# Convolution-Based Model-Solving Method for Three-Dimensional, Unsteady, Partial Differential Equations

**Wenshu Zha**
*wszha@hfut.edu.cn*
**Wen Zhang**[*]
*2019111279@mail.hfut.edu.cn*
**Daolun Li**
*ldaol@ustc.edu.cn*
**Yan Xing**
*xy1128@126.com*
**Lei He**
*hlei80@163.com*
**Jieqing Tan**
*jqtan@mail.hf.ah.cn*
*Hefei University of Technology, Hefei, Anhui, 230009, China*

**Neural networks are increasingly used widely in the solution of partial differential equations (PDEs). This letter proposes 3D-PDE-Net to solve the three-dimensional PDE. We give a mathematical derivation of a three-dimensional convolution kernel that can approximate any order differential operator within the range of expressing ability and then conduct 3D-PDE-Net based on this theory. An optimum network is obtained by minimizing the normalized mean square error (NMSE) of training data, and L-BFGS is the optimized algorithm of second-order precision. Numerical experimental results show that 3D-PDE-Net can achieve the solution with good accuracy using few training samples, and it is of highly significant in solving linear and nonlinear unsteady PDEs.**

## 1 Introduction

In the disciplines of fluid mechanics, geophysics, and biology, the partial differential equation (PDE) model is an important tool to characterize its properties. The two most typical problems in the solution and recovery of PDEs are inverse problems for each other.

Only a handful of PDEs can be solved accurately. Therefore, in practical applications, the solutions of these equations are often approximated

[*]Wen Zhang is the corresponding author.

https://doi.org/10.1162/neco_a_01462

numerically. Many scholars have studied and developed a variety of numerical calculation methods for various types of equations, including the Runge-Kutta method (Verwer, 1996), the grid-free method (Liewa & Cheng, 2009; Liu & Cheng, 2019), the predictor correction method, the finite difference method (Wu & White, 2004), the finite element method, and B-spline. Finite difference algorithms solving high-dimensional PDEs have been improved and developed many times. For example, for parabolic equations, the classical difference formats were crank-Nicolson and Douglas-Gunn (Douglas & Gunn, 1964). Some authors (Dai & Nassar, 1998) advocated solving two-dimensional partial differential equations by an alternating direction implicit (ADI) scheme of second-order precision. The high-order compact ADI (HOC-ADI; Karaa & Zhang, 2004) has been developed by researchers and continuously improved to form many second-, fourth- or even higher-order PDE-solving methods, such as the DHOC ADI format (Karaa, 2006), the PHOC ADI format (Cao & Ge, 2011), and the EHOC ADI format (Ge, Tian, & Zhang, 2013).

Due to powerful data processing and self-learning capabilities, there has been a great surge of interest in developing neural networks. Lagaris and Likas (1998) sought to use artificial neural networks (ANNs) to solve the initial boundary value problem. The radial basis function neural network (RBF; Mai-Duy & Tran-Cong, 2001) was a method to obtain the numerical solution to differential equations. In recent years, various algorithms have combined with neural networks to find the numerical solution of PDE. And the neural network introducing physical information (PINN) was used to approximate the solution of nonlinear PDE (Raissi et al., 2017a). For solving the problem of heterogeneous elliptic PDE in different domains, a convolutional neural network with quantitative uncertainty was presented (ConvPDE-UQ; Winovich, Ramani, & Lin, 2019). The Bernstein neural network (BeNN; Sun et al., 2019), combined with the extreme learning machine algorithm (ELM), was used to solve PDEs, which makes the computational complexity of the BeNN model less than other traditional numerical methods. All of these methods cope with the initial-boundary value problem. Inputs are spatial coordinates or time variables, and outputs are numerical solutions. A network is used to approximate the solution in the computational domain. The initial boundary value conditions of PDEs are merged with the neural network to construct a trial solution. The optimization goal has two parts: the boundary loss of the trial solution and the equation loss on the interior.

