Feature Transportation Improves Graph Neural Networks

Moshe Eliasof[1,2*], **Eldad Haber**[3], **Eran Treister**[2]

[1] Department of Applied Mathematics and Theoretical Physics, University of Cambridge, United Kingdom
[2] Department of Computer Science, Ben-Gurion University of the Negev, Israel
[3] Department of Earth, Ocean, and Atmospheric Sciences, University of British Columbia, Canada
me532@cam.ac.uk, ehaber@eoas.ubc.ca, erant@cs.bgu.ac.il

Abstract

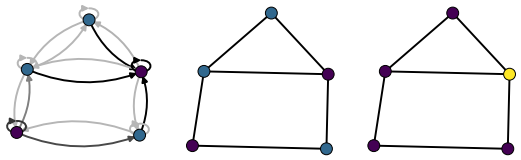
Graph neural networks (GNNs) have shown remarkable success in learning representations for graph-structured data. However, GNNs still face challenges in modeling complex phenomena that involve feature transportation. In this paper, we propose a novel GNN architecture inspired by Advection-Diffusion-Reaction systems, called ADR-GNN. Advection models feature transportation, while diffusion captures the local smoothing of features, and reaction represents the non-linear transformation between feature channels. We provide an analysis of the qualitative behavior of ADR-GNN, that shows the benefit of combining advection, diffusion, and reaction. To demonstrate its efficacy, we evaluate ADR-GNN on real-world node classification and spatio-temporal datasets, and show that it improves or offers competitive performance compared to state-of-the-art networks.

1 Introduction

Recently, GNNs have been linked to ordinary and partial differential equations (ODEs and PDEs) in a series of works (Chamberlain et al. 2021; Eliasof, Haber, and Treister 2021; Rusch et al. 2022; Wang et al. 2022; Giovanni et al. 2023; Gravina, Bacciu, and Gallicchio 2023). These works propose to view GNN layers as the time discretization of ODEs and PDEs, and as such they offer both theoretical and practical advantages. First, ODE and PDE based models allow to reason about the behavior of existing GNNs. For instance, as suggested in (Chamberlain et al. 2021), it is possible to view GCN (Kipf and Welling 2017) and GAT (Veličković et al. 2018) as discretizations of the non-linear heat equation. This observation helps to analyze and understand the oversmoothing phenomenon in GNNs (Oono and Suzuki 2020; Cai and Wang 2020). Second, ODE and PDE based GNNs pave the path to the construction and design of GNNs that satisfy desired properties, such as energy-preservation (Eliasof, Haber, and Treister 2021; Rusch et al. 2022), attraction and repulsion forces modeling (Wang et al. 2022; Giovanni et al. 2023), anti-symmetry (Gravina, Bacciu, and Gallicchio 2023), as well as reaction-diffusion systems (Choi et al. 2022). Nonetheless, the aforementioned architectures still rely on controlled diffusion or wave propagation, as well as non-linear pointwise convolutions. Therefore, as discussed in (Rusch, Bronstein, and Mishra 2023), while there are methods that can alleviate oversmoothing, they may lack expressiveness. We now provide a simple example, known as the graph node feature transportation task (LeVeque 1990), where diffusion, wave propagation, and reaction networks may fail. In this task, the goal is to gather the node information (i.e., features) from several nodes to a single node. Clearly, no diffusion process can express or model such a phenomenon, because diffusion spreads and smooths, rather than transports information (Evans 1998; Ascher 2008). Likewise, a wave-propagation approach cannot express such a phenomenon, because it lacks directionality (Ascher 2008), which is required for this task. An instance of this problem is illustrated in Figure 1, where we show the source and target node features, and the learned advection weights that can achieve the desired target. Later, in Figure 2, we show that popular operators such as diffusion or reaction cannot model the transition from the source to the target node features, while advection can. Furthermore, the concept of advection appears in many real-world problems and data, such as traffic-flow and control (Betts 2001), quantity transportation in computational biology (Uys 2009), and rainfall forecasting (Seed 2003). Motivated by the previously discussed observations and examples, we propose, in addition to learning and combining diffusion and reaction terms, to develop a learnable, neural *advection* term, also known as a *transportation* term (Evans 1998; Ascher 2008; LeVeque 1990), that is suited to model feature transportation from the data in a task-driven fashion. The resulting architecture, called *ADR-GNN*, can therefore express various phenomena, from advection, diffusion, to pointwise reactions, as well as their compositions.

Contributions. The contributions of this paper are threefold. (1) We develop a novel graph neural advection operator that is mass-preserving, stable, and consistent with continuous advection PDEs. This operator enables the modeling of phenomena that involve feature transportation on graphs by learning the direction of the transportation. (2) We propose ADR-GNN, a GNN based on learnable advection-diffusion-reaction (ADR) systems, that can express a wide range of phenomena, including learned directional information flow, diffusion, and pointwise reactions. (3) We demonstrate the

[*] Work done while being a Ph.D. student at Ben-Gurion University of the Negev.



(a) Advection (b) Source (c) Target

Figure 1: An example of node feature transportation on a graph. Applying the advection weights in (a) to the source (b), yields the target (c). Darker edge colors in (a) indicate greater advection weights.

efficacy of ADR-GNN on node classification, and spatio-temporal forecasting datasets, achieving improved or competitive results compared to state-of-the-art models.

2 Related Work

Advection-Diffusion-Reaction. An Advection Diffusion Reaction system is a mathematical model that describes the simultaneous existence of three processes: (1) the advection (transport) of information in a medium, (2) the diffusion (smoothing) of information within that medium, and (3) pointwise (self) reactions. These systems are used to study and model a wide range of physical, chemical, and biological phenomena. For example, ADR systems can be utilized to track and estimate the location of fish swarms (Adam and Sibert 2004), model ecological trends (Cosner 2014), and the modeling of turbulent flames in supernovae (Khokhlov 1995). However, the aforementioned works rely on a low-dimensional, hand-crafted, non-neural ADR system to be determined, typically by trial and error, often requiring a domain expert. In contrast, in this paper, we propose to learn the ADR system for various graph types and tasks.

Graph Neural Networks as Dynamical Systems. Adopting the interpretation of convolutional neural networks (CNNs) as discretizations of ODEs and PDEs (Ruthotto and Haber 2020; Chen et al. 2018b) to GNNs, works like GRAND (Chamberlain et al. 2021), PDE-GCN$_{\text{D}}$ (Eliasof, Haber, and Treister 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, allowing to control the diffusion (smoothing) in the network, to understand oversmoothing (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$_{\text{M}}$ (Eliasof, Haber, and Treister 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, Bronstein, and Mishra 2023), besides alleviating oversmoothing, it is also important to design GNN architectures with improved expressiveness. Recent examples of such networks are (Gravina, Bacciu, and Gallicchio 2023) that propose an anti-symmetric GNN to alleviate over-squashing (Alon and Yahav 2021), and (Wang et al. 2022; Choi et al. 2022) that formulate a reaction-diffusion GNN to enable non-trivial pattern growth. In this paper, we build on the properties of ADR PDEs, that in addition to modeling diffusive and reactive processes, also allow to capture advective processes such as the transportation of node features.

