On The Temporal Domain of Differential Equation Inspired Graph Neural Networks

Moshe Eliasof[1] **Eldad Haber**[2] **Eran Treister**[3] **Carola-Bibiane Schönlieb**[1]
[1] University of Cambridge [2] University of British Columbia [3] Ben-Gurion University of the Negev

Abstract

Graph Neural Networks (GNNs) have demonstrated remarkable success in modeling complex relationships in graph-structured data. A recent innovation in this field is the family of Differential Equation-Inspired Graph Neural Networks (DE-GNNs), which leverage principles from continuous dynamical systems to model information flow on graphs with built-in properties such as feature smoothing or preservation. However, existing DE-GNNs rely on first or second-order temporal dependencies. In this paper, we propose a neural extension to those pre-defined temporal dependencies. We show that our model, called TDE-GNN, can capture a wide range of temporal dynamics that go beyond typical first or second-order methods, and provide use cases where existing temporal models are challenged. We demonstrate the benefit of learning the temporal dependencies using our method rather than using pre-defined temporal dynamics on several graph benchmarks.

1 INTRODUCTION

Graph neural networks (GNNs) are now ubiquitous in diverse applications from social media to chemistry and physical systems, see Wu et al. (2020); Wang et al. (2021) and references within. In recent years, it has been shown that GNNs can be viewed as dynamical systems. Specifically, Ordinary Differential Equations (ODE) based methods have been found to be useful, providing understandable behavior, such as smoothing (Poli et al., 2019; Chamberlain et al., 2021), energy conservation (Eliasof et al., 2021; Rusch et al., 2022), anti-symmetry (Gravina et al., 2023), pattern formation (Wang et al., 2022; Choi et al., 2023b), and more. We refer to this family of architectures as DE-GNNs.

Many works in the field of DE-GNNs consider *stationary data*, that is, data that is not time dependent. As such, most works focus on the *spatial* interactions between nodes, and how to better model them. At the same time, the majority of existing DE-GNNs employ first-order temporal dynamics, which, as we show later in Example 1, can be limiting. Therefore, in this paper, we study the importance of the time domain of DE-GNNs, and propose a novel mechanism to model the temporal order and dependencies of the underlying ODEs of GNN layers in a data-driven fashion. As we show, the utilization of the proposed advanced temporal domain learning mechanism offers a practical performance advantage, while also naturally bridging between works that have been proposed to handle tasks for *non-stationary data* by time-dependent graph neural networks (TD-GNNs), as in Taheri and Berger-Wolf (2019); Guan et al. (2022); Xiong et al. (2020); Pilva and Zareei (2022); Longa et al. (2023).

The goal of this work is to develop and study a novel mechanism to model the time domain of DE-GNNs. Our approach, called *TDE-GNN*[1] , is based on learning (i) the *temporal order*, and, (ii) the *temporal dependency* in a data-driven fashion. The temporal order defines the order of the underlying dynamics as favored by the data, while the temporal dependency specifies the relationship between intermediate DE steps. To the best of our knowledge, this is the first work to study the temporal domain of DE-GNNs, in the sense that it learns higher-order DEs in a general manner. All DE-GNNs known to us, utilize either first or second-order time dependencies. In other words, existing DE-GNNs assume fixed temporal behavior, which is constant in time and is pre-defined. This shortcoming, as we show later, can be rather limiting when complex phenomena are to be modeled. Our TDE-GNN can also be viewed as an extension for Residual Networks (He et al., 2016), which is aimed to incorporate node features from

Proceedings of the 27th International Conference on Artificial Intelligence and Statistics (AISTATS) 2024, Valencia, Spain. PMLR: Volume 238.

[1]Read as Teddy-GNN.

previous time steps (i.e., layers) in a learnable manner.

2 RELATED WORK

Graph Neural Networks Inspired by Differential Equations. Adopting the interpretation of convolutional neural networks (CNNs) as discretizations of ODEs and PDEs (Ruthotto and Haber, 2018; Chen et al., 2018b; Zhang et al., 2019) to GNNs, works like GCDE (Poli et al., 2019), GODE (Zhuang et al., 2020), GRAND (Chamberlain et al., 2021), PDE-GCN$_{\mathrm{D}}$ (Eliasof et al., 2021), GRAND++ (Thorpe et al., 2022) and others, propose to view GNN layers as time steps in the integration of the non-linear heat equation. This perspective allows to control the diffusion (smoothing) in the network, to understand oversmoothing (Nt and Maehara, 2019; Oono and Suzuki, 2020; Cai and Wang, 2020) in GNNs. Thus, works like Chien et al. (2021); Luan et al. (2022); Giovanni et al. (2023) propose to utilize a *learnable* diffusion term, thereby alleviating oversmoothing. Other architectures like PDE-GCN$_{\mathrm{M}}$ (Eliasof et al., 2021) and GraphCON (Rusch et al., 2022) propose to mix diffusion and oscillatory processes (e.g., based on the wave equation) to avoid oversmoothing by introducing a feature energy preservation mechanism. Nonetheless, as noted in Rusch et al. (2023), besides alleviating oversmoothing, it is also important to design GNN architectures with improved expressiveness. Recent examples of such networks are Gravina et al. (2023) that propose an anti-symmetric GNN to alleviate over-squashing (Alon and Yahav, 2021), Wang et al. (2022); Choi et al. (2023b) that formulate a reaction-diffusion GNN to enable non-trivial pattern growth, Zhao et al. (2023) that propose a convection-diffusion based GNN, advection-reaction-diffusion to allow directed information transportation (Eliasof et al., 2023), and Maskey et al. (2023) that formalize a fractional Laplacian ODE based GNN with improved expressiveness. A common theme of most of the aforementioned works, is the focus on the *spatial* term of the ODE, while the temporal term is set to be of first or second order. In this work, we propose to extend the family of ODE-inspired GNNs from the perspective of the *temporal* domain.

The Temporal Domain in Graph Neural Networks. In recent years, GNNs for spatio-temporal data were developed. Some examples are Chen et al. (2018a); Seo et al. (2018); Zhao et al. (2019) that combine graph convolution with LSTM mechanisms, and other combines graph attention with temporal mechanisms, as in Zhu et al. (2020a). Other works like Pareja et al. (2020); Bai et al. (2020) propose adaptive graph convolutions for temporal graphs. It has also been shown in Gutteridge et al. (2023) that adjacency matrix update according to intermediate node features is useful for long-range benchmarks. Furthermore, recent works have shown that GNNs for temporal graph datasets can benefit from the interpretation and construction of ordinary differential equations. For example, it was shown in Xiong et al. (2023); Sun et al. that reaction and diffusion systems can improve traffic prediction, and it was shown in Choi et al. (2023a) that advection and diffusion can improve weather forecasting performance. However, all the considered works discussed here utilize first-order temporal dynamics, while focusing on the spatial term of the ordinary differential equation. In this paper, we explore and study the temporal domain in the context of DE-GNNs, and show its importance to model complex systems and improve performance.

Multihop Graph Neural Networks. Multihop GNN architectures were extensively studied in previous years, leading to several popular architectures such as JK-Net (Xu et al., 2018) and MixHop (Abu-El-Haija et al., 2019). These works take inspiration from earlier works like DenseNets (Huang et al., 2017), where the main idea is to consider a combination of feature maps from multiple layers, instead of only considering the last layer feature map as in ResNets (He et al., 2016). By interpreting layers as time steps, similarities between higher-order DE-inspired GNNs and multihop methods can be established. We distinguish our TDE-GNN from JK-Net and MixHop in a three-fold manner. First, these methods do not stem from an ODE perspective that allows to construct higher-order DE-GNNs. Second, methods like JK-Net can become computationally expensive if many layers are used within a network, as it considers all previous layers. On the other hand, our TDE-GNN is bounded by a maximal order hyperparameter, as we discuss later. Third, in TDE-GNN we propose a novel attention mechanism to learn the relations between layers which was not studied in previous works.

3 MATHEMATICAL BACKGROUND AND MOTIVATION

In this section, we provide a related mathematical overview, and motivate the necessity of our TDE-GNN architecture that enables higher-order DE-GNNs, by an example where first-order models are challenged.

Notations. We consider a graph $G = (V, E)$, where V is a set of n nodes, and $E \subseteq V \times V$ is a set of m edges. The i-th node is associated with a possibly time-dependent hidden feature vector $f_i(t) \in \mathbb{R}^k$. Let $F(t) = [f_0(t), \ldots, f_{n-1}(t)]^\top$ be a $n \times k$ matrix that represents the node state (features) at time t.

Differential Equations Inspired GNNs (DE-GNNs). The basic idea of the DE-GNN family of architectures is to discretize the following ODE:

$$\frac{dF}{dt} = s\left(F(t); G\right) \tag{1a}$$

$$F(t=0) = \mathbf{F}^{(0)} \tag{1b}$$

where $s(F(t); G)$ is a spatial operator that depends on the graph G and the node features $F(t)$. Specifically, it is common to employ graph diffusion, combined with a channel mixing operator implemented by a multilayer perceptron (MLP). Some examples of such methods were proposed in Chamberlain et al. (2021); Eliasof et al. (2021); Thorpe et al. (2022); Choi et al. (2023b), and others. Because we focus on the *temporal* component of the ODE in this paper, we employ a similar spatial term that combines diffusion and channel mixing, as discussed later. Then, the graph ODE in Equation (1) is discretized in time, until time T, typically with the forward Euler method. The chosen discretization times are considered as GNN layers, with a total of L time steps with step size h such that $T = hL$.

For stationary problems (e.g., node classification), the input consists of a single time step at time T_0. Given input features $\mathbf{I}^{(0)} \in \mathbb{R}^{n \times k_{in}}$, we embed them using an MLP to obtain the initial conditions of the ODE, denoted by $\mathbf{F}^{(0)} \in \mathbb{R}^{n \times k}$. For spatio-temporal tasks (e.g., forecasting node quantities), the node features $[\mathbf{I}^{(0)}, \ldots, \mathbf{I}^{(r)}]$ are provided at sampled times $[T_0, \ldots, T_r]$, and embedded in latent space. The node features at the final GNN layer $\mathbf{F}^{(L)}$ are then fed to a classifier to output the desired shaped prediction to be compared with the labeled data, depending on the task.