A kind of inverse problem corresponding to solving PDEs is the recovery of PDEs. Currently, there are two approaches to this problem. One is to obtain a dictionary of possible terms for the physical process described by the PDE with a certain prior knowledge and use sparse regression technology or other methods for feature selection and parameter estimation (Rudy, Brunton, Proctor, & Kutz, 2017; Hayden, 2017; Brunton, Proctor, & Kutz, 2016). The goals of these methods are usually the same. The combined data-driven and data-assimilation method (Chang & Zhang, 2019) proposed for uncovering equations and the local interpolation combining the symbol regression method (Reinbold Patrick & Grigoriev Roman, 2019) were also similar to sparse regression. The second approach is to use the neural network as an approximator. For example, the network-introduced physical information (PINN; Raissi, Perdikaris & Karniadakis, 2017b) could determine the unknown parameters of PDEs while solving equations in which the partial derivative term was calculated by combining the chain rule and automatic differential technology. On this basis, a method that combined two neural networks (Raissi, Perdikaris, & Karniadakis, 2018) was used to simultaneously approximate the solution and parameters of PDE. More recently, the residual network (ResNet) was discovered and developed rapidly. Some researchers (Wu & Xiu, 2020) explored using deep ResNet to numerically approximate the evolution operator of PDEs and then approximate the nonlinear dynamic system as represented by the PDEs. PDE-Net was a new feedforward neural network (Long, Lu, Ma, & Dong, 2018; Long, Lu, & Dong, 2019) that used the convolution kernel to approximate a differential operator. It has been discovered to uncover PDE and accurately predict the dynamics of complex systems, but these two previous studies mainly focused on two-dimensional problems. Inspired by these two papers, we propose a network to solve three-dimensional PDE.

In this letter, we propose a convolutional neural network that we call the 3D-PDE-Net for solving the unsteady PDE. We extended the PDE-Net structure and related concepts from two- to three-dimensional by Taylor′s expansion of multivariate function and theory of vanishing moments of order. Subsequently, we use it to solve unsteady linear convection, diffusion equations, and a forced Burgers problem, respectively, and then observe how the proposed network performs in a noisy environment.

## 2 Description of the Neural Network Method

Prior to this, all theoretical research and numerical experiments were aimed at two-dimensional problems. Our letter constructs a 3D-PDE-Net and uses it to obtain the solution of three-dimensional, unsteady PDEs. In the process of theoretical analysis, we assume that the highest order of the PDE is a finite positive integer.

During neural network training, only a few spatiotemporal measurements of the system state are needed, that is, more solutions are obtained from a few solutions of PDEs. Suppose the unsteady PDE studied here adopts general format expressed by equation 2.1:

$$u_t(t, x, y, z) = F(x, y, z, u_x, u_y, u_z, u_{xx}, u_{yy}, u_{zz}, u_{xy}, u_{xz}, u_{yz}, \ldots). \tag{2.1}$$

Given a series of measurement $\{u(t,\cdot) : t = t_0, t_1, \ldots\}$ in the domain $(x, y, z) \in \Omega \subset R^3$ as training samples, train the neural network to approximate the PDE. The solutions of PDEs in future time can be calculated by using the trained neural network.

**2.1 3D Convolutions and Differentiations..** Under the frame of variation and PDE, Cai, Dong, Osher, and Shen (2012) and Dong, Jiang, and Shen (2017) analyzed the relationship between convolution and differentiation. Different filters can represent different difference formats. For the convenience of writing, here we take the $2^3$ filter as an example to briefly explain the relationship between the convolution kernel and the differential operator. The $2^3$ Haar tight frame bank has filters,

$$q_{000} = \frac{1}{8}\begin{pmatrix} A_1 \\ A_1 \end{pmatrix}, q_{100} = \frac{1}{8}\begin{pmatrix} A_2 \\ A_2 \end{pmatrix}, q_{010} = \frac{1}{8}\begin{pmatrix} A_3 \\ A_3 \end{pmatrix}, q_{001} = \frac{1}{8}\begin{pmatrix} A_1 \\ A_4 \end{pmatrix},$$

where

$$A_2 = \begin{pmatrix} 1 & -1 \\ 1 & -1 \end{pmatrix}, A_3 = \begin{pmatrix} 1 & 1 \\ -1 & -1 \end{pmatrix}, A_4 = \begin{pmatrix} -1 & -1 \\ -1 & -1 \end{pmatrix},$$

and $q_{000}$, $q_{100}$, $q_{010}$, and $q_{001}$ all denote three-dimensional filters composed of two two-dimensional filters.

