Second-Order Graph ODEs for Multi-Agent Trajectory Forecasting

Song Wen Hao Wang Di Liu Qilong Zhangli Dimitris Metaxas
Rutgers University

Abstract

Trajectory forecasting of multiple agents is a fundamental task that has applications in various fields, such as autonomous driving, physical system modeling and smart cities. It is challenging because agent interactions and underlying continuous dynamics jointly affect its behavior. Existing approaches often rely on Graph Neural Networks (GNNs) or Transformers to extract agent interaction features. However, they tend to neglect how the distance and velocity information between agents impact their interactions dynamically. Moreover, previous methods use RNNs or first-order Ordinary Differential Equations (ODEs) to model temporal dynamics, which may lack interpretability with respect to how each agent is driven by interactions. To address these challenges, this paper proposes the Agent Graph ODE, a novel approach that models agent interactions and continuous second-order dynamics explicitly. Our method utilizes a variational autoencoder architecture, incorporating spatial-temporal Transformers with distance information and dynamic interaction graph construction in the encoder module. In the decoder module, we employ GNNs with distance information to model agent interactions, and use coupled second-order ODEs to capture the underlying continuous dynamics by modeling the relationship between acceleration and agent interactions. Experimental results show that our proposed Agent Graph ODE outperforms state-of-the-art methods in prediction accuracy. Moreover, our method performs well in sudden situations not seen in the training dataset.

1. Introduction

Multi-agent trajectory forecasting has become increasingly important due to its wide-ranging applications in fields such as autonomous driving [14], physics system modeling [15], and sports modeling [3]. To accurately model multiple interacting agents, we must consider the coupled and complex spatial and temporal dimensions. The spatial dimension refers to the nonlinear and time-dependent agent interactions, while the temporal dimension describes how these interactions impact agent motion.

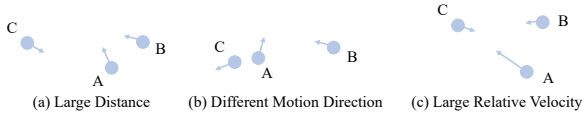


Figure 1. **Impact of distance and velocity information.** (a) Agent A has a greater distance from Agent C than from B, so the interaction between agents A and C may be weaker. (b) Agent A and C are moving in different directions, so they should not affect each other despite their proximity. (c) Agent A and C have a higher relative velocity towards each other, so their interaction should be stronger.

However, accurately modeling multiple interacting agents poses significant challenges due to the complexity and uncertainty of agent interactions and underlying temporal dynamics.

Recently, data-driven methods have attempted to address this problem by jointly modeling agent interactions and underlying temporal dynamics. Some methods use graph neural networks (GNNs) [18] or Transformers [31] to learn agent interactions. However, these approaches often overlook how agent interactions evolve with distance and velocity information among agents, which can lead to poor performance. For example, as shown in Figure 1, when two agents are far away from each other, their interaction should be weak. On the other hand, when they are close to each other but are moving in opposite directions, they may not affect each other. Moreover, previous methods often use RNNs [1], Transformers [31] or first-order Ordinary Differential Equations (ODEs) [12] to model simplified temporal dynamics. Unfortunately, for temporal dynamics, RNNs or Transformers are designed for discrete data, while dynamic under multiple agents are usually continuous. Additionally, the first-order ODEs do not adequately fit the physical motion law. According to Newton's Second Law, the force, which is regarded as agent interactions in our approach, affects acceleration, while first-order ODEs only consider the distance information which affects velocity.

To address these limitations, we propose Agent Graph ODE, a novel approach that models both agent interactions

and continuous second-order temporal dynamics explicitly. Our proposed Agent ODE is a variational autoencoder architecture that uses spatial-temporal Transformers as the encoder to transfer input real-world trajectories to latent vectors as the initial values for ODEs. The decoder first solves the ODEs based on the initial values to generate latent trajectories and then recovers them to real-world trajectories. To leverage distance and velocity information explicitly, we construct dynamic interaction graphs $G_t = (V_t, E_t)$ in real-world space at each time step, where the value of E is 0 or 1 to denote whether two agents affect each other. Additionally, we incorporate distance and velocity information in the spatial attention module by adding information from the interaction graphs. To ensure that the temporal dynamics follow Newton's Second Law, we use second-order Graph ODEs as the decoder, where the acceleration is influenced by the agent interactions. In our Agent Graph ODEs formulation, the agent interactions also depend on interaction graphs whose structure is dynamic and is based on the dynamically evolving distance and velocity.

In summary, our contributions can be listed as follows:

- We explicitly incorporate distance and velocity information to model agent interactions by constructing dynamic interaction graphs in real-world space. In the encoder, the dynamic interaction graphs contribute to the spatial attention module, and in the decoder, the agent interactions in latent space evolve with distance and velocity information, combining dynamic interaction graphs in real-world space.

- We model continuous temporal dynamics using second-order ODEs. Following Newton's Second Law, we use second-order ODEs where the second-order derivative of the latent state (acceleration) is influenced by agent interactions, while agent interactions are based on their relative distances and velocities.

- We conduct an extensive empirical study, demonstrating that our proposed Agent Graph ODE outperforms previous methods on several datasets in terms of forecasting accuracy. Furthermore, our method can handle unexpected events (e.g., sudden obstacles not in the training dataset) better, indicating that it learns more accurately interactions among multiple agents.

2. Related Work

We briefly introduce prior work on multi-agent trajectory forecasting and neural ODEs.