The right-hand side of Equation (1a) describes the *spatial* behavior of the DE-GNN, and has been thoroughly studied in previous works, as discussed in Section 2. The left-hand side, which describes the temporal order and dynamics of the ODE, however, did not receive significant attention, to the best of our knowledge. Most of the works known to us, also discussed in Section 2, consider only first-order time dynamics, with the exception of Eliasof et al. (2021); Rusch et al. (2022) that limit their models to second-order dynamics. Thus, in this work, we focus on the left-hand side of Equation (1a), which describes the temporal *order* and *dynamics* of DE-GNNs. We will show that learning the temporal domain of the DE-GNN offers two major benefits: (i) interpretable learned weights in the time domain, and (ii) improved downstream task performance.

Problem Formulation. The downstream tasks considered in this work aim to predict node values, either by regression or classification. The common theme among the considered tasks is that we view them as the prediction of the time and space evolution of the node features, given past and current node features. A popular approach is to treat the problem by combining a GNN with time series mechanisms such as LSTM (Hochreiter and Schmidhuber, 1997) or GRU (Cho et al., 2014), to predict the future state by using the current state, as discussed in Section 2. While such techniques have shown promising results, we provide Example 1, where a standard GNN-LSTM is challenged, in the sense that it does not perform better than a naive solution. We attribute this shortcoming to the basic assumption of models like LSTM, that future predictions can be based on the previous state, effectively assuming first-order dynamics, which may not be sufficient to model higher-order phenomena, as shown below.

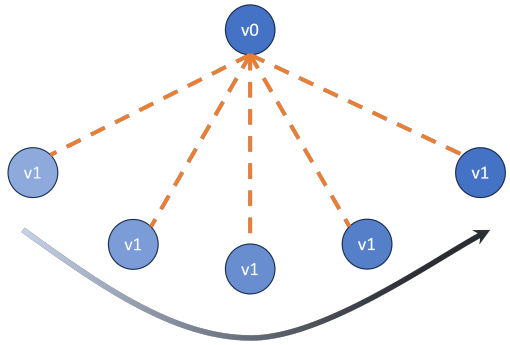


Figure 1: An illustration of a pendulum.

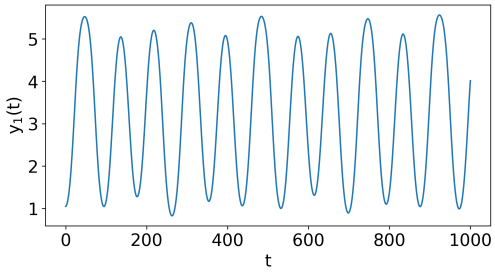
Example 1. (Nonlinear Pendulum) *Let us consider the problem of a nonlinear pendulum. The pendulum can be treated as a graph with two nodes. The first node v_0 is fixed at $(0, 0)$, and the second, v_1, is located at $(x_1(t), y_1(t))$ that evolve in time. We illustrate the pendulum system in Figure 1. Evaluating the coordinates of the nodes v_0, v_1 at time t can be done by solving the Newtonian mechanics that define the nonlinear pendulum's motion. Specifically, it is described by the following equation:*

$$\frac{\partial^2 F}{\partial t^2} = q(F), \tag{2}$$

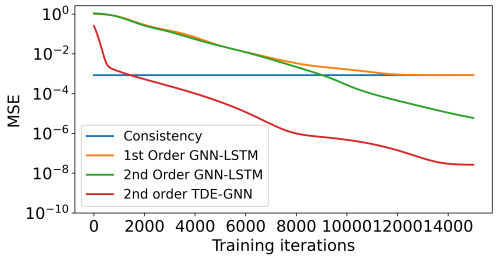
where $q(F)$ is a gradient of the energy that characterizes the behavior of the pendulum discussed in Appendix C.1.

We discretize Equation (2) using the leapfrog method (Ascher, 2008), to generate a time series data of the pendulum vertices locations. Recall that node v_0 is static, and remains in (0,0), while v_1 moves according to Equation (2). We plot the location of v_1 in Figure 2a.

We define a task whose inputs are observed locations of the pendulum nodes, and the goal is to predict the



(a)



(b)

Figure 2: The pendulum location prediction example. (a). Pendulum $y_1(t)$ coordinate vs. time (b) The prediction performance of naive, 1st, and 2nd order models. Higher-order models offer improved predictions.

future locations of nodes. We consider four possible prediction models to address this task: (i) A *naive* prediction model, oftentimes called a *consistency* model, that simply outputs the latest available state. Formally, it is given by $\mathbf{F}^{(l+1)} = \mathbf{F}^{(l)}$. (ii) A GNN-LSTM, similar to Seo et al. (2018), (iii) a second order GNN-LSTM discussed in Appendix C.1, and, (iv) our TDE-GNN limited to second order for a fair comparison, to be defined later in Section 4.

We report the obtained prediction mean squared error (MSE) compared to the ground-truth data in Figure 2b. We observe that the first-order GNN-LSTM model performs as good as the naive model of consistency, thereby not offering improved results as one would like. To understand the limitations of GNN-LSTM, it is key to recall Equation (2), and see that a pendulum's motion involves a second-order system. That is, to predict a future location $\mathbf{F}^{(l+1)}$, one is required to use both $\mathbf{F}^{(l)}$ and $\mathbf{F}^{(l-1)}$. However, the first-order GNN-LSTM mechanism considers only the latest state (node features) $\mathbf{F}^{(l)}$. Indeed, when considering a second-order GNN-LSTM that involves both $\mathbf{F}^{(l)}$ and $\mathbf{F}^{(l-1)}$ one can obtain improved performance. Finally, we see that our TDE-GNN limited to second-order offers further prediction performance improvement. The convergence curves of the considered networks are plotted in Figure 2b.

This example demonstrates that the order (length) of the history used for prediction is important. If the history used is too short, it may be impossible to accurately predict the behavior of systems with order higher than the available history length. Since for many problems, the order is unknown and can vary in time, we allow TDE-GNN to learn the order from the data.

4 TIME DEPENDENT DIFFERENTIAL EQUATIONS INSPIRED GNNS

Example 1 demonstrates the importance of utilizing higher-order temporal behavior to fit complex data. Motivated by this example, we propose and study, a method that can learn the *temporal* domain of the DE-GNNs from the data, in addition to leveraging useful spatial terms, as proposed in other works and discussed in Section 2. Therefore, we call our method *TDE-GNN*.

4.1 TDE-GNN Learns Higher-Order DEs

As discussed, previous works have so far mostly considered first order time dependent GNNs as in Equation (1), whose forward Euler discretization reads:

$$\mathbf{F}^{(l+1)} = \mathbf{F}^{(l)} + hs(\mathbf{F}^{(l)}; G), \tag{3}$$

where $\mathbf{F}^{(l)} \in \mathbb{R}^{n\times k}$ are the node features at the l-th layer, h is a positive discretization step size, and s is the spatial term. To focus on the proposed temporal mechanism, in this work we follow previous works that combine graph diffusion with channel mixing. We elaborate on the implementation of s in Appendix B.3.

In this paper, we generalize and study the time order and dynamics, and introduce a TDE-GNN layer that stems from the following ODE, with a maximal order of $o \geq 1$, which is a hyperparameter:

$$\sum_{p=1}^{o} c_p \frac{d^p F}{dt^p} = s\left(F(t); G\right), \tag{4}$$

accompanied by the initial conditions

$$\left.\frac{d^p F}{dt^p}\right|_{t=0} = F^{(p)}(t=0) \quad p = 0, \ldots, o-1. \tag{5}$$

The forward Euler discretization of Equation (4) yields a layer of our extended, higher-order, TDE-GNN layer:

$$\mathbf{F}^{(l+1)} = \sum_{p=1}^{o} c_p(\mathcal{H}_o^{(l)})\mathbf{F}^{(l-p+1)} + hs(\mathbf{F}^{(l)}; G). \tag{6}$$

Here, we define $\mathbf{c}(\mathcal{H}_o^{(l)}) = [c_1(\mathcal{H}_o^{(l)}), \ldots, c_o(\mathcal{H}_o^{(l)})] \in \mathbb{R}^o$, which are learned weights based on previous node features of up to order o, formally denoted by:

$$\mathcal{H}_o^{(l)} = [\mathbf{F}^{(l)} \| \mathbf{F}^{(l-1)} \| \ldots \| \mathbf{F}^{(l-o+1)}] \in \mathbb{R}^{o\times n\times k}, \tag{7}$$

where $\|$ denotes the stacking operation. It is important to note that Equation (6) models the dynamics of the l-th layer with $o-1$ previous layers, and therefore yields a discretization of an ODE of order o. For stability of computation, and for interoperability, we demand that $\sum_{p=1}^{o} c_p = 1$. This constraint ensures that the coefficients approximate derivatives of the data up to o-th order (Evans, 1998). Because in this paper we focus on the temporal behavior of the underlying graph ODE, in Section 4.2 we describe our mechanism that learns the temporal order and dynamics encoded by $\mathbf{c}(\mathcal{H}_o^{(l)})$, and for completeness, in Appendix B.3 we discuss the spatial term of our TDE-GNN.