The Haar wavelet frame transform corresponding to the above filters is defined as

$$Du = \{q_{ijt}[-\cdot] \otimes u : 0 \le i + j + t \le 1\},$$

where $\otimes$ stands for the convolution kernel. The high-frequency coefficients of the Haar wavelet frame transform on $u$ are discrete approximations of differential operators.

$$\begin{aligned} \frac{2}{\varepsilon}(q_{100}[-\cdot] \otimes u)[x, y, z] = &\frac{1}{4\varepsilon}(u(x, y, z) - u(x - \varepsilon, y, z)) \\ &+ \frac{1}{4\varepsilon}(u(x, y - \varepsilon, z) - u(x - \varepsilon, y - \varepsilon, z)) \\ &+ \frac{1}{4\varepsilon}(u(x, y, z - \varepsilon) - u(x - \varepsilon, y, z - \varepsilon)) \\ &+ \frac{1}{4\varepsilon}(u(x, y - \varepsilon, z - \varepsilon) - u(x - \varepsilon, y - \varepsilon, z - \varepsilon)) \\ \approx\ & u_x(x, y, z) \quad as\ \varepsilon \to 0 \end{aligned}$$

Similarly, we can deduce easily that

$$\frac{2}{\varepsilon}(q_{010}[-\cdot] \otimes u) \approx u_y(x, y, z),\ \frac{2}{\varepsilon}(q_{001}[-\cdot] \otimes u) \approx u_z(x, y, z).$$

Among them, $\varepsilon$ represents spatial grid size. The grid scales of the three dimensions can be different, but for convenience, they are written as consistent here.

In the following context, we discuss the relationship of the convolution with differentiation. The order of the vanishing moments of the $N^3$ filter (Anger, 1990) is defined as follows.

**Definition 1.** *For a filter $q$, we say $q$ has vanishing moments of order $\alpha = (\alpha_1, \alpha_2, \alpha_3) \in Z_+^3$,*

$$\sum_{k \in Z^3} k^\beta q[k] = 0, \tag{2.2}$$

*for all $\beta = (\beta_1, \beta_2, \beta_3) \in Z_+^3$ with $|\beta| < |\alpha|$ and with $|\beta| = |\alpha|$ but $\beta \neq \alpha$ equation 2.2 holds. The $\alpha = (\alpha_1, \alpha_2, \alpha_3)$ that meets the above conditions is defined as the vanishing moment of order $q$. If equation 2.2 holds for all $\beta \in Z_+^3$ with $\beta < |K|$ except for $\beta \neq \beta_0$ with certain $\beta_0 \in Z_+^3$ and $|\beta_0| = J < K$, then we define the total vanishing moments of order of filter $q$ as $K \backslash \{J + 1\}$.*

The following proposition associates the vanishing moment of order of the three-dimensional filter with the differential operator:

**Proposition 1.** *Let $q$ be a filter with vanishing moment of order $\alpha \in Z_+^3$; then for a smooth function $F(x) \in R^3$, we have*

$$\frac{1}{\varepsilon^{|\alpha|}} \sum_{k \in Z^3} q[k] F(x + k\varepsilon) = C_\alpha \frac{\partial^\alpha F(x)}{\partial x^\alpha} + o(\varepsilon),\ \varepsilon \to 0, \tag{2.3}$$

*where $C_\alpha$ is the constant defined by $C_\alpha = \frac{1}{\alpha!} \sum_{k \in Z^3} k^\alpha q[k]$. Additionally, if $q$ has total vanishing moments of order $K \backslash \{|\alpha| + 1\}$ for some $K > |\alpha|$, then*

$$\frac{1}{\varepsilon^{|\alpha|}} \sum_{k \in Z^3} q[k] F(x + k\varepsilon) = C_\alpha \frac{\partial^\alpha F(x)}{\partial x^\alpha} + o(\varepsilon^{K - |\alpha|}),\ \varepsilon \to 0. \tag{2.4}$$

