Advection Diffusion Reaction Graph Neural Networks for Spatio-Temporal Data

Moshe Eliasof
University of Cambridge
me532@cam.ac.uk

Eldad Haber
University of British Columbia
eldadhaber@gmail.com

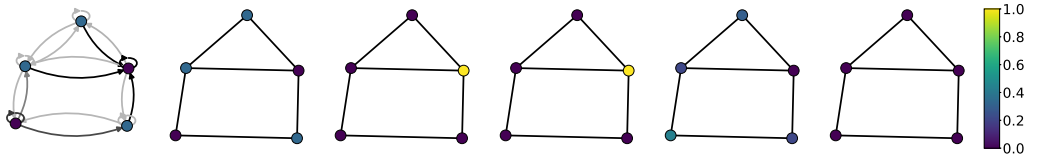
Eran Treister
Ben-Gurion University of the Negev
erant@cs.bgu.ac.il

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 advection. In this paper, we propose a novel GNN architecture based on Advection-Diffusion-Reaction systems, and demonstrate its efficacy on real-world spatio-temporal datasets.

1 Introduction

Recently, GNNs have been linked to ordinary and partial differential equations (ODEs and PDEs) in a series of works [1–7]. 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. For instance, ODE and PDE based models allow to reason about the behavior of existing GNNs. Nonetheless, the aforementioned architectures still rely on controlled diffusion or wave propagation, as well as non-linear pointwise convolutions, which, as shown in [8], may lack expressiveness. We now provide a simple example, known as the graph node feature transportation task [9], where diffusion, 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 [10, 11]. 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, while diffusion and reaction terms fail to model the target configuration. Furthermore, the concept of advection appears in many real-world problems and data, such as traffic-flow and-control [12], quantity transportation in computational biology [13], and rainfall forecasting [14]. 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 [9–11], 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.



(a) Advection weights **(b)** Source **(c)** Target **(d)** Advection output **(e)** Diffusion output **(f)** Reaction output

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.

M. Eliasof et al., Advection Diffusion Reaction Graph Neural Networks for Spatio-Temporal Data (Extended Abstract). Presented at the Second Learning on Graphs Conference (LoG 2023), Virtual Event, November 27–30, 2023.

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 [15], modeling ecological trends [16], and the modeling of turbulent flames in supernovae [17]. 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 [18–20] to GNNs, works like GODE [1], GRAND [2], PDE-GCN$_{\text{D}}$ [3], GRAND++ [21] 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 [22–24] in GNNs. Thus, works like [6, 25–27] propose to utilize a *learnable* diffusion term, thereby alleviating oversmoothing. Other architectures like PDE-GCN$_{\text{M}}$ [3] and GraphCON [4] propose to mix diffusion and oscillatory processes to avoid oversmoothing. Nonetheless, as noted in [8], besides alleviating oversmoothing, it is also important to design GNN architectures with improved expressiveness. Recent examples of such networks are [7] that propose an anti-symmetric GNN to alleviate over-squashing [28], and [5, 29] that formulate a reaction-diffusion GNN to enable non-trivial pattern growth.

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 [30, 31] are available. The advection operator has shown its effectiveness in classical (i.e., non-neural) graph methods, from blood-vessel simulations [32], to traffic flow prediction [33]. 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

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. 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.
- **Diffusion.** We denote the continuous Laplacian operator by Δ. The Laplacian is scaled with a diagonal matrix $K = \text{diag}(\kappa_1, \ldots, \kappa_c) \in \mathbb{R}^{c \times c},\ \ \kappa_i \geq 0$ 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

Space Discretization of the ADR-PDE. Equation (1) is defined in the continuum. We now use a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ to discretize Ω. The nodes $\mathcal{V}$ are a discretization of X, and the edges $\mathcal{E}$ represent the topology of Ω. Then, the *spatial*, graph discretization of Equation (1) is:

$$\frac{d\mathbf{U}(t)}{dt} = \mathbf{DIV}\left(\mathbf{V}\left(\mathbf{U}(t), t; \boldsymbol{\theta}_a(t)\right)\mathbf{U}(t)\right) - \hat{\mathbf{L}}\mathbf{U}(t)\mathbf{K}(t; \boldsymbol{\theta}_d(t)) + f(\mathbf{U}(t), \mathbf{X}, t; \boldsymbol{\theta}_r(t)). \quad (2)$$