Understanding the learned coefficients $\mathbf{c}(\mathcal{H}_o^{(l)})$. Before we proceed with implementation details, it is important to note that Equation (6) extends the idea of residual networks. Typical residual networks (as in ResNet (He et al., 2016)) can be obtained by Equation (6) with $o = 1$ and $c_1 = 1$, which yields the forward Euler method that is often used to discretize diffusive GNNs (Chamberlain et al., 2021; Eliasof et al., 2021). Also, second order oscillatory GNNs as proposed in Eliasof et al. (2021); Rusch et al. (2022), can be implemented by Equation (6) with $o = 2$ and $\mathbf{c} = [c_1, c_2] = [2, -1]$. Overall, our TDE-GNN can implement, as well as extend, both of these types of architectures by learning higher-order dynamics with adaptive coefficients $\mathbf{c}(\mathcal{H}_o^{(l)})$. Those extensions allow our TDE-GNN to model a diverse family of dynamics that cannot be obtained with the aforementioned methods. We now provide Example 2 that shows how a third-order DE-GNN is implemented by our method and how the learned coefficients can be interpreted.

Example 2. (3rd Order TDE-GNN) *We now draw a link between a third-order TDE-GNN (with an order hyperparameter $o = 3$), and a third-order ODE. Note that every set of coefficients $\{c_1, c_2, c_3\}$ that sum to 1, with a step size $h = 1$ can be spanned by the basis:*

$$\begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} = \alpha_1 \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} + \alpha_2 \begin{pmatrix} 2 \\ -1 \\ 0 \end{pmatrix} + \alpha_3 \begin{pmatrix} 2 \\ -2 \\ 1 \end{pmatrix}, \quad (8)$$

with the constraint $\sum_{i=1}^{3} \alpha_i = 1$. Note that the vectors that multiply α_1, α_2 and α_3 correspond to a first-order, second-order, and third-order finite difference, respectively (i.e., $\frac{dF(t)}{dt}$, $\frac{d^2F(t)}{dt^2}$, and $\frac{d^3F(t)}{dt^3}$). Since the basis in Equation (8) is complete, for any c_1, c_2, c_3 that sum to 1, there exists a 3rd order differential equation whose discretization yields the same coefficients.

Thus, treating the temporal domain in DE-GNNs using our learnable framework allows us to *reveal and understand the order of the underlying time-dependent process in a data-driven fashion.* Furthermore, as we have shown in Example 1, such a treatment can be crucial to accurately model data that stems from higher-order phenomena. Namely, if the system we intend to predict is of order o and we do not expose the network to a history of at least o time steps, then accurate predictions may not be possible.

4.2 Implementing $\mathbf{c}(\mathcal{H}_o^{(l)})$

At the core of our TDE-GNN stands the learning of the temporal coefficients $\mathbf{c}(\mathcal{H}_o^{(l)})$, with two key requirements for a valid implementation: (i) the vector $\mathbf{c}$ sums to 1, i.e., $\sum_{p=1}^{o} c_p(\mathcal{H}_o^{(l)}) = 1$, and, (ii) the entries of $\mathbf{c}(\mathcal{H}_o^{(l)})$ can be any real-valued number. These requirements offer both training stability (due to the normalization in requirement (i)), and the approximation of a finite order derivative (Evans, 1998). We now discuss two implementations that we consider in this paper, and later, in our experiments in Section 5, we compare their performance. In appendix B.1 we discuss a possible implementation that transforms a higher-order ODE into a system of first-order ODEs, and our rationale for using maintaining the view of a higher-order ODE.

Direct parameterization. Perhaps the most intuitive implementation is obtained by *direct* parameterization, where we directly learn a vector $\tilde{\mathbf{c}} \in \mathbb{R}^o$, and divide it by the sum of its entries (to satisfy requirement (i)), leading to the temporal coefficients vector:

$$\mathbf{c} = \frac{\tilde{\mathbf{c}}}{\sum_{p=1}^{o} \tilde{c}_p}. \quad (9)$$

Note that in this case, the coefficients vector $\mathbf{c}$ is not directly influenced by the history $\mathcal{H}_o^{(l)}$, but it is still optimized according to the history via backpropagation.

Attention-based parameterization. In addition to the direct parameterization, we propose a novel mechanism that leverages an attention mechanism as in Vaswani et al. (2017). A key feature of the attention mechanism is that it outputs a pairwise score of its input. The *novelty* here is to apply the attention mechanism on the *temporal* dimension. To this end, by collecting and appropriately shaping the node features of the previous o layers, denoted by $\mathcal{H}_o^{(l)} \in \mathbb{R}^{o \times n \times k}$ as defined in Equation (6), and feeding it to an attention layer (Vaswani et al., 2017), one obtains a pairwise score map $\mathcal{S} \in [0, 1]^{o \times o}$. The last row in $\mathcal{S}$ represents the temporal scores of the l-th layer with the previous layers $o - 1$. Clearly, requirement (i) is met by the SoftMax function used in the attention mechanism in Vaswani et al. (2017). However, a SoftMax function yields non-negative pairwise values, which do not satisfy requirement (ii). Such a limitation will prevent, for

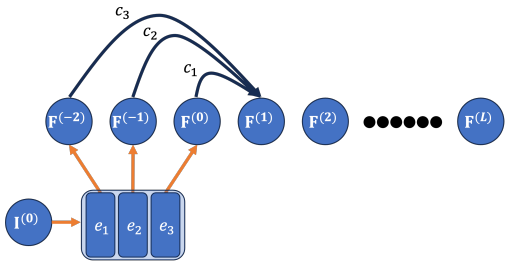


Figure 3: The embedding of input features $\mathbf{I}$ using an MLPs (e_1, e_2, e_3) to obtain $o = 3$ initial conditions, followed by our TDE-GNN for stationary problems.

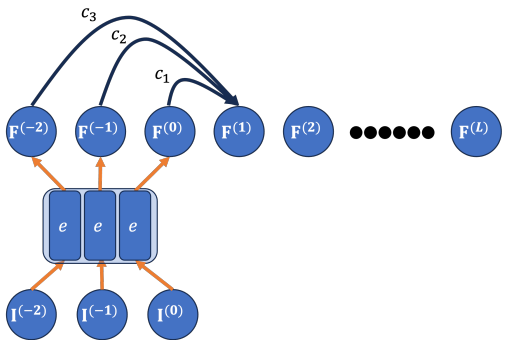


Figure 4: The initialization of TDE-GNN for spatio-temporal data with a history of $o = 3$.

instance, the ability to implement the oscillatory equation discussed in 4.1 using our TDE-GNN. Therefore, we follow the same implementation as in Vaswani et al. (2017) up to the SoftMax step. Instead, we only apply a normalization of the obtained pairwise interaction map by dividing it by its sum. This procedure satisfies both requirements (i) and (ii). We provide further details about the implementation in Appendix B.2.

Initial conditions. When considering high-order ODEs, the aspect of the initial conditions of the model is important (Ascher and Petzold, 1998). We consider two use cases that are treated differently. First, when solving a stationary problem such as node classification, where only a single initial temporal condition is available, we use o MLPs to embed this single state into o states, and then use the network in Equation (6). This initialization, as well as the application of a TDE-GNN at the first layer, is illustrated in Figure 3.

For time series graph problems where we have a time series as input, we use at least o historical data in order to initialize the states. In this case, the frequency of the *observed input* can be different than the frequency of the hidden space $\mathbf{F}^{(l)}$ that discretizes the ODE, in the sense that we can use more hidden layers than observed inputs. Upon receiving at least o input observations, we embed them using an MLP to obtain o hidden initial conditions. In Figure 4, we illustrate the described process as well as the application of a TDE-GNN layer to the inputs. Past the initialization step, in both stationary and non-stationary cases, the features update relies on previously computed hidden node features, as described in Equation (6).

Complexity. Compared to existing DE-GNN methods, our TDE-GNN involves additional $o-1$ additions of previous node features, to allow modeling differential equations of order o, and achieve improved performance, as we show in Section 5. If the coefficients $\mathbf{c}$ are obtained using the *direct* parameterization, then $o-1$ scalar multiplications are required. If the *attention* based mechanism is utilized to learn and evaluate $\mathbf{c}$, adding $\mathcal{O}(n \cdot k \cdot o^2)$ multiplications. We note that o, the order hyperparameter of TDE-GNN, is typically significantly smaller than the number of channels k and nodes n, because it is bounded by the number of layers L. In Appendix C.4 we report the training and inference runtimes of the proposed implementations.

Properties of TDE-GNN. Our TDE-GNN draws inspiration from a stable discrete process of ODE integration, that generates future time values, which is regarded as the node features evolution throughout the layers. Therefore, a natural question that arises is whether the obtained network is stable. Indeed, if the proposed architecture is unstable, then it may be difficult to fit the data, or, the network may not generalize well (see Haber and Ruthotto (2017) for stability definition and a thorough discussion). To this end, we prove the following theorem in Appendix A.

Theorem 1 *(Stability of TDE-GNN). For the discretization of Equation* (6)*, there exists a vector* $\mathbf{c} = [c_1, \ldots, c_o]$ *such that the discrete solution is stable.*

In our ablation study in Section 5.3, we verify Theorem 1, and show that the learnable weights $\mathbf{c}$ can be interpreted as finite difference derivatives.