On another note, CNNs and GNNs are also used to accelerate PDE solvers (Raissi 2018; Long et al. 2018), as well as to generate (Sanchez-Gonzalez et al. 2019) physical simulations. In this paper, we focus on the view of GNNs as the discretization of ADR PDEs, rather than using GNNs to solve PDEs.

Advection on Graphs. Advection is a term used in Physics to describe the transport of a substance in a medium. In the context of graphs, advection is used to express the transport of information (features) on the graph nodes. The underlying process of advection is described by a continuous PDE, and several graph discretization techniques (Chapman and Chapman 2015) are available. The advection operator has shown its effectiveness in classical (i.e., non-neural) graph methods, from blood-vessel simulations (Deepa Maheshvare, Raha, and Pal 2022), to traffic flow prediction (Borovitskiy et al. 2021). In this paper, we develop a neural advection operator that is combined with neural diffusion and reaction operators, called ADR-GNN.

3 Method

In this section, we first describe the general outline of a continuous ADR system in Section 3.1, and present its graph discrete analog, named *ADR-GNN* in Section 3.2. We discuss ADR-GNN components in detail in Sections 3.3-3.4.

Notations. We define a graph by $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is a set of n nodes and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is a set of m edges. We denote the 1-hop neighborhood of the i-th node by $\mathcal{N}_i$, and the node features by $\mathbf{U} \in \mathbb{R}^{n \times c}$, where c is the number of features. The symmetric graph Laplacian reads $\mathbf{L} = \mathbf{D} - \mathbf{A}$, and the symmetric normalized Laplacian is given by $\hat{\mathbf{L}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{L}\mathbf{D}^{-\frac{1}{2}}$, where $\mathbf{D}$ is the degree matrix.

3.1 Continuous Advection-Diffusion-Reaction Systems

The continuous PDE that describes an ADR system is given by:

$$\frac{\partial U}{\partial t} = \underbrace{\nabla \cdot (VU)}_{\text{Advection}} + \underbrace{K\Delta U}_{\text{Diffusion}} + \underbrace{f(U, X, \theta_r)}_{\text{Reaction}}, \tag{1}$$

where $X \in \Omega$, $t \in [0, T]$, accompanied by initial conditions $U(X, t = 0)$ and boundary conditions. Here, $U(X, t) = [u_1(X, t), \ldots, u_c(X, t)] : \mathbb{R}^{\Omega \times [0,T]} \to \mathbb{R}^c$ is a density function, written as a vector of scalar functions $u_s(X, t)$, $s = 1, \ldots, c$, that depend on the initial location X and time t. The spatial domain Ω can be $\mathbb{R}^d$ or a manifold $\mathcal{M} \subseteq \mathbb{R}^d$. From a neural network perspective, u_s is referred to as a channel, interacting with other channels. The left-hand side of Equation (1) is a time derivative that represents the change in features in time, as discussed in Section 2. The right-hand side includes three terms:

- **Advection.** Here, V denotes a velocity function that transports the density U in space, and $\nabla\cdot$ is the divergence operator.
- **Diffusion.** We denote the continuous Laplacian operator by Δ. The Laplacian is scaled with a diagonal matrix $K = \mathrm{diag}(\kappa_1, \ldots, \kappa_c) \in \mathbb{R}^{c\times c}$, $\kappa_i \geq 0$ of non-negative diffusion coefficients, each independently applied to its corresponding channel in U.
- **Reaction.** Here, $f(U, X, \theta_r)$ is a non-linear pointwise function parameterized by θ_r.

3.2 Advection-Diffusion-Reaction on Graphs

Equation (1) is defined in the continuum. We now use a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ to discretize Ω. The nodes $\mathcal{V}$ can be regarded as a discretization of X, that is, the i-th node is located in $\mathbf{X}_i$, and the edges $\mathcal{E}$ represent the topology of Ω. Then, the *spatial*, graph discretization of Equation (1) is:

$$\frac{d\mathbf{U}(t)}{dt} = \mathbf{DIV}\left(\mathbf{V}\left(\mathbf{U}(t), t; \boldsymbol{\theta}_a(t)\right)\mathbf{U}(t)\right) \tag{2a}$$
$$- \hat{\mathbf{L}}\mathbf{U}(t)\mathbf{K}(t; \boldsymbol{\theta}_d(t)) + f(\mathbf{U}(t), \mathbf{X}, t; \boldsymbol{\theta}_r(t)),$$
$$\mathbf{U}(0) = \mathbf{U}^{(0)} = g_{\text{in}}(\mathbf{X}, \boldsymbol{\theta}_0). \tag{2b}$$

Here, $\mathbf{U}(t) \in \mathbb{R}^{n\times c}$ is a matrix that describes the node features at time t. The advection term depends on the velocity function $\mathbf{V}$ parameterized by learnable weights $\boldsymbol{\theta}_a(t)$. The precise discretization of the advection operator $\mathbf{DIV}(\mathbf{V}\cdot)$ is discussed in Section 3.4. The diffusion is discretized using the symmetric normalized Laplacian[1] $\hat{\mathbf{L}}$ that is scaled with a diagonal matrix with non-negative learnable diffusion coefficients on its diagonal $\mathbf{K}(t; \boldsymbol{\theta}_d(t)) = \mathrm{diag}(\mathrm{hardtanh}(\boldsymbol{\theta}_d(t), 0, 1)) \geq 0$, where $\mathrm{hardtanh}(\boldsymbol{\theta}_d(t), 0, 1)$ clamps each element in $\boldsymbol{\theta}_d \in \mathbb{R}^c$ to be between 0 and 1. The reaction term f from Equation (2a) is a pointwise non-linear function realized by a multilayer-perceptron (MLP) parameterized by learnable weights $\boldsymbol{\theta}_r(t)$. To obtain initial node embedding $\mathbf{U}^{(0)} \in \mathbb{R}^{n\times c}$ from the input features $\mathbf{X} \in \mathbb{R}^{n\times c_{in}}$, we use a fully-connected layer g_{in} in Equation (2b).

In this work we focus on static and temporal node-level tasks, and we note that typically, c, the number of hidden channels of $\mathbf{U}(T) \in \mathbb{R}^{n\times c}$, is different than c_{out}, the number of channels of the target output $\mathbf{Y} \in \mathbb{R}^{n\times c_{out}}$. Therefore the output of neural network, $\tilde{\mathbf{Y}}$, is given by

$$\tilde{\mathbf{Y}} = g_{\text{out}}(\mathbf{U}(T), \boldsymbol{\theta}_{\text{out}}) \in \mathbb{R}^{n\times c_{out}}, \tag{3}$$

where g_{out} is a fully-connected layer with weights $\boldsymbol{\theta}_{\text{out}}$.