Here, $\mathbf{U}(t) \in \mathbb{R}^{n\times c}$ are the node features at time t. The advection term depends on the velocity $\mathbf{V}$ parameterized by learnable weights $\boldsymbol{\theta}_a(t)$, discussed in Section 3.2. The diffusion is discretized using the Laplacian $\hat{\mathbf{L}}$ that is scaled with a diagonal matrix with non-negative learnable diffusion coefficients on its diagonal $\mathbf{K}(t; \boldsymbol{\theta}_d(t)) = \mathrm{diag}(\mathrm{hardtanh}(\boldsymbol{\theta}_d(t), 0, 1)) \geq 0$. The reaction term f is a pointwise non-linear function realized by a multilayer-perceptron (MLP) with weights $\boldsymbol{\theta}_r(t)$.

Time Discretization of the ADR-ODE. 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 [34–36]. That is, upon discrete time integration of Equation (2), we replace the notion of time t with l layers, and a step size h, that is a positive scalar hyperparameter. Here we use the common operator splitting discretization technique [11] for Equation (2), that yields a graph neural ADR layer, summarized in Algorithm 1. We defer the details of diffusion and reactions terms to Appendix A, as those are well-known in the GNN literature.

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: $\mathbf{U}^{(l+1)} = \mathbf{U}^{(l+2/3)} + hf(\mathbf{U}^{(l+2/3)}, \mathbf{U}^{(0)}, t; \boldsymbol{\theta}_r^{(l)})$.

Neural Graph Advection Operator. We extend the non-learnable advection operator from [30], 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 $0 \leq \mathbf{V}_{i\to j} \leq 1$. 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 (2), that operates on the learned edge weights $\mathbf{V}$:

$$\mathbf{DIV}_i(\mathbf{VU}) = \textstyle\sum_{j\in\mathcal{N}_i} \mathbf{V}_{j\to i} \odot \mathbf{U}_j - \mathbf{U}_i \odot \sum_{j\in\mathcal{N}_i} \mathbf{V}_{i\to j} = \sum_{j\in\mathcal{N}_i} \mathbf{V}_{j\to i} \odot \mathbf{U}_j - \mathbf{U}_i, \quad (3)$$

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

$$\mathbf{U}_i^{(l+1/3)} = \mathbf{U}_i^{(l)} + h\mathbf{DIV}_i(\mathbf{V}^{(l)}\mathbf{U}^{(l)}) = \mathbf{U}_i^{(l)} + h\left(\textstyle\sum_{j\in\mathcal{N}_i} \mathbf{V}_{j\to i}^{(l)} \odot \mathbf{U}_j^{(l)} - \mathbf{U}_i^{(l)}\right). \quad (4)$$

The updated node features are obtained by adding the $\mathbf{V}_{j\to i}$ weighted inbound features, while removing the $\mathbf{V}_{i\to j}$ weighted outbound features.To learn a *consistent* advection operator that mimics the directional behavior of the advection in Equation (1), we craft an edge weight $\mathbf{V}$ mechanism, parametrized by fully connected layers $\boldsymbol{\theta}_a^{(l)} = \{\mathbf{A}_1^{(l)}, \mathbf{A}_2^{(l)}, \mathbf{A}_3^{(l)}, \mathbf{A}_4^{(l)}\}$, as shown in Algorithm 2. This mechanism yields direction-oriented weights, i.e., $\mathbf{V}_{i\to j} \neq \mathbf{V}_{j\to i}$, unless they are zeros.