Multi-Agent Trajectory Forecasting. Several recent works have employed graph-based methods to model multiple interacting agents. For instance, Sun et al. proposed Graph-VRNN [25], which used graph networks and RNNs to model interaction systems. Kipf et al. [15] used a variational auto-encoder and underlying interaction graphs to infer interactions, while Graber et al. [9] addressed the limitation of static graphs by inferring dynamic relation graphs for each timestep. Huang et al. [12] used coupled graph ODEs to learn the dynamics of nodes and edges, while Trajectron++ [22] utilized a graph-structured recurrent model to account for environmental information. EvolveGraph [18] recognized the relational structure among multiple heterogeneous agents and made predictions via graphs, while Yildiz et al. [30] combined graph ODEs and Gaussian processes to infer both agent interactions and temporal dynamics with uncertainty estimates. In addition to these graph-based models, AgentFormer [31] used Transformers to jointly model agent interactions and temporal dynamics. Social ODEs [27] utilize first-order ODEs to capture temporal dynamics and use distance information in the latent space to model agent interactions. Some work improves the forecasting accuracy by incorporating environmental information [13].

However, some of these previous works used RNNs [9, 15, 18, 22, 28] or CNNs [2], which are discrete models, to model continuous temporal dynamics. Others used ODEs [12, 13, 30] to model temporal dynamics, but used first-order ODEs that do not model accurately the physics of motion. Some concurrent work [20] uses second-order ODEs, but they do not learn temporal dynamics and agent interactions from previous trajectories. Moreover, these methods did not consider distance and velocity information jointly when modeling agent interactions. In contrast, our proposed Agent Graph ODE uses second-order graph ODEs to model the physics of motion and explicitly incorporate relative distance and velocity information.

Neural Ordinary Differential Equations. Neural ODEs [6] were introduced by Chen et al. as a continuous version of residual networks or RNNs, and have been used to model continuous time series data. Following their work, Rubanova et al. [21] proposed Latent ODEs to model irregularly-sampled time series data, and Yildiz et al. [29] used second-order ODEs and Bayesian neural networks to model high-dimensional trajectories with complex temporal dynamics. Augmented Neural ODEs [8] preserve the topology of the input space and can generalize better than Neural ODEs. Brouwer et al. proposed GRU-ODE-Bayes [7] to handle sporadic observations. Neural Controlled Differential Equations (CDEs) introduce a general framework to deal with irregular time series, similar to RNNs. Apart from time series, Neural ODEs also have many applications. For instance, Neural ODEs have been used in generative models, such as Normalizing Flows [5] and Score-Based Generative Modeling [24]. Furthermore, Shi et al. [23], Liang et al. [19], and Vorbach et al. [26] have applied Neural ODEs for trajectory modeling or planning.

Motivated by these approaches, we propose the Agent Graph ODE based on Latent ODE, which is a variational autoencoder framework. However, unlike the Latent ODE, which models only one agent at a time, Agent Graph ODE models agent interactions using dynamic graphs to account for agent interactions.

3. Approach

Our objective is to model multiple interacting agents by predicting future states $X_f = \{x_1^i, x_2^i, ..., x_{T_f}^i, i = 1, 2, .., N\}$ of N agents given their previous states $X_p = \{x_{-T_p}^i, x_{-T_p+1}^i, ..., x_0^i, i = 1, 2, .., N\}$, considering the dynamics of each agent and its interaction with other agents. Here, T_p denotes the number of previous time steps provided as input, while T_f denotes the number of future time steps to forecast. Furthermore, x_t^i denotes the state of agent i at time t, consisting of its position and velocity, i.e., $x_t^i = \{p_t^i, u_t^i\}$. We distinguish between *real-world-space* variables and *latent-space* variables. We use p_t^i and u_t^i to denote the position and the velocity in real-world space, respectively. In the latent space, we use h_t^i and v_t^i to denote the latent position and latent velocity, respectively.

3.1. Overview

We propose Agent Graph ODE, which models agent interactions and underlying continuous temporal dynamics explicitly. Agent Graph ODE model is based on Latent ODE, a variational autoencoder architecture. It consists of three components shown in Figure 2:

Interaction graph construction. We first use position and velocity information in real-world space to construct dynamic interaction graphs $G_t = (V_t, E_t)$ for each time t. The interaction graphs are used in the encoder and decoder to extract agent interaction features.

Encoder. We use the spatial-temporal Transformers as our encoder to transform the real-world trajectories X_p into latent vectors z, which are used for computing initial values for ODEs. We utilize the interaction graphs in the spatial attention module to incorporate distance and velocity information.

Decoder. The decoder module includes two parts: (1) building and solving ODEs in the latent space; and (2) recovering real-world trajectories from the latent-space trajectories. We build second-order Graph ODEs combining interaction graphs. Then we solve ODEs to generate latent trajectories, which are determined by initial values and ODEs.

As an overview, our model for each agent i can be summarized as follows (more details in later sections):

$$G_t = f_g(p_t, u_t), \tag{1}$$

$$\mu_{z^i}, \sigma_{z^i} = g_{enc}(x^i_{-T_p:T_f}, x^{-i}_{-T_p:T_f}, G_t), \tag{2}$$

$$z^i \sim N(\mu_{z^i}, \sigma_{z^i}), \tag{3}$$

$$h^i_{-T_p} = f_h(x^i_{-T_p}, z^i), v^i_{-T_p} = f_v(x^i_{-T_p}, z^i), \tag{4}$$

$$h^i_{-T_p}, h^i_{-T_p+1}, ..., h^i_{T_f} = \text{ODESolve}(h^i_{-T_p}, v^i_{-T_p}, g_\theta, t_{-T_p:T_f}, G_{-T_p:T_f}), \tag{5}$$

$$\hat{x}^i_t \sim p(\hat{x}^i_t | h^i_t) \text{ for each time step } t, \tag{6}$$

where Eq 1 denotes dynamic interaction graph construction, Eq 2 is the encoder module, and Eq 3∼6 are the decoder module. In Eq 2, $x^i_{-T_p:0}$ denotes the previous trajectory of agent i, while $x^{-i}_{-T_p:0}$ denotes the previous trajectories of all agents except i. Eq 3 and Eq 4 are used to sample the initial values from the encoder. With these sampled initial values, we then use Eq 5 to solve the ODEs and generate the predicted latent trajectories. Finally, we recover the latent trajectories back to real-world trajectories using Eq 6. The ODESolver is a numerical ODE solver. We solve the ODEs given the function g_θ and the initial values $h^i_{-T_p}, v^i_{-T_p}$.

