.26704**	0.53371

Table 5: Root Mean Square Error (RMSE) for Cumulative Deaths

	Pred Date	UCLA-SuEIR	UT-Mobility	Columbia	IHME	LSTM	NRI	VGRNN	CG-ODE
1-week -ahead	Nov.29-Dec.05	145.60	117.56	**77.06**	-	417.15	375.59	400.86	103.36
	Dec.07-Dec.12	**84.76**	223.05	123.03	86.14	460.03	391.26	418.89	146.45
	Dec.14-Dec.19	111.47	310.27	**76.32**	-	415.17	447.66	481.33	192.71
	Dec.21-Dec.26	**111.35**	413.73	212.14	-	347.99	363.05	406.59	150.31
	Average	113.30	266.15	122.14	**86.14**	410.09	394.39	426.92	148.21
2-weeks -ahead	Nov.29-Dec.12	605.54	394.57	442.38	-	992.42	823.33	854.72	**230.70**
	Dec.07-Dec.19	539.26	350.78	626.48	439.99	981.73	902.39	939.26	**302.21**
	Dec.14-Dec.26	558.03	**261.12**	413.86	-	876.60	888.33	930.85	358.41
	Average	567.61	335.49	494.24	439.99	950.25	871.35	861.86	**297.11**
3-weeks -ahead	Nov.29-Dec.19	1129.14	892.55	879.45	-	745.35	591.99	618.16	**170.61**
	Dec.06-Dec.26	1010.10	696.32	1069.29	850.02	1483.57	1343.04	1408.38	**455.30**
	Average	1069.52	794.43	974.37	850.02	1114.46	967.52	1013.27	**312.96**

C Additional Results

Table 4 and 5 shows the Root Mean Square Error (RMSE) for the COVID-19 and social network dataset respectively. We can observe a similar trendency that CG-ODE tends to perform better on long range predictions.