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:
$\mathbf{Z}_{ij}^{(l)} = \mathrm{ReLU}(\mathbf{U}_i^{(l)}\mathbf{A}_1^{(l)} + \mathbf{U}_j^{(l)}\mathbf{A}_2^{(l)})\mathbf{A}_3^{(l)}$, $\mathbf{Z}_{ji}^{(l)} = \mathrm{ReLU}(\mathbf{U}_j^{(l)}\mathbf{A}_1^{(l)} + \mathbf{U}_i^{(l)}\mathbf{A}_2^{(l)})\mathbf{A}_3^{(l)}$.
$\mathbf{V}_{i\to j}^{(l)} = \mathrm{ReLU}(\mathbf{Z}_{ij}^{(l)} - \mathbf{Z}_{ji}^{(l)})\mathbf{A}_4^{(l)}$, $\mathbf{V}_{j\to i}^{(l)} = \mathrm{ReLU}(-\mathbf{Z}_{ij}^{(l)} + \mathbf{Z}_{ji}^{(l)})\mathbf{A}_4^{(l)}$.

2: Normalize to obtain edge weights: $\mathbf{V}_{i\to j}^{(l)} \leftarrow \frac{\exp(\mathbf{V}_{i\to j}^{(l)})}{\sum_{k\in\mathcal{N}_i}\exp(\mathbf{V}_{i\to k}^{(l)})}$, $\mathbf{V}_{j\to i}^{(l)} \leftarrow \frac{\exp(\mathbf{V}_{j\to i}^{(l)})}{\sum_{k\in\mathcal{N}_j}\exp(\mathbf{V}_{j\to k}^{(l)})}$.

Dataset	Method	Horizon 3			Horizon 6			Horizon 12		
		MAE	RMSE	MAPE	MAE	RMSE	MAPE	MAE	RMSE	MAPE
METR-LA	FC-LSTM	3.44	6.30	9.60%	3.77	7.23	10.09%	4.37	8.69	14.00%
	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%
	STSGCN	3.31	7.62	8.06%	4.13	9.77	10.29%	5.06	11.66	12.91%
	GMAN	2.80	5.55	7.41%	3.12	6.49	8.73%	3.44	7.35	10.07%
	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%
	ADR-GNN	**2.53**	**4.85**	**6.51%**	**2.81**	**5.82**	**7.39%**	**3.19**	**6.89**	**9.10%**
PEMS-BAY	FC-LSTM	2.05	4.19	4.80%	2.20	4.55	5.20%	2.37	4.96	5.70%
	DCRNN	1.38	2.95	2.90%	1.74	3.97	3.90%	2.07	4.74	4.90%
	STGCN	1.36	2.96	2.90%	1.81	4.27	4.17%	2.49	5.69	5.79%
	Graph WaveNet	1.30	2.74	2.73%	1.63	3.70	3.67%	1.95	4.52	4.63%
	ASTGCN	1.52	3.13	3.22%	2.01	4.27	4.48%	2.61	5.42	6.00%
	STSGCN	1.44	3.01	3.04%	1.83	4.18	4.17%	2.26	5.21	5.40%
	GMAN	1.34	2.91	2.86%	1.63	3.76	3.68%	1.86	4.32	4.37%
	MTGNN	1.32	2.79	2.77%	1.65	3.74	3.69%	1.94	4.49	4.53%
	GTS	1.34	2.83	2.82%	1.66	3.78	3.77%	1.95	4.43	4.58%
	STEP	1.26	2.73	2.59%	1.55	3.58	3.43%	1.79	4.20	4.18%
	ADR-GNN	**1.13**	**2.36**	**2.30%**	**1.39**	**3.13**	**3.01%**	**1.68**	**3.81**	**3.82%**

Table 1: Multivariate time series forecasting on METR-LA and PEMS-BAY.

4 Experimental Results

We demonstrate our ADR-GNN on two spatio-temporal node forecasting datasets. Architecture and training details are provided in Appendix B. We use a grid search to select hyperparameters, discussed in Appendix C. Datasets details and statistics are reported in Appendix D.

