Solving Nonlinear Conservation Laws of Partial Differential Equations Using Graph Neural Networks

Qing Li §*[1], Jiahui Geng §[2], Steinar Evje[1], and Chunming Rong[2]

[1]University of Stavanger, Department of Energy and Petroleum, Group of Computational Engineering, Stavanger, Norway
[2]University of Stavanger, Department of Electrical Engineering and Computer Science, Stavanger, Norway

Abstract

Nonlinear Conservation Laws of Partial Differential Equations (PDEs) are widely used in different domains. Solving these types of equations is a significant and challenging task. Graph Neural Networks (GNNs) have recently been established as fast and accurate alternatives for principled solvers when applied to standard equations with regular solutions. There have been few investigations on GNNs implemented for complex PDEs with nonlinear conservation laws. Herein, we explore GNNs to solve the following problem

$$u_t + f(u,\beta)_x = 0 \qquad (*)$$

where $f(u,\beta)$ is the nonlinear flux function of the scalar conservation law, u is the main variable, and β is the physical parameter. The main challenge of nonlinear conservation laws is that solutions typically create shocks. That is, one or several jumps in the form (u_L, u_R) with $u_L \neq u_R$ moving in space and probably changing over time such that information about $f(u)$ in the interval associated with this jump is not present in the observation data. We demonstrate that GNNs could achieve accurate estimates of PDEs solutions based on new initial conditions and physical parameters within a specific parameter range.

*Corresponding Author: qing.li@uis.no
§The authors contributed equally to this work.

1 Introduction

Machine learning methods have been widely used to solve PDEs in science and engineering, for example, aerodynamics [16, 3], electromagnetism [13], geophysics [17] and weather prediction [1], etc. According to [18, 7, 14, 10], GNNs have recently been introduced and made much progress in this area, offering faster runtimes than principled solvers. Compared to grid-based convolutional neural networks(CNNs) [20, 21], GNNs demonstrated better adaptivity in the simulation scenarios [22, 2, 5]. However, most currently published papers use GNNs to resolve PDEs with regular solutions, such as the Wave equation, the Poisson's equation, and the Navier-Stokes equations. These tests have demonstrated that GNNs can resolve these partial differential equations with high efficiency and precision. How would GNNs perform if we try to use it in the context of nonlinear conservation laws?

In this paper, we implement GNN variants to solve PDEs with nonlinear flux function $f(u,\beta)$ that is involved in general scalar nonlinear conservation laws. We restrict to the one-dimensional case given by

$$u_t + f(u,\beta)_x = 0 \qquad (1)$$

where $u = u(x,t)$ is the main variable and β is a parameter of a physical phenomenon. We train GNNs based on observation data $u(x_j, t_i, \beta_k)$ on a spatial grid $x_j,\ j = 1,\dots,N_x$ at specified times $t_i,\ i = 1,\dots,N_{obs}$ with some specific parameters $\beta_k,\ k = 1,\dots,N_\beta$. Using learned GNNs, we pre-

https://doi.org/10.7557/18.6808

dict the solutions of Eq. (1) given new values of parameter β and initial state u_0.

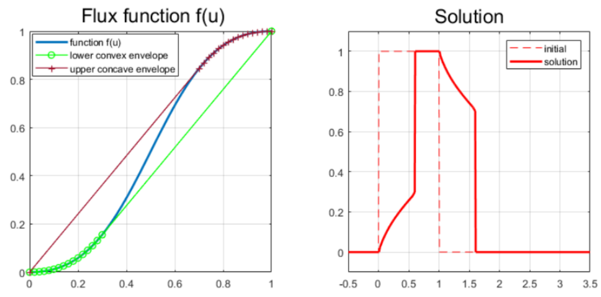
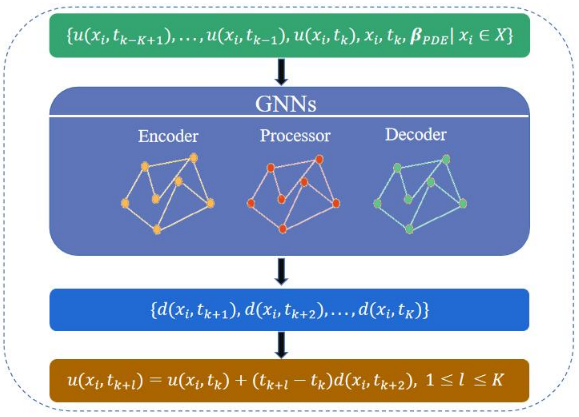


Figure 1: Left: Example of nonlinear flux function $f(u) = \frac{u^2}{u^2+(1-u)^2}$ (blue curve). Upper concave envelope (brown curve) and lower convex envelope (green curve) are also included. Right: The solution of Eq. (1) at time $T = 0.5$ is shown (red solid curve) together with its initial data $u_0(x)$ (red dashed line).

