Probabilistic Continuous-Time Whole-Graph Forecasting

Zhihan Gao*
HKUST
Hong Kong
zhihan.gao@connect.ust.hk

Hao Wang
Rutgers University
USA
hw488@cs.rutgers.edu

Yuyang Wang
Amazon Web Services
USA
yuyawang@amazon.com

Xingjian Shi
Amazon Web Services
USA
xjshi@amazon.com

Dit-Yan Yeung
HKUST
Hong Kong
dyyeung@cse.ust.hk

ABSTRACT

Dynamic graph forecasting has found a wide range of applications including social media, recommendation systems, and computational finance. However, existing dynamic graph models typically focus on discrete-time dynamic graphs, treating dynamic graphs as temporally discrete graph snapshots. We argue that such discrete treatment is inadequate for capturing the underlying dynamics which are intrinsically continuous. To overcome such deficiency, we extend fully connected neural ordinary differential equations (FC-NODE) to graph-connected neural ordinary differential equations (GNODE), which considers graph structures in the input space, output space, and the transition in the latent space. Experiments show that our GNODE naturally captures the continuous-time dynamics in graph sequences and consistently outperforms state-of-the-art graph forecasting methods.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**.

KEYWORDS

graph, time series, Bayesian learning

ACM Reference Format:
Zhihan Gao, Hao Wang, Yuyang Wang, Xingjian Shi, and Dit-Yan Yeung. 2022. Probabilistic Continuous-Time Whole-Graph Forecasting. In *Proceedings of 8th SIGKDD International Workshop on Mining and Learning from Time Series – Deep Forecasting: Models, Interpretability, and Applications, 2022 (SIGKDD-MiLeTS '22).* ACM, New York, NY, USA, 14 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Graph neural networks (GNNs) are widely applied to graph-related tasks and has achieved many successes [5, 9, 42, 45, 46]. While the majority of these work focus on static graphs, dynamically evolving graphs are gaining more and more attention in recent years [16, 17] due to its close connection to numerous real-world applications. For instance, in social media, communication events such as emails and text messages are streaming while friendship relations evolve over time. In recommendation systems, new products, new users and new ratings appear every day. In computational finance, transactions happen at every second and supply chain relations are continuously changing. Learning an effective representation of the temporal evolution of the graphs is essential for understanding the underlying dynamics.

However, most existing research on dynamic graphs have been focusing on discrete-time dynamic graphs (DTDGs). Typically, they treat a dynamic graph as a sequence of graph snapshots and rely on the combination of recurrent neural networks (RNNs) and GNNs to perform discrete-time dynamic graph forecasting [8, 27, 37]. Specifically, a GNN will generate node embeddings according to the current graph snapshot, which are then fed into an RNN to handle the dynamics. We argue that while graphs are intrinsically discrete structures, the embeddings of nodes in the graph are actually continuously evolving. Therefore it is desirable to extend DTDG to a continuous setting, which we call continuous-time dynamic graphs (CTDGs). Unfortunately, discrete methods above, while powerful for modeling DTDG, fail to handle continuous-time dynamic graphs (CTDGs).

On the other hand, continuous-time sequence modeling has also gained much attention in recent years with the success of Neural Ordinary Differential Equation (Neural ODEs) [2, 13, 18]. Neural ODE generalizes discrete-time state transitions in RNNs to continuous-time ones, enabling a neural network to parameterize the continuous trajectory of an ODE solution. A Neural ODE can be trained the same way as conventional neural networks, i.e., can be trained end-to-end efficiently with stochastic gradient descent optimization.

In this work, instead of simply modeling graph sequences as a series of discrete *AddEdge* events, i.e., events of adding an edge to the graph, we take a step further to enable dynamic graph forecasting in a continuous-time fashion. Inspired by the efficient continuous-time sequence modeling of neural ODE, we propose a novel and general architecture, dubbed graph-connected neural ordinary differential equation (GNODE), to learn the continuous latent dynamics of evolving graphs. The key idea is to extend the fully connected neural ODE (FC-NODE) to graph-connected neural ODE (GNODE), which considers graph structures in the input space, output space, and more importantly the continuous transition in the latent space.

*Work done while being an intern at Amazon Web Services.

SIGKDD-MiLeTS '22, August 15, 2022, Convention Center, Washington DC

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/XXXXXXX.XXXXXXX

Intuitively GNODE allows nodes in a graph to *continuously* interact with each other, thereby *continuously* changing their node embeddings. Therefore, GNODE is capable of handling more general CTDGs and modeling the underlying continuous dynamics which DTDG models can not model. We further extend GNODE to construct an GNODE-augmented RNN encoder, which we call GNODE^e. Together with the proposed vanilla GNODE as a decoder, our full model can achieve further performance boost.

We summarize our contributions as follows:

- We identify the problem of continuous-time whole-graph forecasting and propose GNODE to address this problem.
- We propose a technique, dubbed decoded adjacency matrices, to update the adjacency matrix continuously and simultaneously with the evolution of node latent representations.
- We evaluate GNODE on continuous-time dynamic graph forecasting tasks and demonstrate the superiority of GNODE over state-of-the-art methods.

2 RELATED WORKS

Discrete-Time Dynamic Graphs. There is a rich literature on learning and modeling discrete-time dynamic graphs (DTDG). Typically, they treat a dynamic graph as a sequence of graph snapshots and rely on the combination of recurrent neural networks (RNN) and GNN to perform discrete-time dynamic graph forecasting [8, 27, 37]. We categorize them into three groups: stacked graph recurrent neural networks (stacked GRNN or SGRNN), integrated graph recurrent neural networks (integrated GRNN or IGRNN), and graph neural networks with other temporal constraints. SGRNN extracts features from each snapshot directly using a static graph model and then feed them into a Recurrent Neural Network (RNN) [24, 27, 38, 40]. Another group of models, IGRNN, replaces the fully-connected input-to-state and state-to-state transitions with graph convolutions [1, 8, 26, 38]. The third group of approaches impose temporal constraints besides using RNN: DySAT [37] uses spatial attention and temporal attention mechanism; DynGEM [6] uses Net2Net [3] to add constraints on the architecture of the static graph model for each snapshot; Evolve GCN [31] updates the parameters of the static graph model using RNN.

Timestamp-Augmented Graph RNNs. Besides the typical combination of RNN and GNN, there have been some recent works that replace traditional discrete-time RNN with either timestamp-augmented RNN or stochastic processes to better incorporate available timestamps from graph snapshots. However, these models represent graph sequences as a series of '*AddEdge*' event, i.e., the event of adding a single edge at a certain non-overlapping timestamp, and directly use continuous-time RNNs or stochastic processes to model such sequences [14, 15, 27, 29, 41]. However, such methods are rather limited because they (1) fail to handle simultaneous appearances of multiple edges since timestamps are non-overlapping, (2) fail to model the removal of edges, which is prevalent in practice, and (3) cannot effectively capture the interactions between nodes described by the dynamic graphs. Therefore they are not suitable for forecasting continuous-time dynamic graphs.

Neural ODE for Graphs. There are some recent works that integrate neural ODE into GCNs. Continuous graph neural networks (CGNN) [43] generalize multi-layer GCNs with residual connections by replacing discrete layers with neural ODE. There is also theoretical analysis on the behavior of the CGNN at the limit of the number of layers going to infinity. Ordinary differential equations on graph networks (GODE) [47] combine GCN with neural ODE in the same way as CGNN, while proposing a new gradient estimation algorithm to reduce the numerical errors of the adjoint sensitivity method. Graph neural ordinary differential equations (GDE) [32] apply the combination of GCN and neural ODE to both static and dynamic graphs. However when dealing with dynamic graphs with non-stationary topological structures, GDE takes fixed discrete steps and degenerates into discrete GRNNs with small time interval. Therefore none of the methods above is capable of performing continuous-time dynamic graph forecasting, which is the focus of our proposed method.

3 PRELIMINARIES

Notation. In the following we use bold-face upper-case letters, such as $\boldsymbol{Z}$, to denote matrices, and bold-face lower-case letters, such as $\boldsymbol{v}$, to denote vectors. We use $\boldsymbol{A} \in \{0,1\}^{N\times N}$ to denote the adjacency matrix of N nodes, $\boldsymbol{X} \in \mathbb{R}^{N\times D}$ to denote the D-dimensional input features of these N nodes, and $\boldsymbol{Z} \in \mathbb{R}^{N\times C}$ to denote their corresponding C-dimensional latent representations. We use $\boldsymbol{Z}^{(i)}$ to denote the i'th discrete graph snapshot in the DTDG setting and use $\boldsymbol{Z}(t)$ to denote the graph snapshot at timestamp t in the CTDG setting. We use 'hat' such as $\widehat{G}$ to denote the output predictions from models. For recurrent update in RNNs and GRNNs, we use 'tilde' such as $\widetilde{\boldsymbol{Z}}^{(i)}$ to denote the external input feature at the i'th step, to distinguish it from the latent state $\boldsymbol{Z}^{(i)}$ after the i'th recurrent update.