3.2. Interaction Graph Construction

We use distance and velocity information to construct interaction graphs $G_t = (V_t, E_t)$, with e_{ij} as the edge between agent i and j. Intuitively, the influence of other agents on agent i may change over time. Two agents will not affect each other if they move away from each other ($e_{ij} = 0$). Otherwise if they are moving towards each other, then agent i and agent j are connected ($e_{ij} = 1$). Formally we have:

$$e^t_{ij} = \mathbb{1}[\big((p_j - p_i) \cdot u_i\big) + \big((p_i - p_j) \cdot u_j\big)], \tag{7}$$

where p and u denote position and velocity in the real-world space. $\mathbb{1}(\cdot)$ denotes the unit step function, i.e., $\mathbb{1}(x) = 1$ if $x > 0$ and $\mathbb{1}(x) = 0$ otherwise. If $(p_j - p_i) \cdot u_i$ is larger than 0, it means agent i is moving to agent j. Therefore, if $((p_j - p_i) \cdot u_i) + ((p_i - p_j) \cdot u_j)$ is larger than 0, then two agents are moving closer and e^t_{ij} is set to 1.

3.3. Encoder

The encoder encodes previous real-world trajectories for each agent i into latent vectors (features), which are used as initial values for our ODEs. We use spatial-temporal Transformers to extract features for each agent. Different from the standard attention module, we incorporate interaction graphs in spatial Transformers to model agent interactions more explicitly.

Spatial Transformers. The purpose is to extract agent interaction features among multiple agents. To model agent

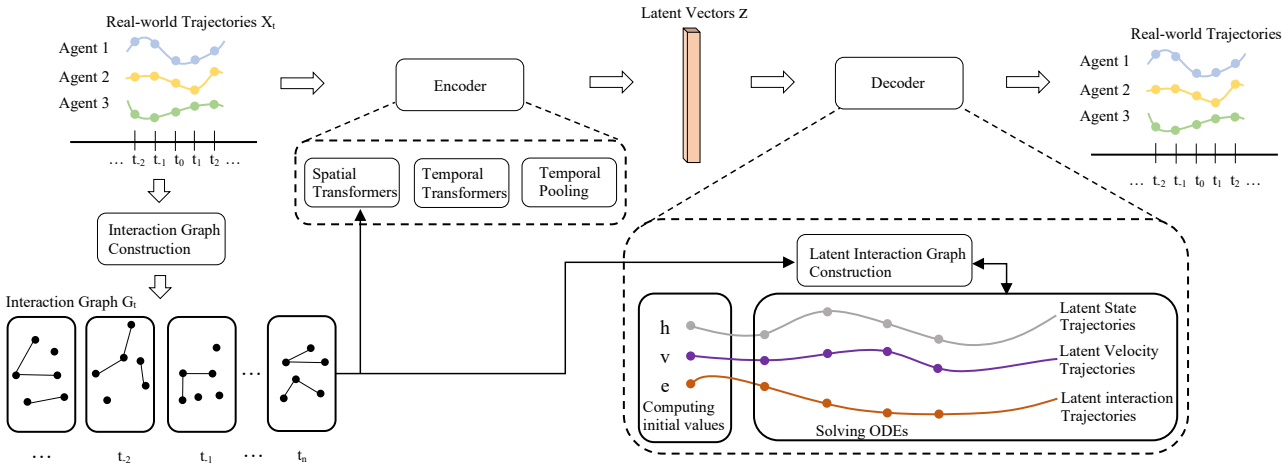


Figure 2. Overview of the proposed Agent Graph ODE, which is composed of graph construction, encoder, and decoder. First, we construct interaction graphs based on real-world distance and velocity information. Then, we use the encoder to produce latent vectors by inputting real-world trajectories. In the latent space, we build and solve ODEs to generate latent trajectories. Finally, the latent trajectories are recovered to real-world trajectories.

i, we consider the interactions between agent i and all other agents at time t. The positions of agent i and agent j are denoted as p^i and p^j, respectively. The feature of layer l is denoted as f_i^l. The attention mechanism works as follows:

$$q_i = w_q^l f_i^l,\ k_i = w_k^l f_i^l, \tag{8}$$

$$m_{ij} = \frac{(p_j - p_i)\cdot u_i + (p_i - p_j)\cdot u_j}{||p^j - p^i||_2^2 + \epsilon} e_{ij}, \tag{9}$$

$$\bar{m}_i = \frac{1}{\sum_j e_{ij}} \sum_j m_{ij}, \tag{10}$$

$$\text{attention}_{\text{ij}} = \frac{e_{ij}\exp(m_{ij} q_i^T k_j / \sqrt{d_k}\bar{m}_{ij})}{\sum_j e_{ij}\exp(m_{ij} q_i^T k_j / \sqrt{d_k}\bar{m}_{ij})}, \tag{11}$$

where m_{ij} denotes the ratio of relative velocity, in the direction along agent i and j, to the respective distance if they are moving to each other. Eq 11 implies that the attention should be large if the relative velocity of moving together is large or the distance is small. Here, ϵ is used to avoid numerical instability leading to large m_{ij}. Similar to $\sqrt{d_k}$, $\bar{m}_i$ is used to scale and prevent the softmax logits from becoming too large. If $e_{ij} = 0$, the attention between these two agents is ignored. We set $m_{ii} = \bar{m}_i$. Therefore, our proposed attention mechanism explicitly incorporates distance and velocity information.