The qualitative behavior of ADR-GNN. The ADR-GNN model combines the learning of three powerful terms. Namely, the learned parameters are $\boldsymbol{\theta}_a$ the advection parameters, $\boldsymbol{\theta}_d$ the diffusion parameters, and $\boldsymbol{\theta}_r$ the reaction parameters. Therefore, an ADR-GNN layer can express and model various phenomena. For example, if we set $\boldsymbol{\theta}_d(t) = 0$, then there is no diffusion in the system, and the method is dominated by advection and reaction. If on the other hand, one learns a very small advection (i.e., the learned $\mathbf{V}$, to be discussed later, tends to retain all features in place), then a reaction-diffusion oriented system is obtained. Similarly, other combinations of advection, diffusion, and reaction can be achieved, because of the learning of the parameters of the system. Thus, ADR-GNN can be adopted to solve a host of problems, depending on dynamics and patterns mandated by the data, as we show later in our experiments in Section 4.

[1] In PDE theory, the Laplacian is a negative operator, while in graph theory it is positive. Therefore it is required to multiply $\hat{\mathbf{L}}$ by a negative sign in Equation (2a) compared to Equation (1).

3.3 From an ODE to a Graph Neural Network - Time Discretization of ADR-GNN

Equation (2) *spatially* discretizes the PDE in Equation (1), yielding an ODE defined on the graph. The *time* discretization of the ODE yields a sequential process that can be thought of as layers of neural networks (Haber and Ruthotto 2017; Chen et al. 2018b). That is, upon discrete time integration of Equation (2a), we replace the notion of time t with l layers, and a step size h, that is a positive scalar hyperparameter.

While it is possible to use many ODE discretization methods (see, e.g., (Haber and Ruthotto 2017; Chen et al. 2018b; Chamberlain et al. 2021) and references within), in various applications where an ADR system arises, from flow in porous media (Coats 2000), to PDE-based image segmentation (Vese and Chan 2002), and multiphase flow (Kadioglu et al. 2011), an operator-splitting (OS) (Ascher 2008) is utilized. We therefore also use an OS time discretization for Equation (2a), that yields a graph neural ADR layer, summarized in Algorithm 1. Composing several neural ADR layers leads to ADR-GNN. We further discuss the properties of the OS approach in the Appendix. The exact discretizations of the ADR terms are derived in Section 3.4.

Algorithm 1: Graph Neural Advection-Diffusion-Reaction Layer

Input: Node features $\mathbf{U}^{(l)} \in \mathbb{R}^{n\times c}$
Output: Updated node features $\mathbf{U}^{(l+1)} \in \mathbb{R}^{n\times c}$
1: Advection:
$\mathbf{U}^{(l+1/3)} = \mathbf{U}^{(l)} + h\mathbf{DIV}(\mathbf{V}(\mathbf{U}^{(l)}, t; \boldsymbol{\theta}_a^{(l)})\mathbf{U}^{(l)})$.
2: Diffusion:
$\mathbf{U}^{(l+2/3)} = \mathrm{mat}\left((\mathbf{I} + h\mathbf{K}(t; \boldsymbol{\theta}_d^{(l)}) \otimes \hat{\mathbf{L}})^{-1}\mathrm{vec}(\mathbf{U}^{(l+1/3)})\right)$.
3: Reaction:
$\mathbf{U}^{(l+1)} = \mathbf{U}^{(l+2/3)} + hf(\mathbf{U}^{(l+2/3)}, \mathbf{U}^{(0)}, t; \boldsymbol{\theta}_r^{(l)})$.

3.4 Discretized Graph Operators

We now elaborate on the discretized graph operators utilized in our ADR-GNN, summarized in Algorithm 1. Besides the combination of the learnable advection, diffusion, and reaction terms, which, to the best of our knowledge, was not studied in the context of GNNs, the main innovation here is the *consistent, mass preserving, and stable* discretization of the advection operator.

Advection. To define the graph discretized advection operator, we extend the non-learnable advection operator from (Chapman and Chapman 2015), into a learnable, neural advection operator. Our advection operator transports node

features based on learned directed edge weights (velocities) $\{(\mathbf{V}_{i\to j}, \mathbf{V}_{j\to i})\}_{(i,j)\in\mathcal{E}}$, where each $\mathbf{V}_{i\to j}, \mathbf{V}_{j\to i} \in \mathbb{R}^c$, such that $0 \le \mathbf{V}_{i\to j} \le 1$. The notation $i \to j$ implies that the weight transfers features from the i-th to j-th node. We further demand that the outbound edge weights associated with every node, per channel, sum to 1, i.e., $\sum_{j\in\mathcal{N}_i} \mathbf{V}_{i\to j} = 1$. This constraint suggests that a node can at most transfer the total of its features to other nodes. First, we define the discretized divergence from Equation (2a), that operates on the learned edge weights $\mathbf{V}$:

$$\begin{aligned}\mathbf{DIV}_i(\mathbf{VU}) &= \textstyle\sum_{j\in\mathcal{N}_i} \mathbf{V}_{j\to i} \odot \mathbf{U}_j - \mathbf{U}_i \odot \sum_{j\in\mathcal{N}_i} \mathbf{V}_{i\to j} \\ &= \textstyle\sum_{j\in\mathcal{N}_i} \mathbf{V}_{j\to i} \odot \mathbf{U}_j - \mathbf{U}_i,\end{aligned} \tag{4}$$

where $\odot$ is the elementwise Hadamard product. Then, the graph advection operator in Algorithm 1 is:

$$\begin{aligned}\mathbf{U}_i^{(l+1/3)} &= \mathbf{U}_i^{(l)} + h\mathbf{DIV}_i(\mathbf{V}^{(l)}\mathbf{U}^{(l)}) \\ &= \mathbf{U}_i^{(l)} + h\left(\textstyle\sum_{j\in\mathcal{N}_i} \mathbf{V}_{j\to i}^{(l)} \odot \mathbf{U}_j^{(l)} - \mathbf{U}_i^{(l)}\right).\end{aligned} \tag{5}$$

Namely, the updated node features are obtained by adding the $\mathbf{V}_{j\to i}$ weighted inbound node features, while removing the $\mathbf{V}_{i\to j}$ weighted outbound node features, and h is a positive step size. The scheme in Equation (5) is the forward Euler discretization. We now show that the proposed graph neural advection operator is *mass conserving*, *stable* and *consistent* [2], meaning that our advection operator is adherent to the continuous advection PDE (LeVeque 1990).

Lemma 1 *Define the mass of the graph node features $\mathbf{U}^{(l)} \in \mathbb{R}^{n\times c}$ as the scalar $\rho^{(l)} = \sum \mathbf{U}^{(l)}$. Then the advection operator in Equation* (5) *is mass conserving, i.e., $\rho^{(l+1/3)} = \rho^{(l)}$.*

Lemma 2 *The advection operator in Equation* (5) *is stable.*

To learn a *consistent* advection operator, i.e., an operator that mimics the directional behavior of the advection in Equation (1), we craft an edge weight $\mathbf{V}$ mechanism, shown in Algorithm 2, that yields direction-oriented weights, i.e., we ensure that $\mathbf{V}_{i\to j} \neq \mathbf{V}_{j\to i}$, unless they are zeroes.