Problem Setup. We use a sequence of I graph-timestamp pairs to represent a dynamic graph,

$$\mathcal{G} = \{(G^{(i)}, t_i)\}_i, \quad i = 1, 2, \cdots, I,$$

where $G^{(i)}$ is the graph snapshot at time t_i. Note that such continuous representations contain exact temporal information; this is in contrast to the discrete representation $\mathcal{G} = \{G^{(1)}, G^{(2)}, \ldots, G^{(I)}\}$, which only contains the temporal order of the graph snapshots. Continuous representations of graph sequences are particularly important when the time intervals are unevenly spaced, i.e., $t_i - t_{i-1} \neq t_j - t_{j-1}$, in which case DTDG methods often fail to effectively capture the continuous dynamics of graph sequences. Other kinds of continuous representations [10, 11, 28] such as event-based representations and contact sequence representations can also be converted to this kind of general representation without loss of information.

In this work, we focus on the multi-step graph forecasting (prediction) task, the performance of which can also evaluate how well different methods capture the continuous evolution of dynamic graphs. We formulate the task as follows: given an observation sequence $\mathcal{G}_{\text{in}} = \{(G^{(i)}, t_i)\}_i$, we are interested in how the dynamics would evolve in the future $t' > t_I$, where t' denotes future timestamps. Specifically, the goal is to predict the most likely graphs at

multiple specified future timestamps $\{t'_{i'}\}_{i'}$:

$$\{\widehat{G}^{(i')}\}_{i'} = \underset{\{G^{(i')}\}_{i'}}{\operatorname{argmax}}\, p(\{G^{(i')}\}_{i'}|\{(G^{(i)}, t_i)\}_i, \{t'_{i'}\}_{i'}), \tag{1}$$

where p denotes the predictive distribution of the future graph sequences.

GCN. Graph convolutional networks (GCN) [21] are extension of convolutional neural networks to the graph domain. Given a graph, each layer in GCN essentially aggregates the features from the local neighbors to each node, followed by typical linear and nonlinear transformation of these features. We summarize the operations of a GCN as:

$$\boldsymbol{Z} = \mathrm{GCN}(G) = \mathrm{GCN}(\boldsymbol{X}, \boldsymbol{A}). \tag{2}$$

Given a graph $G = (\boldsymbol{A}, \boldsymbol{X})$ represented by its adjacency matrix $\boldsymbol{A} \in \{0,1\}^{N\times N}$ and the node features $\boldsymbol{X} \in \mathbb{R}^{N\times D}$, as shown in Eqn. 2, GCN embeds $\mathbf{X}$ to corresponding latent representations $\boldsymbol{Z} \in \mathbb{R}^{N\times C}$. The latent state $\boldsymbol{Z}$ is expected to contain the information about the original graph, especially its topological structure. A graph decoder, denoted as $D_g(\cdot)$ below and usually taking the form of an inner product function or a multi-layer perceptron (MLP), is able to reconstruct the graph from the latent representation $\boldsymbol{Z}$ as follows:

$$p(\widehat{G}|\boldsymbol{Z}) = D_g(\boldsymbol{Z}). \tag{3}$$

Two GRNN Variants: SGRNN and IGRNN. GCN and its variants are models for static graphs, while in most DTDG problems, dynamic graphs are represented by a sequence of graph snapshots $\{G^{(i)}\}_i$. Therefore, a straightforward solution of dealing with such a graph snapshot sequence is to combine GCN with RNN to first encode each graph snapshot into a latent state and then process the sequence of latent states using RNN. As Eqn. 4 illustrates, the RNN can be a vanilla RNN, GRU, LSTM, etc. We refer to this approach as stacked graph RNN (i.e., SGRNN in Sec. 'Related Works') [39]. In single recurrent step, the input graph G is encoded as $\widetilde{\boldsymbol{Z}} = \mathrm{GCN}(G^{(i)}) \in \mathbb{R}^{N\times C}$, flattened into a vector $\mathrm{vec}(\widetilde{\boldsymbol{Z}}) \in \mathbb{R}^{NC}$, and then updated to $\boldsymbol{Z}$ from step $i-1$ to i through fully-connected (FC) operations:

$$\begin{aligned}\boldsymbol{Z}^{(i)} &= \mathrm{RNN}(\boldsymbol{Z}^{(i-1)}, \widetilde{\boldsymbol{Z}}^{(i)}) \\ &= \sigma(\boldsymbol{W}\cdot\boldsymbol{Z}^{(i-1)} + \boldsymbol{U}\cdot\mathrm{vec}(\widetilde{\boldsymbol{Z}}^{(i)})).\end{aligned} \tag{4}$$

In contrast to SGRNN that uses graph convolution only for input-to-state transitions (but uses fully connected state-to-state transitions), IGRNN uses graph convolutions for both input-to-state and state-to-state transitions, as shown in Eqn. 5 below:

$$\begin{aligned}\boldsymbol{Z}^{(i)} &= \mathrm{IGRNN}(\boldsymbol{Z}^{(i-1)}, \widetilde{\boldsymbol{Z}}^{(i)}, \boldsymbol{A}^{(i)}) \\ &= \sigma(\mathrm{GCN}_{\boldsymbol{W}}(\boldsymbol{Z}^{(i-1)}, \boldsymbol{A}^{(i)}) + \mathrm{GCN}_{\boldsymbol{U}}(\widetilde{\boldsymbol{Z}}^{(i)}, \boldsymbol{A}^{(i)})).\end{aligned} \tag{5}$$

Neural ODE. To some degree, neural ODE [2] can be seen as continuous-time extension to traditional RNN, which can only handle discrete-time sequences by iteratively updating the latent states as:

$$\boldsymbol{Z}^{(i)} = \boldsymbol{Z}^{(i-1)} + f(\boldsymbol{Z}^{(i-1)}, t_i - t_{i-1}). \tag{6}$$

where $\boldsymbol{Z}^{(i)}$ is the latent state at time t_i. As the time interval $t_i - t_{i-1}$ gets smaller, the trajectory of the latent state gets more fine-grained, with more but smaller discrete updates. In the limit, the trajectory of the latent states becomes continuous, which is specified by a neural ODE:

$$\frac{d\boldsymbol{Z}(t)}{dt} = f(\boldsymbol{Z}(t), t). \tag{7}$$

The derivative of $\boldsymbol{Z}$ w.r.t. t is parameterized by a neural network $f(\cdot)$. By solving the ODE initial value problem using ODE solvers [7, 23, 36], the value of $\boldsymbol{Z}(t)$ is available at any timestamp t. The adjoint sensitivity method [33] efficiently computes the gradients of the network parameters and hence allows training Neural ODE using stochastic gradient descent optimization algorithms.

4 METHOD

In this section, we present our proposed GNODE. We start with an overview of our model, introduce the core GNODE model in detail, and then briefly discuss how GNODE can be integrated into GRNN to construct a GNODE-augmented GRNN encoder for continuous-time graph sequences.

4.1 Overview

We adopt a variational autoencoder architecture to model the conditional distribution of the future graphs given the observation graph-timestamp pairs, i.e., $p(\{G^{(i')}\}_{i'}|\{(G^{(i)}, t_i)\}_i, \{t'_{i'}\}_{i'})$. Fig. 1 shows an overview of our method. It mainly consists of an encoder and a decoder. The encoder GNODE^e is a hybrid of GNODE and GRNN (details in Sec. 'GNODE^e: A GNODE-Augmented GRNN Encoder'); the decoder is a GNODE (details in Sec. 'Graph Neural Ordinary Differential Equation'). The equations below formally describe the generative process from the observations $\{G^{(i)}, t_i\}_i$ to the predictions $\widehat{G}^{(i')}$ step by step:

$$\boldsymbol{Z}(t_I) = \mathrm{GNODE}^e(\{(G^{(i)}, t_i)\}_i), \tag{8}$$

$$\boldsymbol{Z}(t'_{0'}) \sim q(\boldsymbol{Z}(t'_{0'})|\{(G^{(i)}, t_i)\}_i) = q(\boldsymbol{Z}(t'_{0'})|\boldsymbol{Z}(t_I)), \tag{9}$$

$$\boldsymbol{Z}(t') = \mathrm{GNODE}(\boldsymbol{Z}(t'_{0'}), t'), \tag{10}$$

$$\widehat{G}^{(i')} \sim p(G^{(i')}|\boldsymbol{Z}(t'_{i'})) = p(G^{(i')}|\boldsymbol{Z}(t')). \tag{11}$$

The encoder GNODE^e first encodes the observation sequence $\{(G^{(i)}, t_i)\}_i$ into latent states $\boldsymbol{Z}(t_I)$ (Eqn. 8). $\boldsymbol{Z}(t_I)$ then parameterizes the variational distribution $q(\boldsymbol{Z}(t'_{0'})|\{G^{(i)}, t_i\}_i)$, from which the future initial latent states are sampled (Eqn. 9). Next, the sampled initial latent states are fed into the decoder GNODE to predict the future evolution of the latent states $\boldsymbol{Z}(t')$ (Eqn. 10). Here $\boldsymbol{Z}(t')$ is a continuous trajectory, where the value of $\boldsymbol{Z}(t'_{i'})$ is always available for any $t'_{i'} > t_I$; a probabilistic graph decoder then generates the graph from the final latent states from the conditional distribution $p(G^{(i')}|\boldsymbol{Z}(t'_{i'}))$ (Eqn. 11 and Eqn. 3).