Classical ADR models are widely utilized to predict and model spatio-temporal phenomena [15, 37]. We therefore evaluate our ADR-GNN on two popular spatio-temporal node forecasting datasets, namely, the traffic speed prediction datasets METR-LA [38] and PEMS-BAY [39]. We consider the mean-absolute-error (MAE) loss as in [40]. We report the MAE, root mean squared error (RMSE), and mean absolute percentage error (MAPE). To demonstrate the effectiveness of ADR-GNN 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 with various methods like FC-LSTM [41], DCRNN [40], Graph WaveNet [42], ASTGCN [43], STSGCN [44], GMAN [45], MTGNN [46], GTS [47], and STEP [48]. We find that our ADR-GNN offers lower (better) metrics than the considered methods. For instance, on METR-LA, ADR-GNN reduces the MAE achieved by the recent STEP method from 3.37 to 3.19. We summarize the results in Table 1.

The Impact of Advection, Diffusion, and Reaction. We study the influence of each of the proposed terms in Equation (2) on real-world datasets, independently and jointly. The results, reported in Table 2 further show the significance of the advection term due to its offering of increased performance.

A	D	R	METR-LA	PEMS-BAY
✓	✗	✗	1.84	3.39
✗	✓	✗	1.93	3.67
✗	✗	✓	2.19	4.24
✓	✓	✗	1.79	3.30
✗	✓	✓	1.82	3.46
✓	✗	✓	1.71	3.21
✓	✓	✓	**1.68**	**3.19**

Table 2: Impact of Advection ($\mathbf{A}$), Diffusion ($\mathbf{D}$), and Reaction ($\mathbf{R}$) on the MAE on METR-LA and PEMS-BAY.

5 Summary

We presented a novel GNN architecture that is based on the ADR PDE. 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. We show the significance of ADR-GNN using two spatio-temporal forecasting datasets.

References

[1] Juntang Zhuang, Nicha Dvornek, Xiaoxiao Li, and James S Duncan. Ordinary differential equations on graph networks. 2020. 1, 2

[2] Benjamin Paul Chamberlain, James Rowbottom, Maria Gorinova, Stefan Webb, Emanuele Rossi, and Michael M Bronstein. GRAND: Graph neural diffusion. In *International Conference on Machine Learning (ICML)*, pages 1407–1418. PMLR, 2021. 2, 8

[3] 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. 2

[4] 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. 2

[5] 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. 2

[6] Francesco Di Giovanni, James Rowbottom, Benjamin P Chamberlain, Thomas Markovich, and Michael M Bronstein. Graph neural networks as gradient flows. *arXiv preprint arXiv:2206.10991*, 2022. 2

[7] Alessio Gravina, Davide Bacciu, and Claudio Gallicchio. Anti-symmetric dgn: a stable architecture for deep graph networks. *arXiv preprint arXiv:2210.09789*, 2022. 1, 2

[8] T Konstantin Rusch, Michael M Bronstein, and Siddhartha Mishra. A survey on oversmoothing in graph neural networks. *arXiv preprint arXiv:2303.10993*, 2023. 1, 2

[9] R.J. LeVeque. *Numerical Methods for Conservation Laws*. Birkhauser, 1990. 1

[10] L. C. Evans. *Partial Differential Equations*. American Mathematical Society, San Francisco, 1998. 1

[11] Uri M Ascher. *Numerical methods for evolutionary differential equations*. SIAM, 2008. 1, 3, 8, 9

[12] J.T. Betts. *Practical Methods for Optimal Control using Nonlinear Programming*. Advances in Design and Control. SIAM, Philadelphia, 2001. 1

[13] Lafras Uys. *Coupling kinetic models and advection-diffusion equations to model vascular transport in plants, applied to sucrose accumulation in sugarcane*. PhD thesis, Stellenbosch: University of Stellenbosch, 2009. 1

[14] AW Seed. A dynamic and spatial scaling approach to advection forecasting. *Journal of Applied Meteorology and Climatology*, 42(3):381–388, 2003. 1