Compared to equations with regular solutions, Eq. (1) has some characteristics that make it more challenging to solve. Typically, Eq. (1) generates shock wave solutions $u(x,t)$ in finite time, i.e., solutions that contain one or more discontinuities expressed as a jump (u_L, u_R) with $u_L \neq u_R$, though the initial value $u_0(x)$ is smooth [8, 6]. In particular, the specific form of $f(u)$ in the interval $[\min(u_L, u_R), \max(u_L, u_R)]$ is not used in the construction of the entropy solution, only the slope $s = \frac{f(u_L)-f(u_R)}{u_L-u_R}$. As jumps arise and disappear in the solution over the period for which observation data is collected, the data may lack information about $f(u)$. This situation is illustrated in Fig. 1. In the left panel, we plot the flux function $f(u) = u^2/(u^2 + (1-u)^2)$. In the right panel, the entropy solution is shown after a time $T = 0.5$. At the time $t = 0$, the initial data $u_0(x)$ involves one jump at $x = 0$ and another at $x = 1$. The initial jump at $x = 0$ is instantly transformed into a solution that is a combination of a continuous wave solution (rarefaction wave) and a discontinuous wave $(u_L, u_R) \approx (0.3, 1.0)$, as dictated by the lower convex envelope shown in the left panel (green curve) [8]. Similarly, the initial jump at $x = 1$ is transformed into a solution that is a combination of a continuous wave solution (rarefaction wave) and a discontinuous wave $(u_L, u_R) \approx (0.7, 0)$, by the upper concave envelope illustrated in the left panel (brown curve) [8]. From this example, we see that we have no observation data that directly can reveal the shape of $f(u)$ in the interval $u \in [0.3, 0.7]$ (approximately).

2 Related Works

Several machine learning approaches are proposed to solve the PDEs. Raissi et al. [15] introduced the physics informed neural networks (PINNs) for solving solutions of PDEs and learning the parameters in PDEs. However, the neural network methods struggle to learn the nonlinear hyperbolic PDE that governs two-phase transport in porous media [4]. They experimentally indicated that this shortcoming of PINNs for hyperbolic PDEs is not due to the specific architecture or to the choice of the hyperparameters, but rather to the lack of regularity in the solution. Long et al. [11, 12] proposed a PDE-Net that combines numerical approximations of differential operators and a symbolic multi-layer neural network. They employed convolutions to approximate differential operators and deep networks to approximate the nonlinear response. However, in our case, $f(u)_x$ can not be written by $f'(u)u_x$ as $f(u)$ is not in general a differentiable function in our problem. Recently, more and more GNNs methods are being used to solve PDE problems. Gao et al. presented a novel discrete PINN framework based on Graph Convolutional Networks (GCNs) and the variational structure of PDEs in [5]. This framework could solve forward and inverse PDEs in a unified manner. Zhao et al. used GNNs in conjunction with autodecoder style priors to tackle PDE-constrained inverse problems [22]. In [2], authors combined GNNs with some classical numerical methods, such as finite differences, finite volumes, and WENO schemes to solve PDEs problems. They experimentally proved that this method has fast, stable, and accurate performance across different domain topologies on various fluid-related flow problems. However, these current works mainly use GNNs on some typical equations with regular solutions. Several complex equations with discontinuous solutions still need to be researched. In a recent work [9], we introduced a framework coined ConsLaw-Net that combines a symbolic multi-layer neural network and an entropy-satisfying discrete scheme to learn the non-



$\{u(x_i, t_{k-K+1}), \ldots, u(x_i, t_{k-1}), u(x_i, t_k), x_i, t_k, \boldsymbol{\beta}_{PDE} | x_i \in X\}$
GNNs
Encoder
Processor
Decoder
$\{d(x_i, t_{k+1}), d(x_i, t_{k+2}), \ldots, d(x_i, t_K)\}$
$u(x_i, t_{k+l}) = u(x_i, t_k) + (t_{k+l} - t_k) d(x_i, t_{k+2}),\ 1 \leq l \leq K$

Figure 2: The pipeline of the approach. For each node c_i, at time point t_k, the last K solutions $(u(x_i, t_{k-K+1}), ..., u(x_i, t_{k-1}), u(x_i, t_k))$, position x_i, current time t_k, and equation parameters $\boldsymbol{\beta}_{PDE}$ are fed into GNNs. After the operators of *Encoder*, *Processor* and *Decoder*, we get the information increment of this node c_i at the next K time points: $(d(x_i, t_{k+1}), d(x_i, t_{k+2}), ..., d(x_i, t_{k+K}))$. Finally, we get solutions of the next K time points by $u(x_i, t_{k+l}) = u(x_i, t_k) + (t_{k+l} - t_k) d(x_i, t_{k+l}), 1 \leq l \leq K$.

linear, unknown flux function $f(u)$ without parameter β.