Temporal Transformers. Once the data have been processed by the Spatial Transformers, we obtain a feature sequence for each agent. Each feature represents the embedding of agent interactions at different time steps. We then utilize the Temporal Transformers to encode temporal information for the feature sequence. In the Temporal Transformers, we use the timestamp in the positional encoding.

Following the Spatial-Temporal Transformers described above, we obtain a feature sequence that includes both agent interactions and temporal information. We use average pooling to obtain a single vector for each agent.

3.4. Decoder: Build and Solve ODEs

After sampling the latent vectors and computing value, the decoder builds and solves the ODEs to obtain future latent trajectories.

Second-Order Graph ODEs. We build the second-order ODEs based on graph message passing with interaction graphs $G_t^l = (h_t, a_t, n_t)$ in the latent space, where each node in the graph corresponds to the latent states h_t of an agent and the edges in the graph corresponds to agent interactions a_t. Here, $n_t \in \{0, 1\}$ denotes the connectivity. The agent state and interactions evolve in a coupled manner. Our ODEs and initial values are shown as:

$$\frac{dh_t^i}{dt} = v_t^i, \tag{12}$$

$$\frac{dv_t^i}{dt} = \sum_{j\neq i} \frac{1}{||h_t^i - h_t^j||} a_t^{ij} n_t^{ij} + \lambda_2 f_v(h_t^i), \tag{13}$$

$$\frac{da_t^{ij}}{dt} = f_{a1}(h_t^i, h_t^j, v_t^i, v_t^j) + f_{a2}(a_t^{ij}), \tag{14}$$

$$h_{-T_p}^i = f_{h_0}(x_{-T_p}^i, z^i), \tag{15}$$

$$v_{-T_p}^i = f_{v_0}(x_{-T_p}^i, z^i), \tag{16}$$

$$a_{-T_p}^{ij} = f_{a_0}(h_{-T_p}^i, h_{-T_p}^j, v_{-T_p}^i, v_{-T_p}^j), \tag{17}$$

$$n_t^{ij} = h_n(h_t^i, h_t^j, v_t^i, v_t^j), \tag{18}$$

where h is the latent state for each agent and v is the latent velocity. The evolution of latent state h is modeled by our *second-order ODE*, governed by Eq 12 (*velocity as the first-order dynamics*) and Eq 13 (*acceleration as the second-order dynamics*), while the agent interaction in the latent

space a_{ij} is modeled by the first-order ODE in Eq 14. The agent state h and agent interaction a_{ij} evolve in a coupled manner: the second-order derivative of the agent state, also seen as the acceleration, depends on agent interaction a_{ij}; the equations are inspired by Newton's Second Law. Meanwhile, the derivative of agent interaction also depends on the agent state h. We compute the initial values of these three equations using Eq 15, 16 and 17, where z is the output of the encoder. We also use a classifier in Eq 18 to construct the interaction graph n_{ij} in the latent space, which should be the same as e_{ij} in the real-world space.

We solve the ODEs above to generate latent trajectories $h_{-T_p:T_f}$. Finally, we recover them back to the real-world trajectories.

3.5. Loss Function

Our method builds upon the Conditional Variational Autoencoder (CVAE) model, which aims to approximate the conditional probability $p_\theta(X_{1:T_f}|X_{-T_p:0})$. We employ the negative evidence lower bound (ELBO) L_{elbo} in CVAE as part of our loss function:

$$\begin{aligned} L_{elbo} = &- E_{q_\phi(h_{-T_p}|X_{-T_p:0},X_{1:T_f})}[\log p_\theta(X_{1:T_f}|X_{-T_p:0},h_{-T_p})] \\ &+ KL(q_\phi(h_{-T_p}|X_{-T_p:0},X_{1:T_f})||p_\theta(h_{-T_p}|X_{-T_p:0})). \end{aligned} \tag{19}$$

Here, $q_\phi(h_{-T_p}|X_{-T_p:0},X_{1:T_f})$ denotes the approximate posterior distribution given the whole trajectories, while $p_\theta(h_{-T_p}|X_{-T_p:0})$ denotes the posterior distribution given *only the previous trajectories*. We use $q_\phi(h_{-T_p}|X_{-T_p:0},X_{1:T_f})$ as the encoder module during *training* to generate the initial value h_{-T_p} given the trajectories $X_{-T_p:0},X_{1:T_f}$ and use $p_\theta(h_{-T_p}|X_{-T_p:0})$ the encoder module during *inference* to generate the initial value h_{-T_p} given only the previous trajectories $X_{-T_p:0}$. $p_\theta(X_{1:T_f}|X_{-T_p:0},h_{-T_p})$ denotes the decoder module, which forecasts the future trajectories $X_{1:T_f}$ given previous trajectories $X_{-T_p:0}$ and the initial value h_{-T_p}. We compute the trajectories by solving ODEs (Section 3.4); since this is determined by the initial values and the equations, the future trajectories $X_{1:T_f}$ only depend on the initial value h_{-T_p}. For the decoder, we have

$$\begin{aligned} \log p_\theta(X_{1:T_f}|X_{-T_p:0},h_{-T_p})] &= \log p_\theta(X_{1:T_f}|h_{-T_p})] \\ &= k\sum_i ||\hat{x}^i_{1:T_f} - x^i_{1:T_f}||_2^2, \end{aligned} \tag{20}$$

where $\hat{x}^i_{1:T_f}$ is the estimated future trajectories, and k is the constant (details in supplementary). The second term of Eq 19 means that the encoder should generate a similar initial value h_{-T_p} whether it uses as input only previous trajectories $X_{-T_p:0}$ or the whole trajectories $X_{-T_p:T_f}$, because they belong to the same trajectories.

