PhyGNNet: Solving spatiotemporal PDEs with Physics-informed Graph Neural Network

Longxiang Jiang, Liyuan Wang, Xinkun Chu, Yonghao Xiao and Hao Zhang*

Institute of Computer Application, China Academy of Engineering Physics, Mianyang 621900, China

ARTICLE INFO

Keywords:
Physics-informed neural networks
Partial differential equation
Graph neural networks
Deep learning

ABSTRACT

Solving partial differential equations (PDEs) is an important research means in the fields of physics, biology, and chemistry. As an approximate alternative to numerical methods, PINN has received extensive attention and played an important role in many fields. However, PINN uses a fully connected network as its model, which has limited fitting ability and limited extrapolation ability in both time and space. In this paper, we propose PhyGNNet for solving partial differential equations on the basics of a graph neural network which consists of encoder, processer, and decoder blocks. In particular, we divide the computing area into regular grids, define partial differential operators on the grids, then construct pde loss for the network to optimize to build PhyGNNet model. What's more, we conduct comparative experiments on Burgers equation and heat equation to validate our approach, the results show that our method has better fit ability and extrapolation ability both in time and spatial areas compared with PINN.

1. Introduction

With the development of physics, biology, chemistry, and other fields, a large number of partial differential equations [17] (PDEs) have been accumulated in related fields, and the evolution process and results of specific problems can be obtained by solving PDEs. However, solving PDEs is a challenging task, except for a few equations which exist analytical solutions, most of the equations are solved numerically [18]. Numerical methods are highly specialized, building numerical solutions have a very high threshold for users in non-related fields, it often takes a lot of time to derive solution methods and build calculation tools, which is not friendly enough. Along with the rapid progress and popularization of deep learning, a series of methods has emerged which solves PDEs based on the neural network by learning mapping functions of input and output. These methods can be divided into traditional data-driven methods [20, 19, 6], and methods that only rely on PDEs but no data [21, 22]. In practice, the acquisition of data often requires experimental mensuration or numerical solution of PDEs, which is cumbersome. Combining neural networks and PDEs to build solving tools is a popular research field nowadays.

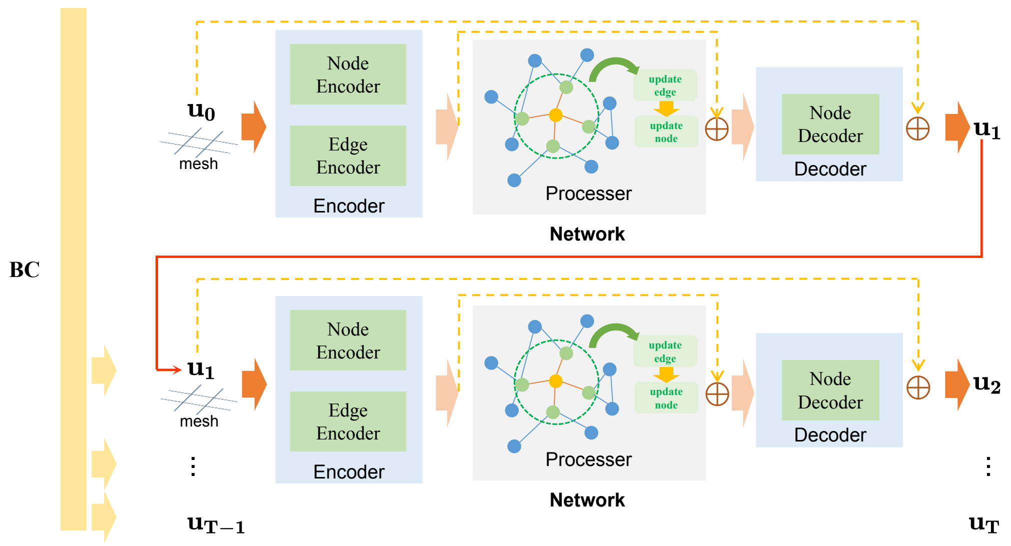
The existing mainstream method is PINN [23], which relies on the automatic differentiation tools provided by the neural network framework to embed PDEs into neural networks via minimizing loss function, which has been widely used in solving problems such as fluid flow [24] and heat conduction [25]. However, there exists two limitations in PINN. Firstly, to calculate the higher-order differential, the automatic differential needs to save the differential calculation graph, which consumes a lot of storage space and calculation time during training, due to the limitation of hardware resources, the network structure of PINN is usually very concise, thus fully connected network is mainly adopted, which limits the fitting accuracy of PINN. Secondly, the PINN directly learns the function mapping of spatial coordinates and time to solution and does not pay attention to the locality and timing of the physical evolution process, resulting in the lack of time and space extrapolation ability of the trained model.