3 Method

3.1 Graph Neural Networks (GNNs)

In this paper, we leverage GNNs with encoder processor decoder structure to learn a fast-forward model that predicts solutions of PDEs. We model the domain X as a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with node $c_i \in \mathcal{V}$, edges $l_{ij} \in \mathcal{E}$. The features of node c_i are remarked by $\boldsymbol{f}_i \in R^c$ and the edges l_{ij} define local neighborhoods. An overview of the approach that we use is shown in Fig. 2.

Encoder *Encoder* computes node embeddings. For each node c_i, the operator of *Encoder* maps the last K solution values $(u(x_i, t_{k-K+1}), ..., u(x_i, t_{k-1}), u(x_i, t_k))$, node position x_i, current time t_k, and equation parameters $\boldsymbol{\beta}_{PDE}$ to node embedding vector

$$\begin{aligned} \boldsymbol{f}_i = &\boldsymbol{\epsilon}([u(x_i, t_{k-K+1}), ..., u(x_i, t_{k-1}), u(x_i, t_k) \\ &, x_i, t_k, \boldsymbol{\beta}_{PDE}]) \end{aligned} \tag{2}$$

where $\boldsymbol{\epsilon}$ representing the operator of *Encoder*.

Processor *Processor* contains M layers with intermediate graphs $\mathcal{G}^m$, $m = 1, 2, ..., M$. Eq. (3) and Eq. (4) is the messaging of edges and the information update of node c_i, respectively.

- edge $c_j \to c_i$ message:

$$\begin{aligned} \boldsymbol{m}_{ij}^m = \boldsymbol{\phi}(&\boldsymbol{f}_i^m, \boldsymbol{f}_j^m, u(x_i, t_{k-K+1}) - u(x_j, t_{k-K+1}) \\ &, ..., u(x_i, t_k) - u(x_j, t_k), x_i - x_j, \boldsymbol{\beta}_{PDE}) \end{aligned} \tag{3}$$

- node c_i update:

$$\boldsymbol{f}_i^{m+1} = \boldsymbol{\psi}\left(\boldsymbol{f}_i^m, \sum_{j \in N(i)} \boldsymbol{m}_{ij}^m, \boldsymbol{\beta}_{PDE}\right) \tag{4}$$

where $N(i)$ holds the neighbors of node c_i, and $\boldsymbol{\phi}$ and $\boldsymbol{\psi}$ are multilayer perceptrons (MLPs). Using relative positions $x_i - x_j$ can be justified by the translational symmetry of the PDEs we consider. Solution differences $u_i - u_j$ make sense by thinking of the message passing as a local difference operator, like a numerical derivative operator.

Decoder After *Processor*, we use a shallow convolutional network to output the K next timesteps information increment at grid point $x_i \in X$. The result is a new vector $\boldsymbol{d}_i = (d(x_i, t_{k+1}), d(x_i, t_{k+2}), ..., d(x_i, t_{k+K}))$ with each element $d(x_i, t_{k+l}), 1 \leq l \leq K$ corresponding to different time point t_{k+l}.

Readout After GNNs with the structure of encoder processor decoder, we get solutions of the next K time points by

$$u(x_i, t_{k+l}) = u(x_i, t_k) + (t_{k+l} - t_k) d(x_i, t_{k+l}) \tag{5}$$

where $1 \leq l \leq K$.

3.2 Data Generator

We investigate a discretization of the spatial domain $[0, L]$ in terms of $\{x_i\}_{i=0}^{N_x - 1}$ where $x_i = (1/2 + i)\Delta x$ for $i = 0, \ldots, N_x - 1$ with $\Delta x = L/N_x$.