We assume $q_\phi(h_{-T_p}|X_{-T_p:0},X_{1:T_f}) = N(\mu_q,\sigma_q)$ and $p_\theta(h_{-T_p}|X_{-T_p:0}) = N(\mu_p,\sigma_p)$, where μ_q,σ_q are the output of the encoder given the whole trajectories and μ_p,σ_p are the output given only previous trajectories. We then have

$$\begin{aligned} &KL(q_\phi(h_{-T_p}|X_{-T_p:0},X_{1:T_f})||p_\theta(h_{-T_p}|X_{-T_p:0})) \\ &= -\frac{1}{2}\sum_{j=1}^{J}[\log\frac{\sigma^2_{q,j}}{\sigma^2_{p,j}} - \frac{\sigma^2_{q,j}}{\sigma^2_{p,j}} - \frac{(\mu_{q,j}-\mu_{p,j})^2}{\sigma^2_{p,j}} + 1], \end{aligned} \tag{21}$$

where J is the dimension of the latent initial value h_{-T_p}.

The ELBO in CVAE only considers the future trajectories. To stabilize training and prevent overfitting, our model is also trained to reconstruct the previous trajectories $x^i_{-T_p:0}$ given the initial value in latent space. This leads to an additional loss term using the Mean Squared Error (MSE) between ground-truth previous trajectories $x^i_{-T_p:0}$ and reconstructed previous trajectories $\hat{x}^i_{-T_p:0}$ as follows:

$$L_{mse} = \sum_i ||\hat{x}^i_{-T_p:0} - x^i_{-T_p:0}||_2^2. \tag{22}$$

Additionally, we should also ensure that the dynamic graph in the latent space is consistent with the graph in the real-world space. Denoting as $e^{ij}_t \in \{0,1\}$ the edge between agent i and j in the graph at time t in the real-world space, we then use the Binary Cross-Entropy (BCE) to enforce the graph consistency:

$$L_g = \sum_{i,j,t}[e^{ij}_t \log n^{ij}_t + (1-e^{ij}_t)\log(1-n^{ij}_t)], \tag{23}$$

where $n^{ij}_t \in \{0,1\}$ denotes the edge between agent i and j in the graph at time t in the latent space.

Therefore, the overall loss function is:

$$L = \alpha_1 L_{elbo} + \alpha_2 L_{mse} + \alpha_3 L_g, \tag{24}$$

where α_1, α_2 and α_3 are the coefficients.

4. Experiments

This section presents results that validate the performance of our proposed Agent Graph ODE on the traffic trajectory datasets and the sport dataset.

4.1. Implementation Details

We conducted experiments on four datasets from two distinct areas, transportation and sports, to assess the effectiveness of our proposed method. These datasets present diverse challenges for motion prediction, such as agent behavior variability, interaction complexities, and environmental factors impact. For all evaluated methods, we apply data augmentation techniques, including translation and rotation, and normalize all coordinates to the range of 0 to

1. This approach ensures that the input data is standardized and more amenable to analysis.

During the training phase, we input the whole trajectories $X_{-T_p:T_f}$ and the previous trajectories $X_{-T_p:0}$. The output is estimated whole trajectories $\hat{X}_{-T_p:T_f}$. Conversely, during the inference period, we only input the previous trajectories $X_{-T_p:0}$ to recover the input and forecast the future trajectories $\hat{X}_{1:T_f}$.

4.2. Baselines

We compare our proposed Agent Graph ODE model against state-of-the-art methods including: 1) **Social LSTM** [1], which utilizes social pooling of the hidden states in an LSTM to model interactions between agents. 2) **Social GAN** [11], which combines Generative Adversarial Networks (GAN) with LSTM encoder-decoder to determine whether the predicted trajectories are realistic. 3) **DenseTNT** [10], which uses a graph to model the relationship among agents, with each node in the graph representing a trajectory. 4) **AgentFormer** [31], which is a Transformer-based model based on Conditional Variational Autoencoder (CVAE) and utilizes spatial-temporal attention to jointly extract time and social dimensions. 5) **Social ODE** [27], which is based on neural ODEs and model agent interaction in the latent space.

4.3. Evaluation Metrics

Average Displacement Error. To evaluate the prediction accuracy of our proposed model, we follow common practice in motion prediction and use Average Displacement Error (ADE) as our metric. ADE is calculated as the average Euclidean distance between the predicted positions of the agents and their corresponding ground-truth positions over the prediction horizon:

$$\text{ADE} = \frac{1}{TN} \sum_{t,i} ||x_t^i - \hat{x}_t^i||, \tag{25}$$

where x_t^i denotes the ground-truth position of agent i at time t, and $\hat{x}_t^i$ denotes the corresponding prediction made by the model.

Collision Rate. To assess the extent to which our proposed model learns agent interactions explicitly and how it reacts to sudden situations that were not present in the training data, we introduce new agents as obstacles to the original agent trajectories. We consider it a collision event if the distance between our agent and the obstacle is less than a threshold. The collision rate is the ratio of the number of collision events to the total number of sudden events.

4.4. Traffic Trajectory Data

We use three datasets - inD [4], rounD [17], and highD [16] - which contain naturalistic road user trajectories collected by a drone. Unlike other datasets, these datasets feature varying numbers of agents in the recorded area over time, as some agents may enter or leave the area. To evaluate our method, we split each dataset into 80% for training and validation, and 20% for testing. Each trajectory lasts for 8 seconds with one data point every 0.4 seconds, resulting in 20 points (frames) for agents present throughout the entire trajectory. The input data for our model comprised the trajectories in the first 4 seconds (10 frames), while the ground truth was based on the trajectories in the next 4 seconds (10 frames).