5 EXPERIMENTS

To demonstrate the efficacy of TDE-GNN, we experiment with two tasks: (i) node classification, and, (ii) spatio-temporal node forecasting, on several benchmarks. We provide benchmark details and statistics in Appendix C.2. The hyperparameters are determined using a grid search, as discussed in Appendix C.3. Because we propose two possible implementations of the temporal learning mechanism in Section 4.2, we denote the *direct* parameterization variant by TDE-GNN$_{\text{D}}$ and the *attention* based parameterization variant by TDE-GNN$_{\text{A}}$. A detailed de-

Dataset Homophily	Squirrel 0.22	Film 0.22	Chameleon 0.23	Citeseer 0.71	Pubmed 0.74	Cora 0.81
General GNNs						
GCN	23.96±2.01	26.86±1.10	28.18±2.24	73.68±1.36	88.13±0.50	85.77±1.27
GAT	30.03±1.55	28.45±0.89	42.93±2.50	74.32±1.23	87.62±1.10	86.37±0.48
GCNII*	38.47±1.58	32.87±1.30	60.61±3.04	77.13±1.48	90.30±0.43	88.49±1.25
Geom-GCN*	38.32±0.92	31.63±1.15	60.90±2.81	77.99±1.15	90.05±0.47	85.27±1.57
GGCN	55.17±1.58	37.81±1.56	71.14±1.84	77.14±1.45	89.15±0.37	87.95±1.05
H2GCN	36.48±1.86	35.70±1.00	60.11±1.71	77.11±1.57	89.49±0.38	87.87±1.20
FAGCN	42.59±0.69	34.87±1.35	55.22±2.11	74.01±1.85	76.57±1.88	86.34±0.67
GPRGNN	31.61±1.24	34.63±1.22	46.58±1.71	77.13±1.67	87.54±0.38	87.95±1.18
LINKX	61.81±1.80	36.10±1.55	68.42±1.38	73.19±0.99	87.86±0.77	84.64±1.13
ACMII*	67.40±2.21	37.09±1.32	74.76±2.20	77.12±1.58	89.71±0.48	88.25±0.96
Multihop GNNs						
MixHop	43.80±1.48	32.22±2.34	60.50±2.53	76.26±1.33	85.31±0.61	87.61±0.85
JK-Net	45.03±1.73	35.14±1.37	63.79±2.27	76.05±1.37	88.41±0.45	85.96±0.83
GNNs Inspired by DEs						
GRAND	40.05±1.50	35.62±1.01	54.67±2.54	76.46±1.77	89.02±0.51	87.36±0.96
PDE-GCN*	N/A	N/A	66.01±2.11	78.45±1.98	89.93±0.62	88.60±1.77
GRAND++	40.06±1.70	33.63±0.48	56.20±2.15	76.57±1.46	88.50±0.35	88.15±1.22
NSD*	56.34±1.32	37.79±1.15	68.68±1.58	77.14±1.57	89.49±0.40	87.14±1.13
GRAFF*	59.01±1.31	37.11±1.08	71.38±1.47	77.30±1.85	90.04±0.41	88.01±1.03
GREAD*	59.22±1.44	37.90±1.17	71.38±1.30	77.60±1.81	90.23±0.55	88.57±0.66
CDE*	55.04±1.73	40.08±1.49	68.45±2.47	80.04±1.75	90.05±0.64	87.19±1.44
FLODE	64.23±1.84	37.16±1.42	73.60±1.55	78.07±1.62	89.02±0.38	86.44±1.17
Vanilla baseline						
DE-GNN	63.97±1.77	36.04±1.08	70.99±2.27	76.58±1.89	89.92±0.59	87.03±1.14
TDE-GNN (ours)						
TDE-GNN$_{\text{D}}$	70.19±1.74	37.29±1.19	77.38±2.05	77.66±1.91	90.28±0.53	87.99±1.02
TDE-GNN$_{\text{A}}$	71.38±1.93	37.02±1.27	78.48±2.11	77.47±1.82	90.08±0.49	87.93±0.95

Table 1: Node classification accuracy (%). ↑. * denotes the best result out of several variants.

scription of the TDE-GNN architectures is given in Appendix B.4. Our implementation is available at `https://github.com/MosheEliasof/TDE-GNN`.

We compare TDE-GNN with a *baseline* model that we call DE-GNN and is implemented according to Equation (6) with $o = 1$ and $c_1 = 1$, that is, it considers only first-order dynamics, similarly to existing GNNs inspired by DEs. The inclusion of this baseline model to our experiments helps to directly quantify the contribution of our work, TDE-GNN, and our results show the consistent improvement of TDE-GNN over the first-order baseline. Additionally, we compare the obtained performance with other GNNs and in particular to other DE-GNNs such as GRAND (Chamberlain et al., 2021), PDE-GCN (Eliasof et al., 2021), GRAND++ (Thorpe et al., 2022), GREAD (Choi et al., 2023b), CDE (Zhao et al., 2023), and FLODE (Maskey et al., 2023), as well as other GNNs, as described below.

5.1 Node Classification

We experiment with homophilic and non-homophilic datasets. The homophilic datasets are Cora (McCallum et al., 2000), Citeseer (Sen et al., 2008), and Pubmed (Namata et al., 2012). The non-homophilic datasets are Chameleon, Squirrel, and Film from Rozemberczki et al. (2021a). In all cases, we use the 10 splits from Pei et al. (2020), and report their average accuracy and standard deviation in Table 1. We consider four types of baselines: (i) 'general' GNN architectures, such as GCN (Kipf and Welling, 2017), GAT (Veličković et al., 2018), GCNII (Chen et al., 2020), Geom-GCN (Pei et al., 2020), GGCN (Yan et al., 2021), H2GCN (Zhu et al., 2020b), FAGCN (Bo et al., 2021), GPRGNN (Chien et al., 2021), LINKX (Lim et al., 2021), and ACMII (Luan et al., 2022). (ii) Multihop GNNs such as MixHop (Abu-El-Haija et al., 2019) and JK-Net (Xu et al., 2018). (iii) GNNs inspired by differential equations (DEs), including: GRAND (Chamberlain et al., 2021), PDE-GCN (Eliasof et al., 2021), GRAND++ (Thorpe et al., 2022), NSD (Bodnar et al., 2022), GRAFF (Giovanni et al., 2023), GREAD (Choi et al., 2023b), CDE (Zhao et al., 2023), and FLODE (Maskey et al., 2023). The common theme of those methods is that all consider first-order temporal behavior with $o = 1, c_1 = 1$, except for PDE-GCN, which considers a second-order model, where $o = 2, \mathbf{c} = [c_1, c_2] = [2, -1]$. In contrast,

Dataset	Chickenpox Hungary	PedalMe London	Wikipedia Math
Temporal GNNs			
DCRNN	1.124±0.015	1.463±0.019	0.679±0.020
GConvGRU	1.128±0.011	1.622±0.032	0.657±0.015
GC-LSTM	1.115±0.014	1.455±0.023	0.779±0.023
DyGrAE	1.120±0.021	1.455±0.031	0.773±0.009
EGCN-O	1.124±0.009	1.491±0.024	0.750±0.014
A3T-GCN	1.114±0.008	1.469±0.027	0.781±0.011
T-GCN	1.117±0.011	1.479±0.012	0.764±0.011
MPNN LSTM	1.116±0.023	1.485±0.028	0.795±0.010
AGCRN	1.120±0.010	1.469±0.030	0.788±0.011
GNNs Inspired by DEs			
GRAND	1.068±0.021	1.557±0.049	0.798±0.034
GREAD	0.983±0.027	1.291±0.055	0.704±0.016
CDE	0.848±0.020	0.810±0.063	0.694±0.028
Vanilla baseline			
DE-GNN	0.998±0.022	1.329±0.041	0.714±0.019
TDE-GNN (ours)			
TDE-GNN$_{\text{D}}$	0.792±0.028	1.096±0.057	0.614±0.023
TDE-GNN$_{\text{A}}$	0.787±0.018	0.714±0.051	0.565±0.017

Table 2: The performance of spatio-temporal networks evaluated by the average MSE and standard deviation (↓) of 10 experimental repetitions.

our TDE-GNN can learn the vector of coefficients $\mathbf{c}$ with maximal order o, and unless otherwise specified, we set the order to be the number of layers in the network, i.e., $o = L$. The fourth baseline we consider is (iv) a vanilla version of our TDE-GNN, where $o = 1, c_1 = 1$, and we call this variant DE-GNN. Our results in Table 1 suggest that for homophilic graphs which are known to benefit from diffusion (Gasteiger et al., 2019), TDE-GNN performs similarly to other first-order differential equations inspired GNNs, including our baseline DE-GNN, since a diffusion process can be described using a first-order ODE. We find that the significance of learning higher-order dynamics is more pronounced for non-homophilic graphs which may stem from more complex phenomena than homophilic graphs. For instance, we find that our TDE-GNN$_{\text{A}}$ achieves an accuracy of 78.48%, compared to the baseline vanilla, first order DE-GNN with 70.99% – a considerable improvement.

5.2 Spatio-Temporal Forecasting

We now focus on the applicability of TDE-GNN to spatio-temporal datasets, where the goal is to forecast future node values, given time-series data. We use the Chickenpox-Hungary, PedalMe-London, and Wikipedia-Math datasets from Rozemberczki et al. (2021b). We use incremental training, mean-squared-error (MSE) loss, and testing procedure from Rozemberczki et al. (2021b). We report the prediction performance of TDE-GNN, in terms of MSE, in Table 2, and compare it with recent methods like DCRNN (Li et al., 2018), GConv (Seo et al., 2018), GC-LSTM (Chen et al., 2018a), DyGrAE (Taheri et al., 2019; Taheri and Berger-Wolf, 2019), EGCN Pareja et al. (2020), A3T-GCN (Zhu et al., 2020a), T-GCN (Zhao et al., 2019), MPNN LSTM (Panagopoulos et al., 2021), and AGCRN (Bai et al., 2020).

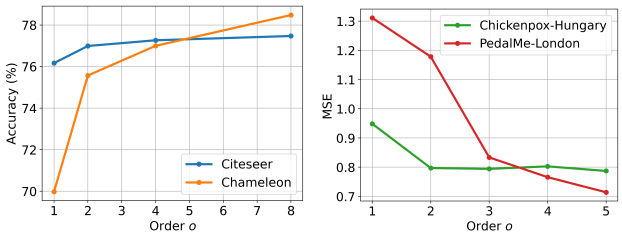
We also provide a comparison with recent DE-inspired methods such as GRAND, GREAD, and CDE. We also provide the important baseline of a vanilla TDE-GNN that utilizes $o = 1, c_1 = 1$. As previously discussed, this architecture is similar to other first-order DE-inspired GNN models, and we therefore call it DE-GNN. We also note again that including the baseline of DE-GNN allows to *directly* measure the contribution of our TDE-GNN with the learnable temporal in Equation (6), and therefore it provides an objective and accurate comparison to our TDE-GNN. Our results are reported in Table 2, and they show improvement over existing temporal GNN models, as well as other DE-inspired GNNs, and the vanilla baseline of DE-GNN. These results further highlight the importance of learning higher-order dynamics offered by our TDE-GNN. Also, we see that the attention-based TDE-GNN$_{\text{A}}$ offers improved performance compared to the directly parameterized TDE-GNN$_{\text{D}}$, which can be attributed to the increased complexity of the attention module.