Note that the ground-truth graphs $\{G^{(i')}\}_{i'}$ at specified timestamps $\{t'_{i'}\}_{i'}$ are available for training. We maximize the evidence lower bound (ELBO) in Eqn. 12 below to jointly train all model

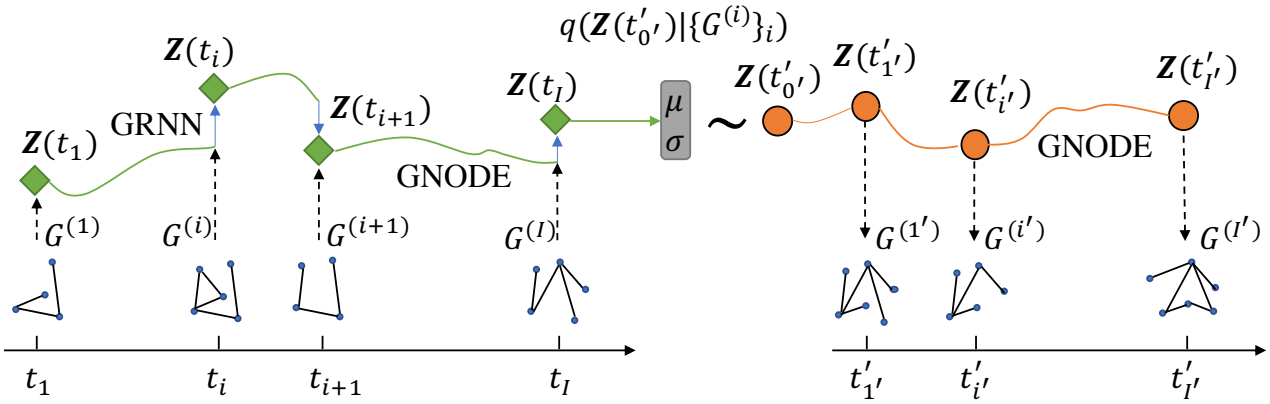

Figure 1: An encoder-decoder architecture for continuous-time dynamic graph forecasting. The green part is the GNODE^e encoder, the orange part is the GNODE decoder. The curves in both the encoder and decoder parts are the continuous trajectories modeled by GNODEs. The blue arrows are discrete GRNN updates in the GNODE^e encoder.

components:

$$\text{ELBO} = \mathbb{E}_{Z(t'_{0'})\sim q(Z(t'_{0'})|\{(G^{(i)},t_i)\}_i)}\Big(\sum_{i'} \log p(G^{(i')}|Z(t'_{i'}))\Big) - \text{KL}(\log q(Z(t'_{0'})|\{(G^{(i)},t_i)\}_i)\|p(Z(t'_{0'}))). \tag{12}$$

In practice, the variational distribution $q(Z(t'_{0'})|Z(t_I))$ is parameterized as a Gaussian distribution:

$$q(Z(t'_{0'})|Z(t_I)) = \mathcal{N}(Z(t'_{0'})|\mu(Z(t_I)), \sigma(Z(t_I))),$$

where the mean $\mu(Z(t_I))$ and the standard deviation $\sigma(Z(t_I))$ are the calculated by two FC layers respectively. The model is trained using the reparameterization trick [20]. Below, we introduce the encoder part and the decoder part in detail respectively.

4.2 Graph Neural Ordinary Differential Equation

Motivation. The most important components that distinguish our architecture from others, as shown in Fig. 1, are continuous trajectories in both encoder (in green) and decoder (in orange) part. Both are modeled by our proposed graph neural ordinary differential equations (GNODE). Different from the original work [2] which uses a fully-connected layer to parameterizes the derivative, GNODE extends the fully-connected neural ODE (referred to as FC-NODE to distinguish from GNODE) by using GCN as state-to-state transitions.

Intuitively GNODE allows different nodes in a graph to *continuously* interact with each other according their adjacency, leading to continuously evolving node embeddings. Such extension is non-trivial since the graph that governs the interaction is also *continuously* changing.

Method. Recall that from Eqn. 6 to Eqn. 7, neural ODE replaces the discrete recurrent update with continuous trajectories. Similarly for GNODE, the derivative of the latent state is parameterized by a learnable GCN, as shown below:

$$Z^{(i)} = Z^{(i-1)} + \text{GCN}(Z^{(i-1)}, \boldsymbol{A}^{(i)}) \tag{13}$$

$$\Rightarrow \begin{array}{ll} \frac{dZ}{dt} & = \text{GCN}(Z, \boldsymbol{A}), \\ Z(t) & = \text{GNODE}(Z(t_0), t), \end{array} \tag{14}$$

where $\boldsymbol{A}^{(i)}$ denotes the adjacency matrix of $G^{(i)}$. Given an initial state $z(t_0)$, a neural ODE extrapolates the trajectory of latent z, and can be evaluated at any specified timestamps. Similar to FC-NODE, the input of GNODE is an an initial state $Z(t_0)$. GNODE extrapolates the trajectory of latent $Z(t)$, which can be evaluated at any specified timestamps t_i to output $Z(t_i)$ by calling an ODE solver.

Key Challenge. However, enabling graph-to-graph transitions in GNODE is not straightforward. Unlike typical neural ODE where $\frac{dZ}{dt}$ is only related to Z (and possibly time t), in GNODE $\frac{dZ}{dt}$ is also related to the adjacency matrix $\boldsymbol{A}$. The solution would have been straightforward if $\boldsymbol{A}$ is fixed across all timestamps. However, this is not the case; what makes it even more involved is that $\boldsymbol{A}$ and Z interact with each other continuously. On the one hand, since Z is the latent representation of nodes in the graph $G = (\boldsymbol{A}, \boldsymbol{X})$, as the latent state Z evolves, the adjacency matrix $\boldsymbol{A}$ should also change continuously over time. On the other hand, since only the derivative of Z is specified, directly solving Eqn. 14 as a typical neural ODE is equivalent to fixing $\boldsymbol{A}$ (i.e., treating $\boldsymbol{A}$ as a constant), which is obviously incorrect.

Naive Approach: Piecewise Fixed Adjacency Matrix. Observing that the challenge would not have existed if the adjacency matrix $\boldsymbol{A}$ is fixed, one naive approach is to select a set of timestamps $\{t_i\}_i$, only update $\boldsymbol{A}$ at these timestamps while fixing $\boldsymbol{A}$ during each interval $[t_i, t_{i+1})$. This leads to the following equation:

$$\frac{dZ}{dt} = \text{GCN}(Z, \boldsymbol{A}^{(i)}), \ t \in [t_i, t_{i+1}), \ \forall i, \tag{15}$$

where $\boldsymbol{A}$ is only updated at the end of each interval. This approach is not practical for continuously graph forecasting due to the following two reasons [32]: (a) The choice of timestamps may have great

Algorithm 1 GNODE^e: GNODE-Augmented GRNN

Input: $\{(G^{(i)}, t_i)\}_i$
Set initial state $\bar{Z}(t_1)$
for $i = 1$ to $I - 1$ **do**
  $Z(t_i) = \text{GRNN}(\bar{Z}(t_i), G^{(i)})$
  $\bar{Z}(t_{i+1}) = \text{GNODE}(Z(t_i), \{t_i, t_{i+1}\})$
end for
$Z(t_I) = \text{GRNN}(\bar{Z}(t_I), G^{(I)})$
Return: $Z(t_I)$

impact on the final solution of the ODE. For example, if the time interval $[t_i, t_{i+1})$ is large while the derivative of $Z(t_i)$ is also large, the ODE solver may over-extrapolate from $Z(t_i)$ to $Z(t_{i+1})$ with the derivative $\text{GCN}(Z, A^{(i)})$, reaching a poor estimate of $Z(t_{i+1})$. Such error also accumulates over time, making the solution highly volatile and inexactly. (b) To implement piecewise fixed A, intermediate results at all timestamps $\{t_i\}_i$ need to be saved so that the gradients are available to perform back-propagation. Therefore with T timestamp, the memory cost w.r.t. T for this naive approach is $O(T)$ rather than the original $O(1)$ for the adjoint sensitivity method. This is impractical because to avoid poor estimates of $Z(t_{i+1})$, as stated in reason (a), one needs an extremely large T, leading to huge memory cost.