In addition to the PINN method that relies on automatic differentiation, there are also computation methods of differential that adopt a predefined discrete format. The method in [1] regards a convolutional kernel with fixed weights as the differentiation operator and constructs a convolutional neural network (CNN) to fit PDEs, which converts the spatial differential calculation into a convolutional calculation. On the basis of [1], to solve the time-dependent PDE, [2] proposes a method that approximates the time derivative by backward difference and introduces LSTM [3] to deal with the series information. Furthermore, with FV discretization scheme and two-point flux approximation [5], [4] applies CNN to solve transient Darcy flows.

It is well known that the significant difference between FNN and CNN is that convolutional operations are spatially localized, and training CNN enables the network to learn spatially localized evolution, which is consistent with physical processes and has better interpretability. In addition to CNNs, graph neural networks (GNN) [9, 10] are also spatially localized. GNN operates on graph data that learns local computation based on the message passing mechanism of nearest neighboring nodes or connected edges. Works in [6, 7, 8] indicate that GNN has the desirable fitting ability and generalization performance in modeling the evolution of physical processes.

In this paper, we address the problem of solving spatiotemporal PDEs and propose a method to solve PDEs with GNN. Specifically, we divide the computational region of the PDE into a regular mesh, then construct spatial differentiation on the grid of the mesh and time derivative with backward difference, following, we regard the mesh as a graph

*Corresponding author
jianglx@whu.edu.cn (L. Jiang); linusec@163.com (H. Zhang)
ORCID(s):



🟥	detach
🟨	residual connection

$$R(\mathbf{u}^{\theta}_{t+\Delta t};\mathbf{u}_t,\theta)=\frac{\mathbf{u}^{\theta}_{t+\Delta t}-\mathbf{u}_t}{\Delta t}+\mathcal{N}[\mathbf{u}^{\theta}_{t+\Delta t},\cdots,\nabla\mathbf{u}^{\theta}_{t+\Delta t},\Delta\mathbf{u}^{\theta}_{t+\Delta t},\nabla\mathbf{u}^{\theta}_{t+\Delta t}\cdot\mathbf{u}^{\theta}_{t+\Delta t},\cdots;\boldsymbol{\lambda}]$$

$$L(\mathbf{u}^{\theta}_{t+\Delta t};\mathbf{u}_t,\theta)=\frac{1}{N}\sum_{s=1}^{T}||R(\mathbf{u}^{\theta}_{t+\Delta t};\mathbf{u}_t,\theta)||_2$$

Figure 1: The framework of our proposed method. Given initial and boundary conditions, we predict the solutions of multi time steps with the network consists of encoder, processer and decoder blocks, where processer is a graph neural network.

and organize pde loss to train GNN to make the network have the ability of inference solutions that satisfies the PDE equation. Moreover, we conduct several experiments on burgers and heat equations, which indicates that our approach has better fit ability and extrapolates well on both spatial area and time than PINN.

2. Method

The general form of PDE can be expressed as:

$$\frac{\partial \mathbf{u}}{\partial t}+\mathcal{N}\left[\mathbf{u},\cdots,\nabla\mathbf{u},\Delta\mathbf{u},\nabla\mathbf{u}\cdot\mathbf{u},\cdots;\lambda\right]=\mathbf{0} \tag{1}$$

where, $\mathbf{u}\in\mathbb{R}^{N\times d}$ denotes solutions which have d-dims in area Ω on N mesh grid points, $\frac{\partial \mathbf{u}}{\partial t}$ is time derivative, $\nabla\mathbf{u}$ represents the gradient in space. $\Delta\mathbf{u}$ is Laplace item, which is euqal to $\nabla^2\mathbf{u}$. λ is parameters of the PDE. In addition, the Initial Condition (IC) and Boundary Condition (BC) have the following definitions:

$$\mathcal{I}(\mathbf{u},\nabla\mathbf{u},\Delta\mathbf{u},\cdots;t=0)=\mathbf{0} \tag{2}$$

$$\mathcal{B}(\mathbf{u},\nabla\mathbf{u},\Delta\mathbf{u},\cdots;\mathbf{x}\in\partial\Omega)=\mathbf{0} \tag{3}$$

where, $\partial\Omega$ denotes the boundary area of Ω.

Given the above PDE equation and Initial/Boundary Conditions, a solution can be found with several approaches. However, In this paper, we aim to solve the problem with GNN. Similar to PINN, we yearn to devise a method that is unsupervised to solve PDEs.

The framework of our approach to solve the above equation is illustrated in Fig.1. We first generate a regular mesh and express the mesh as a graph, then assign the solution in time step s to the nodes of the graph, note that at the very first time step, the solution is the IC. The network takes the graph and transforms the node attributions and edges into features, then maps the features into solutions for the next time step, the procedure is detailed in Sec.2.1. When training, the BC is assigned to the predicted solutions to compute PDE loss, the solution is then detached from the calculation graph and fed into the network to predict solutions of all T time steps by repeating the above procedure, and the losses of T steps are cumulated to update the parameters of the network at once. The above process is repeated many times until the specified number of times is reached or the network converges to get the final model.