[15] M Shiham Adam and John R Sibert. Use of neural networks with advection-diffusion-reaction models to estimate large-scale movements of skipjack tuna from tagging data. Technical report, Marine Research Centre, Ministry of Fisheries, Agriculture and Marine Resources, 2004. 2, 4

[16] Chris Cosner. Reaction-diffusion-advection models for the effects and evolution of dispersal. *Discrete & Continuous Dynamical Systems*, 34(5):1701, 2014. 2

[17] Alexei M Khokhlov. Propagation of turbulent flames in supernovae. *The Astrophysical Journal*, 449:695, 1995. 2

[18] Lars Ruthotto and Eldad Haber. Deep neural networks motivated by partial differential equations. *Journal of Mathematical Imaging and Vision*, 62:352–364, 2020. 2

[19] Weikai Chen, Xiaoguang Han, Guanbin Li, Chao Chen, Jun Xing, Yajie Zhao, and Hao Li. Deep rbfnet: Point cloud feature learning using radial basis functions. *arXiv preprint arXiv:1812.04302*, 2018.

[20] 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. 2

[21] 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. 2

[22] Hoang Nt and Takanori Maehara. Revisiting graph neural networks: All we have is low-pass filters. *arXiv preprint arXiv:1905.09550*, 2019. 2

[23] 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.

[24] Chen Cai and Yusu Wang. A note on over-smoothing for graph neural networks. *arXiv preprint arXiv:2006.13318*, 2020. 2

[25] 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. 2

[26] Sitao Luan, Mingde Zhao, Chenqing Hua, Xiao-Wen Chang, and Doina Precup. Complete the missing half: Augmenting aggregation filtering with diversification for graph convolutional networks. *arXiv preprint arXiv:2008.08844*, 2020.

[27] 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. 2

[28] 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. 2

[29] Jeongwhan Choi, Seoyoung Hong, Noseong Park, and Sung-Bae Cho. Gread: Graph neural reaction-diffusion equations. *arXiv preprint arXiv:2211.14208*, 2022. 2, 8

[30] Airlie Chapman and Airlie Chapman. Advection on graphs. *Semi-Autonomous Networks: Effective Control of Networked Systems through Protocols, Design, and Modeling*, pages 3–16, 2015. 2, 3

[31] Radim Hošek and Jonáš Volek. Discrete advection–diffusion equations on graphs: Maximum principle and finite volumes. *Applied Mathematics and Computation*, 361:630–644, 2019. 2

[32] M Deepa Maheshvare, Soumyendu Raha, and Debnath Pal. A graph-based framework for multiscale modeling of physiological transport. *Frontiers in Network Physiology*, 1:18, 2022. 2

[33] Viacheslav Borovitskiy, Iskander Azangulov, Alexander Terenin, Peter Mostowsky, Marc Peter Deisenroth, and Nicolas Durrande. Matern gaussian processes on graphs. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 2021. 2

[34] E. Haber and L. Ruthotto. Stable architectures for deep neural networks. *Inverse Problems*, 34 (1), 2017. 3

[35] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K. Duvenaud. Neural ordinary differential equations. *CoRR*, abs/1806.07366, 2018. URL http://arxiv.org/abs/1806.07366.

[36] E Weinan. A Proposal on Machine Learning via Dynamical Systems. *Communications in Mathematics and Statistics*, 5(1):1–11, March 2017. 3

[37] Bernold Fiedler and Arnd Scheel. Spatio-temporal dynamics of reaction-diffusion patterns. *Trends in nonlinear analysis*, pages 23–152, 2003. 4

[38] Hosagrahar V Jagadish, Johannes Gehrke, Alexandros Labrinidis, Yannis Papakonstantinou, Jignesh M Patel, Raghu Ramakrishnan, and Cyrus Shahabi. Big data and its technical challenges. *Communications of the ACM*, 57(7):86–94, 2014. 4

[39] Chao Chen, Karl Petty, Alexander Skabardonis, Pravin Varaiya, and Zhanfeng Jia. Freeway performance measurement system: mining loop detector data. *Transportation Research Record*, 1748(1):96–102, 2001. 4