GNODE's Approach: Decoded Adjacency Matrix. Instead of using manually selected timestamps and a piecewise fixed adjacency matrix, we propose a more natural approach that updates the adjacency matrix *continuously* and simultaneously with the evolution of Z. The key idea is to replace the input adjacency matrix A with the output of a graph decoder which is similar to D_g in Eqn. 3 and Eqn. 11:

$$\frac{dZ}{dt} = \text{GCN}(Z, \widetilde{D}_g(Z)). \tag{16}$$

Notice that the graph decoder here is slightly different from the one in Eqn. 11 which specifies the conditional probability $p(G|Z)$. $\widetilde{D}_g(\cdot)$ replaces the last softmax layer in conventional graph decoder $D_g(\cdot)$ with a Gumbel-Softmax layer [12], which outputs binary adjacency matrix while keeping differentiable. With $\widetilde{D}_g(\cdot)$, GNODE in Eqn. 11 can be solved exactly the same way as neural ODE, while enabling continuous interaction between Z and A.

4.3 GNODE^e: GNODE-Augmented GRNN Encoder

While the decoder takes only the initial state $Z(t'_{0'})$ as input, the encoder is fed with the whole input sequence $\{(G^{(i)}, t_i)\}_i$. To better encode the whole graph sequence into the final state $Z(t_I)$ with consideration to continuous dynamics, we adopt a similar algorithm as ODE-RNN [35] to integrate GNODE into GRNN, creating a GNODE-augmented graph sequence encoder, which we call GNODE^e. Alg. 1 and the left of Fig. 1 illustrate how GNODE^e encodes each input graph $(G^{(i)}, t_i)$ in the sequence one by one and updates the state $Z(t_i)$. Note that in Alg. 1 we distinguish $\bar{Z}^{(i)}$, which is the state before the discrete GRNN update, from $Z^{(i)}$, which is the state after the discrete GRNN update.

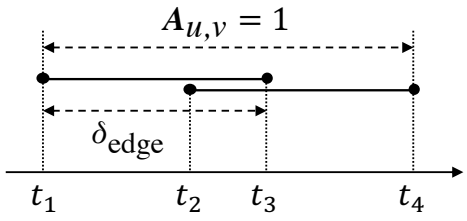

Figure 2: Converting email records to sequences of graph-timestamp pairs. δ_{edge} is the duration of a relation. $A_{u,v} = 1$ at time t if t is covered by at least one relation, otherwise $A_{u,v} = 0$. In this figure, $A_{u,v} = 1$ during $[t_1, t_4]$.

5 EXPERIMENTS

We perform continuous dynamic graph forecasting tasks on two real-world datasets under two settings respectively, to demonstrate the effectiveness of proposed approaches. Since there are no node features X in both datasets, we use identity matrix as X as suggested by [21]. When predicting future graphs, we only predict the adjacency matrices.

5.1 Datasets

We use two real-world datasets to evaluate our methods and baselines.

- *Enron* [22, 25] is a dataset containing the email records of Enron Corporation. We use a processed version[1] which consists of around 500,000 email messages among 184 Enron employees from 1998 to 2002.
- *EU-core* [30] is a dataset with email records from a large European research institution. It contains around 300,000 records from 986 members over 803 days.

Both datasets include timestamps of emails and are therefore suitable for evaluating different methods on CTDG forecasting tasks.

5.2 Experiment Settings

Email Records to CTDG Format. For fair comparison with baselines, we follow [8, 34, 44] to process the datasets and construct graph sequences. As illustrated in Fig. 2, there is a predefined time interval parameter δ_{edge} (e.g., one week), which represents how long a 'relation' built between two nodes can last if no further interaction occurs. As an example, when node u contacts node v at timestamp t_1, they build a relation which lasts for δ_{edge}. Later, if u contacts v again at t_2, where $t_2 < t_3 = t_1 + \delta_{\text{edge}}$, such an interaction will extend the duration of the relation until $t_4 = t_2 + \delta_{\text{edge}}$. Therefore, $A_{u,v} = 1$ during the time interval $[t_1, t_4]$. Note that if after t_4, u does not contact v any more, the relation will terminate and the edge $A_{u,v}$ will disappear at timestamp t_4.

With the procedure above, one can obtain the graph snapshot at any timestamp t_i, based on which one can then construct datasets for CTDG forecasting task. Specifically, we adopt two different evaluation settings with different ways of sampling graph-timestamp pairs $\{(G^{(i)}, t_i)\}_i$. Below we describe these two settings, the time-based and edge-based settings, in detail.

[1] http://www.cis.jhu.edu/ parky/Enron/

Table 1: AUC and AP for different methods in the time-based setting on Enron.

Metric	AUC					AP				
σ_t	0	1	2	3	4	0	1	2	3	4
SGRNN-SGRNN [4]	0.8870	0.8709	0.8585	0.8780	0.8673	0.2141	0.1886	0.1522	0.2024	0.1931
SGRNN-SGRNN$_v$ [4]	0.8756	0.8685	0.8679	0.8551	0.8658	0.2173	0.2163	0.2086	0.1915	0.2279
IGRNN-IGRNN [38]	0.8437	0.8416	0.8439	0.8332	0.8370	0.2576	0.2577	0.2516	0.2507	0.2592
IGRNN-IGRNN$_v$ [8]	0.8399	0.8364	0.8304	0.8291	0.8314	0.2604	0.2584	0.2445	0.2542	0.2570
DySAT [37]	0.8751	0.8781	0.8753	0.8750	0.8805	0.1637	0.1695	0.1656	0.1664	0.1732
GNODE-GNODE (ours)	0.8862	0.8805	0.8778	0.8725	0.8770	0.2935	0.2894	0.2870	**0.2910**	**0.2991**
GNODE-IGRNN$_v$ (ours)	0.8398	0.8343	0.8331	0.8261	0.8238	0.2604	0.2545	0.2500	0.2488	0.2535
IGRNN-GNODE$_v$ (ours)	**0.8912**	**0.8865**	0.8823	0.8784	**0.8818**	**0.2996**	**0.2908**	0.2861	0.2846	0.2931
GNODE-GNODE$_v$ (ours)	0.8910	0.8856	**0.8831**	**0.8789**	0.8791	0.2965	0.2907	**0.2871**	0.2905	0.2929

Table 2: AUC and AP for different methods in the time-based setting on Eu-core.

Metric	AUC					AP				
σ_t	0	1	2	3	4	0	1	2	3	4
IGRNN-IGRNN [38]	0.8794	0.8785	0.8652	0.8692	0.8886	0.3202	0.3143	0.2748	0.2922	0.3207
IGRNN-IGRNN$_v$ [8]	0.8635	0.8729	0.8669	0.8689	0.8667	0.3223	0.3198	0.3219	0.3284	0.3255
DySAT [37]	0.9001	0.8997	0.9017	0.9020	0.6008	0.1143	0.1130	0.1157	0.1148	0.0761
GNODE-GNODE (ours)	0.9036	0.9023	0.9013	0.9037	0.9050	0.3517	0.3511	0.3438	0.3506	0.3475
GNODE-IGRNN$_v$ (ours)	0.8641	0.8711	0.8663	0.8613	0.8552	0.3245	0.3145	0.3223	0.3002	0.2900
IGRNN-GNODE$_v$ (ours)	**0.9285**	**0.9201**	**0.9266**	0.9310	**0.9295**	0.3645	0.3592	**0.3611**	0.3716	0.3689
GNODE-GNODE$_v$ (ours)	0.9278	0.9196	0.9261	**0.9312**	0.9288	**0.3664**	**0.3624**	0.3609	**0.3727**	**0.3692**

Time-Based Setting. In the time-based setting, we first sample equally spaced timestamps $\{t_i\}_i$ where $t_{i+1} - t_i = t_i - t_{i-1}$, $\forall i$, and then obtain corresponding graph snapshots to construct the final graph sequence $\{(G^{(i)}, t_i)\}_i$. This setting is similar to that of DTDG forecasting task, allowing fair comparison with DTDG models. To further evaluate different methods' robustness to irregularly sampled timestamps, we also add Gaussian noise with certain standard deviation to each timestamp, i.e., $t_i \leftarrow t_i + \mathcal{N}(0, \sigma_t^2)$, before obtaining the graph snapshots. Ideally, the accuracy of a robust method will remain almost unchanged as the standard deviation increases.

Edge-Based Setting. In the edge-based setting, we sample graph-timestamp pairs $\{(G^{(i)}, t_i)\}_i$ such that the number of changed edges between any two consecutive graph snapshots ($G^{(i)}$ and $G^{(i+1)}$) is identical. We define the number of changed edges between $G^{(i)}$ and $G^{(i+1)}$ as 'stride'. For example, stride = 100 means there are 100 edges changed between $(G^{(i)}, t_i)$ and $(G^{(i+1)}, t_{i+1})$ for any i. Note that this setting is in favor of DTDG baselines because the degree of changes in the graph is roughly constant for any pair of consecutive timestamps. Interestingly our method is still able to outperform DTDG methods in this setting.