2.1. Network

We regard the mesh with the grid as an undirected graph to train the network. That is, we express the grid points as graph nodes, and assign edges to nodes that are nearest neighbors to each other. Similar to [6], the features of nodes consist of the current solution and 2-dimensional one-hot code of node type, indicating whether a specific node is located on $\partial\Omega$. The features of edges contain euclidean distance and the coordinate difference between sender and receiver nodes.

The framework of our network is shown in Fig.1. The network put the solution at time t to obtain solution at next time step $t+\Delta t$. It mainly has three parts. The encoder transforms node and edges features mentioned above with MLP, the processer predicts latent feature variation of nodes via Graph Network [9] (GN) and the decoder decodes node features with MLP as correction of the input $\mathbf{u}_t$ to create final predicts. The dotted lines in the figure represent residual connection.

Specifically, the calculation process of GN in the framework contains edges update and nodes update steps. To illustrate the procedure, here, we define the edge $\mathbf{e}_{i,j}$ connected to two nodes with features $\mathbf{v}_i$ and $\mathbf{v}_j$ respectively. The GN conduct edges update step at first, which can be described as:

$$\tilde{\mathbf{e}}_{i,j} = f_e(\mathbf{e}_{i,j}||\mathbf{v}_i||\mathbf{v}_j) \tag{4}$$

where $||$ denotes concatenating operator and f_e denotes MLP for edges. With the updated edge features, GN then update node features as:

$$\tilde{\mathbf{v}}_i = f_n(\mathbf{v}_i|| \sum_{j\in\mathcal{N}(i)} \tilde{\mathbf{e}}_{i,j}) \tag{5}$$

and, f_n denotes MLP for nodes.

2.2. Discrete format

The above PDE equation consists of three derivative operators, namely is $\mathbf{u}_t$, $\nabla\mathbf{u}$ and $\Delta\mathbf{u}$. The three operators can be approximated by the following discrete format.

For operator $\mathbf{u}_t$, it can be approximated with backward difference on time t, denoted as:

$$\frac{\partial\mathbf{u}}{\partial t} = \frac{\mathbf{u}_{t+\Delta t} - \mathbf{u}_t}{\Delta t} \tag{6}$$

where Δt denotes the single time iterval in the time evolution process, is a superparameter.

Operators $\nabla\mathbf{u}$ and $\Delta\mathbf{u}$ are differential on spatial area, when the area Ω is divided into uniform areas, the operators can be defined on the grid as well. For simplicity, we only take two-dimension as an example to illustrate the solution method of the operator, for others, the same. For $\mathbf{x} \notin \partial\Omega$, the gradients can be defined by the nearest neighbor grid points on the grid as follows:

$$\frac{\partial\mathbf{u}_{i,j}}{\partial x} = \frac{\mathbf{u}_{i+1,j} - \mathbf{u}_{i-1,j}}{2\Delta x} \tag{7}$$

$$\frac{\partial\mathbf{u}_{i,j}}{\partial y} = \frac{\mathbf{u}_{i,j+1} - \mathbf{u}_{i,j-1}}{2\Delta y} \tag{8}$$

In the equation, Δx and Δy denote the grid point distance. And the Laplace operator has the following form:

$$\Delta\mathbf{u}_{i,j} = \frac{\mathbf{u}_{i+1,j} + \mathbf{u}_{i-1,j} - 2\mathbf{u}_{i,j}}{(\Delta x)^2} + \frac{\mathbf{u}_{i,j+1} + \mathbf{u}_{i,j-1} - 2\mathbf{u}_{i,j}}{(\Delta y)^2} \tag{9}$$

Here, we follow the bias that the grid is regular thus $\Delta y = \Delta x$, and the above equation be been rewritten as:

$$\begin{aligned}\Delta\mathbf{u}_i &= \frac{\mathbf{u}_{i+1,j} + \mathbf{u}_{i-1,j} + \mathbf{u}_{i,j+1} + \mathbf{u}_{i,j-1} - 4\mathbf{u}_{i,j}}{(\Delta x)^2} \\ &= \sum_{\tilde{\mathbf{u}}\in\mathcal{N}(\mathbf{u}_{i,j})} \frac{1}{(\Delta x)^2} \times (\tilde{\mathbf{u}} - \mathbf{u}_{i,j}) && (10) \\ &= \sum_{j=1}^{N} \frac{\mathbf{A}_{i,j}}{(\Delta x)^2} \times (\tilde{\mathbf{u}} - \mathbf{u}_{i,j}) && (11) \\ &= \frac{1}{(\Delta x)^2}[(\mathbf{A} - \mathbf{D})\mathbf{u}]_i && (12) \\ &= -\frac{1}{(\Delta x)^2}[\mathbf{L}\mathbf{u}]_i && (13)\end{aligned}$$

where, $\mathcal{N}(\mathbf{u}_{i,j})$ denotes the nearest neighbors of $\mathbf{u}_{i,j}$. $\mathbf{A} \in \{0,1\}^{N\times N}$ denotes the adjacent matrix, $\mathbf{A}_{i,j} = 1$ if grid point j is one of the nearest neighbors. Matrix $\mathbf{D}$ is the diagonal degree matrix of $\mathbf{A}$, and $\mathbf{L}$ is laplacian matrix [11].