[40] 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. 4, 8

[41] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, 2014. 4

[42] Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, and Chengqi Zhang. Graph wavenet for deep spatial-temporal graph modeling. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, IJCAI'19, page 1907–1913. AAAI Press, 2019. ISBN 9780999241141. 4

[43] Shengnan Guo, Youfang Lin, Ning Feng, Chao Song, and Huaiyu Wan. Attention based spatial-temporal graph convolutional networks for traffic flow forecasting. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 922–929, 2019. 4

[44] Chao Song, Youfang Lin, Shengnan Guo, and Huaiyu Wan. Spatial-temporal synchronous graph convolutional networks: A new framework for spatial-temporal network data forecasting. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 914–921, 2020. 4

[45] Chuanpan Zheng, Xiaoliang Fan, Cheng Wang, and Jianzhong Qi. Gman: A graph multi-attention network for traffic prediction. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 1234–1241, 2020. 4

[46] Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, Xiaojun Chang, and Chengqi Zhang. 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*, pages 753–763, 2020. 4

[47] Chao Shang, Jie Chen, and Jinbo Bi. Discrete graph structure learning for forecasting multiple time series. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=WEHSlH5mOk. 4

[48] Zezhi Shao, Zhao Zhang, Fei Wang, and Yongjun Xu. 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*, pages 1567–1577, 2022. 4

[49] Toshiyuki Koto. Imex runge–kutta schemes for reaction–diffusion equations. *Journal of Computational and Applied Mathematics*, 215(1):182–195, 2008. 8

[50] Eldad Haber, Keegan Lensink, Eran Treister, and Lars Ruthotto. IMEXnet a forward stable deep neural network. In *International Conference on Machine Learning*, pages 2525–2534. PMLR, 2019. 8

[51] G.H. Golub and C.F. van Loan. *Matrix Computations*. Johns Hopkins University Press, 1988. 8

[52] Siddhant M. Jayakumar, Wojciech M. Czarnecki, Jacob Menick, Jonathan Schwarz, Jack Rae, Simon Osindero, Yee Whye Teh, Tim Harley, and Razvan Pascanu. Multiplicative interactions and where to find them. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rylnK6VtDH. 8

[53] Ido Ben-Shaul, Tomer Galanti, and Shai Dekel. Exploring the approximation capabilities of multiplicative neural networks for smooth functions. *arXiv preprint arXiv:2301.04605*, 2023. 8

[54] 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. 8

A Diffusion and Reaction Terms

We now discuss the diffusion and reaction terms in our ADR-GNN.

Diffusion. To discretize the diffusion term from Equation (2), both explicit and implicit time discretizations can be used [11]. An explicit forward Euler discretization yields the following layer:

$$\mathbf{U}^{(l+2/3)} = \mathbf{U}^{(l+1/3)} - h\left(\hat{\mathbf{L}}\mathbf{U}^{(l+1/3)}\mathbf{K}^{(l)}\right). \tag{5}$$

However, an explicit scheme requires using a small step size $h > 0$, as it is marginally stable [11]. We therefore harness an implicit scheme, which guarantees the stability of the diffusion [1], and reads:

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

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[2] [11, 51]. 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 [29, 52, 53]. Hence, we define

$$f(\mathbf{U}^{(l+2/3)}, \mathbf{U}^{(0)}; \boldsymbol{\theta}_r^{(l)}) = \sigma(\mathbf{U}^{(l+2/3)}\mathbf{R}_1^{(l)} + \tanh(\mathbf{U}^{(l+2/3)}\mathbf{R}_2^{(l)}) \odot \mathbf{U}^{(l+2/3)} + \mathbf{U}^{(0)}\mathbf{R}_3^{(l)}), \tag{7}$$

as our reaction term in Equation (2). 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.

B Architecture and Training Details

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 $\mathbf{X}_{temporal} \in \mathbb{R}^{n \times \tau_{in} c_{in}}$, where τ_{in} is the number of input (observed) time frames, and the goal is to predict τ_{out} time frames ahead, i.e., $\mathbf{Y}_{\mathrm{temporal}} \in \mathbb{R}^{n \times \tau_{out} c_{out}}$.