Here, $\boldsymbol{\theta}_a^{(l)} = \{\mathbf{A}_1^{(l)}, \mathbf{A}_2^{(l)}, \mathbf{A}_3^{(l)}, \mathbf{A}_4^{(l)}\}$ are learnable fully connected layers, and the $\exp$ is computed channel-wise. We note that the sign of $\mathbf{Z}_{ij} - \mathbf{Z}_{ji}$ is opposite than that of $-\mathbf{Z}_{ij} + \mathbf{Z}_{ji}$ in Algorithm 2. Hence, after the $\mathrm{ReLU}(\cdot)$ activation, one of the edge weights, either $\mathbf{V}_{i\to j}$ or $\mathbf{V}_{j\to i}$ is guaranteed to be equal to zero, and the other will be non-negative. This allows the architecture to create significant asymmetry in the edge weights $\mathbf{V}$, as also seen in Figure 1.

Diffusion. To discretize the diffusion term from Equation (2a), both explicit and implicit time discretizations can be used (Ascher 2008). An explicit forward Euler discretization yields the following layer:

$$\mathbf{U}^{(l+2/3)} = \mathbf{U}^{(l+1/3)} - h\left(\hat{\mathbf{L}}\mathbf{U}^{(l+1/3)}\mathbf{K}^{(l)}\right). \tag{6}$$

However, an explicit scheme requires using a small step size $h > 0$, as it is marginally stable (Ascher 2008). We therefore harness an implicit scheme, which guarantees the stability of the diffusion [3], and reads:

$$\mathbf{U}^{(l+2/3)} = \mathrm{mat}\left((\mathbf{I} + h\mathbf{K}^{(l)} \otimes \hat{\mathbf{L}})^{-1}\mathrm{vec}(\mathbf{U}^{(l+1/3)})\right). \tag{7}$$

Here, $\otimes$ is the Kronecker product, $\mathrm{vec}()$ is a flattening operator, and $\mathrm{mat}()$ reshapes a vector to a matrix. The computation of $\mathbf{U}^{(l+2/3)}$ requires the solution of a linear system, solved by conjugate gradients[4] (Ascher 2008). In our experiments we found 5 iterations to be sufficient.

Reaction. Our reaction term is realized using MLPs. Recent works showed that utilizing both additive and multiplicative MLPs yields improved performance (Choi et al. 2022). Hence, we define

$$\begin{aligned}&f(\mathbf{U}^{(l+2/3)}, \mathbf{U}^{(0)}; \boldsymbol{\theta}_r^{(l)}) = \sigma(\mathbf{U}^{(l+2/3)}\mathbf{R}_1^{(l)} \\ &+ \tanh(\mathbf{U}^{(l+2/3)}\mathbf{R}_2^{(l)}) \odot \mathbf{U}^{(l+2/3)} + \mathbf{U}^{(0)}\mathbf{R}_3^{(l)}),\end{aligned} \tag{8}$$

as our reaction term in Equation (2a). Here, $\boldsymbol{\theta}_r^{(l)} = \{\mathbf{R}_1^{(l)}, \mathbf{R}_2^{(l)}, \mathbf{R}_3^{(l)}\}$ are trainable fully-connected layers, and σ is non-linear activation function (ReLU in our experiments), that can also be coupled with batch-normalization. This term is integrated via forward Euler as in Algorithm 1.

Algorithm 2: Learning directional edge weights.

Input: Node features $\mathbf{U}^{(l)} \in \mathbb{R}^{n\times c}$

Output: Edge weights $\mathbf{V}_{i\to j}^{(l)}, \mathbf{V}_{j\to i}^{(l)} \in \mathbb{R}^c$

1: Compute edge features:
$\mathbf{Z}_{ij}^{(l)} = \mathrm{ReLU}(\mathbf{U}_i^{(l)}\mathbf{A}_1^{(l)} + \mathbf{U}_j^{(l)}\mathbf{A}_2^{(l)})\mathbf{A}_3^{(l)}$.
$\mathbf{Z}_{ji}^{(l)} = \mathrm{ReLU}(\mathbf{U}_j^{(l)}\mathbf{A}_1^{(l)} + \mathbf{U}_i^{(l)}\mathbf{A}_2^{(l)})\mathbf{A}_3^{(l)}$.
2: Compute relative edge features:
$\mathbf{V}_{i\to j}^{(l)} = \mathrm{ReLU}(\mathbf{Z}_{ij}^{(l)} - \mathbf{Z}_{ji}^{(l)})\mathbf{A}_4^{(l)}$.
$\mathbf{V}_{j\to i}^{(l)} = \mathrm{ReLU}(-\mathbf{Z}_{ij}^{(l)} + \mathbf{Z}_{ji}^{(l)})\mathbf{A}_4^{(l)}$.
3: Normalize to obtain edge weights:
$\mathbf{V}_{i\to j}^{(l)} \leftarrow \frac{\exp(\mathbf{V}_{i\to j}^{(l)})}{\sum_{k\in\mathcal{N}_i}\exp(\mathbf{V}_{i\to k}^{(l)})}$.
$\mathbf{V}_{j\to i}^{(l)} \leftarrow \frac{\exp(\mathbf{V}_{j\to i}^{(l)})}{\sum_{k\in\mathcal{N}_j}\exp(\mathbf{V}_{j\to k}^{(l)})}$.

4 Experimental Results

We demonstrate our ADR-GNN on two types of tasks on real-world datasets: node classification, and spatio-temporal node forecasting. Architectures and training details are provided in the Appendix, and the runtimes and complexity of ADR-GNN are discussed in the Appendix. We use a grid search to select hyperparameters, discussed in the Appendix. Datasets details and statistics are reported in the Appendix.

[2] See stability definition and proofs in the Appendix.

[3] See (Haber et al. 2019; Chamberlain et al. 2021) for details on implicit vs. explicit schemes for diffusion processes and in neural networks.

[4] We note that the matrix $\mathbf{I} + h\mathbf{K}_l \otimes \hat{\mathbf{L}}$ is positive definite and invertible, because the identity matrix is positive definite, h is positive, $\mathbf{K}_l$ is non-negative, and the graph Laplacian $\hat{\mathbf{L}}$ is positive semi-definite.

Overall, we propose the two following ADR-GNN architectures:

- ADR-GNN$_S$. Here we follow a similar approach to typical neural networks, where different weights are learned for each layer. From a dynamical system perspective, this can be interpreted as an unrolled ADR iteration (Mardani et al. 2018). This architecture is suitable for 'static' datasets that do not involve temporal information, such as Cora, and is specified in the Appendix.
- ADR-GNN$_T$. A time-dependent ADR-GNN for temporal datasets. Compared to ADR-GNN$_S$, it also utilizes temporal embedding, discussed in the Appendix.