5.3 Ablation Study

We now study and report two important aspects of our TDE-GNN: the influence of the order o, and an analysis of the learned values in $\mathbf{c}$.

The influence of the order o. As shown in Example 1, having sufficiently high order can be crucial to modeling complex data. In that example, we have demonstrated this significance via a synthetic task where the order is known. We now supplement this study by reporting the obtained performance as a function of the order o on real-world datasets, where the exact order of the underlying process that generated the data is unknown. To provide a comprehensive study of the impact of o, we report the results on both the node classification and spatio-temporal node forecasting tasks considered in this work, on several datasets. The results are reported in Figure 5. Our results indicate, in congruence with Example 1, that higher-order models can improve performance compared to using a first-order model only, as is common in GNNs. We also note that interestingly, for the Chickenpox-Hungary dataset, we see that a second-order model performs almost as well as a third, fourth, or fifth-order model. While we do not know the exact order of such a real-world dataset, the empirical results may hint that the actual underlying process of the spread of the Chickenpox disease in this dataset is of second order.

Inspecting the values $\mathbf{c}(\mathcal{H}_o^{(l)})$. Following Example 1, where the underlying process of the data is known



(a) (b)

Figure 5: The impact of the model order o on the performance of TDE-GNN.

	c_1	c_2	c_3	c_4	c_5
$o=2$	2	-1	–	–	–
$o=3$	1.4	0.2	-0.6	–	–
$o=4$	0.975	0.675	-0.25	-0.4	–
$o=5$	-0.08	1.68	0.153	0.006	-0.759

Table 3: The learned coefficients $\mathbf{c}$ with a varying order $o \in \{2,3,4,5\}$ when solving Example 1.

to be second-order, we now inspect and analyze the obtained coefficients $\mathbf{c}$ for a varying order $o \in \{2,3,4,5\}$.[2] As we show in Appendix C.5, it is possible to verify that the learned coefficients in Table 3 yield a valid discretization of the second-derivative operator $\frac{\partial^2 F}{\partial t^2}$, revealing the true order of the differential equation of the nonlinear pendulum.

6 CONCLUSIONS

In this paper, we studied the temporal domain of GNNs inspired by differential equations. We showed that incorporating higher-order models can be crucial to model data that arises from complex phenomena. This understanding motivated us to develop a novel architecture called TDE-GNN, which utilizes a temporal dynamics learning mechanism, that allows modeling higher-order ODE behaviors in GNNs. Our experimental results show the significance of higher-order GNNs, especially on non-homophilic, and spatio-temporal datasets. Furthermore, the learned temporal coefficients in TDE-GNN allow us to interpret and explain the underlying time-dependent process hidden in the data.

Acknowledgements

ME is a Blavatnik-Cambridge fellow, and is funded by the British Council and the Blavatnik Family Foundation. The authors thank Haggai Maron and Beatrice Bevilacqua for the discussions on the attention-based implementation. The authors thank Beatrice Bevilacqua for the discussions and comments on the manuscript.

[2]Recall that an order of $o = 1$ cannot describe the second-order data in Example 1.

References

Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In *international conference on machine learning*, pages 21–29. PMLR, 2019.

Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=i80OPhOCVH2`.

U. Ascher, R. Mattheij, and R. Russell. *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations*. SIAM, Philadelphia, 1995.

Uri M Ascher. *Numerical methods for evolutionary differential equations*. SIAM, 2008.

Uri M Ascher and Linda R Petzold. *Computer methods for ordinary differential equations and differential-algebraic equations*, volume 61. Siam, 1998.

Lei Bai, Lina Yao, Can Li, Xianzhi Wang, and Can Wang. Adaptive Graph Convolutional Recurrent Network for Traffic Forecasting. *Advances in Neural Information Processing Systems*, 33, 2020.

Deyu Bo, Xiao Wang, Chuan Shi, and Huawei Shen. Beyond low-frequency information in graph convolutional networks. In *AAAI*. AAAI Press, 2021.

Cristian Bodnar, Francesco Di Giovanni, Benjamin Chamberlain, Pietro Liò, and Michael Bronstein. Neural sheaf diffusion: A topological perspective on heterophily and oversmoothing in gnns. *Advances in Neural Information Processing Systems*, 35:18527–18541, 2022.

Chen Cai and Yusu Wang. A note on over-smoothing for graph neural networks. *arXiv preprint arXiv:2006.13318*, 2020.

Ben Chamberlain, James Rowbottom, Maria I Gorinova, Michael Bronstein, Stefan Webb, and Emanuele Rossi. Grand: Graph neural diffusion. In *International Conference on Machine Learning*, pages 1407–1418. PMLR, 2021.

Jinyin Chen, Xuanheng Xu, Yangyang Wu, and Haibin Zheng. GC-LSTM: Graph Convolution Embedded LSTM for Dynamic Link Prediction. *arXiv preprint arXiv:1812.04206*, 2018a.

Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 1725–1735. PMLR, 13–18 Jul 2020. URL http://proceedings.mlr.press/v119/chen20v.html.

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, 2018b.

Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. Adaptive universal generalized pagerank graph neural network. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=n6jl7fLxrP.

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1724–1734. ACL, 2014. doi: 10.3115/v1/d14-1179. URL https://doi.org/10.3115/v1/d14-1179.

Hwangyong Choi, Jeongwhan Choi, Jeehyun Hwang, Kookjin Lee, Dongeun Lee, and Noseong Park. Climate modeling with neural advection–diffusion equation. *Knowledge and Information Systems*, 65(6): 2403–2427, 2023a.

Jeongwhan Choi, Seoyoung Hong, Noseong Park, and Sung-Bae Cho. GREAD: Graph neural reaction-diffusion networks. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 5722–5747. PMLR, 23–29 Jul 2023b. URL https://proceedings.mlr.press/v202/choi23a.html.

Moshe Eliasof, Eldad Haber, and Eran Treister. PDE-GCN: Novel architectures for graph neural networks motivated by partial differential equations. *Advances in Neural Information Processing Systems*, 34:3836–3849, 2021.

Moshe Eliasof, Eldad Haber, and Eran Treister. Adrgnn: Advection-diffusion-reaction graph neural networks. *arXiv preprint arXiv:2307.16092*, 2023.

L. C. Evans. *Partial Differential Equations*. American Mathematical Society, San Francisco, 1998.

Johannes Gasteiger, Stefan Weißenberger, and Stephan Günnemann. Diffusion improves graph learning. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

Francesco Di Giovanni, James Rowbottom, Benjamin Paul Chamberlain, Thomas Markovich, and Michael M. Bronstein. Understanding convolution on graphs via energies. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=v5ew3FPTgb.

Alessio Gravina, Davide Bacciu, and Claudio Gallicchio. Anti-symmetric DGN: a stable architecture for deep graph networks. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=J3Y7cgZOOS.

Mingyu Guan, Anand Padmanabha Iyer, and Taesoo Kim. Dynagraph: dynamic graph neural networks at scale. In *Proceedings of the 5th ACM SIGMOD Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA)*, pages 1–10, 2022.

Benjamin Gutteridge, Xiaowen Dong, Michael M Bronstein, and Francesco Di Giovanni. Drew: dynamically rewired message passing with delay. In *International Conference on Machine Learning*, pages 12252–12267. PMLR, 2023.

Eldad Haber and Lars Ruthotto. Stable architectures for deep neural networks. *Inverse problems*, 34(1): 014004, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=SJU4ayYgl.

Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting. In *International Conference on Learning Representations*, 2018.

Derek Lim, Felix Matthew Hohne, Xiuyu Li, Sijia Linda Huang, Vaishnavi Gupta, Omkar Prasad Bhalerao, and Ser-Nam Lim. Large scale learning on non-homophilous graphs: New benchmarks and strong simple methods. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=DfGu8WwT0d.

Antonio Longa, Veronica Lachi, Gabriele Santin, Monica Bianchini, Bruno Lepri, Pietro Lio, Franco Scarselli, and Andrea Passerini. Graph neural networks for temporal graphs: State of the art, open challenges, and opportunities. *arXiv preprint arXiv:2302.01018*, 2023.

Sitao Luan, Chenqing Hua, Qincheng Lu, Jiaqi Zhu, Mingde Zhao, Shuyuan Zhang, Xiao-Wen Chang, and Doina Precup. Revisiting heterophily for graph neural networks. *Conference on Neural Information Processing Systems*, 2022.

Sohir Maskey, Raffaele Paolino, Aras Bacho, and Gitta Kutyniok. A fractional graph laplacian approach to oversmoothing. *arXiv preprint arXiv:2305.13084*, 2023.

Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval*, 3(2):127–163, 2000.

Galileo Namata, Ben London, Lise Getoor, Bert Huang, and U Edu. Query-driven active surveying for collective classification. In *10th International Workshop on Mining and Learning with Graphs*, volume 8, page 1, 2012.

Hoang Nt and Takanori Maehara. Revisiting graph neural networks: All we have is low-pass filters. *arXiv preprint arXiv:1905.09550*, 2019.

Kenta Oono and Taiji Suzuki. Graph neural networks exponentially lose expressive power for node classification. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=S1ldO2EFPr.

George Panagopoulos, Giannis Nikolentzos, and Michalis Vazirgiannis. Transfer Graph Neural Networks for Pandemic Forecasting. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence*, 2021.

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.

Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-gcn: Geometric graph convolutional networks. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=S1e2agrFvS.