Algorithm 1: CFL

```
Input: $L$: length of the spatial domain; $N_x$: the number of spatial grid cells;
       $f(u)$: the nonlinear flux function; $T$: computational time period;
Output: $\Delta t$: local time interval

Function CFL($L, N_x, f(u), T$):
    $\Delta x = L/N_x$
    $M = \max_u |f'(u)|$
    $dt = (\frac{3}{4}\Delta x)/(M + 0.0001)$
    $n\_time = \lfloor T/dt \rfloor$
    $\Delta t = T/n\_time$
    return $\Delta t$;
End Function
```

Furthermore, we consider time lines $\{t^n\}_{n=0}^{N_t}$ with $N_t\Delta t = T$. The discretization of Eq. (1) is based on the Rusanov scheme [8] which is expressed as

$$u_j^{n+1} = u_j^n - \lambda(F_{j+1/2}^n - F_{j-1/2}^n), \qquad \lambda = \frac{\Delta t}{\Delta x},$$
$$u_1^{n+1} = u_2^{n+1}, \qquad u_{N_x}^{n+1} = u_{N_x-1}^{n+1} \tag{6}$$

with $j = 2, \ldots, N_x - 1$ and the Rusanov flux takes the form

$$F_{j+1/2}^n = \frac{f(u_j^n) + f(u_{j+1}^n)}{2} - \frac{\max\{|f'(u_j^n)|, |f'(u_{j+1}^n)|\}}{2}(u_{j+1}^n - u_j^n).$$

The Courant–Friedrichs–Lewy(CFL) condition [8] determines the magnitude of Δt for a given Δx. We detail the CFL condition in Algorithm 1. We illustrate how to learn the solution $U = \{u(x_j, t^n)\}$ of the discrete conservation law Eq. (6) in Algorithm 2.

Algorithm 2: DataGenerator

```
Input: $T$: computational time period; $N_x$: the number of spatial grid cells; $L$:
       length of the spatial domain;
       $u_0 = \{u_0(x_j)\}_{j=1}^{N_x}$: initial state set of dimension $N_x$; $f(u)$: the flux
       function;
Output: $U = \{u_j^n\}$: the solution based on initial state $u_0$;

$\Delta t$ = CFL($L$, $N_x$, $f(u)$, $T$)
$\Delta x = L/N_x$
$U[0] = u_0$
$\tilde{u} = u_0$
for n = 1,...,$T/\Delta t$ do
    for j = 1,...,$N_x$ - 1 do
        $F_{j+1/2} = \frac{1}{2}\left(f(\tilde{u}_j) + f(\tilde{u}_{j+1})\right) - \frac{\max\{|f'(\tilde{u}_j)|,|f'(\tilde{u}_{j+1})|\}}{2}\left(\tilde{u}_{j+1} - \tilde{u}_j\right)$
    end
    for j = 2,...,$N_x$ - 1 do
        $u_j = \tilde{u}_j - \frac{\Delta t}{\Delta x}\left(F_{j+1/2} - F_{j-1/2}\right)$
    end
    $u_1 = u_2$
    $u_{N_x} = u_{N_x-1}$
    $\tilde{u} = u$
    $U[i] = u$
end
```

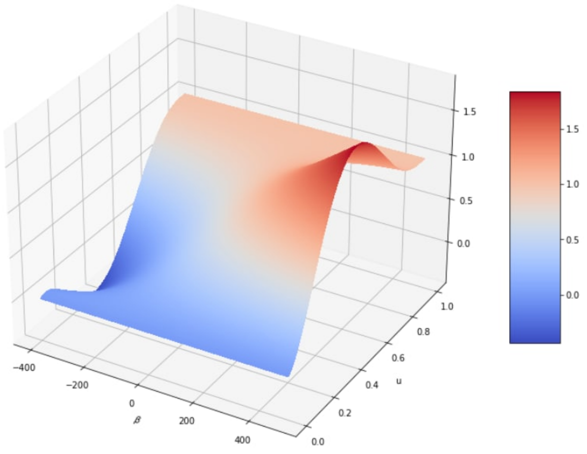
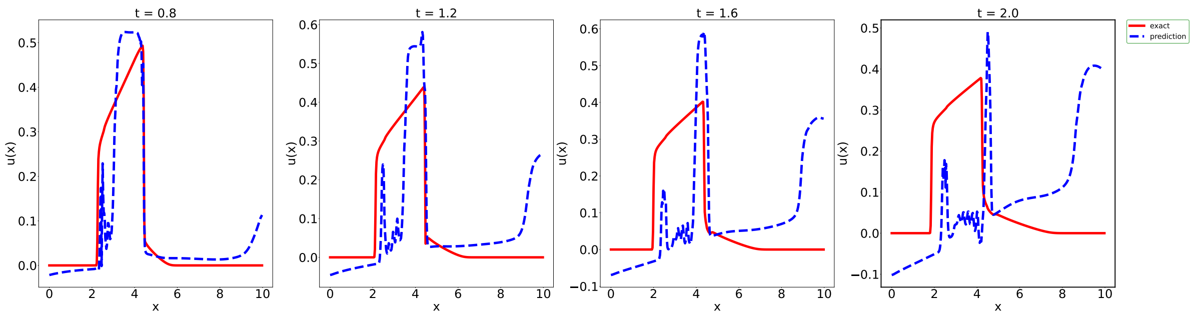