2.3. PDE loss construction

The objective of our approach is to solve PDE on a regular grid with GNN when the PDE equation, IC, and BC are explicitly presented. In PINN, the PDE equation, IC, and BC are softly satisfied when minimizing a multi-objective loss function. However, the loss function requires tuning the weight parameters of multiple losses carefully to avoid falling into local minima. In this paper, we construct loss function with the same methodology as [2], given $\mathbf{u}_t$, the network predicts $\mathbf{u}^{\theta}_{t+\Delta t}$, the loss value of grid points has the following form:

$$\begin{aligned}&R(\mathbf{u}^{\theta}_{t+\Delta t}; \mathbf{u}_t, \theta) && (14)\\ &= \frac{\mathbf{u}^{\theta}_{t+\Delta t} - \mathbf{u}_t}{\Delta t} \\ &+\mathcal{N}\left[\mathbf{u}^{\theta}_{t+\Delta t}, \cdots, \nabla\mathbf{u}^{\theta}_{t+\Delta t}, \Delta\mathbf{u}^{\theta}_{t+\Delta t}, \nabla\mathbf{u}^{\theta}_{t+\Delta t}\cdot\mathbf{u}^{\theta}_{t+\Delta t}, \cdots; \lambda\right]\end{aligned}$$

and the loss function to optimize is:

$$L(\mathbf{u}^{\theta}_{t+\Delta t}; \mathbf{u}_t, \theta) = \frac{1}{N}\sum_{s=1}^{T} ||R(\mathbf{u}^{\theta}_{t+\Delta t}; \mathbf{u}_t, \theta)||_2 \tag{15}$$

Notice that when training, we update the network parameters by cumulating gradients of multi time-steps. For each time step, the input is the network prediction of the last time step, and the IC is as input at the very first step of the network. Also, similar to [2], the BC is hard assigned when organizing loss. In detail, the values of boundary nodes in prediction are assigned according to the BC the ahead of PDE loss computing, that is, by minimizing the PDE loss, the internal nodes changes along with the variations of boundary value in the term of time.

3. Experiments

In this section, we conduct several numerical experiments on two typical PDEs to evaluate our proposed method. One is burgers equation [14] to solve the propagation and reflection of waves. Another is heat equation [12, 13] to solve heat diffusion problem as an example.

3.1. Setup

As aforementioned, there are MLPs in the encoder, processor, and decoder of the neural network. In our experiments, the MLPs are with two hidden layers, each with 64