Table 1 shows the results. Trajectories are categorized into two classes: curve and straight. As shown in the table, our proposed Agent Graph ODE outperforms Social LSTM, Social GAN and Social ODE, in all time lengths. When forecasting for longer periods, such as 4 seconds and 8 seconds, our model outperforms all other methods in curve trajectories, indicating its ability to handle more complex scenarios.

4.5. Sports Data

We conducted additional experiments on the NBA SportVU dataset, which consists of player and ball trajectories. In contrast to the naturalistic road user trajectories dataset, the number of agents in the NBA dataset remains constant throughout the dataset. However, sports agents exhibit more frequent changes, and the models therefore need to handle more complex situations. We input a 2-second interval and predict movements for the subsequent 2 to 4 seconds. As seen in Table 2, our proposed Agent Graph ODE outperforms other methods when forecasting for 4 seconds. This result verifies the capability of our model to handle more complex scenarios.

4.6. Sudden Obstacle

In many real-world scenarios, the number of agents present in a given environment can change over time. Unfortunately, most previous methods for motion prediction assume a constant number of agents; this limits their applicability in certain situations. For example, in the context of autonomous driving, the sudden appearance of a pedestrian or cyclist on the road can significantly impact the behavior of other agents, such as vehicles or other pedestrians. To evaluate the effectiveness of our proposed Agent Graph ODE in such situations, we conduct an experiment using instances from the test dataset. In this experiment, we place a static or moving agent in the predicted trajectory and make the moving agent move directly towards the agent being modeled, simulating an obstacle. We then measure the collision rate of each method.

Results in Table 3 show that our proposed Agent Graph ODE model achieves the lowest collision rate when encountering both static and moving obstacles. This suggests that our model is effective in extracting agent interactions and

Method	length	inD		highD		rounD	
		straight	curve	straight	curve	straight	curve
Social LSTM	2s	0.2474	0.8537	0.2846	0.8347	0.2367	0.8986
Social GAN		0.2537	0.8236	0.2564	0.8977	0.2679	0.8876
DenseTNT		0.2367	**0.8046**	0.2465	0.8546	0.2268	0.8464
AgentFomer		**0.2346**	0.8124	**0.2368**	0.8263	**0.2140**	**0.8259**
Social ODE		0.2408	0.8147	0.2406	0.8135	0.2254	0.8357
Agent Graph ODE		0.2389	0.8101	0.2402	**0.8123**	0.2225	0.8316
Social LSTM	4s	0.7973	3.1463	0.9525	3.5364	0.7268	2.6473
Social GAN		0.7861	3.1583	0.8367	3.4637	0.7483	2.6940
DenseTNT		0.7794	3.1578	0.7431	3.1778	0.6543	2.4764
AgentFomer		0.7604	3.1483	**0.6814**	3.1527	**0.5924**	2.4748
Social ODE		0.7728	3.1417	0.6873	3.1509	0.6005	2.4738
Agent Graph ODE		**0.7581**	**3.1402**	0.6831	**3.1487**	0.6001	**2.4733**
Social LSTM	8s	2.7536	8.3456	2.4570	9.3365	2.5583	9.1346
Social GAN		2.6573	8.2478	2.3279	9.6437	2.9546	8.9446
DenseTNT		2.6644	8.1475	2.1345	9.3464	2.7854	8.4677
AgentFomer		2.3474	8.1457	**2.1167**	9.3258	**2.5337**	8.3464
Social ODE		2.6064	8.1208	2.1384	9.3203	2.6447	8.3384
Agent Graph ODE		**2.3412**	**8.1030**	2.1251	**9.3116**	2.5538	**8.3127**

Table 1. Evaluation on inD, rounD and highD traffic datasets. The results in bold indicate the best performance.

Method	2s	4s
Social LSTM	0.9738	2.7441
Social GAN	0.9537	2.6318
DenseTNT	0.8442	2.4873
AgentFomer	0.7384	2.3427
Social ODE	0.7393	2.0511
Agent Graph ODE	**0.7323**	**1.9402**

Table 2. Evaluation on NBA SportVU dataset. The results in bold indicate the best performance.

can adapt to sudden situations not seen in the training data. Furthermore, Figure 3 provides examples of how different models handle the obstacle avoidance task. It shows that only the Agent Graph ODE model can successfully avoid the static obstacle, demonstrating its ability to capture complex agent interactions and make informed decisions in situations not seen in the training data.

4.7. Ablation Study

In this section, we evaluate the effectiveness of the proposed Agent Graph ODE model by conducting a set of ablation studies. Specifically, we study each component of our method:

Dynamic Graph. We examine the dynamic graph generation module by comparing the performance with and without it. In the absence of the dynamic graph generation module, we use the complete graph at each time step, which considers the interaction between any two agents. As shown in Table 4, we observe a decrease in performance without the dynamic graph generation module, highlighting the importance of this module in capturing the changing relationships between agents.