4.1 Node Classification

Homophilic graphs. We experiment with Cora, Citeseer, and Pubmed datasets. We use the 10 splits from (Pei et al. 2020) with train/validation/test split ratios of $48\%/32\%/20\%$, and report their average accuracy in Table 1. In the Appendix, we also provide the accuracy standard deviation. As a comparison, we consider multiple recent methods, such as GCN (Kipf and Welling 2017), GAT (Veličković et al. 2018), Geom-GCN (Pei et al. 2020), GCNII (Ming Chen, Zengfeng Huang, and Li 2020), PDE-GCN (Eliasof, Haber, and Treister 2021), H2GCN (Zhu et al. 2020b), GGCN (Yan et al. 2022), DMP (Yang et al. 2021), GREAD (Choi et al. 2022), LINKX (Lim et al. 2021a), ACMII (Luan et al. 2022), Ord. GNN (Song et al. 2023), and FLODE (Maskey et al. 2023). We see that our ADR-GNN$_S$ outperforms all methods on the Cora and Pubmed datasets, and achieves close (0.12% accuracy difference) to the best performing PDE-GCN on Citeseer.

Heterophilic graphs. While our ADR-GNN offers competitive accuracy on homophilic datasets, as discussed in Section 2, ADR systems are widely used to model non-smooth phenomena and patterns, as often appear in heterophilic datasets by their definition (Pei et al. 2020). We therefore utilize 10 heterophilic datasets from various sources. In Table 2 we compare the average accuracy of our ADR-GNN$_S$ with recent GNNs on the Squirrel, Film, and Chameleon from (Rozemberczki, Allen, and Sarkar 2021), as well as the Cornell, Texas and Wisconsin datasets from (Pei et al. 2020), using the 10-splits from (Pei et al. 2020) we train/validation/test split ratios of $48\%/32\%/20\%$. We include more comparisons and the accuracy standard deviation in the Appendix. In addition to the previously considered methods, we also compare with GRAFF (Giovanni et al. 2023), and ACMP-GCN (Wang et al. 2022). We see that ADR-GNN$_S$ offers accuracy that is in line with recent state-of-the-art methods. In addition, we evaluate ADR-GNN$_S$ on the Twitch-DE, deezer-europe, Penn94, and arXiv-year datasets from (Lim et al. 2021b,a) to further demonstrate the efficacy of our method, in the Appendix.

4.2 Spatio-Temporal Node Forecasting

Classical ADR models are widely utilized to predict and model spatio-temporal phenomena (Fiedler and Scheel 2003; Adam and Sibert 2004). We therefore now evaluate our temporal ADR-GNN$_T$ on several spatio-temporal node forecasting datasets. To this end, we harness the software package PyTorch-Geometric-Temporal (Rozemberczki et al. 2021) that offers a graph machine learning pipeline for spatio-temporal graph tasks. In our experiments, we use the Chickenpox Hungary, PedalMe London, and Wikipedia Math datasets from (Rozemberczki et al. 2021), as well as the traffic speed prediction datasets METR-LA (Jagadish et al. 2014) and PEMS-BAY (Chen et al. 2001).

For the first three datasets, we follow the incremental training mode, mean-squared-error (MSE) loss, and testing procedure from (Rozemberczki et al. 2021). We report the performance of ADR-GNN$_T$ and other models, in terms of MSE, in Table 3. We compare with several recent methods, namely, DCRNN (Li et al. 2018), GConv (Seo et al. 2018), GC-LSTM(Chen et al. 2018a), A3T-GCN (Zhu et al. 2020a), AGCRN (Bai et al. 2020), and DIFFormer (Wu et al. 2023). Our results in Table 3 show improved performance, further revealing the significance of neural ADR systems on graphs.

On the METR-LA and PEMS-BAY datsets, we follow the same training and testing procedures, and mean-absolute-error (MAE) loss as in (Li et al. 2018). We report the MAE, root mean squared error (RMSE), and mean absolute percentage error (MAPE). To demonstrate the effectiveness of ADR-GNN$_T$ for varying time frame predictions, we report the results on 3, 6, and 12 future frame traffic speed prediction, where each time frame equates to 5 minutes. We compare ADR-GNN$_T$ with various methods like DCRNN (Li et al. 2018), Graph WaveNet (Wu et al. 2019), AST-GCN (Guo et al. 2019), MTGNN (Wu et al. 2020), GTS (Shang, Chen, and Bi 2021), STEP (Shao et al. 2022), and STAEformer (Liu et al. 2023). We find that our ADR-GNN$_T$ offers lower (better) metrics than the considered methods. For instance, on METR-LA, ADR-GNN$_T$ reduces the MAE achieved by the recent STEP method from 3.37 to 3.19. We provide the results on METR-LA in Table 4, with additional comparisons as well as results on PEMS-BAY in the Appendix.

4.3 Ablation Studies

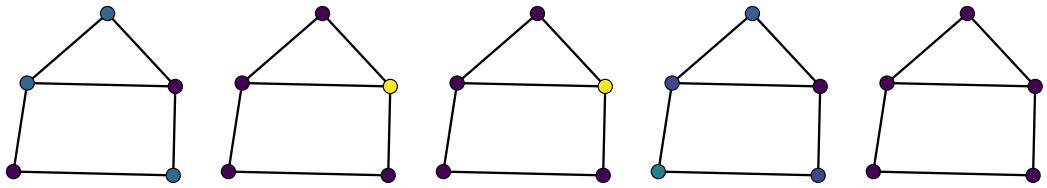
Synthetic Feature Transportation. The benefits of diffusion and reaction are known in GNNs (see (Chamberlain et al. 2021; Choi et al. 2022) and references within). However, the significance of neural advection was not studied in GNNs prior to our work, to the best of our knowledge. Therefore, and following the discussion of the task of feature transportation in Section 1 and Figure 1, we now compare the behavior of the advection, diffusion, and reaction terms on this task. Although this experiment is conceptually simple, it is evident from Figure 2, that diffusion and reaction terms in GNNs are limited in modeling such a behavior. This result, however, is not surprising. Employing diffusion smooths, rather than directly *transferring* node features. Similarly, the reaction term can only learn to scale the node features in this experiment. On the contrary, Figure 2 that the advection term, that by definition, transports information, achieves an exact fit. More experimental details are given in the Appendix.

Method Homophily	Cora 0.81	Citeseer 0.80	Pubmed 0.74
GCN	85.77	73.68	88.13
GAT	86.37	74.32	87.62
GCNII†	88.49	77.13	90.30
Geom-GCN†	85.27	77.99	90.05
PDE-GCN†	88.60	**78.48**	89.93
GRAND	87.36	76.46	89.02
GRAND++	88.15	76.57	88.50
GGCN	87.95	77.14	89.15
H2GCN	87.87	77.11	89.49
GRAFF†	88.01	77.30	90.04
DMP†	86.52	76.87	89.27
GREAD†	88.57	77.60	90.23
LINKX	84.64	73.19	87.86
ACMII†	88.25	77.12	89.71
Ord. GNN	77.31	90.15	88.37
FLODE	78.07	89.02	86.44
ADR-GNN$_S$	**89.43**	78.36	**90.55**

Table 1: Node accuracy (%) on homophilic datasets.