5.3 Baselines, GNODE Variants, and Implementation

Baselines. As mentioned in Sec. 'Related Works', SGRNN [24, 27, 38, 40] and IGRNN [1, 8, 26, 38] are typical dynamic graph forecasting baselines. We therefore replace the encoder and decoder in our model (Fig. 1) with SGRNN or IGRNN, and use such models as baselines. Specifically, we use **dyngraph2vec** [4] and **VGRNN** [8] for SGRNN and IGRNN, respectively. Our baselines include

- **SGRNN-SGRNN$_v$** [4], which uses an SGRNN encoder and an SGRNN decoder.
- **IGRNN-IGRNN$_v$** [8], which uses an IGRNN encoder and an IGRNN decoder.
- **SGRNN-SGRNN** [4], which is SGRNN-SGRNN$_v$'s deterministic counterpart.
- **IGRNN-IGRNN** [38], which is IGRNN-IGRNN$_v$'s deterministic counterpart.
- **DySAT** [37], which is a GRNN-free baseline that uses attention mechanism for dynamic graph forecasting tasks.

For Eu-core dataset, we exclude SGRNN models due to their high memory and computation cost.

Table 3: AUC and AP for different methods in the edge-based setting on Enron.

Metric	AUC				AP			
Stride	30	50	100	150	30	50	100	150
SGRNN-SGRNN [4]	**0.9932**	**0.9910**	0.9803	0.9784	0.6546	0.5660	0.4326	0.4082
SGRNN-SGRNN$_v$ [4]	0.9907	0.9899	0.9754	0.9711	0.6341	0.5619	0.4336	0.3891
IGRNN-IGRNN [38]	0.9915	0.9897	0.9819	0.9786	0.7361	0.6763	0.5566	0.5217
IGRNN-IGRNN$_v$ [8]	0.9907	0.9896	0.9819	0.9781	0.7570	0.6801	0.5735	0.5310
DySAT [37]	0.9326	0.9189	0.9011	0.8903	0.2639	0.2187	0.1986	0.1803
GNODE-GNODE (ours)	0.9922	0.9902	0.9833	0.9802	0.7453	0.6803	0.5785	0.5336
GNODE-IGRNN$_v$ (ours)	0.9907	0.9896	0.9816	0.9779	0.7573	0.6801	0.5736	0.5309
IGRNN-GNODE$_v$ (ours)	0.9932	0.9907	0.9849	0.9822	**0.7776**	**0.6897**	**0.6056**	0.5654
GNODE-GNODE$_v$ (ours)	0.9930	0.9906	**0.9852**	**0.9824**	0.7707	0.6874	0.6043	**0.5662**

Table 4: AUC and AP for different methods in the edge-based setting on Eu-core.

Metric	AUC					AP				
Stride	100	200	300	400	500	100	200	300	400	500
IGRNN-IGRNN [38]	0.9884	0.9830	0.9817	0.9770	0.9710	0.8585	0.7704	0.6911	0.5869	0.5220
IGRNN-IGRNN$_v$ [8]	0.9833	0.9772	0.9749	0.9692	0.9629	0.8351	0.7462	0.6736	0.5732	0.5230
DySAT [37]	0.9457	0.9353	0.9234	0.9300	0.9129	0.1627	0.1473	0.1401	0.1424	0.1256
GNODE-GNODE (ours)	0.9883	0.9851	0.9839	0.9794	0.9755	**0.8647**	0.7759	0.7080	0.6028	0.5424
GNODE-IGRNN$_v$ (ours)	0.9835	0.9782	0.9750	0.9693	0.9627	0.8359	0.7467	0.6731	0.5748	0.5230
IGRNN-GNODE$_v$ (ours)	**0.9901**	0.9869	0.9866	**0.9827**	**0.9785**	0.8604	0.7823	0.7090	**0.6046**	0.5432
GNODE-GNODE$_v$ (ours)	0.9901	**0.9869**	**0.9866**	0.9826	0.9783	0.8624	**0.7832**	**0.7096**	0.6039	**0.5440**

Proposed GNODE Variants. Similarly, we evaluate four variants of our proposed GNODE. They are

- **GNODE-IGRNN$_v$**, which uses GNODE^e (introduced in Sec. 'GNODE^e: A GNODE-Augmented GRNN Encoder') as the encoder and IGRNN as the decoder.
- **GNODE-GNODE$_v$**, which uses GNODE^e as the encoder and GNODE as the decoder.
- **IGRNN-GNODE$_v$**, which uses IGRNN as the encoder and GNODE as the decoder.
- **GNODE-GNODE**, which is the deterministic version (i.e., without variational autoencoders) of GNODE-GNODE$_v$.

Implementation and Metrics. For all experiments, the models receive observed graph sequences of length 10 as input. In the time-based setting, they will forecast the next 5 future graph snapshots, while in the edge-based setting they will forecast the next 10 future graph snapshots; this is because the time-based setting is much more challenging than the edge-based setting. We use area under the ROC curve (AUC) and average precision (AP) as evaluation metrics. We optimize the cross-entropy loss using the Adam optimizer [19]. More experiment configuration details are included in the Supplement.

5.4 Experiment Results

We show the results in the time-based setting in Table 1 and Table 2, and the results in edge-based setting in Table 3 and Table 4. All AUCs and APs are calculated based on the whole adjacency matrices. We use three different random seeds for each experiment and report the mean scores, with standard deviations included in the Supplement. We mark the best result within a specific setting with **bold face** and the second best result by underlining.

Results in the Time-Based Setting. As shown in Table 1 and Table 2, our proposed model GNODE-GNODE$_v$ consistently outperforms baseline models. As shown in the first and second columns of Fig. 3, in most cases, GNODE-GNODE$_v$ is better than GNODE-GNODE, which demonstrates the advantage of the variational encoder-decoder architecture against its deterministic counterpart. By comparing GNODE-GNODE$_v$ against GNODE-IGRNN$_v$ and IGRNN-GNODE$_v$ against IGRNN-IGRNN$_v$, we can see that models with the GNODE decoder consistently outperforms models with the IGRNN decoder, demonstrating the effectiveness of GNODE as a decoder. Interestingly, we find that with IGRNN as the decoder, GNODE-IGRNN$_v$ actually underperforms IGRNN-IGRNN$_v$. However, if we replace the decoder of GNODE-IGRNN$_v$ with GNODE,

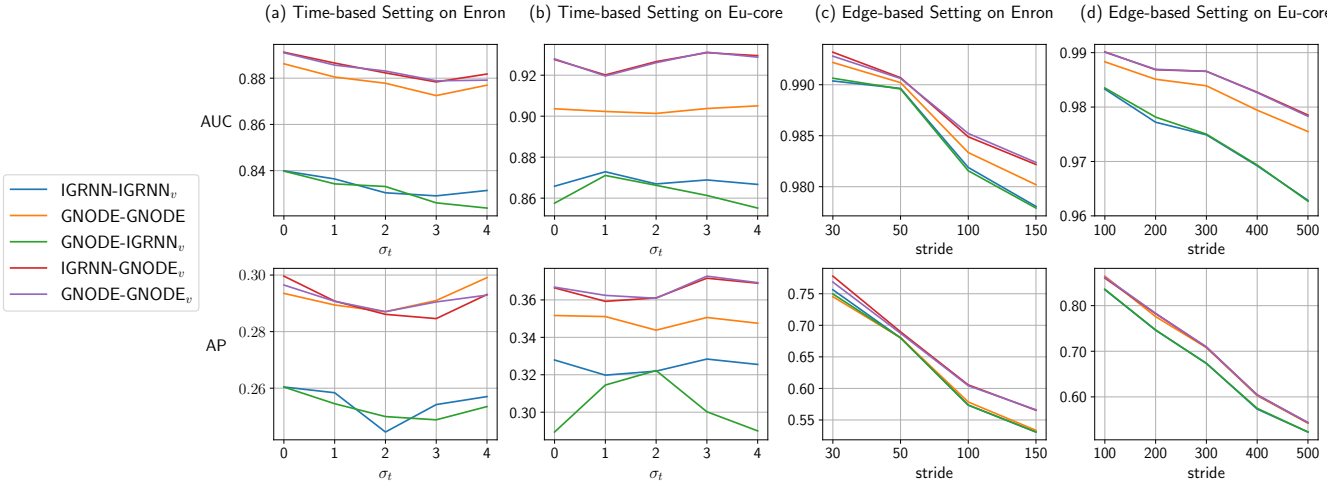

Figure 3: Results of the ablation study. We visualize the performance of GNODE variants (GNODE-GNODE, GNODE-IGRNN$_v$ IGRNN-GNODE$_v$ and GNODE-GNODE$_v$) and the baseline IGRNN-IGRNN$_v$ in different settings. Two rows show AUC and AP respectively. (a) and (b) are results in time-based setting with different σ_t, (c) and (d) are results in edge-based setting with different stride.

leading to GNODE-GNODE$_v$, the performance is dramatically improved, surpassing all other methods. Such results suggests that GNODE may prefer a symmetric encoder-decoder architecture. This is consistent with prior work on ODE-RNN [35].