First-Order ODEs. We investigate the impact of using second-order ODEs for modeling agent interactions. We compare the performance of our proposed Agent Graph ODE model using first-order ODEs and second-order ODEs. Instead of Eq 12 $\sim$ Eq 14, the equation for the first-order model is:

$$\frac{dh_t^i}{dt} = \sum_{j \neq i} \frac{1}{||h_t^i - h_t^j||} a_t^{ij} + \lambda_2 f_h(h_t^i), \tag{26}$$

$$\frac{da_t^{ij}}{dt} = f_{a1}(h_t^i, h_t^j) + f_{a2}(a_t^{ij}), \tag{27}$$

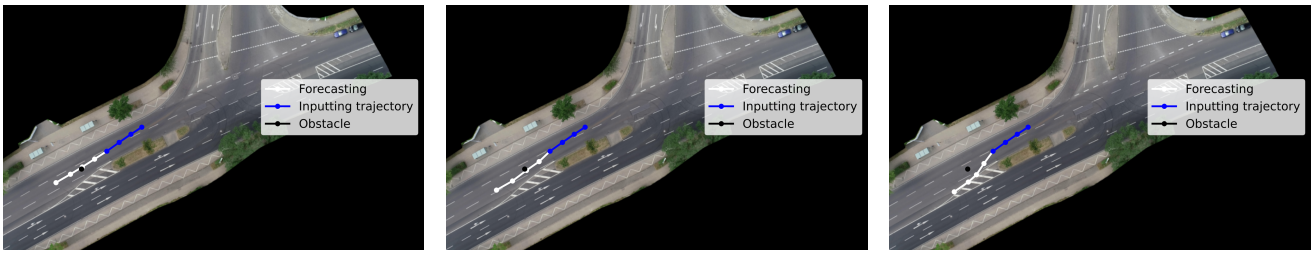
$$a_{-T_p}^{ij} = f_{a_0}(h_{-T_p}^i, h_{-T_p}^j). \tag{28}$$

The results in Table 4 show that the second-order ODEs lead to better performance.

Third-Order ODEs. We extend our model to higher-order ODEs. Specifically, we upgrade our model to the third order by modifying Eq 12 to $\frac{dh_t^i}{dt} = \dot{h}_t^i$ and $\frac{d\dot{h}_t^i}{dt} = v_t^i$ while maintaining Eq 13 $\sim$ Eq 18 unchanged. The results presented in Table 4 reveal that the performance of the third-order model is inferior to that of its second-order counterparts. This suggests that second-order ODEs more accurately capture temporal dynamics and agent interactions.

Method	Social LSTM	Social GAN	DenseTNT	AgentFormer	Social ODE	Agent Graph ODE
Static obstacle	17.4%	19.2%	15.2%	16.8%	7.6%	**7.0%**
Moving obstacle	21.2%	23.4%	18.8%	20.6%	8.4%	**7.8%**

Table 3. The collision rate when introducing a sudden obstacle into the trajectory. Bold text indicates the best performing method.



(a) AgentFormer (b) Social ODE (c) Agent Graph ODE

Figure 3. Visualization of a sudden obstacle scenario. Each image shows the input trajectory in blue and the predicted trajectory in white. The black point represents the obstacle that suddenly appears. In this scenario, Agent Graph ODE can avoid the obstacle, while the other methods fail to do so.

Method	inD		highD		rounD	
	straight	curve	straight	curve	straight	curve
w/o dynamic graph	0.7735	3.1647	0.7149	3.2357	0.6327	2.5380
w/ first-order ODEs	0.7602	3.1411	0.6852	3.1503	0.6048	2.4863
w/ third-order ODEs	0.7662	3.1516	0.6893	3.1723	0.6072	2.4885
w/o distance information in Transformers	0.7596	3.1409	0.6845	3.1492	0.6002	2.4737
w/o graph consistency in loss function	0.7731	3.1620	0.7018	3.1968	0.6269	2.5024
Agent Graph ODE	**0.7581**	**3.1402**	**0.6831**	**3.1487**	**0.6001**	**2.4733**

Table 4. Assessment of the impact of varying certain components on the inD, rounD, and highD traffic datasets. The prediction horizon is 4 seconds. The best performance is marked in boldface.

Distance Information in Transformers. We also study the effect of adding distance information into the spatial Transformer module by comparing the performance with and without distance information. The results in Table 4 demonstrate that adding distance information improves the prediction accuracy.

Graph Consistency in the Loss Function. We study the effect of graph consistency in the loss function by comparing the performance of our proposed model with and without the graph consistency loss. We change the loss function in Eq 24 to $L = \alpha_1 L_{elbo} + \alpha_2 L_{mse}$ for the ablated model. The results in Table 4 show that the graph consistency loss indeed improves the performance.

5. Conclusion

In this paper, we have identified several limitations in existing multi-agent trajectory forecasting methods. Specifically, we observed that previous methods often overlook distance and velocity information to learn agent interactions. We also observed that earlier ODE-based methods often rely on first-order modeling of temporal dynamics, which fails to capture the motion physics. To address these limitations, we then propose Agent Graph ODE, a VAE architecture that explicitly models agent interactions and temporal dynamics. We begin by constructing dynamic interaction graphs based on distance and velocity in real-world space. These graphs are then incorporated in the spatial Transformers in the encoder and Graph ODEs in the decoder. Furthermore, we utilize second-order ODEs to model temporal dynamics, which adhere to Newton's Second Law and also integrate distance and velocity information. Our experiments on traffic trajectory and sports datasets demonstrate that the proposed Agent Graph ODE outperforms other methods in terms of forecasting accuracy, particularly in complex environments. Additionally, it exhibits superior capability in effectively handling sudden situations.