**Proof.** The proof is available by Taylor's expansion of multivariate function and definition 1.1,

$$\begin{aligned}
&\frac{1}{\varepsilon^{|\alpha|}}\sum_{k\in Z^3} q[k]F(x+k\varepsilon)\\
&=\frac{1}{\varepsilon^{|\alpha|}}\sum_{k\in Z^3} q[k]\sum_{i=0}^{N-1}\left(\frac{\partial^i F}{\partial x^i}\bigg|_{(x)}\frac{(k\varepsilon)^i}{i!}\right)+o(\varepsilon^{N-\alpha})\\
&=\sum_{i=0}^{N-1}\frac{1}{i!}\left(\sum_{k\in Z^3}k^i q[k]\right)\frac{\partial^i F}{\partial x^i}\bigg|_{(x)}\frac{\varepsilon^i}{\varepsilon^{|\alpha|}}+o(\varepsilon^{N-\alpha})\\
&=\sum_{i=0}^{\alpha-1}\frac{1}{i!}\left(\sum_{k\in Z^3}k^i q[k]\right)\frac{\partial^i F}{\partial x^i}\bigg|_{(x)}\frac{\varepsilon^i}{\varepsilon^{|\alpha|}}+\frac{1}{\alpha!}\left(\sum_{k\in Z^3}k^\alpha q[k]\right)\frac{\partial^\alpha F}{\partial x^\alpha}\bigg|_{(x)}\frac{\varepsilon^\alpha}{\varepsilon^{|\alpha|}}\\
&\quad+\sum_{i=\alpha+1}^{N-1}\frac{1}{i!}\left(\sum_{k\in Z^3}k^i q[k]\right)\frac{\partial^i F}{\partial x^i}\bigg|_{(x)}\frac{\varepsilon^i}{\varepsilon^{|\alpha|}}+o(\varepsilon^{N-\alpha})\\
&=\frac{1}{\alpha!}\left(\sum_{k\in Z^3}k^\alpha q[k]\right)\frac{\partial^\alpha F}{\partial x^\alpha}\bigg|_{(x)}+o(\varepsilon)=C_\alpha\frac{\partial^\alpha F(x)}{\partial x^\alpha}+o(\varepsilon),\ \varepsilon\to 0.
\end{aligned}$$

The accuracy can be conducted by the definition of total vanishing moments of order. □

**Definition 2.** *For an $N\times N\times N$ filter $q$, we say that the moment matrix of $q$ as $M(q)=(m_{i,j,t})_{N\times N\times N}$ will be imposed on filters in the 3D-PDE-Net, which is defined by*

$$m_{i,j,t}=\frac{1}{i!j!t!}\sum_{k_1,k_2,k_3=-\frac{N-1}{2}}^{\frac{N-1}{2}}k_1^i k_2^j k_3^t q[k_1,k_2,k_3]\quad i,j,t=0,1,\ldots N-1. \tag{2.5}$$

For a smooth function $f:R^3\to R$, we apply the convolution operation with respect to filter $q$ on the spatial sampled point of $f$. While retaining the precision of the finite derivative order, the following formula can be obtained by using Taylor's expansion according to the trinomial expansion rule

$$\begin{aligned}
&\sum_{k_1,k_2,k_3=-\frac{N-1}{2}}^{\frac{N-1}{2}} q[k_1,k_2,k_3]f(x+k_1\delta x,y+k_2\delta y,z+k_3\delta z)\\
&=\sum_{k_1,k_2,k_3=-\frac{N-1}{2}}^{\frac{N-1}{2}} q[k_1,k_2,k_3]\left(\sum_{i,j,t=0}^{N-1}\frac{1}{i!j!t!}\frac{\partial^{i+j+t}f}{\partial x^i\partial y^j\partial z^t}\bigg|_{(x,y,z)}(k_1\delta x)^i(k_2\delta y)^j(k_3\delta z)^t\right.\\
&\quad\left.+o(|\delta x|^{N-1}+|\delta y|^{N-1}+|\delta z|^{N-1})\right)
\end{aligned}$$