Method Homophily	Squi. 0.22	Film 0.22	Cham. 0.23	Corn. 0.30	Texas 0.11	Wisc. 0.21
GCN	23.96	26.86	28.18	52.70	52.16	48.92
GAT	30.03	28.45	42.93	54.32	58.38	49.41
GCNII†	38.47	32.87	60.61	74.86	69.46	74.12
Geom-GCN†	38.32	31.63	60.90	60.81	67.57	64.12
PDE-GCN†	–	–	66.01	89.73	93.24	91.76
GRAND	40.05	35.62	54.67	82.16	75.68	79.41
GRAND++	40.06	33.63	56.20	81.89	77.57	82.75
GGCN	55.17	37.81	71.14	85.68	84.86	86.86
H2GCN	36.48	35.70	60.11	82.70	84.86	87.65
GRAFF†	59.01	37.11	71.38	84.05	88.38	88.83
DMP†	47.26	35.72	62.28	89.19	89.19	92.16
GREAD†	59.22	37.90	71.38	87.03	89.73	89.41
ACMP-GCN	–	–	–	85.40	86.20	86.10
LINKX	61.81	36.10	68.42	77.84	74.60	75.49
ACMII†	67.40	37.09	74.76	86.49	88.38	88.43
Ord. GNN	62.44	37.99	72.28	–	–	–
FLODE	64.23	37.16	73.60	–	–	–
ADR-GNN$_S$	**72.54**	**39.16**	**79.91**	**91.89**	**93.61**	**93.46**

Table 2: Node accuracy (%) on heterophilic datasets.



(a) Source (b) Target (c) Advection (d) Diffusion (e) Reaction

Figure 2: Source and target node features, and their fit using advection, diffusion, and reaction.

Method	Chickenpox	PedalMe	Wikipedia
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
A3T-GCN	1.114±0.008	1.469±0.027	0.781±0.011
AGCRN	1.120±0.010	1.469±0.030	0.788±0.011
DIFFormer	0.920±0.001	–	0.720±0.036
ADR-GNN$_T$	**0.817±0.012**	**0.598±0.050**	**0.571±0.014**

Table 3: The performance of spatio-temporal networks evaluated by the average MSE of 10 experimental repetitions and standard deviations, calculated on 10% forecasting horizons.

The Impact of Advection, Diffusion, and Reaction. We study the influence of each of the proposed terms in Equation (2a) on real-world datasets, independently and jointly. The results, reported in Table 5 further show the significance of the advection term. For datasets that are homophilic like Cora, we see minor accuracy improvement when incorporating the advection term. This is in line with known findings regarding the benefits of diffusion for homophilic datasets (Chamberlain et al. 2021). More importantly, we see that mostly for heterophilic datasets like Chameleon, as well as traffic prediction datasets like PEMS-BAY, utilizing the advection significantly improves the performance of the network. Overall, we see that allowing all terms to be learned leads to favorable performance, indicating an implicit balancing of the terms obtained in the training stage.

The Influence of Number of Layers. The design of ADR-GNN can alleviate oversmoothing in two ways. First, by learning the diffusion coefficients $\mathbf{K}$, ADR-GNN controls the amount of smoothing, and can also achieve no smoothing if $\mathbf{K}$ is zero, depending on the data. Second, note that the advection and reaction terms can increase the frequency of the node features, because they are not limited to smoothing processes. To verify our observation, we evaluate ADR-GNN$_S$ on Cora and Citeseer with 2 to 64 layers, to see if its performance degrades as more layers are added, an issue that is associated with oversmoothing. We report the

Dataset	Method	Horizon 3			Horizon 6			Horizon 12		
		MAE	RMSE	MAPE	MAE	RMSE	MAPE	MAE	RMSE	MAPE
METR-LA	DCRNN	2.77	5.38	7.30%	3.15	6.45	8.80%	3.60	7.60	10.50%
	STGCN	2.88	5.74	7.62%	3.47	7.24	9.57%	4.59	9.40	12.70%
	Graph WaveNet	2.69	5.15	6.90%	3.07	6.22	8.37%	3.53	7.37	10.01%
	ASTGCN	4.86	9.27	9.21%	5.43	10.61	10.13%	6.51	12.52	11.64%
	MTGNN	2.69	5.18	6.88%	3.05	6.17	8.19%	3.49	7.23	9.87%
	GTS	2.67	5.27	7.21%	3.04	6.25	8.41%	3.46	7.31	9.98%
	STEP	2.61	4.98	6.60%	2.96	5.97	7.96%	3.37	6.99	9.61%
	STAEformer	2.65	5.11	6.85%	2.97	6.00	8.13%	3.34	7.02	9.70%
	ADR-GNN$_{\mathrm{T}}$	**2.53**	**4.85**	**6.51%**	**2.81**	**5.82**	**7.39%**	**3.19**	**6.89**	**9.10%**

Table 4: Multivariate time series forecasting on the METR-LA. Additional results are provided in the Appendix.

A	**D**	**R**	Cora	Cham.	METR-LA	PEMS-BAY
✓	✗	✗	86.69	66.79	1.84	3.39
✗	✓	✗	88.21	65.08	1.93	3.67
✗	✗	✓	77.76	52.28	2.19	4.24
✓	✓	✗	88.92	73.33	1.79	3.30
✗	✓	✓	89.33	72.08	1.82	3.46
✓	✗	✓	88.02	73.46	1.71	3.21
✓	✓	✓	**89.43**	**79.91**	**1.68**	**3.19**

Table 5: Impact of Advection (**A**), Diffusion (**D**), and Reaction (**R**) on the Accuracy (%) on Cora and Chameleon, MAE on METR-LA and PEMS-BAY.

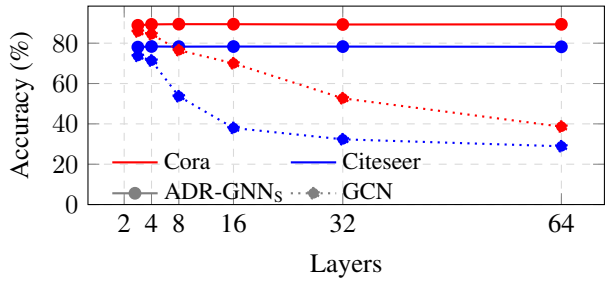


Figure 3: Accuracy (%) vs. model depth.

obtained accuracy in Figure 3, where no performance drop is evident. For reference, we also report the results obtained with GCN (Kipf and Welling 2017). Also, we define and report the measured Dirichlet energy in the Appendix, which shows that ADR-GNN does not oversmooth.