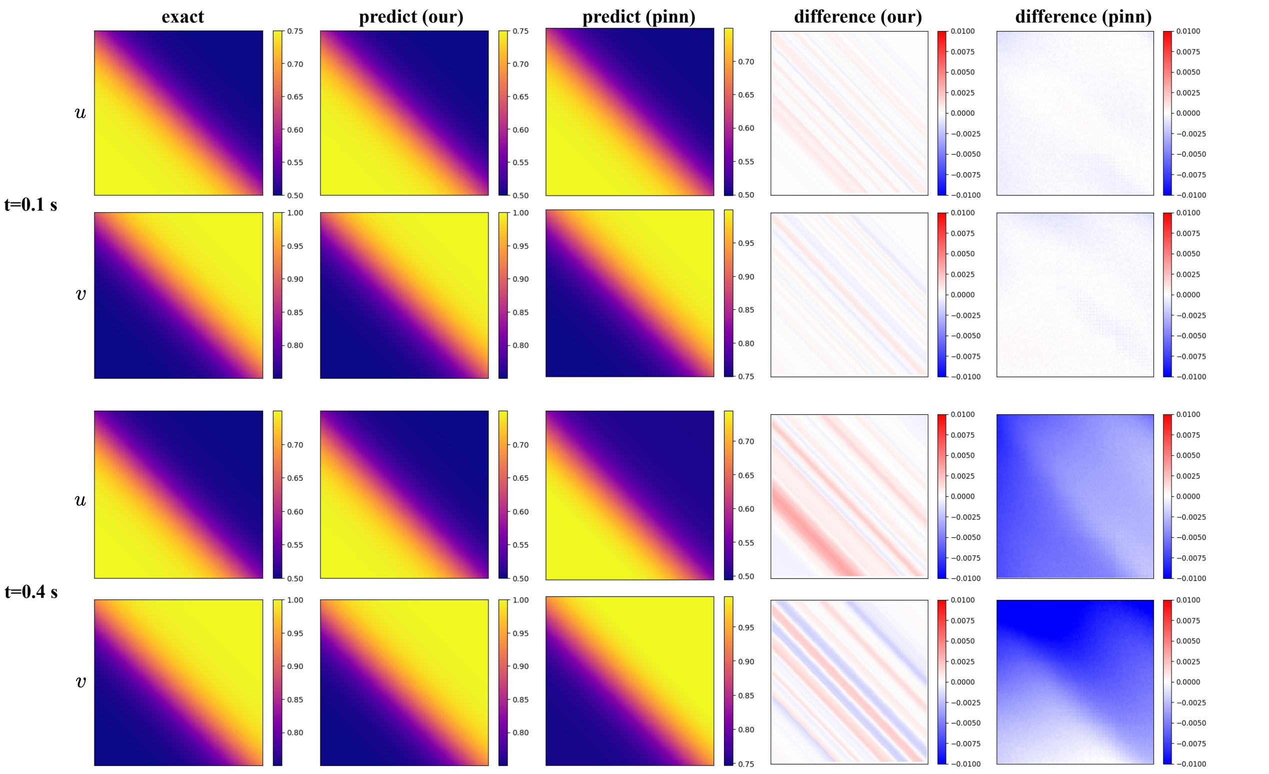


Figure 2: The reuslts of burgers equation with u, v components at different time steps. The predicted results are compared with the exact analytical solutions and the difference is also presented.

neurons, and the ReLU activation function is applied to transform the output of the input layer and hidden layer. For the variable parameters in discrete format, the time interval Δt is set to 0.001, Δx and Δy is set to 0.01 with $x, y \in [0, 1]$ and there are 10000 nodes in total. The learning rate of our method is set to 1×10^{-3} if is not mentioned.

3.2. Burgers Equation

Here, we consider the two-dimensional burgers equation as example, which has the following form:

$$\begin{aligned} u_t + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} - \frac{1}{R}\Delta u = 0 \\ v_t + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} - \frac{1}{R}\Delta v = 0 \end{aligned} \tag{16}$$

where the parameter R is Reynolds number, which controls the wave dynamics. Specifically, In order to verify the accuracy of our method, we choose to solve the equation with exact solution same as [14], that is, the solution is shown as below:

$$\begin{aligned} u(x, y, t) &= \frac{3}{4} - \frac{1}{4\left(1 + e^{(R(-t-4x+4y))/32}\right)} \\ v(x, y, t) &= \frac{3}{4} + \frac{1}{4\left(1 + e^{(R(-t-4x+4y))/32}\right)} \end{aligned} \tag{17}$$

The IC is the exact solution at time $t = 0$ and the boundary value changes along with time t based on the solution. The parameter R is set to 80. We train the network to solve the pde at the first 10 time steps and expect the model to have the ability to infer the subsequent solutions. In detail, we set $T = 10$ at this situation and repeat the train process 10000 epochs, taking the model with minimal pde loss to evaluate performance.

In addition, based on DeepXDE [15] framework, we construct a PINN baseline as a comparison of our method, that is, we build an MLP model with 4 hidden layers, each layer containing 20 neurons and we choose the Tanh as activation function. When training, for fair comparison, the computation area, and train time steps are the same as the items in our approach, we randomly sample 1.2×10^5 collocation points and train the model with two stages, in the first stage, we train the model 1×10^4 epochs by Adam optimizer with $1e{-4}$ learning rate and then, in the second stage, the L-BFGS optimizer is adopted to further minimize the loss, which repeats 1×10^5 epochs.

The results are demonstrated in Tab.1, which contains the RMSE (Root Mean Square Error) [6, 2] at different time steps. Note that the model is built at the very first 10 steps, thus the results are an extrapolation of the model. As is shown in the Table, our method maintains a low level of error over time, demonstrating the better fitting ability and long-time generalization performance of the model compared with the PINN approach. Besides, we also visualize our results in Fig.2. In the Figure, the first two rows in the figure are the results at 0.1s and the remaining indicates the results at 0.4s. As we can observe, the predicted patterns are closed to the exact solutions, and the differences are near to zero.