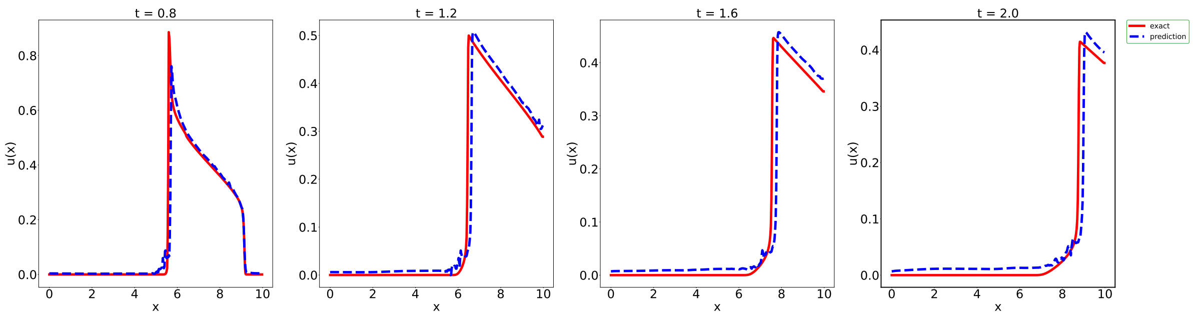
Figure 3: Flux function Eq. (7).

4 Experiments

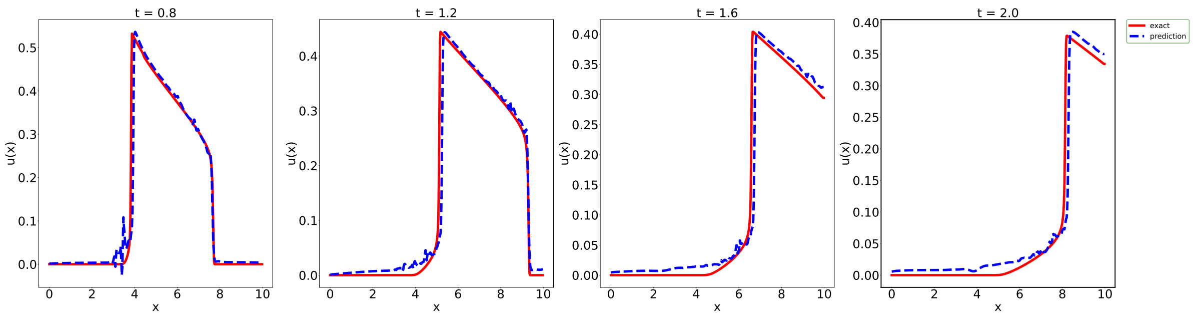
Experiment Setup
In this section, we study a class of nonlinear conservation laws that are naturally from the problems where one fluid is displaced by another fluid in a vertical domain. The displacement process involves a balance between buoyancy and viscous forces. Depending on the properties of the used fluids, there could be various displacement processes. One can derive a family of flux functions $f(u, \beta)$ in



(a) $\beta = -252$ in flux function (7). Solutions of (1) based on initial state $u_0 = \begin{cases} 0.5, & \text{if } x \in [2.575, 4.3] \\ 0.0, & \text{otherwise} \end{cases}$.



(b) $\beta = 264$ in flux function (7). Solutions of (1) based on initial state $u_0 = \begin{cases} 0.88, & \text{if } x \in [4.675, 6.4] \\ 0.0, & \text{otherwise} \end{cases}$.