Pourya Pilva and Ahmad Zareei. Learning time-dependent pde solver using message passing graph neural networks. *arXiv preprint arXiv:2204.07651*, 2022.

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

Benedek Rozemberczki, Carl Allen, and Rik Sarkar. Multi-Scale Attributed Node Embedding. *Journal of Complex Networks*, 9(2), 2021a.

Benedek Rozemberczki, Paul Scherer, Yixuan He, George Panagopoulos, Alexander Riedel, Maria Astefanoaei, Oliver Kiss, Ferenc Beres, Guzmán López, Nicolas Collignon, et al. Pytorch geometric temporal: Spatiotemporal signal processing with neural machine learning models. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 4564–4573, 2021b.

T Konstantin Rusch, Ben Chamberlain, James Rowbottom, Siddhartha Mishra, and Michael Bronstein. Graph-coupled oscillator networks. In *International Conference on Machine Learning*, pages 18888–18909. PMLR, 2022.

T Konstantin Rusch, Michael M Bronstein, and Siddhartha Mishra. A survey on oversmoothing in graph neural networks. *arXiv preprint arXiv:2303.10993*, 2023.

Lars Ruthotto and Eldad Haber. Deep neural networks motivated by partial differential equations. *arXiv preprint arXiv:1804.04272*, 2018.

Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

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.

Yue Sun, Chao Chen, Yuesheng Xu, Sihong Xie, Rick S. Blum, and Parv Venkitasubramaniam. Reaction-diffusion graph ordinary differential equation networks: Traffic-law-informed speed prediction under mismatched data. URL https://par.nsf.gov/biblio/10466683.

Aynaz Taheri and Tanya Berger-Wolf. Predictive Temporal Embedding of Dynamic Graphs. In *Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pages 57–64, 2019.

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*, WWW '19, page 301–307, 2019.

Matthew Thorpe, Tan Minh Nguyen, Hedi Xia, Thomas Strohmer, Andrea Bertozzi, Stanley Osher, and Bao Wang. GRAND++: Graph neural diffusion with a source term. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=EMxu-dzvJk`.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=rJXMpikCZ`.

Jianian Wang, Sheng Zhang, Yanghua Xiao, and Rui Song. A review on graph neural network methods in financial applications. *arXiv preprint arXiv:2111.15367*, 2021.

Yuelin Wang, Kai Yi, Xinliang Liu, Yu Guang Wang, and Shi Jin. Acmp: Allen-cahn message passing for graph neural networks with particle phase transition. *arXiv preprint arXiv:2206.05437*, 2022.

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.

Ni Xiong, Yan Yang, Yongquan Jiang, and Xiaocao Ouyang. Diffusion graph neural ordinary differential equation network for traffic prediction. In *2023 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2023. doi: 10.1109/IJCNN54540.2023.10191212.

Xi Xiong, Kaan Ozbay, Li Jin, and Chen Feng. Dynamic origin–destination matrix prediction with line graph neural networks and kalman filter. *Transportation Research Record*, 2674(8):491–503, 2020.

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5453–5462. PMLR, 10–15 Jul 2018. URL `http://proceedings.mlr.press/v80/xu18c.html`.

Yujun Yan, Milad Hashemi, Kevin Swersky, Yaoqing Yang, and Danai Koutra. Two sides of the same coin: Heterophily and oversmoothing in graph convolutional neural networks. *arXiv preprint arXiv:2102.06462*, 2021.

Kuangen Zhang, Ming Hao, Jing Wang, Clarence W de Silva, and Chenglong Fu. Linked dynamic graph cnn: Learning on point cloud via linking hierarchical features. *arXiv preprint arXiv:1904.10014*, 2019.

K. Zhao, Q. Kang, Y. Song, R. She, S. Wang, and W. P. Tay. Graph neural convection-diffusion with heterophily. In *Proc. International Joint Conference on Artificial Intelligence*, Macao, China, Aug 2023.

Ling Zhao, Yujiao Song, Chao Zhang, Yu Liu, Pu Wang, Tao Lin, Min Deng, and Haifeng Li. T-GCN: A Temporal Graph Convolutional Network for Traffic Prediction. *IEEE Transactions on Intelligent Transportation Systems*, 21(9):3848–3858, 2019.

Jiawei Zhu, Yujiao Song, Ling Zhao, and Haifeng Li. A3T-GCN: Attention Temporal Graph Convolutional Network for Traffic Forecasting. *arXiv preprint arXiv:2006.11583*, 2020a.

Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. Beyond homophily in graph neural networks: Current limitations and effective designs. *Advances in Neural Information Processing Systems*, 33:7793–7804, 2020b.

Juntang Zhuang, Nicha Dvornek, Xiaoxiao Li, and James S Duncan. Ordinary differential equations on graph networks. 2020.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Yes]
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Yes]
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [Yes]

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. [Yes]
   (b) Complete proofs of all theoretical results. [Yes]
   (c) Clear explanations of any assumptions. [Yes]

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [Yes]
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Yes]
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [Yes]
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Yes]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. [Yes]
   (b) The license information of the assets, if applicable. [Not Applicable]
   (c) New assets either in the supplemental material or as a URL, if applicable. [Not Applicable]
   (d) Information about consent from data providers/curators. [Not Applicable]
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Not Applicable]

5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. [Not Applicable]
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Not Applicable]
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Not Applicable]

Supplementary Materials: On The Temporal Domain of Differential Equation Inspired Graph Neural Networks

A Proof to Theorem 1

We first define the stability of a solution of an ODE:

Definition 1 *(Stability of an ODE solution). A solution $\mathbf{F}(t)$ of an ODE equipped with appropriate initial condition $\mathbf{F}(0)$, is stable if for any $\omega > 0$, there exists $\delta > 0$ such that any other solution $\tilde{\mathbf{F}}(t)$ of the ODE with initial condition $\tilde{\mathbf{F}}(0)$ satisfying $|\mathbf{F}(0) - \tilde{\mathbf{F}}(0)| \leq \delta$ also satisfies $|\mathbf{F}(t) - \tilde{\mathbf{F}}(t)| \leq \omega$, for all $t \geq 0$.*

We now prove Theorem 1 from the main paper.

Proof: Let us write the discrete DE in Equation (6) explicitly as

$$\mathbf{F}^{(l+1)} = c_o\mathbf{F}^{(l-o+1)} + \ldots c_1\mathbf{F}^{(l)} + hs(\mathbf{F}^{(l)}; G). \tag{10}$$

Similar to the proofs for multistep ODE methods (Ascher et al., 1995), to prove the stability of the method, assuming the Jacobians of s have non-positive real part[3], it is sufficient to consider only the temporal term:

$$\mathbf{F}^{(l+1)} = c_o\mathbf{F}^{(l-o+1)} + \ldots + c_1\mathbf{F}^{(l)}$$

This is a linear, constant-coefficient differential equation, and it must be stable for Equation (6) to be stable. Also, as common in proofs of multistep ODE methods (Ascher et al., 1995), we start from a solution of the form

$$\mathbf{F}^{(l)} = \xi^l$$

(meaning ξ to the power of l). Substituting we obtain

$$\xi^{l+1} = c_o\xi^{l-o+1} + \ldots c_1\xi^l$$

Dividing by ξ^{l-o+1} we obtain the polynomial equation

$$\xi^{o+1} - c_1\xi^o - \ldots - c_o = 0$$

This is a polynomial of degree $o+1$ with coefficients $[1, -c_1, \ldots, -c_o]$ where $\sum c_i = 1$. Let $\rho(c)$ be the roots of the polynomial. It is straightforward to see (by substitution) that 1 is a root of the polynomial, and therefore a constant is a valid solution of Equation (10). For the solution to be stable, we need to have that:

$$\begin{aligned}
&\mathbf{F}^{(n)} = \xi^n \\
&|\mathbf{F}^{(n)}| \leq |\mathbf{F}^{(n-1)}| \\
&|\xi^n| \leq |\xi^{n-1}| \quad \rightarrow \quad |\xi| \leq 1
\end{aligned}$$

In multi-step methods for ODEs, this condition is referred to as the root condition. Furthermore, as shown in Ascher et al. (1995), since the coefficients c are to be determined (learned, in the case of TDE-GNN), there always exists a set of coefficients such that the root condition is satisfied. □

Remark 1 *While it is difficult to verify the root condition analytically, it is possible to compute it numerically, thus revealing the order of the process we learn, as we also show in C.5.*

[3] If the Jacobian has a positive real part, then the underlying ODE is unstable, and therefore its discretization is also unstable.

B Implementation Details

B.1 Higher-Order DE-GNN by First Order ODEs System

As is known (Evans, 1998), it is possible to transform higher-order ODEs into a system of first-order ODEs. Thus, theoretically, it would be possible to model higher-order DE-GNNs as our TDE-GNN using this approach. In our implementation, we chose to maintain the view of a higher-order ODE instead for the following reasons: (i) Using a system of first-order ODEs requires explicit higher-order time differentiation. In the presence of discrete input data (as in our experiments), computing high-order derivatives is prone to noise amplification. In contrast, our computation of the coefficients $\mathbf{c}$ directly from the data allows us to generate stable architectures, as shown in Theorem 1 and validated in our ablation study. (ii) Additionally, using derivatives to express higher-order models results in a system of $o \times k$ channels, leading to channel mixing operations (MLPs) with a cost of $\mathcal{O}(o^2 \times k^2)$, while our implementation is of cost $\mathcal{O}(o \times k^2)$. (iii) Our implementation of TDE-GNN allows a direct interpretation of the time-dependent process by inspecting the values of $\mathbf{c}$.