$$
\begin{aligned}
&= \sum_{i,j,t=0}^{N-1} \left( \frac{1}{i!j!t!} \sum_{k_1,k_2,k_3=-\frac{N-1}{2}}^{\frac{N-1}{2}} k_1^i k_2^j k_3^t q[k_1,k_2,k_3] \right) \left. \frac{\partial^{i+j+t} f}{\partial x^i \partial y^j \partial z^t} \right|_{(x,y,z)} (\delta x)^i (\delta y)^j (\delta z)^t \\
&\quad + o(|\delta x|^{N-1} + |\delta y|^{N-1} + |\delta z|^{N-1}) \\
&= \sum_{i,j,t=0}^{N-1} m_{i,j,t} \left. \frac{\partial^{i+j+t} f}{\partial x^i \partial y^j \partial z^t} \right|_{(x,y,z)} (\delta x)^i (\delta y)^j (\delta z)^t + o(|\delta x|^{N-1} + |\delta y|^{N-1} + |\delta z|^{N-1}).
\end{aligned}
\tag{2.6}
$$

From equation 2.6 and proposition 1, it can be seen that the three-dimensional filter constrained by $M(q)$ can be used to approximate any order differential operator within the representation capability of the filter. For example, the $3 \times 3 \times 3$ convolution kernel cannot theoretically represent differential operators of seventh order or higher. Long and Dong et al. (2019) gave a detailed explanation on the realization of the two-dimensional moment matrix constraint, and the three-dimensional is similar. The process for using the convolution kernel to approximate the differential operator does not need other prior knowledge except for the maximum possible order (max_order) of the differential equation. Given the hyperparameter max_order, we can get convolution kernels that satisfy the parameter constraint. For example, when max_order = 2, it has 10 convolution kernels: $D_{000}$, $D_{001}$, $D_{010}$, $D_{100}$, $D_{002}$, $D_{011}$, $D_{020}$, $D_{110}$, $D_{200}$, $D_{101}$. Long et al. (2018) proved that the more accurate the hyperparameter constraint is, the smaller the solution error is.

**2.2 3D-PDE-Net Method.** Considering the evolution of PDE equation 2.1, we use forward Euler as the time discretization method for concise demonstration.

*2.2.1 Basic Module:* $1 - \delta t$ *Block.* Let $\hat{u}(t_{i+1}, \cdot)$ be the computed value of $u$ at time $t_{i+1}$ based on the value of $u$ at time $t_i$. Then the network structure is expressed as

$$
\hat{u}(t_{i+1}, \cdot) = D_0 u(t_i, \cdot) + \delta t \times F(x, y, D_{000}u, D_{100}u, D_{010}u, D_{001}u, \ldots), \tag{2.7}
$$

where the operators $D_0$ and $D_{ijt}$ represent filters corresponding to the $q_0$ and $q_{ijt}$ convolution operators, which are used to approximate the differential operator, that is, $D_0 u = q_0 \otimes u$ and $D_{ijt} u = q_{ijt} \otimes u \approx \frac{\partial^{i+j+t} u}{\partial x^i \partial y^j \partial z^t}$. $D_0$ and $D_{000}$ are average operators, and after the introduction of an average operator, the expression ability of the network is better than that of the network using identity mapping. Therefore, we introduce the average convolution operator directly in the same way as in a two-dimensional network. $D_0$ and