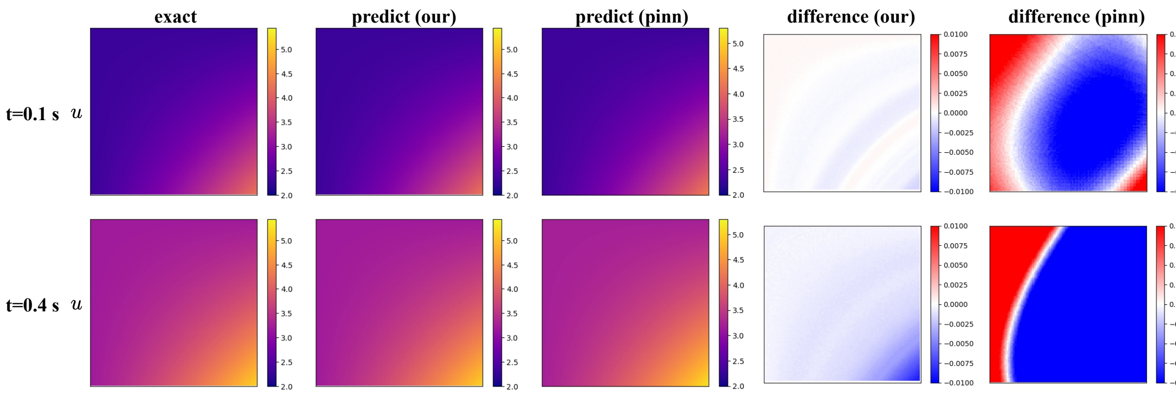


Figure 3: The reuslts of heat equation with u components at different time steps. The predicted results are compared with the exact analytical solutions and the difference is also presented.

Table 1
The RMSE errors at different time steps of Burges Equation.

Step	1	100	200	300	400
PINN	1.88e-04	2.13e-04	5.40e-04	1.28e-03	2.47e-03
OUR	3.93e-06	2.28e-04	4.03e-04	5.43e-04	6.53e-04

Table 2
The RMSE errors at different time steps of Heat Equation.

Step	1	100	200	300	400
PINN	6.19e-04	4.13e-03	1.24e-02	2.37e-02	3.51e-02
OUR	7.42e-06	2.93e-04	4.00e-04	5.81e-04	8.06e-04

3.3. Heat Equation

We introduce another PDE equation here which depicts the thermal conductivity dynamics, we consider the problem under one-dimension as example, which has the following form [16]:

$$u_t + \gamma \Delta u + f(x, y, t) = 0 \tag{18}$$

where, γ denotes the diffusion coefficient, f is a function of both spatial area x, y and time t. Here we set γ to 1 and $f(x, y, t)$ to $-3-2\gamma(x+y)$, following, the exact solution that conform to the above form is:

$$u(x, y, t) = xy^2 + yx^2 + 3t \tag{19}$$

The IC and BC setting have the same strategies as that is in Sec 3.2, in view of the above exact solution. To conduct experiments, we fix the γ to 1, and set the parameters T to 10, likewise, we train the network 1000 epochs and keep the model with minimal pde loss for evaluation. Moreover, we also construct a PINN model with the help of the DeepXDE framework. The network parameters and training configures of PINN are the same as Sec.3.2, which we don't go into detail here. It is worth noting that the L-BFGS optimizer doesn't minimize the pde loss to convergence in practice, we train the network with Adam for 1×10^5 epochs as a substitute, and the learning rate is set to 1×10^{-4}.

The results in quantitative form are shown in the Tab.2, which denotes the RMSE at different time steps compared with the exact solution. As we can see, the error in step 1 is on 10^{-6} level, which indicates the excellent fitting ability. Specifically, when compared with the PINN method, our approach fits better and has better accuracy even when extrapolating at time. The visualized results are shown in Fig.3, which demonstrates the extrapolation results of our approach and PINN method at 0.1s and 0.4s. As shown in the figure, the patterns are close to the exact solution of both our and PINN results, however, the differences of PINN are much higher than our approach.

3.4. Scale up

The above results show that our approach extrapolates well along with time increases compared with the PINN. However, in practice, the spatial extrapolation ability is considered as well. Limited by time or hardware resources, the model is usually trained on a small spatial area, then applied to an area that is usually larger than the training area. Here, we conduct experiments to explore the spatial extrapolation ability of our method.

We use the burgers equation as an example. Here, we directly apply the models of our approach and PINN mentioned in Sec.3.2 for evaluation. In detail, we train both the two models on the spatial area of $(x, y) \in [0, 1]^2$, after that, we specify the IC and BC same as Sec.3.2 and adopt the trained models for inference multi time steps results on larger scale spatial areas of $[0, 2]^2$ and $[0, 4]^2$ respectively, and evaluate the accuracy with the RMSE metric. The results are shown in Fig.3.4. That is, the solid lines in the figure denote the results of our approach, meanwhile, the dashed lines denote the PINN results. It can be seen that the RMSE of our method remains basically of the same magnitude on spatial regions of different scales, while the PINN method has a large RMSE on the non-training spatial scale. Also, the figure shows that our method has better time extrapolation capability as mentioned above. The extrapolation ability of space mainly benefits from the local computational property of the graph neural network, that is, the calculation of the