B.2 Implementing the Temporal Term with Attention

We now describe the implementation of TDE-GNN$_{\text{A}}$, i.e., the TDE-GNN implemented by an attention mechanism. Namely, to learn the dynamics between the node features at a current layer l and the previous $o-1$ layers, we utilize a multi-head self-attention mechanism (Vaswani et al., 2017) that assigns scores between the considered layer l and the $o-1$ layers. The difference in our implementation compared to a standard attention module as in Vaswani et al. (2017) is that we remove the SoftMax normalization step, as discussed in 4.2, and it is required to allow both positive and negative numbers. We denote the attention mechanism by MHA. The MHA computes a score for each pair of layers $(l_i, l_j) \in o \times o$. As an input to the MHA, we use the history feature tensor as described in Equation (7), which is comprised of the stacking of the current and previous (layer-wise) $o-1$ node features. Then, the output of the attention module is given by:

$$\tilde{\mathcal{S}}_o^{(l)} = \text{MHA}(\mathcal{H}_o^{(l)}) \in \mathbb{R}^{o \times o}. \tag{11}$$

The (l_i, l_j)-th entry in $\mathcal{S}_o^{(l)}$ represents the connection between the l_i-th and l_j-th layers. Specifically, the last row of $\mathcal{S}_o^{(l)}$ represents the connection between the current layer l and the previous $o-1$ layers. Therefore we define the unnormalized layer coefficients vector as the last row of $\tilde{\mathcal{S}}_o^{(l)}$, that can be extracted using the following Python notations:

$$\tilde{\mathbf{c}}(\mathcal{H}_o^{(l)}) = \tilde{\mathcal{S}}_o^{(l)}[-1, :] \in \mathbb{R}^o \tag{12}$$

In order to satisfy condition (i) that demands the weights to sum to 1, described in 4.2, we also add a normalization step, such that the coefficients are defined as:

$$\mathbf{c}(\mathcal{H}_o^{(l)}) = \frac{\tilde{\mathbf{c}}(\mathcal{H}_o^{(l)})}{\sum \tilde{\mathbf{c}}(\mathcal{H}_o^{(l)})}. \tag{13}$$

B.3 Implementing the Spatial Term

The temporal mechanism developed in this paper is generic and can possibly be applied to various DE-GNNs, and is the novelty of our work. However, a GNN inspired by differential equations, and therefore also our TDE-GNN, is not complete without the spatial term that propagates node features across the graph. As discussed in Equation (1), our spatial aggregation function s is based on the combination of a channel mixing operation realized by an MLP and the symmetric normalized graph Laplacian $\mathbf{L} = \mathbf{D}^{-\frac{1}{2}}(\mathbf{D} - \mathbf{A})\mathbf{D}^{-\frac{1}{2}}$ where $\mathbf{D}$ is the degree matrix, and $\mathbf{A}$ is the adjacency matrix of the graph G. Formally, the spatial aggregation is given by:

$$s(\mathbf{F}^{(l)}; G) = \sigma\left(\left(\mathbf{F}^{(l)} - h\mathbf{L}\mathbf{F}^{(l)}\right)\mathbf{W}^{(l)}\right), \tag{14}$$

where σ is a non-linear activation function, ReLU in our implementation, and h is a positive step size, and $\mathbf{W}^{(l)} \in \mathbb{R}^{k \times k}$ are the learnable weights of the MLP.

Therefore, substituting the prescribed temporal in Equation (14) into Equation (6) leads to our TDE-GNN layer, as follows:

$$\mathbf{F}^{(l+1)} = \sum_{p=0}^{o-1} c_p(\mathcal{H}_o^{(l)})\mathbf{F}^{(l-p)} + h\sigma\left(\left(\mathbf{F}^{(l)} - h\mathbf{L}\mathbf{F}^{(l)}\right)\mathbf{W}^{(l)}\right). \tag{15}$$

B.4 Architecture details

Node classification architecture. We now elaborate on the TDE-GNN architecture for stationary problems, as used in our node classification experiments. The overall architecture flow is similar to standard GNN architectures for node classification, such as GCN (Kipf and Welling, 2017) and GCNII (Chen et al., 2020). It is composed of initial embedding layers $e_1, \ldots, e_o$, followed by L TDE-GNN layers, and a classifier implemented by a linear layer denoted by e_{out}. The complete flow of this architecture is described in Algorithm 1. We train this architecture on node classification datasets by minimizing the cross-entropy loss between the ground-truth node labels $\mathbf{Y}$ and the predicted node labels $\tilde{\mathbf{Y}}$.

Algorithm 1 TDE-GNN for stationary problems with order o.

Input: Node features $\mathbf{I}^{(0)} \in \mathbb{R}^{n \times k_{in}}$
Output: Predicted node labels $\tilde{\mathbf{Y}} \in \mathbb{R}^{n \times k_{out}}$

procedure TDE-GNN
  $\mathbf{I}^{(0)} \leftarrow \text{Dropout}(\mathbf{I}^{(0)}, p)$
  $\mathbf{F}^{(-o+1)} = e_1(\mathbf{I}^{(0)});\ \mathbf{F}^{(-o+2)} = e_2(\mathbf{I}^{(0)}); \ldots;\ \mathbf{F}^{(0)} = e_o(\mathbf{I}^{(0)})$
  Initialize history tensor $\mathcal{H}_o^{(0)}$ according to Equation (7).
  for $l = 0 \ldots L-1$ **do**
    $\mathbf{F}^{(l)} \leftarrow \text{Dropout}(\mathbf{F}^{(l)}, p)$
    Compute coefficients $\mathbf{c}(\mathcal{H}_0^{(l)})$ according to Section 4.2.
    Update features $\mathbf{F}^{(l+1)}$ according to Equation (15).
    Update history tensor $\mathcal{H}_o^{(l+1)}$ according to Equation (7).
  end for
  $\mathbf{F}^{(L)} \leftarrow \text{Dropout}(\mathbf{F}^{(L)}, p)$
  $\tilde{\mathbf{Y}} = e_{out}(\mathbf{F}^{(L)})$
  Return $\tilde{\mathbf{Y}}$
end procedure

Spatio-Temporal Node Forecasting. The typical task in spatio-temporal datasets is to predict future quantities (e.g., driving speed) given several previous time steps (also called frames). Formally, one is given an input tensor $\mathbf{I}^{in}_{temporal} = [\mathbf{I}^{(0)}, \ldots \mathbf{I}^{(r)}] \in \mathbb{R}^{n \times rk_{in}}$, where r is the number of input (observed) time frames, and the goal is to predict a time frames ahead, i.e., the ground-truth is given by $\mathbf{I}^{gt}_{temporal} = [\mathbf{I}^{(r+1)}, \ldots, \mathbf{I}^{(r+a)}] \in \mathbb{R}^{n \times ak_{in}}$. This is in contrast to stationary datasets such as Cora (McCallum et al., 2000), where input node features $\mathbf{I}^{(0)} \in \mathbb{R}^{n \times k_{in}}$ are given, and the goal is to fit to some ground-truth $\mathbf{Y} \in \mathbb{R}^{n \times k_{out}}$ which can also be of different dimensionality in its output space. In this context, a stationary dataset can be thought of as setting $r = a = 1$ for the non-stationary settings. We show the overall flow of our TDE-GNN architecture for non-stationary problems in Algorithm 2 [4].

In this architecture, we update the hidden state feature matrix $\mathbf{F}^{(l)}_{\text{state}}$ based on the hidden historical feature matrix $\mathbf{F}^{(l)}_{\text{hist}}$. The reason for this construction is that we want to continue from the current, most recent feature $\mathbf{F}^{(l)}_{\text{state}}$, but also consider the given historical data encoded in $\mathbf{F}^{(l)}_{\text{hist}}$.

Similarly to Attention models (Vaswani et al., 2017), we incorporate time embedding based on the concatenation of sine and cosine function evaluations with varying frequencies multiplied by the time of the input frames, as input to our TDE-GNN, denoted by $\mathbf{T}_{\text{emb}} \in \mathbb{R}^{n \times rk_{temb}}$, where we choose the number of frequencies to be 10, and by the concatenation of both sine and cosine lead to $k_{temb} = 20$. We note that the time embedding is computed

[4] In Algorithm 2, $\oplus$ denotes channel-wise concatenation.

in a pre-processing fashion. To initialize the hidden feature matrices $\mathbf{F}^{(0)}_{\text{state}}$, $\mathbf{F}^{(0)}_{\text{hist}}$, we embed the input data $\mathbf{I}^{in}_{\text{temporal}}$, concatenated with $\mathbf{T}_{\text{emb}}$, using two fully connected layers denoted by e^{state} and e^{hist}.

During training, we minimize the mean squared error (MSE) between the ground truth future node quantities and the predicted quantities by TDE-GNN, similar to the training procedure of the rest of the considered methods in Table 2. Specifically, following Rozemberczki et al. (2021b), the goal is to predict the node quantities of the next time frame given 4 previous time frames.

Algorithm 2 TDE-GNN for non-stationary problems with order o.