Results in the Edge-Based Setting. Table 3 and Table 4 show the AUC and AP for various methods in the edge-based setting. Results are consistent with the time-based setting. On the Enron dataset, though SGRNN-SGRNN achieves the highest AUC in some experiment settings, our GNODE-GNODE$_v$ gets almost the same AUCs (differences smaller than 10^{-2}) and much higher APs. Additionally, we find that our GNODE-GNODE$_v$ outperforms the baseline IGRNN-IGRNN$_v$ by a larger gap as the stride increases, as shown in the third and fourth column of Fig. 3. This observation demonstrates the advantage and robustness of GNODE-GNODE$_v$ in face of more complex and long-term dynamics.

6 CONCLUSION

In this paper, we have formulated CTDG forecasting problem in a more general way and proposed GNODE as a novel model that captures the underlying continuous dynamics more efficiently. Empirical results demonstrate the superiority of our model over state-of-the-art baselines. For future work, a straightforward extension is to evaluate GNODE on dynamic graph datasets with node features to explore whether GNODE is also capable to forecast future node features in high quality. We also point out a limitation of GNODE here: GNODE requires predefined set of nodes, i.e., it can not handle previously unseen nodes. Addressing such a limitation is also interesting future work.

REFERENCES

[1] Jinyin Chen, Xuanheng Xu, Yangyang Wu, and Haibin Zheng. Gc-lstm: Graph convolution embedded lstm for dynamic link prediction. *arXiv preprint arXiv:1812.04206*, 2018.
[2] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in neural information processing systems*, pages 6571–6583, 2018.
[3] Tianqi Chen, Ian Goodfellow, and Jonathon Shlens. Net2net: Accelerating learning via knowledge transfer. *arXiv preprint arXiv:1511.05641*, 2015.
[4] Palash Goyal, Sujit Rokka Chhetri, and Arquimedes Canedo. dyngraph2vec: Capturing network dynamics using dynamic graph representation learning. *Knowledge-Based Systems*, 187:104816, 2020.
[5] Palash Goyal and Emilio Ferrara. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, 151:78–94, 2018.
[6] Palash Goyal, Nitin Kamra, Xinran He, and Yan Liu. Dyngem: Deep embedding method for dynamic graphs. *arXiv preprint arXiv:1805.11273*, 2018.
[7] Ernst Hairer, Syvert P Nørsett, and Gerhard Wanner. *Solving ordinary differential equations. 1, Nonstiff problems.* Springer-Vlg, 1993.
[8] Ehsan Hajiramezanali, Arman Hasanzadeh, Krishna Narayanan, Nick Duffield, Mingyuan Zhou, and Xiaoning Qian. Variational graph recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 10700–10710, 2019.
[9] William L Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*, 2017.
[10] Petter Holme. Modern temporal network theory: a colloquium. *The European Physical Journal B*, 88(9):234, 2015.
[11] Petter Holme and Jari Saramäki. Temporal networks. *Physics reports*, 519(3):97–125, 2012.
[12] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.
[13] Junteng Jia and Austin R Benson. Neural jump stochastic differential equations. *Advances in Neural Information Processing Systems*, 2019.
[14] Di Jin, Sungchul Kim, Ryan A Rossi, and Danai Koutra. From static to dynamic node embeddings. *arXiv preprint arXiv:2009.10017*, 2020.
[15] Woojeong Jin, He Jiang, Meng Qu, Tong Chen, Changlin Zhang, Pedro Szekely, and Xiang Ren. Recurrent event network: Global structure inference over temporal knowledge graph. *arXiv preprint arXiv:1904.05530*, 2019.
[16] Seyed Mehran Kazemi, Rishab Goel, Kshitij Jain, Ivan Kobyzev, Akshay Sethi, Peter Forsyth, and Pascal Poupart. Relational representation learning for dynamic (knowledge) graphs: A survey. *arXiv preprint arXiv:1905.11485*, 2019.
[17] Seyed Mehran Kazemi, Rishab Goel, Kshitij Jain, Ivan Kobyzev, Akshay Sethi, Peter Forsyth, and Pascal Poupart. Representation learning for dynamic graphs: A survey. *arXiv preprint arXiv:1905.11485*, 2019.
[18] Patrick Kidger, James Morrill, James Foster, and Terry Lyons. Neural controlled differential equations for irregular time series. *Advances in Neural Information Processing Systems*, 2020.
[19] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
[20] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[21] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[22] Bryan Klimt and Yiming Yang. Introducing the enron corpus. In *CEAS*, 2004.

[23] Wilhelm Kutta. Beitrag zur naherungsweisen integration totaler differentialgleichungen. *Z. Math. Phys.*, 46:435–453, 1901.

[24] Kai Lei, Meng Qin, Bo Bai, Gong Zhang, and Min Yang. Gcn-gan: A non-linear temporal link prediction model for weighted dynamic networks. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pages 388–396. IEEE, 2019.

[25] Jure Leskovec, Kevin J Lang, Anirban Dasgupta, and Michael W Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6(1):29–123, 2009.

[26] Jia Li, Zhichao Han, Hong Cheng, Jiao Su, Pengyun Wang, Jianfeng Zhang, and Lujia Pan. Predicting path failure in time-evolving graphs. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1279–1289, 2019.

[27] Franco Manessi, Alessandro Rozza, and Mario Manzo. Dynamic graph convolutional networks. *Pattern Recognition*, 97:107000, 2020.

[28] Naoki Masuda and Renaud Lambiotte. *Guide To Temporal Networks, A*, volume 6. World Scientific, 2020.

[29] Giang Hoang Nguyen, John Boaz Lee, Ryan A Rossi, Nesreen K Ahmed, Eunyee Koh, and Sungchul Kim. Continuous-time dynamic network embeddings. In *Companion Proceedings of the The Web Conference 2018*, pages 969–976, 2018.

[30] Ashwin Paranjape, Austin R Benson, and Jure Leskovec. Motifs in temporal networks. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, pages 601–610, 2017.

[31] Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao B Schardl, and Charles E Leiserson. Evolvegcn: Evolving graph convolutional networks for dynamic graphs. In *AAAI*, pages 5363–5370, 2020.

[32] Michael Poli, Stefano Massaroli, Junyoung Park, Atsushi Yamashita, Hajime Asama, and Jinkyoo Park. Graph neural ordinary differential equations. *arXiv preprint arXiv:1911.07532*, 2019.

[33] Lev Semenovich Pontryagin. *Mathematical theory of optimal processes*. Routledge, 2018.

[34] Mahmudur Rahman and Mohammad Al Hasan. Link prediction in dynamic networks using graphlet. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 394–409. Springer, 2016.

[35] Yulia Rubanova, Tian Qi Chen, and David K Duvenaud. Latent ordinary differential equations for irregularly-sampled time series. In *Advances in Neural Information Processing Systems*, pages 5321–5331, 2019.

[36] Carl Runge. Über die numerische auflösung von differentialgleichungen. *Mathematische Annalen*, 46(2):167–178, 1895.

[37] Aravind Sankar, Yanhong Wu, Liang Gou, Wei Zhang, and Hao Yang. Dysat: Deep neural representation learning on dynamic graphs via self-attention networks. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, pages 519–527, 2020.

[38] Youngjoo Seo, Michaël Defferrard, Pierre Vandergheynst, and Xavier Bresson. Structured sequence modeling with graph convolutional recurrent networks. In *International Conference on Neural Information Processing*, pages 362–373. Springer, 2018.

[39] Joakim Skarding, Bogdan Gabrys, and Katarzyna Musial. Foundations and modelling of dynamic networks using dynamic graph neural networks: A survey. *arXiv preprint arXiv:2005.07496*, 2020.

[40] Aynaz Taheri, Kevin Gimpel, and Tanya Berger-Wolf. Learning to represent the evolution of dynamic graphs with recurrent models. In *Companion Proceedings of The 2019 World Wide Web Conference*, pages 301–307, 2019.

[41] Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. Dyrep: Learning representations over dynamic graphs. In *International Conference on Learning Representations*, 2018.

[42] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 2020.

[43] Louis-Pascal A. C. Xhonneux, Meng Qu, and Jian Tang. Continuous graph neural networks. *ArXiv*, abs/1912.00967, 2019.

[44] Kevin S. Xu and Alfred O. Hero. Dynamic stochastic blockmodels for time-evolving social networks. *IEEE Journal of Selected Topics in Signal Processing*, 8(4):552–562, Aug 2014.