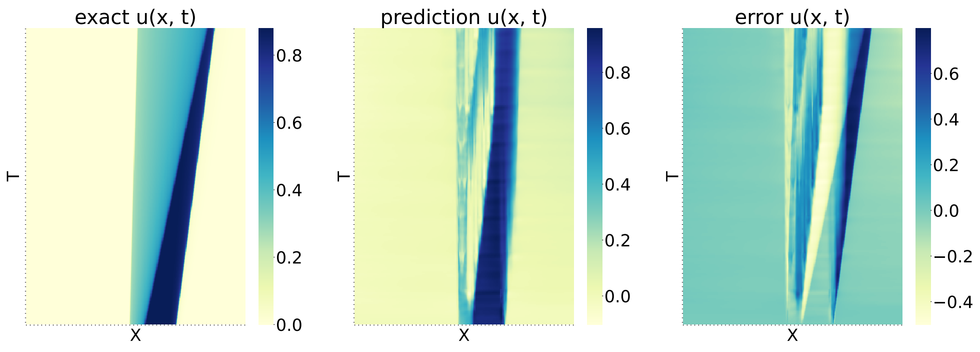
(c) $\beta = 360$ in flux function (7). Solutions of (1) based on initial state $u_0 = \begin{cases} 0.86, & \text{if } x \in [2.675, 4.4] \\ 0.0, & \text{otherwise} \end{cases}$.

Figure 4: Solutions of (1) based on different initial states and values of β at time point $t = 0.8, 1.2, 1.6$ and 2.0, respectively. In each subplot, the solid red line is the true solution obtained by Algorithm 2, and the blue dashed line is the solution predicted by the trained GNNs.

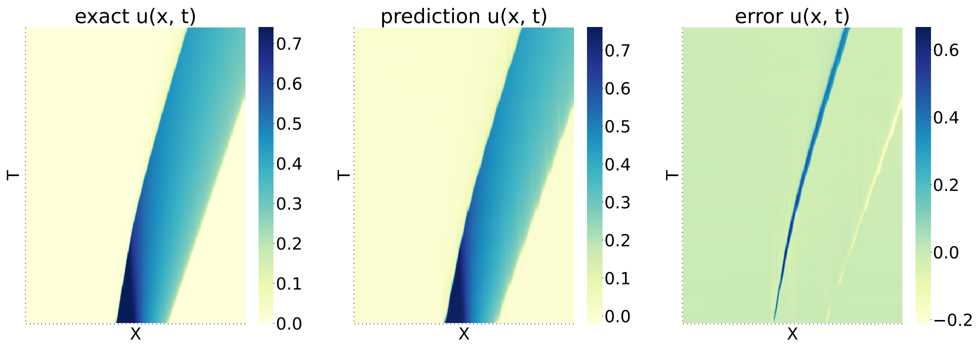
Eq. (1) which takes the form [19]

$$f(u,\beta) = \frac{1}{2}u(3-u^2)+\frac{\beta}{12}u^2\left(\frac{3}{4} - 2u + \frac{3}{2}u^2 - \frac{1}{4}u^4\right), \tag{7}$$

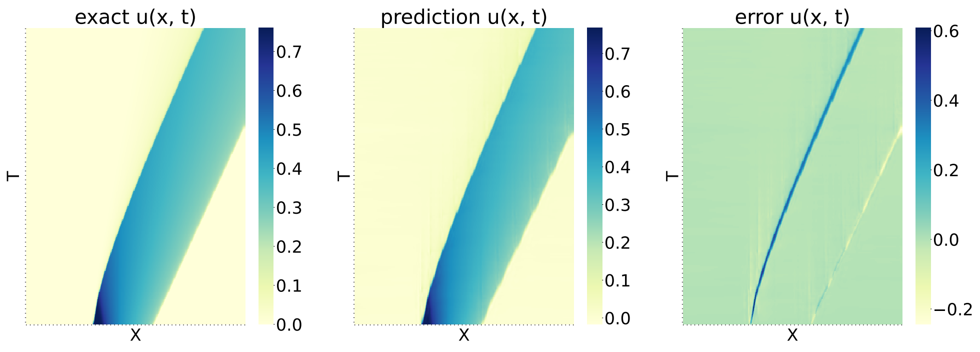
where the parameter $\beta \in (-400, 400)$ represents the balance between gravity (bouyancy) and viscous forces. We study the solution of Eq. (1) at $t \in [0, 2]$ in $x \in [0, 10]$. As shown in Fig. 3, different values of β result in different types of flux



(a) $\beta = -172$ in flux function (7). Solutions of (1) based on initial state $u_0 = \begin{cases} 0.88, & \text{if } x \in [4.675, 6.4] \\ 0.0, & \text{otherwise} \end{cases}$.



(b) $\beta = 192$ in flux function (7). Solutions of (1) based on initial state $u_0 = \begin{cases} 0.74, & \text{if } x \in [3.55, 5.275] \\ 0.0, & \text{otherwise} \end{cases}$.



(c) $\beta = 336$ in flux function (7). Solutions of (1) based on initial state $u_0 = \begin{cases} 0.76, & \text{if } x \in [2.475, 4.2] \\ 0.0, & \text{otherwise} \end{cases}$.