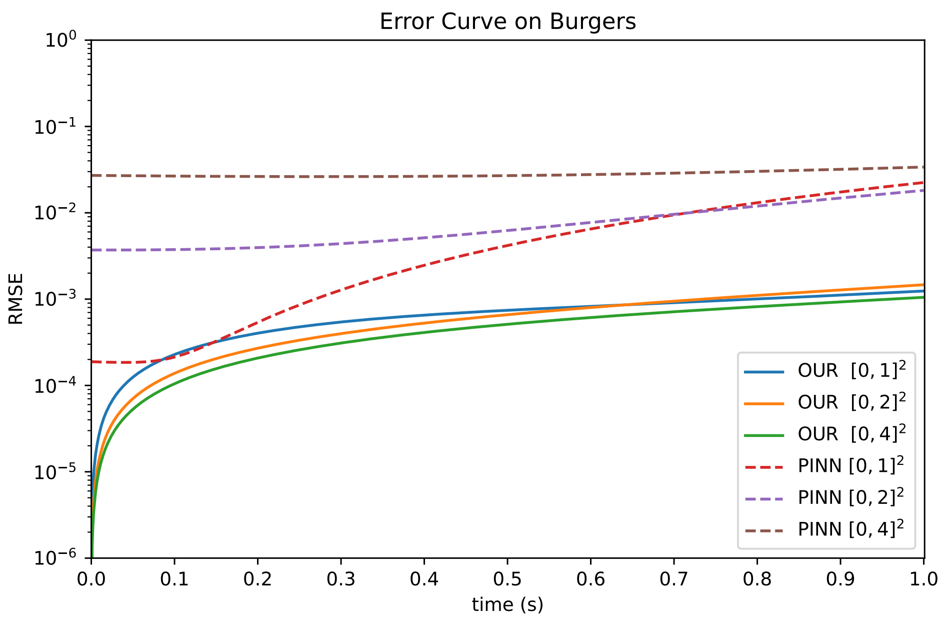


Figure 4: The RMSE results of our method (solid line) and PINN (dashed line) on different spatial areas along with time increases, the $[0,1]^2$ are training area while others are extrapolation.

graph neural network only depends on the adjacent nodes of space and their connected edges, thus is independent of the size of spatial region.

In addition, we also evaluate the time and memory consumption of the algorithm in the inference process. All the results are collected on a single A100 GPU, which are repeated 10 times, and the average values are reported. As shown in Fig.3.4, the bars show the inference time, and the red line indicates peak GPU memory allocated of a single step. It can be seen that with the expansion of the spatial area, the GPU memory and inference time required is increasing, but the magnitude remains at a low level. Even on the area $[0,10]^2$ with one million mesh points, single step inference costs 0.08s, and the GPU memory allocated is about 3 GB, and it is acceptable.

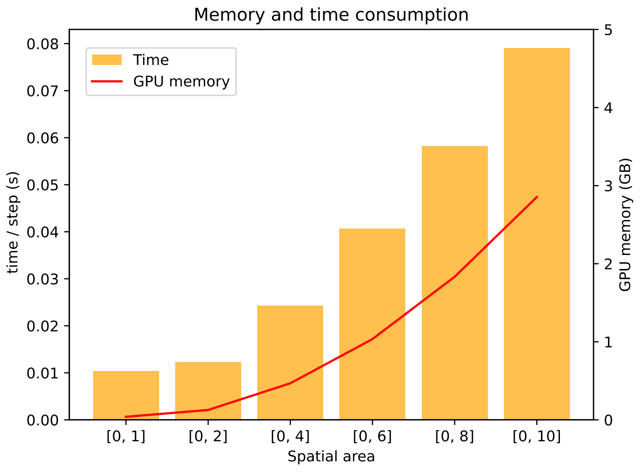


Figure 5: GPU memory and inference time consumption of our method at different spatial areas.

4. Conclusion

In this paper, we proposed a method to solve PDEs with graph neural network. That is, specifying the status of a time step, we supervised the network to have the ability to predict the solution of the next time step by minimizing the PDE loss. Compared with the typical PINN methods, our approach doesn't rely on automatic differentiation, otherwise, the differential term is constructed with spatial and time differences. Moreover, our approach benefits from the calculation of neighbors of the graph neural network, thus the trained model is more suitable for larger scale areas than PINN. The experiments conducted on burgers and heat equations show that our method extrapolates better on both spatial area and time contrast to the PINN method.

Furthermore, there also exists a limitation of our method. That is, according to the definition of the spatial differential term, our approach is only suitable for the square areas with a regular grid. However, in practice, irregular grid and areas are more common. How to compute the spatial differential term on irregular grid and areas remains to be studied in the future.