[45] Ziwei Zhang, Peng Cui, and Wenwu Zhu. Deep learning on graphs: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 2020.

[46] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*, 2018.

[47] Juntang Zhuang, Nicha Dvornek, Xiaoxiao Li, and James S Duncan. Ordinary differential equations on graph networks. 2019.

A DETAILS ON MODEL ARCHITECTURES

The model architecture of GNODE-GNODE and GNODE-GNODE$_v$ is illustrated in Fig. 4. We introduce the notations and the components of the architecture one by one:

- There are two latent layers in both the encoder and the decoder.
- Z represents the collection of low-level and high-level latent states.
- The 'Transform' part between the encoder and the decoder can be either deterministic (GNODE-GNODE) or probabilistic (GNODE-GNODE$_v$).
- An input graph G_i first goes into a GCN layer and then is fed into GNODE$_e$.
- The i'th GNODE in the encoder part between two GRNN updates takes $Z(t_i)$ as the initial states, extrapolates the trajectory to t_{i+1} and outputs $\bar{Z}(t_{i+1})$.
- The trajectory extrapolated by the GNODE in the decoder part is evaluated at $\{t'_{i'}\}_{i'}$, generating $\{Z(t'_{i'})\}_{i'}$, which is then fed into the graph decoder $D_g(Z)$.
- The graph decoder $D_g(Z)$ takes Z as input and generates the prediction $\widehat{G}$ only from high-level latent state.

The model architecture of baselines are illustrated in Fig. 5. For fair comparison, we also adopt two latent layers. The GRNN can be either SGRNN or IGRNN. Unlike the GNODE decoder, GRNN decoder generates predictions step-by-step for multi-step forecasting.

Other GNODE variants are combinations of the above encoders and decoders. For example, GNODE-IGRNN$_v$ uses the encoder in Fig. 4 and the decoder in Fig. 5, while IGRNN-GNODE$_v$ uses the encoder in Fig. 5 and the decoder in Fig. 4.

All models in our experiments were trained using a single NVIDIA Tesla T4 GPU with 16GB of memory.

B EXPERIMENT RESULTS

We train each model using three different random seeds and report the mean as well as the standard deviation of the final scores. We show the mean of the scores in the time-based setting in Table 5 (Table 1 in our paper) and Table 7 (Table 2 in our paper), as well as the mean of the scores in edge-based setting in Table 9 (Table 3 in our paper) and Table 11 (Table 4 in our paper). We show the standard deviation of the scores in the time-based setting in Table 6 and Table 8, as well as the standard deviation of the scores in edge-based setting in Table 10 and Table 12.

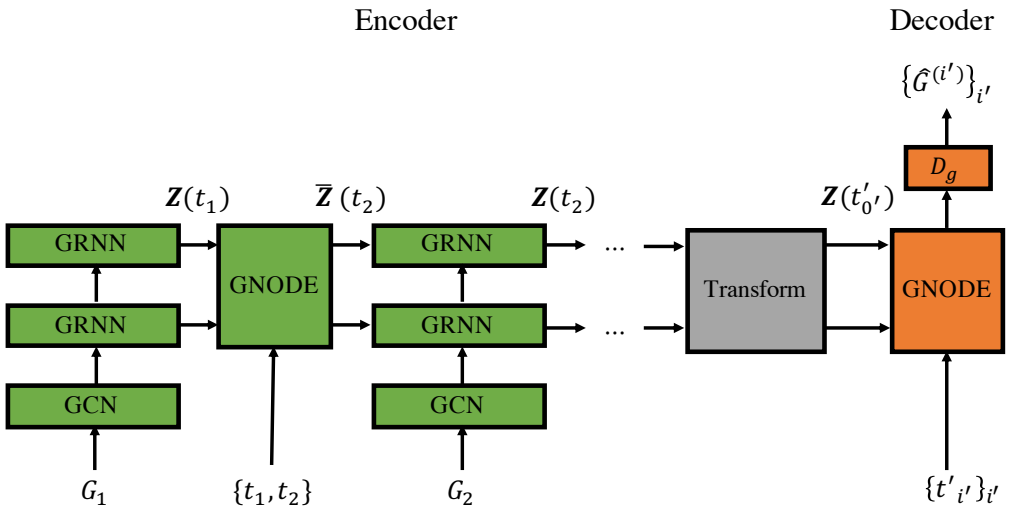

Figure 4: Model architecture of GNODE-GNODE and GNODE-GNODE$_v$.

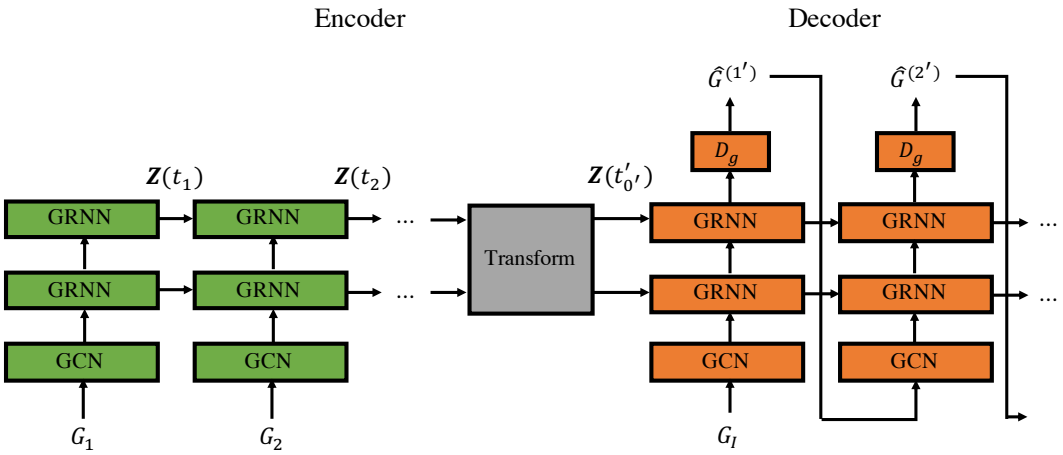

Figure 5: Model architecture of GRNN-GRNN and GRNN-GRNN$_v$.

Table 5: AUC and AP for different methods in the time-based setting on *Enron*.

Metric	AUC					AP				
σ_t	0	1	2	3	4	0	1	2	3	4
SGRNN-SGRNN [4]	0.8870	0.8709	0.8585	0.8780	0.8673	0.2141	0.1886	0.1522	0.2024	0.1931
SGRNN-SGRNN$_v$ [4]	0.8756	0.8685	0.8679	0.8551	0.8658	0.2173	0.2163	0.2086	0.1915	0.2279
IGRNN-IGRNN [38]	0.8437	0.8416	0.8439	0.8332	0.8370	0.2576	0.2577	0.2516	0.2507	0.2592
IGRNN-IGRNN$_v$ [8]	0.8399	0.8364	0.8304	0.8291	0.8314	0.2604	0.2584	0.2445	0.2542	0.2570
DySAT [37]	0.8751	0.8781	0.8753	0.8750	0.8805	0.1637	0.1695	0.1656	0.1664	0.1732
GNODE-GNODE (ours)	0.8862	0.8805	0.8778	0.8725	0.8770	0.2935	0.2894	0.2870	**0.2910**	**0.2991**
GNODE-IGRNN$_v$ (ours)	0.8398	0.8343	0.8331	0.8261	0.8238	0.2604	0.2545	0.2500	0.2488	0.2535
IGRNN-GNODE$_v$ (ours)	**0.8912**	**0.8865**	0.8823	0.8784	**0.8818**	**0.2996**	**0.2908**	0.2861	0.2846	0.2931
GNODE-GNODE$_v$ (ours)	0.8910	0.8856	**0.8831**	**0.8789**	0.8791	0.2965	0.2907	**0.2871**	0.2905	0.2929

Table 6: Confidence intervals for different methods in the time-based setting on *Enron*.

Metric	AUC					AP				
σ_t	0	1	2	3	4	0	1	2	3	4
SGRNN-SGRNN [4]	0.0177	0.0107	0.0037	0.0149	0.0017	0.0455	0.0433	0.0115	0.0339	0.0243
SGRNN-SGRNN$_v$ [4]	0.0024	0.0031	0.0046	0.0077	0.0015	0.0134	0.0232	0.0213	0.0132	0.0126
IGRNN-IGRNN [38]	0.0018	0.0088	0.0065	0.0058	0.0123	0.0015	0.0094	0.0113	0.0089	0.0115
IGRNN-IGRNN$_v$ [8]	0.0029	0.0069	0.0053	0.0045	0.0065	0.0046	0.0057	0.0109	0.0115	0.0123
DySAT [37]	0.0004	0.0056	0.0041	0.0059	0.0043	0.0007	0.0059	0.0056	0.0098	0.0014
GNODE-GNODE (ours)	0.0011	0.0046	0.0048	0.0030	0.0058	0.0033	0.0072	0.0116	0.0115	0.0103
GNODE-IGRNN$_v$ (ours)	0.0019	0.0026	0.0083	0.0030	0.0064	0.0011	0.0028	0.0157	0.0103	0.0113
IGRNN-GNODE$_v$ (ours)	0.0001	0.0039	0.0030	0.0043	0.0013	0.0010	0.0036	0.0124	0.0132	0.0061
GNODE-GNODE$_v$ (ours)	0.0006	0.0025	0.0031	0.0038	0.0036	0.0018	0.0049	0.0120	0.0142	0.0071

Table 7: AUC and AP for different methods in the time-based setting on *Eu-core*.