Figure 5: Solutions of (1) based on different initial states and values of β at time $t \in [0.0, 2.0]$. The left subplot is the real solutions of Eq. (1). The middle subplot is the solutions generated by GNNs, and the right subplot is the error of real and predicted $u(x, t)$.

functions.

We use Algorithm 2 to generate 500 samples with different values of β and initial states. The values of β are randomly selected from $(0, 300)$ and initial states are from $[0.0, 1.0]$. We consider the observation in terms of x-dependent data at fixed times $\{t_i^*\}_{i=1}^{N_{\text{obs}}}$ extracted from the solution U as follows:

$$U_{\text{sub}} = \left\{ u(x_j, t_1^*), u(x_j, t_2^*), \ldots, u(x_j, t_{N_{\text{obs}}}^*) \right\} \quad (8)$$

where $j = 1, \ldots, N_x$, $N_x = 400$ with $\Delta x = 0.025$ and $N_{\text{obs}} = 250$ with $\Delta t^{\text{obs}} = 0.008$.

For training, we split U_{sub} into five parts of length 50 in terms of time. Values of $u(x)$ for the first 25 time points are used to predict $u(x)$ for the following 25 time points, i.e., $K = 25$ in Eq. (5). In the testing, we use the first stage, which contains 25 time points of $u(x)$ to predict the solutions of the second stage, and then predict $u(x)$ at the third phase based on the predicted value of the second stage. This process is repeated until $T = 2$.

Results and discussions

The total number of trainable parameters is 0.28M. We set *epoch* $= 30$ and divide the 100 test samples into three groups according to β, (i)G_1: 50 examples with $\beta \in (-400, 0)$, (ii)G_2: 30 examples with $\beta \in (0, 300)$,(iii)G_3: 20 examples with $\beta \in (300, 400)$.

We randomly select one sample $\beta = -252, 264, 360$ from each group. Fig. 4 displays the exact and predicted $u(x, t)$ of these selected examples with $x \in [0, 10]$ at time $t = 0.8, 1.2, 1.6, 2.0$. All the examples show that the error between exact and predicted values of $u(x)$ increases with time. Notably, the model cannot predict the underlying trend at $t = 1.6$ and $t = 2.0$ on the case selected from G_1. While, it performs very well on G_2 and G_3. The result is also reflected in Table 1 that counts MSE of G_1, G_2, G_3. The MSE of G_1 is 10 times higher than the MSE of G_2 and G_3.

Fig. 5 shows another set of samples from G_1, G_2, G_3 with $x \in [0, 10)$ and $t \in [0.4, 2.0]$. When the solution fluctuates, the model is prone to large prediction errors. This phenomenon is also reflected in Fig. 4. This phenomenon is probably related to the discontinuity of the solutions of 1.

Experimental results show that the model can predict more accurately when β is within a specific range. However, when β deviates too much from that used for the training model, the model's predictive power is significantly reduced.

Group	G_1	G_2	G_3
MSE	0.0268	0.004	0.0015

Table 1: The value of MSE for each group. We calculate MSE of the true and predicted $u(x, t)$ of all samples with $x \in [0, 10]$ and $t \in [0.4, 2.0]$in each group.

5 Conclusion

In this work, we apply GNNs to solve PDEs, which are unique since their solutions usually contain shocks. Our experiments demonstrate that the GNN models can predict accurately when the parameter β is within a specific range. However, when the parameter deviates far from the value for training, the model's predictive performance drops sharply. Furthermore, the model is not very good at predicting the discontinuity of the solutions. There will be numerous small fluctuations around the discontinuity. One of our future work directions is to explore how to improve GNN's behavior when handling discontinuous solutions or develop new methods to deal with PDEs containing shocks.

References

[1] P. Bauer, A. Thorpe, and G. Brunet. The quiet revolution of numerical weather prediction. *Nature*, 525(7567):47–55, 2015. doi: https://doi.org/10.1038/nature14956.

[2] J. Brandstetter, D. E. Worrall, and M. Welling. Message passing neural PDE solvers. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. doi: https://doi.org/10.48550/arXiv.2202.03376. URL `https://openreview.net/forum?id=vSix3HPYKSU`.

[3] T. D. Economon, F. Palacios, S. R. Copeland, T. W. Lukaczyk, and J. J. Alonso. Su2:

An open-source suite for multiphysics simulation and design. *Aiaa Journal*, 54(3):828–846, 2016. doi: https://doi.org/10.2514/1.J053813.

[4] O. Fuks and H. A. Tchelepi. Limitations of physics informed machine learning for nonlinear two-phase transport in porous media. *Journal of Machine Learning for Modeling and Computing*, 1(1), 2020. doi: 10.1615/JMachLearnModelComput.2020033905.