5 Summary and Discussion

In this paper, we present a novel GNN architecture that is based on the Advection-Diffusion-Reaction PDE, called ADR-GNN. We develop a graph neural advection operator that mimics the continuous advection operator, and compose it with learnable diffusion and reaction terms.

We discuss and analyze the properties of ADR-GNN and its flexibility in modeling various phenomena. In particular, we show that the main advantage of the graph advection operator is its ability to transport information over the graph edges through the layers - a behavior that is hard to model using the diffusion and reaction terms that have been used in the literature. To demonstrate the effectiveness of ADR-GNN we experiment with a total of 18 real-world datasets, from homophilic and heterophilic node classification to spatio-temporal node forecasting datasets.

While the gains observed on homophilic datasets are relatively modest, the performance improvements demonstrated on heterophilic datasets are significant, offering 5% accuracy increase in some cases. Moreover, when applied to spatio-temporal node forecasting datasets, our ADR-GNN exhibits notable enhancements in the evaluated metrics compared to other methods. This progress can be attributed to the inherent suitability of ADR-GNN for tasks involving directional transportation of features, making it an intuitive choice for modeling such scenarios.

Acknowledgements

This research was supported by grant no. 2018209 from the United States - Israel Binational Science Foundation (BSF), Jerusalem, Israel, and by the Israeli Council for Higher Education (CHE) via the Data Science Research Center and the Lynn and William Frankel Center for Computer Science at BGU. ME is supported by Kreitman High-tech scholarship.

References

Adam, M. S.; and Sibert, J. R. 2004. Use of neural networks with advection-diffusion-reaction models to estimate large-scale movements of Skipjack tuna from tagging data.

Alon, U.; and Yahav, E. 2021. On the Bottleneck of Graph Neural Networks and its Practical Implications. In *International Conference on Learning Representations*.

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

Bai, L.; Yao, L.; Li, C.; Wang, X.; and Wang, C. 2020. Adaptive Graph Convolutional Recurrent Network for Traffic Forecasting. *Advances in Neural Information Processing Systems*, 33.

Betts, J. 2001. *Practical Methods for Optimal Control using Nonlinear Programming*. Advances in Design and Control. Philadelphia: SIAM.

Borovitskiy, V.; Azangulov, I.; Terenin, A.; Mostowsky, P.; Deisenroth, M. P.; and Durrande, N. 2021. Matern Gaussian Processes on Graphs. In *International Conference on Artificial Intelligence and Statistics*. PMLR.

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

Chamberlain, B. P.; Rowbottom, J.; Gorinova, M.; Webb, S.; Rossi, E.; and Bronstein, M. M. 2021. GRAND: Graph neural diffusion. In *International Conference on Machine Learning (ICML)*, 1407–1418. PMLR.

Chapman, A.; and Chapman, A. 2015. Advection on graphs. *Semi-Autonomous Networks: Effective Control of Networked Systems through Protocols, Design, and Modeling*, 3–16.

Chen, C.; Petty, K.; Skabardonis, A.; Varaiya, P.; and Jia, Z. 2001. Freeway performance measurement system: mining loop detector data. *Transportation Research Record*, 1748(1): 96–102.

Chen, J.; Xu, X.; Wu, Y.; and Zheng, H. 2018a. GC-LSTM: Graph Convolution Embedded LSTM for Dynamic Link Prediction. *arXiv preprint arXiv:1812.04206*.

Chen, T. Q.; Rubanova, Y.; Bettencourt, J.; and Duvenaud, D. K. 2018b. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, 6571–6583.

Chien, E.; Peng, J.; Li, P.; and Milenkovic, O. 2021. Adaptive Universal Generalized PageRank Graph Neural Network. In *International Conference on Learning Representations*.

Choi, J.; Hong, S.; Park, N.; and Cho, S.-B. 2022. GREAD: Graph Neural Reaction-Diffusion Equations. *arXiv preprint arXiv:2211.14208*.

Coats, K. H. 2000. A note on IMPES and some IMPES-based simulation models. *SPE Journal*, 5(03): 245–251.

Cosner, C. 2014. Reaction-diffusion-advection models for the effects and evolution of dispersal. *Discrete & Continuous Dynamical Systems*, 34(5): 1701.

Deepa Maheshvare, M.; Raha, S.; and Pal, D. 2022. A graph-based framework for multiscale modeling of physiological transport. *Frontiers in Network Physiology*, 1: 18.

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

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

Fiedler, B.; and Scheel, A. 2003. Spatio-temporal dynamics of reaction-diffusion patterns. *Trends in nonlinear analysis*, 23–152.

Giovanni, F. D.; Rowbottom, J.; Chamberlain, B. P.; Markovich, T.; and Bronstein, M. M. 2023. Understanding convolution on graphs via energies. *Transactions on Machine Learning Research*.

Gravina, A.; Bacciu, D.; and Gallicchio, C. 2023. Anti-Symmetric DGN: a stable architecture for Deep Graph Networks. In *The Eleventh International Conference on Learning Representations*.

Guo, S.; Lin, Y.; Feng, N.; Song, C.; and Wan, H. 2019. Attention based spatial-temporal graph convolutional networks for traffic flow forecasting. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, 922–929.

Haber, E.; Lensink, K.; Treister, E.; and Ruthotto, L. 2019. IMEXnet a forward stable deep neural network. In *International Conference on Machine Learning*, 2525–2534. PMLR.

Haber, E.; and Ruthotto, L. 2017. Stable architectures for deep neural networks. *Inverse Problems*, 34(1).

Jagadish, H. V.; Gehrke, J.; Labrinidis, A.; Papakonstantinou, Y.; Patel, J. M.; Ramakrishnan, R.; and Shahabi, C. 2014. Big data and its technical challenges. *Communications of the ACM*, 57(7): 86–94.

Kadioglu, S.; Knoll, D.; Sussman, M.; and Martineau, R. 2011. A second order JFNK-based IMEX method for single and multi-phase flows. In *Computational Fluid Dynamics 2010*, 549–554. Springer.

Khokhlov, A. M. 1995. Propagation of turbulent flames in supernovae. *The Astrophysical Journal*, 449: 695.

Kipf, T. N.; and Welling, M. 2017. Semi-supervised classification with graph convolutional networks. *International Conference on Learning Representations (ICLR)*.

LeVeque, R. 1990. *Numerical Methods for Conservation Laws*. Birkhauser.

Li, Y.; Yu, R.; Shahabi, C.; and Liu, Y. 2018. Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting. In *International Conference on Learning Representations*.

Lim, D.; Hohne, F. M.; Li, X.; Huang, S. L.; Gupta, V.; Bhalerao, O. P.; and Lim, S.-N. 2021a. Large Scale Learning on Non-Homophilous Graphs: New Benchmarks and Strong Simple Methods. In Beygelzimer, A.; Dauphin, Y.; Liang, P.; and Vaughan, J. W., eds., *Advances in Neural Information Processing Systems*.