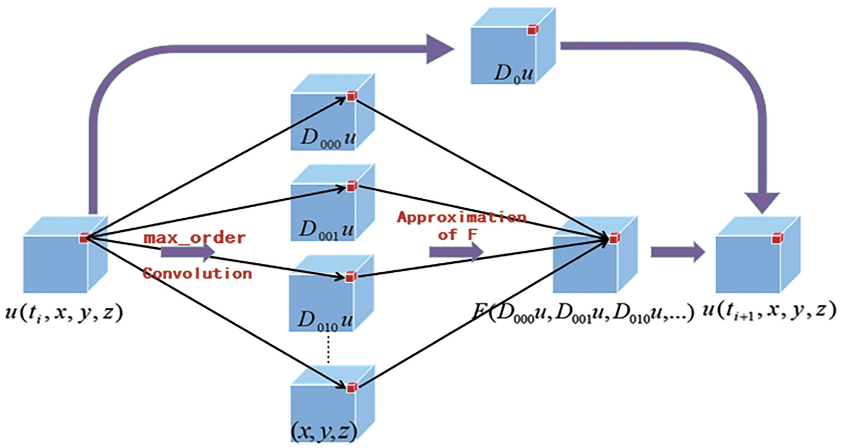

Figure 1: Schematic diagram of a 3D $1-\delta t$ block.

$D_{ijt}$ are the parameters that can be learned during the training process. In addition, the approximate nonlinear term F can be carried out by neural networks or other machine learning methods. The experiments in this letter adopt multivariate interpolation to approximate nonlinear F. The network structure is set out in Figure 1.

If the objective equation is linear with respect to partial derivatives, then it is described as $F = \sum_{0\le i,j,t} f_{ijt}(x,y,z)\frac{\partial^{i+j+t}u(x,y)}{\partial x^i \partial y^j \partial z^t}$. Whether it is a variable coefficient or not, the network structure $\{c_{ijt}\}$ is a three-dimensional matrix as an approximation of $f_{ijt}(x,y,z)$ on $\Omega$, which can be obtained by interpolation or directly learned during training.

*2.2.2 3D-PDE-Net: Multiple-$\delta t$ Blocks.* Here we stack multiple $1-\delta t$ blocks to be 3D-PDE-Net and share parameters in each $1-\delta t$ block across layers. An optimum network is obtained by minimizing the cumulative error $\|u(t_{i+n}) - \hat{u}(t_{i+n})\|_2^2$, where $\hat{u}(t_{i+n})$ is the output of the 3D-PDE-Net ($n-\delta t$ blocks). In the numerical experiments in this letter, the normalized error (*NMSE*) is adopted as the loss function of network training:

$$\frac{\left\|u(t_{i+n}) - \hat{u}(t_{i+n})\right\|_2^2}{\left\|u(t_{i+n}) - \bar{u}(t_{i+n})\right\|_2^2} = \frac{\frac{1}{SNML}\sum_{t,i,j,k}^{S,N,M,L}\left(|\hat{u}-u|^2\right)}{\frac{1}{SNML}\sum_{t,i,j,k}^{S,N,M,L}\left(|u-\bar{u}|^2\right)}, \tag{2.8}$$

Among them, $S$ represents the number of samples for training, and $N, M, L$ represent the number of sampling points in three-dimensional coordinates, respectively.

**2.3 Adaptive Activation Functions (LAAF).** Jagtap et al. (2020a) proposed an adaptive activation function with an additional scalable parameter $na$. The parameter $a \in R$ acts as a slope of activation function. The globally adaptive activation (GAAF) means that it gives an optimized slope for the entire network. And then the authors define this parameter for the local network, including layer-wise and neuron-wise locally adaptive activation functions. The two locally optimized activation functions are defined as follows (Jagtap et al, 2020b):

**Layer-wise locally adaptive activation functions (L-LAAF)**. This scheme defines the parameter for each hidden layer as

$$\sigma(na^k L_k(z^{k-1})), \quad k = 1, 2, \ldots, D-1,$$

where $L_k(z^{k-1})$ is the form of affine transformation between two adjacent layers. This gives additional $D-1$ parameters to be optimized along with weights and biases. Here, every hidden layer has its own slope for the activation function.

**Neuron-wise locally adaptive activation functions (N-LAAF)**. One can also define such an activation function at the neuron level as

$$\sigma(na_i^k (L_k(z^{k-1}))_i), \quad k = 1, 2, \ldots, D-1, i = 1, 2, 3, \ldots, N_k.$$

This gives additional $\sum_{i=1}^{D-1} N_k$ parameters to be optimized.