[5] H. Gao, M. J. Zahr, and J.-X. Wang. Physics-informed graph neural galerkin networks: A unified framework for solving pde-governed forward and inverse problems. *Computer Methods in Applied Mechanics and Engineering*, 390:114502, 2022. doi: https://doi.org/10.1016/j.cma.2021.114502.

[6] H. Holden and N. Risebro. Front tracking for hyperbolic conservation laws. *Springer, Berlin*, 2011.

[7] V. Iakovlev, M. Heinonen, and H. Lähdesmäki. Learning continuous-time pdes from sparse data with graph neural networks. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. doi: https://doi.org/10.48550/arXiv.2006.08956. URL `https://openreview.net/forum?id=aUX5Plaq7Oy`.

[8] R. LeVeque. Finite volume methods for hyperbolic problems. *Cambridge Texts in Applied Mathematics*, 2007. doi: https://doi.org/10.1017/CBO9780511791253.

[9] Q. Li and S. Evje. Learning the nonlinear flux function of a hidden scalar conservation law from data. *Network Heterogeneous Media*, 18, 2023. doi: 10.3934/nhm.2023003.

[10] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Neural operator: Graph kernel network for partial differential equations. *arXiv preprint arXiv:2003.03485*, 2020.

[11] Z. Long, Y. Lu, X. Ma, and B. Dong. Pde-net: Learning PDEs from data. In *International Conference on Machine Learning*, pages 3208–3216. PMLR, 2018. doi: https://doi.org/10.48550/arXiv.1710.09668.

[12] Z. Long, Y. Lu, and B. Dong. PDE-Net 2.0: Learning PDEs from data with a numeric-symbolic hybrid deep network. *Journal of Computational Physics*, 399:108925, 2019. doi: https://doi.org/10.1016/j.jcp.2019.108925.

[13] D. Pardo, L. Demkowicz, C. Torres-Verdin, and M. Paszynski. A self-adaptive goal-oriented hp-finite element method with electromagnetic applications. part ii: Electrodynamics. *Computer methods in applied mechanics and engineering*, 196(37-40):3585–3597, 2007. doi: https://doi.org/10.1016/j.cma.2006.10.016.

[14] T. Pfaff, M. Fortunato, A. Sanchez-Gonzalez, and P. W. Battaglia. Learning mesh-based simulation with graph networks. *arXiv preprint arXiv:2010.03409*, 2020. doi: https://doi.org/10.48550/arXiv.2010.03409.

[15] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse prvoblems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019. doi: https://doi.org/10.1016/j.jcp.2018.10.045.

[16] R. Ramamurti and W. Sandberg. Simulation of flow about flapping airfoils using finite element incompressible flow solver. *AIAA journal*, 39(2):253–260, 2001. doi: https://doi.org/10.2514/2.1320.

[17] C. Schwarzbach, R.-U. Börner, and K. Spitzer. Three-dimensional adaptive higher order finite element simulation for geo-electromagnetics—a marine csem example. *Geophysical Journal International*, 187(1):63–74, 2011. doi: https://doi.org/10.1111/j.1365-246X.2011.05127.x.

[18] S. Seo, C. Meng, and Y. Liu. Physics-aware difference graph networks for sparsely-observed dynamics. In *International Conference on Learning Representations*, 2019.

[19] H. Skadsem and S. Kragset. A numerical study of density-unstable reverse circulation displacement for primary cementing. *J. Energy Resour. Technol*, 144, 2022. doi: https://doi.org/10.1115/1.4054367.

[20] N. Thuerey, K. Weißenow, L. Prantl, and X. Hu. Deep learning methods for reynolds-averaged navier–stokes simulations of airfoil flows. *AIAA Journal*, 58(1):25–36, 2020. doi: https://doi.org/10.2514/1.j058291.

[21] N. Wandel, M. Weinmann, and R. Klein. Learning incompressible fluid dynamics from scratch - towards fast, differentiable fluid models that generalize. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. doi: https://doi.org/10.48550/arXiv.2006.08762. URL `https://openreview.net/forum?id=KUDUoRsEphu`.

[22] Q. Zhao, D. B. Lindell, and G. Wetzstein. Learning to solve pde-constrained inverse problems with graph networks. In K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvári, G. Niu, and S. Sabato, editors, *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 26895–26910. PMLR, 2022. doi: https://doi.org/10.48550/arXiv.2206.00711. URL `https://proceedings.mlr.press/v162/zhao22d.html`.