In our spatio-temporal ADR-GNN, we update the hidden state feature matrix $\mathbf{U}_{\mathrm{state}}^{(l)}$ based on the hidden historical feature matrix $\mathbf{U}_{\mathrm{hist}}^{(l)}$, as shown in Lines 6-9 in Algorithm 3.

Similarly to Attention models [54], 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 ADR-GNN, denoted by $\mathbf{T}_{\mathrm{emb}} \in \mathbb{R}^{n \times \tau_{in} c_t}$, where we choose the number of frequencies to be 10, and by the concatenation of both sine and cosine lead to $c_t = 20$. We note that the time embedding is computed in a pre-processing fashion. To initialize the hidden feature matrices $\mathbf{U}_{\mathrm{state}}^{(0)}$, $\mathbf{U}_{\mathrm{hist}}^{(0)}$, we embed the input data $\mathbf{X}_{\mathrm{temporal}}$, concatenated with $\mathbf{T}_{\mathrm{emb}}$, using two fully connected layers, as described in Lines 3-4 in Algorithm 3. [3]

We minimize the mean absolute error (MAE), similar to [40], where we also follow the standard 12 previous time frames as inputs, and consider 3,6, and 12 future time frames node quantity prediction as output.

C Hyperparameters

All hyperparameters were determined by grid search, and the ranges and sampling mechanism distributions are provided in Table 3. Note, that as discussed after Equation (7), we may add a BatchNorm layer before applying the non-linear activation σ to the reaction term, we therefore treat the use of batchnorm as a hyperparameter in Table 3.

[1] See [2, 49, 50] for details on implicit vs. explicit schemes for diffusion processes and in neural networks.

[2] 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.

[3] In Python notations, $\mathbf{X}_{\mathrm{temporal}}[:, -c_{in}]$ extracts the last c_{in} entries of the second dimension of $\mathbf{X}_{\mathrm{temporal}}$, which returns the features of the last time frame.

Algorithm 3 ADR-GNN Architecture Flow

Input: Node features $\mathbf{X}_{\text{temporal}} \in \mathbb{R}^{n \times \tau_{in} c_{in}}$, time embedding $\mathbf{T}_{\text{emb}} \in \mathbb{R}^{n \times \tau_{in} c_t}$
Output: Predicted future node quantities $\tilde{\mathbf{Y}} \in \mathbb{R}^{n \times \tau_{out} c_{out}}$

procedure ADR-GNN
  $\mathbf{X}_{\text{temporal}} \leftarrow \text{Dropout}(\mathbf{X}_{\text{temporal}}, p)$
  $\mathbf{T}_{\text{emb}} \leftarrow \text{g}^{\text{time-embed}}(\mathbf{T}_{\text{emb}})$
  $\mathbf{U}^{(0)}_{\text{state}} = g^{\text{state}}_{in}(\mathbf{X}_{\text{temporal}}[:, -c_{in}] \oplus \mathbf{T}_{\text{emb}})$
  $\mathbf{U}^{(0)}_{\text{hist}} = g^{\text{hist}}_{in}(\mathbf{X}_{\text{temporal}} \oplus \mathbf{T}_{\text{emb}})$
  for $l = 0 \dots L-1$ **do**
    $\mathbf{U}^{(l)}_{\text{state}} \leftarrow \text{Dropout}(\mathbf{U}^{(l)}_{\text{state}}, p)$
    Advection: $\mathbf{U}^{(l+1/3)}_{\text{state}} = \mathbf{U}^{(l)}_{\text{state}} + h\mathbf{DIV}(\mathbf{V}(\mathbf{U}^{(l)}_{\text{hist}}; \boldsymbol{\theta}^{(l)}_a)\mathbf{U}^{(l)}), \boldsymbol{\theta}_a)\mathbf{U}^{(l)}_{\text{state}}$
    Diffusion: $\mathbf{U}^{(l+2/3)}_{\text{state}} = \text{mat}\left((\mathbf{I} + h\mathbf{K}^{(l)} \otimes \hat{\mathbf{L}})^{-1}\text{vec}(\mathbf{U}^{(l+1/3)}_{\text{state}})\right)$
    Reaction: $\mathbf{U}^{(l+1)}_{\text{state}} = \mathbf{U}^{(l+2/3)}_{\text{state}} + hf(\mathbf{U}^{(l+2/3)}_{\text{hist}}, \mathbf{U}^{(0)}_{\text{hist}}; \boldsymbol{\theta}_r)$
    $\mathbf{U}^{(l+1)}_{\text{hist}} = g^{\text{hist}}_{l}(\mathbf{U}^{(l)}_{\text{hist}} \oplus \mathbf{U}^{(l+1)}_{\text{state}} \oplus \mathbf{T}_{\text{emb}})$
  end for
  $\mathbf{U}^{(L)}_{\text{state}} \leftarrow \text{Dropout}(\mathbf{U}^{(L)}_{\text{state}}, p)$
  $\tilde{\mathbf{Y}} = g^{\text{state}}_{out}(\mathbf{U}^{(L)}_{\text{state}})$
  Return $\tilde{\mathbf{Y}}$
end procedure