Acknowledgements: Research partially funded by research grants to Metaxas from NSF: 2310966, 2235405, 2212301, 2003874, 1951890, AFOSR 23RT0630 and NIH 2R01HL127661

References

[1] Alexandre Alahi, Kratarth Goel, Vignesh Ramanathan, Alexandre Robicquet, Li Fei-Fei, and Silvio Savarese. Social lstm: Human trajectory prediction in crowded spaces. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 961–971, 2016. 1, 6

[2] Inhwan Bae and Hae-Gon Jeon. A set of control points conditioned pedestrian trajectory prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 6155–6165, 2023. 2

[3] Natàlia Balague, Carlota Torrents, Robert Hristovski, Keith Davids, and Duarte Araújo. Overview of complex systems in sport. *Journal of Systems Science and Complexity*, 26:4–13, 2013. 1

[4] Julian Bock, Robert Krajewski, Tobias Moers, Steffen Runde, Lennart Vater, and Lutz Eckstein. The ind dataset: A drone dataset of naturalistic road user trajectories at german intersections. In *2020 IEEE Intelligent Vehicles Symposium (IV)*, pages 1929–1934, 2020. 6

[5] Johann Brehmer and Kyle Cranmer. Flows for simultaneous manifold learning and density estimation. *Advances in Neural Information Processing Systems*, 33:442–453, 2020. 2

[6] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018. 2

[7] Edward De Brouwer, Jaak Simm, Adam Arany, and Yves Moreau. Gru-ode-bayes: Continuous modeling of sporadically-observed time series. *Advances in neural information processing systems*, 32, 2019. 2

[8] Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. Augmented neural odes. *Advances in Neural Information Processing Systems*, 32, 2019. 2

[9] Colin Graber and Alexander Schwing. Dynamic neural relational inference for forecasting trajectories. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 1018–1019, 2020. 2

[10] Junru Gu, Chen Sun, and Hang Zhao. Densetnt: End-to-end trajectory prediction from dense goal sets. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 15303–15312, 2021. 6

[11] Agrim Gupta, Justin Johnson, Li Fei-Fei, Silvio Savarese, and Alexandre Alahi. Social gan: Socially acceptable trajectories with generative adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2255–2264, 2018. 6

[12] Zijie Huang, Yizhou Sun, and Wei Wang. Coupled graph ode for learning interacting system dynamics. In *The 27th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 2021. 1, 2

[13] Zijie Huang, Yizhou Sun, and Wei Wang. Generalizing graph ode for learning complex system dynamics across environments. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 798–809, 2023. 2

[14] Boris Ivanovic, Amine Elhafsi, Guy Rosman, Adrien Gaidon, and Marco Pavone. Mats: An interpretable trajectory forecasting representation for planning and control. In *Conference on Robot Learning*, pages 2243–2256. PMLR, 2021. 1

[15] Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. Neural relational inference for interacting systems. In *International Conference on Machine Learning*, pages 2688–2697. PMLR, 2018. 1, 2

[16] Robert Krajewski, Julian Bock, Laurent Kloeker, and Lutz Eckstein. The highd dataset: A drone dataset of naturalistic vehicle trajectories on german highways for validation of highly automated driving systems. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 2118–2125, 2018. 6

[17] Robert Krajewski, Tobias Moers, Julian Bock, Lennart Vater, and Lutz Eckstein. The round dataset: A drone dataset of road user trajectories at roundabouts in germany. In *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–6, 2020. 6

[18] Jiachen Li, Fan Yang, Masayoshi Tomizuka, and Chiho Choi. Evolvegraph: Multi-agent trajectory prediction with dynamic relational reasoning. *Advances in neural information processing systems*, 33:19783–19794, 2020. 1, 2

[19] Yuxuan Liang, Kun Ouyang, Hanshu Yan, Yiwei Wang, Zekun Tong, and Roger Zimmermann. Modeling trajectories with neural ordinary differential equations. In *IJCAI*, pages 1498–1504, 2021. 2

[20] Yang Liu, Jiashun Cheng, Haihong Zhao, Tingyang Xu, Peilin Zhao, Fugee Tsung, Jia Li, and Yu Rong. Physics-inspired neural graph ode for long-term dynamical simulation. *arXiv preprint arXiv:2308.13212*, 2023. 2

[21] Yulia Rubanova, Ricky TQ Chen, and David K Duvenaud. Latent ordinary differential equations for irregularly-sampled time series. *Advances in neural information processing systems*, 32, 2019. 2

[22] Tim Salzmann, Boris Ivanovic, Punarjay Chakravarty, and Marco Pavone. Trajectron++: Dynamically-feasible trajectory forecasting with heterogeneous data. In *European Conference on Computer Vision*, pages 683–700. Springer, 2020. 2

[23] Ruian Shi and Quaid Morris. Segmenting hybrid trajectories using latent odes. In *International Conference on Machine Learning*, pages 9569–9579. PMLR, 2021. 2

[24] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*, 2020. 2

[25] Chen Sun, Per Karlsson, Jiajun Wu, Joshua B Tenenbaum, and Kevin Murphy. Stochastic prediction of multi-agent interactions from partial observations. *arXiv preprint arXiv:1902.09641*, 2019. 2

[26] Charles Vorbach, Ramin Hasani, Alexander Amini, Mathias Lechner, and Daniela Rus. Causal navigation by continuous-time neural networks. *Advances in Neural Information Processing Systems*, 34, 2021. 2

[27] Song Wen, Hao Wang, and Dimitris Metaxas. 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*, pages 217–233. Springer, 2022. 2, 6

[28] Raymond A Yeh, Alexander G Schwing, Jonathan Huang, and Kevin Murphy. Diverse generation for multi-agent sports games. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4610–4619, 2019. 2

[29] Cagatay Yildiz, Markus Heinonen, and Harri Lahdesmaki. Ode2vae: Deep generative second order odes with bayesian neural networks. *Advances in Neural Information Processing Systems*, 32, 2019. 2

[30] Cagatay Yildiz, Melih Kandemir, and Barbara Rakitsch. Learning interacting dynamical systems with latent gaussian process odes. In *Advances in Neural Information Processing Systems*, 2022. 2

[31] Ye Yuan, Xinshuo Weng, Yanglan Ou, and Kris M Kitani. Agentformer: Agent-aware transformers for socio-temporal multi-agent forecasting. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9813–9823, 2021. 1, 2, 6