Lim, D.; Li, X.; Hohne, F.; and Lim, S.-N. 2021b. New Benchmarks for Learning on Non-Homophilous Graphs. *Workshop on Graph Learning Benchmarks, WWW*.

Liu, H.; Dong, Z.; Jiang, R.; Deng, J.; Deng, J.; Chen, Q.; and Song, X. 2023. Spatio-temporal adaptive embedding makes vanilla transformer sota for traffic forecasting. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, 4125–4129.

Long, Z.; Lu, Y.; Ma, X.; and Dong, B. 2018. PDE-Net: Learning PDEs from Data.

Luan, S.; Hua, C.; Lu, Q.; Zhu, J.; Zhao, M.; Zhang, S.; Chang, X.-W.; and Precup, D. 2022. Revisiting Heterophily For Graph Neural Networks. *Conference on Neural Information Processing Systems*.

Mardani, M.; Sun, Q.; Donoho, D.; Papyan, V.; Monajemi, H.; Vasanawala, S.; and Pauly, J. 2018. Neural proximal gradient descent for compressive imaging. *Advances in Neural Information Processing Systems*, 31.

Maskey, S.; Paolino, R.; Bacho, A.; and Kutyniok, G. 2023. A Fractional Graph Laplacian Approach to Oversmoothing. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Ming Chen, Z. W.; Zengfeng Huang, B. D.; and Li, Y. 2020. Simple and Deep Graph Convolutional Networks.

Oono, K.; and Suzuki, T. 2020. Graph Neural Networks Exponentially Lose Expressive Power for Node Classification. In *International Conference on Learning Representations*.

Pei, H.; Wei, B.; Chang, K. C.-C.; Lei, Y.; and Yang, B. 2020. Geom-GCN: Geometric Graph Convolutional Networks. In *International Conference on Learning Representations*.

Raissi, M. 2018. Deep hidden physics models: Deep learning of nonlinear partial differential equations. *The Journal of Machine Learning Research*, 19(1): 932–955.

Rozemberczki, B.; Allen, C.; and Sarkar, R. 2021. Multi-Scale Attributed Node Embedding. *Journal of Complex Networks*, 9(2).

Rozemberczki, B.; Scherer, P.; He, Y.; Panagopoulos, G.; Riedel, A.; Astefanoaei, M.; Kiss, O.; Beres, F.; López, G.; Collignon, N.; et al. 2021. Pytorch geometric temporal: Spatiotemporal signal processing with neural machine learning models. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, 4564–4573.

Rusch, T. K.; Bronstein, M. M.; and Mishra, S. 2023. A Survey on Oversmoothing in Graph Neural Networks. *arXiv preprint arXiv:2303.10993*.

Rusch, T. K.; Chamberlain, B.; Rowbottom, J.; Mishra, S.; and Bronstein, M. 2022. Graph-coupled oscillator networks. In *International Conference on Machine Learning*, 18888–18909. PMLR.

Ruthotto, L.; and Haber, E. 2020. Deep neural networks motivated by partial differential equations. *Journal of Mathematical Imaging and Vision*, 62: 352–364.

Sanchez-Gonzalez, A.; Bapst, V.; Cranmer, K.; and Battaglia, P. 2019. Hamiltonian graph networks with ode integrators. *arXiv preprint arXiv:1909.12790*.

Seed, A. 2003. A dynamic and spatial scaling approach to advection forecasting. *Journal of Applied Meteorology and Climatology*, 42(3): 381–388.

Seo, Y.; Defferrard, M.; Vandergheynst, P.; and Bresson, X. 2018. Structured Sequence Modeling with Graph Convolutional Recurrent Networks. In *International Conference on Neural Information Processing*, 362–373. Springer.

Shang, C.; Chen, J.; and Bi, J. 2021. Discrete Graph Structure Learning for Forecasting Multiple Time Series. In *International Conference on Learning Representations*.

Shao, Z.; Zhang, Z.; Wang, F.; and Xu, Y. 2022. Pre-training enhanced spatial-temporal graph neural network for multivariate time series forecasting. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 1567–1577.

Song, Y.; Zhou, C.; Wang, X.; and Lin, Z. 2023. Ordered GNN: Ordering Message Passing to Deal with Heterophily and Over-smoothing. In *The Eleventh International Conference on Learning Representations*.

Thorpe, M.; Nguyen, T. M.; Xia, H.; Strohmer, T.; Bertozzi, A.; Osher, S.; and Wang, B. 2022. GRAND++: Graph Neural Diffusion with A Source Term. In *International Conference on Learning Representations*.

Uys, L. 2009. *Coupling kinetic models and advection-diffusion equations to model vascular transport in plants, applied to sucrose accumulation in sugarcane*. Ph.D. thesis, Stellenbosch: University of Stellenbosch.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Liò, P.; and Bengio, Y. 2018. Graph Attention Networks. *International Conference on Learning Representations*.

Vese, L. A.; and Chan, T. F. 2002. A multiphase level set framework for image segmentation using the Mumford and Shah model. *International journal of computer vision*, 50(3): 271–293.

Wang, Y.; Yi, K.; Liu, X.; Wang, Y. G.; and Jin, S. 2022. ACMP: Allen-Cahn Message Passing for Graph Neural Networks with Particle Phase Transition. *arXiv preprint arXiv:2206.05437*.

Wu, Q.; Yang, C.; Zhao, W.; He, Y.; Wipf, D.; and Yan, J. 2023. DIFFormer: Scalable (Graph) Transformers Induced by Energy Constrained Diffusion. In *The Eleventh International Conference on Learning Representations*.

Wu, Z.; Pan, S.; Long, G.; Jiang, J.; Chang, X.; and Zhang, C. 2020. Connecting the dots: Multivariate time series forecasting with graph neural networks. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, 753–763.

Wu, Z.; Pan, S.; Long, G.; Jiang, J.; and Zhang, C. 2019. Graph Wavenet for Deep Spatial-Temporal Graph Modeling. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, IJCAI'19, 1907–1913. AAAI Press. ISBN 9780999241141.

Yan, Y.; Hashemi, M.; Swersky, K.; Yang, Y.; and Koutra, D. 2022. Two sides of the same coin: Heterophily and oversmoothing in graph convolutional neural networks. In *2022 IEEE International Conference on Data Mining (ICDM)*, 1287–1292. IEEE.

Yang, L.; Li, M.; Liu, L.; Wang, C.; Cao, X.; Guo, Y.; et al. 2021. Diverse message passing for attribute with heterophily. *Advances in Neural Information Processing Systems*, 34: 4751–4763.

Zhu, J.; Song, Y.; Zhao, L.; and Li, H. 2020a. A3T-GCN: Attention Temporal Graph Convolutional Network for Traffic Forecasting. *arXiv preprint arXiv:2006.11583*.

Zhu, J.; Yan, Y.; Zhao, L.; Heimann, M.; Akoglu, L.; and Koutra, D. 2020b. Beyond homophily in graph neural networks: Current limitations and effective designs. *Advances in Neural Information Processing Systems*, 33: 7793–7804.