Input: Node features $\mathbf{I}^{in}_{temporal} = [\mathbf{I}^{(0)}, \dots \mathbf{I}^{(r)}] \in \mathbb{R}^{n \times rk_{in}}$, time embedding $\mathbf{T}_{\text{emb}} \in \mathbb{R}^{n \times rk_{temb}}$

Output: Predicted future node quantities $\tilde{\mathbf{I}}^{pred}_{temporal} = [\tilde{\mathbf{I}}^{(r+1)}, \dots, \tilde{\mathbf{I}}^{(r+a)}] \in \mathbb{R}^{n \times ak_{in}}$

procedure TDE-GNN
  $\mathbf{I}^{in}_{\text{temporal}} \leftarrow \text{Dropout}(\mathbf{I}^{in}_{\text{temporal}}, p)$
  $\mathbf{T}_{\text{emb}} \leftarrow \text{e}^{\text{time-embed}}(\mathbf{T}_{\text{emb}})$
  $\mathbf{F}^{(0)}_{\text{state}} = e^{\text{state}}(\mathbf{I}^{(r)} \oplus \mathbf{T}_{\text{emb}})$
  $\mathbf{F}^{(0)}_{\text{hist}} = e^{\text{hist}}(\mathbf{I}^{in}_{\text{temporal}} \oplus \mathbf{T}_{\text{emb}})$
  Initialize history tensor $\mathcal{H}^{(0)}_o$ according to Equation (7).
  for $l = 0 \dots L-1$ **do**
    $\mathbf{F}^{(l)}_{\text{state}} \leftarrow \text{Dropout}(\mathbf{F}^{(l)}_{\text{state}}, p)$
    Compute coefficients $\mathbf{c}(\mathcal{H}^{(l)}_0)$ according to Section 4.2.
    Update features $\mathbf{F}^{(l+1)}_{state}$ according to Equation (15).
    Update history tensor $\mathcal{H}^{(l+1)}_o$ according to Equation (7).
    $\mathbf{F}^{(l+1)}_{\text{hist}} = e^{\text{hist}}_l(\mathbf{F}^{(l)}_{\text{hist}} \oplus \mathbf{F}^{(l+1)}_{\text{state}} \oplus \mathbf{T}_{\text{emb}})$
  end for
  $\mathbf{F}^{(L)}_{\text{state}} \leftarrow \text{Dropout}(\mathbf{F}^{(L)}_{\text{state}}, p)$
  $\tilde{\mathbf{Y}} = e^{\text{state}}_{out}(\mathbf{F}^{(L)}_{\text{state}})$
  Return $\tilde{\mathbf{I}}$
end procedure

C Experimental Details

C.1 Pendulum example problem

The pendulum's motion in Example 1 is modeled by a time-varying frequency pendulum, such that

$$q(F;t) = \sin(\omega(t)F),$$

where $\omega(t) = 1 - 0.04\sin(t)$.

C.2 Benchmarks

Node classification datasets. We report the statistics of the datasets used in our node classification experiments in Table 4. All datasets are publicly available, and appropriate references to the data sources are provided in the main paper. All the datasets considered in our experiments consider a transductive node classification problem. That is, at training time, we have labels available at a fraction of the graph's nodes, and our goal is to predict the labels of the remaining nodes.

Spatio-temporal forecasting datasets. We report the statistics of the datasets used in our spatio-temporal forecasting experiments in Table 5. All datasets are publicly available, and appropriate references to the data sources are provided in the main paper. In this task, our goal is to reduce the MSE between the ground-truth future value (which depends on the dataset) and the predicted value, given previous time snapshots.

Table 4: Node classification datasets statistics.

Dataset	Classes	Nodes	Edges	Features	Homophily
Squirrel	5	5,201	198,493	2,089	0.22
Film	5	7,600	33,544	932	0.22
Chameleon	5	2,277	36,101	2,325	0.23
Citeseer	6	3,327	4,732	3,703	0.80
Pubmed	3	19,717	44,338	500	0.74
Cora	7	2,708	5,429	1,433	0.81

Table 5: Attributes of the spatio-temporal datasets, and information about the number of time periods (T) and spatial units ($|\mathcal{V}|$).

Dataset	Frequency	T	$\lvert\mathcal{V}\rvert$
Chickenpox Hungary	Weekly	522	20
Pedal Me Deliveries	Weekly	36	15
Wikipedia Math	Daily	731	1,068

C.3 Hyperparameters

All hyperparameters were determined by grid search, and the ranges and sampling mechanism distributions are provided in Table 6. Also, unless otherwise specified, in all experiments, we use $L = 8$ layers, and for node classification datasets, we use $o = L$. For the spatio-temporal datasets, we use $o = r$, i.e., the order is set to be equal to the number of historical data given by the task.

Table 6: Hyperparameter ranges

Hyperparameter	Range	Uniform Distribution
input/output embedding learning rate	[1e-4, 1e-1]	log uniform
temporal term **c** learning rate	[1e-4, 1e-1]	log uniform
spatial term learning rate	[1e-4, 1e-1]	log uniform
input/output embedding weight decay	[0, 1e-2]	uniform
temporal term **c** weight decay	[0, 1e-2]	uniform
spatial term weight decay	[0, 1e-2]	uniform
input/output dropout	[0, 0.9]	uniform
hidden layer dropout	[0, 0.9]	uniform
use BatchNorm	{ yes / no }	discrete uniform
step size h	[1e-3, 1]	uniform
hidden channels k	{ 8,16,32,64,128,256 }	discrete uniform

C.4 Runtimes

In addition to the complexity discussion in the main paper, we provide the measured runtimes in Table 7. Learning the temporal order and dynamics requires additional computations compared to the vanilla baseline of DE-GNN, however, it also offers improved performance, as we show in our experiments in Section 5. We report the measured training and inferences runtimes, and the number of parameters on the Cora dataset in Table 7. We measure the runtimes using an Nvidia-RTX3090 with 24GB of memory, which is the same GPU used to conduct our experiments.

Table 7: Training and inference GPU runtimes (milliseconds), and the number of parameters (thousands).

Metric	DE-GNN ($o = 1, c_1 = 1$)	TDE-GNN$_{\text{D}}$ ($o = 8$)	TDE-GNN$_{\text{A}}$ ($o = 8$)
Training time	21.45	23.96	34.55
Inference time	11.84	12.83	16.97
Parameters	125	157	174

C.5 Coefficients analysis

We now conduct an analysis of the learned coefficients $\mathbf{c}$ for the pendulum problem in Example 1. For convenience, we present the learned coefficients again, in Table 8. Our analysis consists of two parts: (i) stability, and, (ii) consistency.

	c_1	c_2	c_3	c_4	c_5
$o = 2$	2	-1	–	–	–
$o = 3$	1.4	0.2	-0.6	–	–
$o = 4$	0.975	0.675	-0.25	-0.4	–
$o = 5$	-0.08	1.68	0.153	0.006	-0.759

Table 8: The learned coefficients $\mathbf{c}$ with a varying order $o \in \{2, 3, 4, 5\}$ when solving Example 1.

Stability analysis. Following the derivations in Theorem 1 proof presented in Appendix A, we examine the root conditions of the learned coefficients. In Table 9 we report the absolute values of the characteristic polynomial with the coefficients from Table 8. We note that for orders $2, 3$, and 4, stability is obtained. However, for $o = 5$ the coefficients have one unstable mode. This result suggests that one should not use $o = 5$ for the pendulum problem in Example 1. However, orders lower than 5 yield stability. We believe that a stable fifth-order model is possible to be learned from the data, however, the incorporation of the root condition to the learning process requires adding constraints to the learning process and therefore is beyond the scope of this work.

	$\lvert r_1\rvert$	$\lvert r_2\rvert$	$\lvert r_3\rvert$	$\lvert r_4\rvert$	$\lvert r_5\rvert$
$o = 2$	1	1	–	–	–
$o = 3$	1	0.6	1	–	–
$o = 4$	1	1	0.629	0.629	–
$o = 5$	1	0.73	0.73	1.4	1

Table 9: The absolute value of the roots $r_1, \ldots, r_5$ of the characteristic polynomial with coefficients from Table 8.

Consistency analysis. There are two conditions that verify the consistency of the learned coefficients. The first condition requires that the sum of the coefficients $\mathbf{c}$ equals to 1, which implies that if the spatial term in Equation (15), the future state (node features) is equal to a weighted average of the previous $o - 1$ states. This implies that a constant solution can always be achieved. The second condition requires that the application of the coefficients $\mathbf{c}$ to a known discrete function yields a consistent approximation to its derivatives of some order.

Note that condition (i) is satisfied by our construction of $\mathbf{c}$. The second condition can be verified numerically, as we show now. To this end, we discretize the function $y = \sin(2\pi t)$ in the interval $[0, 1]$. We then apply the stencils based on the learned coefficients $\mathbf{c}$ as shown in Table 8. As can be depicted in Figure 6, the application of the stencils to the discrete function $y(t)$ yields a scaled version of the second derivative of $y(t)$ (which also equals to y, that is, $\frac{\partial^2 y(t)}{\partial t^2} = \beta y(t)$). This result shows that our TDE-GNN is able to reveal the true order that describes the pendulum's motion, which is a second-order process.

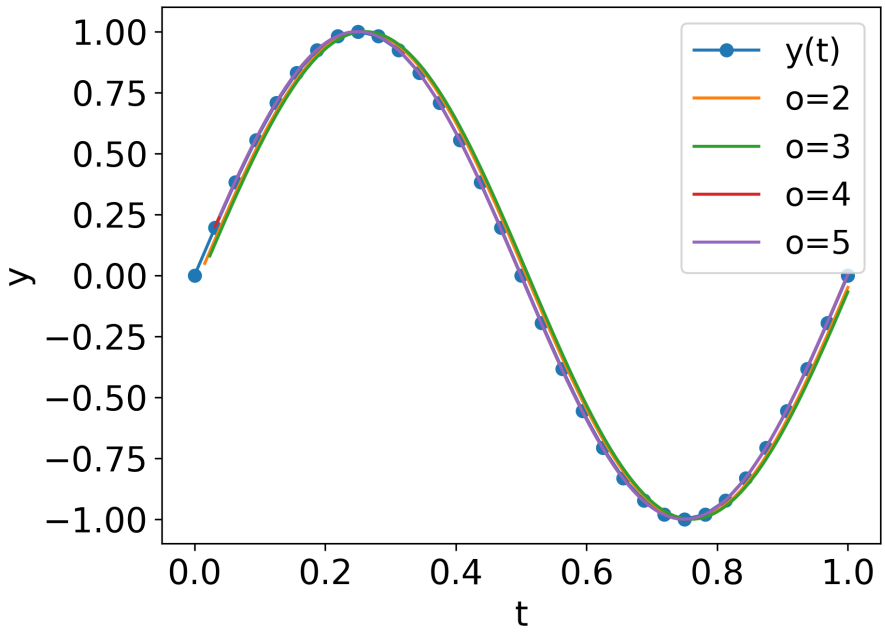


Figure 6: Examining the consistency of the learned coefficients. The learned coefficients model a second-derivative of the test function $y(t) = \sin(2\pi t)$.