Table 3: Hyperparameter ranges

Hyperparameter	Range	Uniform Distribution
input/output embedding learning rate	[1e-4, 1e-1]	log uniform
advection learning rate	[1e-4, 1e-1]	log uniform
diffusion learning rate	[1e-4, 1e-1]	log uniform
reaction learning rate	[1e-4, 1e-1]	log uniform
input/output embedding weight decay	[0, 1e-2]	uniform
advection weight decay	[0, 1e-2]	uniform
diffusion weight decay	[0, 1e-2]	uniform
reaction 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
layers	{ 2,4,8,16,32,64 }	discrete uniform
channels	{ 8,16,32,64,128,256 }	discrete uniform

D Datasets

We report the statistics of the datasets used in our experiments in Table 4.

E Discussion

Advection as 'net transfer' mechanism. The advection term describes the information (feature) transfer from one node to another. It considers both the incoming and outgoing information and therefore it is possible to interpret it as the 'net' feature transfer.

On the composition of the Advection, Diffusion, and Reaction terms in Algorithm 1. In this work we utilize the operator splitting approach to discretize the temporal part of the ODE (i.e., the its right-hand-side). The basic idea of operator splitting is to integrate each part of the differential equation (advection, diffusion, and reaction) one after the other. This is a common scheme when integrating ODEs [11]. From this perspective, it would be fair to say that the diffusion term is influenced by the advection term and similarly for the reaction term. Therefore, this approach may

Table 4: Attributes of the spatio-temporal datasets in our experiments, and information about the number of time periods (T) and spatial units ($|\mathcal{V}|$).

Dataset	Frequency	T	$\lvert\mathcal{V}\rvert$
METR-LA	5-Minutes	34,272	207
PEMS-BAY	5-Minutes	52,116	325

lead to improved results as it uses more updated features. Also, from a machine learning perspective, the prescribed composition of the ADR terms essentially leads to a deeper network, which can also be beneficial.

On Advection vs. Transformers. The use of a weighted fully connected graph (by a transformer) does not necessarily prevent feature smoothing or can replicate the behavior of the advection operator. The crucial detail in modelling an advection operator is the characteristics of the edge weights (V from Equation 1). To generate a proper advective behaviour, the edge weights need to be directionally oriented, as shown in Figure 1. Therefore, while a fully connected graph allows the direct communication between all pairs of nodes, it is not straight forward to generate advective behaviour, and a careful construction of the transformer needs to be done, similarly to the edge weights mechanism proposed in our ADR-GNN.

In addition, changing the graph connectivity by rewiring or full connectivity (by a transformer) adds more complexity. Furthermore, in some cases may not be possible or straight forward to use. For instance, considering the spatiotemporal datasets for traffic prediction, it may not be possible to change the graph, as it represents a physical road network.