Metric	AUC					AP				
σ_t	0	1	2	3	4	0	1	2	3	4
IGRNN-IGRNN [38]	0.8794	0.8785	0.8652	0.8692	0.8886	0.3202	0.3143	0.2748	0.2922	0.3207
IGRNN-IGRNN$_v$ [8]	0.8635	0.8729	0.8669	0.8689	0.8667	0.3223	0.3198	0.3219	0.3284	0.3255
DySAT [37]	0.9001	0.8997	0.9017	0.9020	0.6008	0.1143	0.1130	0.1157	0.1148	0.0761
GNODE-GNODE (ours)	0.9036	0.9023	0.9013	0.9037	0.9050	0.3517	0.3511	0.3438	0.3506	0.3475
GNODE-IGRNN$_v$ (ours)	0.8641	0.8711	0.8663	0.8613	0.8552	0.3245	0.3145	0.3223	0.3002	0.2900
IGRNN-GNODE$_v$ (ours)	**0.9285**	**0.9201**	**0.9266**	0.9310	**0.9295**	0.3645	0.3592	**0.3611**	0.3716	0.3689
GNODE-GNODE$_v$ (ours)	0.9278	0.9196	0.9261	**0.9312**	0.9288	**0.3664**	**0.3624**	0.3609	**0.3727**	**0.3692**

Table 8: Confidence intervals for different methods in the time-based setting on *Eu-core*.

Metric	AUC					AP				
σ_t	0	1	2	3	4	0	1	2	3	4
IGRNN-IGRNN [38]	0.0049	0.0142	0.0276	0.0294	0.0084	0.0054	0.0126	0.0630	0.0506	0.0119
IGRNN-IGRNN$_v$ [8]	0.0045	0.0092	0.0015	0.0016	0.0028	0.0045	0.0014	0.0095	0.0069	0.0110
DySAT [37]	0.0017	0.0023	0.0020	0.0014	0.0007	0.0012	0.0031	0.0015	0.0037	0.0003
GNODE-GNODE (ours)	0.0020	0.0013	0.0027	0.0014	0.0042	0.0005	0.0021	0.0040	0.0121	0.0120
GNODE-IGRNN$_v$ (ours)	0.0047	0.0048	0.0043	0.0118	0.0218	0.0044	0.0147	0.0080	0.0477	0.0607
IGRNN-GNODE$_v$ (ours)	0.0007	0.0124	0.0020	0.0029	0.0011	0.0041	0.0111	0.0036	0.0107	0.0086
GNODE-GNODE$_v$ (ours)	0.0004	0.0112	0.0020	0.0032	0.0007	0.0013	0.0087	0.0059	0.0092	0.0076

Table 9: AUC and AP for different methods in the edge-based setting on *Enron*.

Metric	AUC				AP			
Stride	30	50	100	150	30	50	100	150
SGRNN-SGRNN [4]	**0.9932**	**0.9910**	0.9803	0.9784	0.6546	0.5660	0.4326	0.4082
SGRNN-SGRNN$_v$ [4]	0.9907	0.9899	0.9754	0.9711	0.6341	0.5619	0.4336	0.3891
IGRNN-IGRNN [38]	0.9915	0.9897	0.9819	0.9786	0.7361	0.6763	0.5566	0.5217
IGRNN-IGRNN$_v$ [8]	0.9907	0.9896	0.9819	0.9781	0.7570	0.6801	0.5735	0.5310
DySAT [37]	0.9326	0.9189	0.9011	0.8903	0.2639	0.2187	0.1986	0.1803
GNODE-GNODE (ours)	0.9922	0.9902	0.9833	0.9802	0.7453	0.6803	0.5785	0.5336
GNODE-IGRNN$_v$ (ours)	0.9907	0.9896	0.9816	0.9779	0.7573	0.6801	0.5736	0.5309
IGRNN-GNODE$_v$ (ours)	0.9932	0.9907	0.9849	0.9822	**0.7776**	**0.6897**	**0.6056**	0.5654
GNODE-GNODE$_v$ (ours)	0.9930	0.9906	**0.9852**	**0.9824**	0.7707	0.6874	0.6043	**0.5662**

Table 10: Confidence intervals for different methods in the edge-based setting on *Enron*.

Metric	AUC				AP			
Stride	30	50	100	150	30	50	100	150
SGRNN-SGRNN [4]	0.0004	0.0008	0.0011	0.0030	0.0201	0.0170	0.0161	0.0395
SGRNN-SGRNN$_v$ [4]	0.0002	0.0017	0.0002	0.0011	0.0045	0.0197	0.0016	0.0082
IGRNN-IGRNN [38]	0.0003	0.0001	0.0002	0.0004	0.0010	0.0012	0.0021	0.0039
IGRNN-IGRNN$_v$ [8]	0.0002	0.0000	0.0002	0.0005	0.0006	0.0052	0.0014	0.0021
DySAT [37]	0.0000	0.0002	0.0003	0.0000	0.0001	0.0002	0.0003	0.0001
GNODE-GNODE (ours)	0.0001	0.0002	0.0001	0.0002	0.0023	0.0019	0.0009	0.0029
GNODE-IGRNN$_v$ (ours)	0.0001	0.0002	0.0001	0.0001	0.0057	0.0061	0.0010	0.0019
IGRNN-GNODE$_v$ (ours)	0.0000	0.0007	0.0001	0.0001	0.0008	0.0156	0.0007	0.0021
GNODE-GNODE$_v$ (ours)	0.0002	0.0008	0.0002	0.0001	0.0042	0.0088	0.0034	0.0016

Table 11: AUC and AP for different methods in the edge-based setting on *Eu-core*.

Metric	AUC					AP				
Stride	100	200	300	400	500	100	200	300	400	500
IGRNN-IGRNN [38]	0.9884	0.9830	0.9817	0.9770	0.9710	0.8585	0.7704	0.6911	0.5869	0.5220
IGRNN-IGRNN$_v$ [8]	0.9833	0.9772	0.9749	0.9692	0.9629	0.8351	0.7462	0.6736	0.5732	0.5230
DySAT [37]	0.9457	0.9353	0.9234	0.9300	0.9129	0.1627	0.1473	0.1401	0.1424	0.1256
GNODE-GNODE (ours)	0.9883	0.9851	0.9839	0.9794	0.9755	**0.8647**	0.7759	0.7080	0.6028	0.5424
GNODE-IGRNN$_v$ (ours)	0.9835	0.9782	0.9750	0.9693	0.9627	0.8359	0.7467	0.6731	0.5748	0.5230
IGRNN-GNODE$_v$ (ours)	**0.9901**	0.9869	0.9866	**0.9827**	**0.9785**	0.8604	0.7823	0.7090	**0.6046**	0.5432
GNODE-GNODE$_v$ (ours)	0.9901	**0.9869**	**0.9866**	0.9826	0.9783	0.8624	**0.7832**	**0.7096**	0.6039	**0.5440**

Table 12: Confidence intervals for different methods in the edge-based setting on *Eu-core*.

Metric	AUC					AP				
Stride	100	200	300	400	500	100	200	300	400	500
IGRNN-IGRNN [38]	0.0002	0.0004	0.0008	0.0009	0.0009	0.0017	0.0013	0.0018	0.0025	0.0015
IGRNN-IGRNN$_v$ [8]	0.0005	0.0009	0.0003	0.0010	0.0011	0.0004	0.0010	0.0012	0.0017	0.0005
DySAT [37]	0.0000	0.0001	0.0002	0.0157	0.0001	0.0016	0.0002	0.0003	0.0147	0.0001
GNODE-GNODE (ours)	0.0003	0.0003	0.0001	0.0001	0.0001	0.0012	0.0019	0.0003	0.0009	0.0013
GNODE-IGRNN$_v$ (ours)	0.0003	0.0002	0.0003	0.0009	0.0010	0.0002	0.0011	0.0023	0.0014	0.0024
IGRNN-GNODE$_v$ (ours)	0.0001	0.0001	0.0001	0.0002	0.0000	0.0003	0.0007	0.0001	0.0009	0.0010
GNODE-GNODE$_v$ (ours)	0.0000	0.0002	0.0001	0.0001	0.0002	0.0003	0.0006	0.0008	0.0008	0.0008