CRediT authorship contribution statement

Longxiang Jiang: Designed the algorithm, conducted experiment, and wrote draft paper. **Liyuan Wang:** Discussed the results and wrote draft paper. **Xinkun Chu:** Revised draft paper. **Yonghao Xiao:** Discussed the results. **Hao Zhang:** Designed the algorithm, revised draft paper.

References

[1] Gao, H., Sun, L. & Wang, J. PhyGeoNet: physics-informed geometry-adaptive convolutional neural networks for solving parameterized steady-state PDEs on irregular domain. *Journal Of Computational Physics*. **428** pp. 110079 (2021)

[2] Ren, P., Rao, C., Liu, Y., Wang, J. & Sun, H. PhyCRNet: Physics-informed convolutional-recurrent network for solving spatiotemporal PDEs. *Computer Methods In Applied Mechanics And Engineering*. **389** pp. 114399 (2022)

[3] Gers, F., Schraudolph, N. & Schmidhuber, J. Learning precise timing with LSTM recurrent networks. *Journal Of Machine Learning Research*. **3**, 115-143 (2002)

[4] Zhang, Z. A physics-informed deep convolutional neural network for simulating and predicting transient Darcy flows in heterogeneous reservoirs without labeled data. *Journal Of Petroleum Science And Engineering*. pp. 110179 (2022)

[5] Chen, Z., Huan, G. & Ma, Y. Computational methods for multiphase flows in porous media. (SIAM,2006)

[6] Pfaff, T., Fortunato, M., Sanchez-Gonzalez, A. & Battaglia, P. Learning mesh-based simulation with graph networks. *ArXiv Preprint ArXiv:2010.03409*. (2020)

[7] Sanchez-Gonzalez, A., Godwin, J., Pfaff, T., Ying, R., Leskovec, J. & Battaglia, P. Learning to simulate complex physics with graph networks. *International Conference On Machine Learning*. pp. 8459-8468 (2020)

[8] Seo, S. & Liu, Y. Differentiable physics-informed graph networks. *ArXiv Preprint ArXiv:1902.02950*. (2019)

[9] Battaglia, P., Hamrick, J., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R. & Others Relational inductive biases, deep learning, and graph networks. *ArXiv Preprint ArXiv:1806.01261*. (2018)

[10] Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C. & Philip, S. A comprehensive survey on graph neural networks. *IEEE Transactions On Neural Networks And Learning Systems*. **32**, 4-24 (2020)

[11] Merris, R. Laplacian matrices of graphs: a survey. *Linear Algebra And Its Applications*. **197** pp. 143-176 (1994)

[12] Smoller, J. Shock waves and reaction diffusion-equations. (Springer Science & Business Media,2012)

[13] Chen, X. Generation and propagation of interfaces for reaction diffusion-equations. *Journal Of Differential Equations*. **96**, 116-141 (1992)

[14] Zhu, H., Shu, H. & Ding, M. Numerical solutions of two-dimensional Burgers'equations by discrete Adomian decomposition method. *Computers & Mathematics With Applications*. **60**, 840-848 (2010)

[15] Lu, L., Meng, X., Mao, Z. & Karniadakis, G. DeepXDE: A deep learning library for solving differential equations. *SIAM Review*. **63**, 208-228 (2021)

[16] Langtangen, H. & Logg, A. Solving PDEs in python: the FEniCS tutorial I. (Springer Nature,2017)

[17] Brezis, H. & Brézis, H. Functional analysis, Sobolev spaces and partial differential equations. (Springer,2011)

[18] Dormand, J. Numerical methods for differential equations: a computational approach. (CRC press,2018)

[19] Bar-Sinai, Y., Hoyer, S., Hickey, J. & Brenner, M. Learning data-driven discretizations for partial differential equations. *Proceedings Of The National Academy Of Sciences*. **116**, 15344-15349 (2019)

[20] Raissi, M., Perdikaris, P. & Karniadakis, G. Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations. *ArXiv Preprint ArXiv:1711.10561*. (2017)

[21] Karniadakis, G., Kevrekidis, I., Lu, L., Perdikaris, P., Wang, S. & Yang, L. Physics-informed machine learning. *Nature Reviews Physics*. **3**, 422-440 (2021)

[22] Rao, C., Sun, H. & Liu, Y. Physics-informed deep learning for incompressible laminar flows. *Theoretical And Applied Mechanics Letters*. **10**, 207-212 (2020)

[23] Raissi, M., Perdikaris, P. & Karniadakis, G. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal Of Computational Physics*. **378** pp. 686-707 (2019)

[24] Almajid, M. & Abu-Al-Saud, M. Prediction of porous media fluid flow using physics informed neural networks. *Journal Of Petroleum Science And Engineering*. **208** pp. 109205 (2022)

[25] Zobeiry, N. & Humfeld, K. A physics-informed machine learning approach for solving heat transfer equation in advanced manufacturing and engineering applications. *Engineering Applications Of Artificial Intelligence*. **101** pp. 104232 (2021)