The neuron-wise activation function acts as a vector activation function in each hidden layer, where every neuron has its own slope for the activation function. We take the first approach. However, due to parameter sharing of the circular network, the L-LAAF in this letter is equivalent to the GAAF.

**2.4 Error Evaluation Index.** In order to evaluate the performance of the neural network, the following functions are used as evaluation indexes to measure the error between the exact value $u$ and output of network $\hat{u}$, and $\bar{u}$ represents the average value of the exact value in space:

$$NMSE = \frac{\|\hat{u}-u\|_2^2}{\|u-\bar{u}\|_2^2}, \quad \|e\|_2 = \sqrt{\Delta x \Delta y \Delta z \sum_{i,j,k}^{N,M,L} (\hat{u}-u)^2},$$

$$MAE = \frac{1}{N \times M \times L} \sum_{i,j,k}^{N,M,L} |\hat{u}-u|, \quad MAPE = \frac{1}{N \times M \times L} \sum_{i,j,k}^{N,M,L} \left| \frac{\hat{u}-u}{u} \right|,$$

$$\|e\|_\infty = \max_{i,j,k} |\hat{u}-u|, \quad NL_\infty = \frac{\max_{i,j,k} |\hat{u}-u|}{\|u-\bar{u}\|_2} = \frac{\max_{i,j,k} |\hat{u}-u|}{\|u-\bar{u}\|_2}.$$

*NMSE* represents the standard mean square error, and the smaller this value is, the better the model effect will be. $\|e\|_2$ represents an $L_2$ norm error, which is used to compare the accuracy with other difference schemes. *MAE* represents the average absolute error value. The smaller the value is, the closer it is to the true value. *MAPE* represents the average absolute percentage error. A value less than 0.1 indicates that the model has a good effect. When the value of*MAPE* is large, it cannot indicate that the model is bad (when the local point error is large, it may also lead to a large *MAPE* value); when the value of a point in the real value $u$ is 0, *MAPE* is meaningless (Wang et al., 2020). $\|e\|_\infty$ represents an $L_\infty$ norm error value, namely, the maximum error; $NL_\infty$ represents the standard maximum error: the smaller, the better. *NMSE* and *MAPE* can reflect the overall model and equation fitting effect to a certain extent. $NL_\infty$ represents local characteristics and the relative maximum error. The smaller the error of *MAE*, $L_2$ and $L_\infty$ calculated, the closer the result is to the real value.

## 3 Numerical Experiment 1: Convection Equation

Coefficients can be calculated by multivariate interpolation (Long et al., 2018). In order to simplify the equation and save computing time, the experiment in this letter considers only the constant coefficient convection equation; the variable coefficient convection equation can also be studied in a similar way.

**3.1 Data Preparation, Training, and Testing.** In the computation domain $\Omega = [0, 2] \times [0, 2] \times [0, 2]$, the three-dimensional convection equation is considered,

$$\frac{\partial u}{\partial t} = -p\frac{\partial u}{\partial x} - q\frac{\partial u}{\partial y} - r\frac{\partial u}{\partial z}, \quad \text{where } (x, y, z, t) \in \Omega \times \{t > 0\}, \tag{3.1}$$

where $p = q = r = 1$, the size of regular mesh used to discretize $\Omega$ is $50 \times 50 \times 50$, the spatial step $\Delta x = \Delta y = \Delta z = 2/50$, and training and testing data are all given by the exact solution, equation 3.2:

$$u(x, y, z, t) = \sin(\pi(x + y + z - 3t)). \tag{3.2}$$

The initial conditions are given by the exact solution, equation 3.2, and here we use periodic boundary conditions. Gaussian noise is added to each data sample $u(x, y, z, t)$ for enhancing the expressive ability of the network:

$$\hat{u}(x, y, z, t) = u(x, y, z, t) + 0.01 \times std\,\mathrm{var} \times W, \tag{3.3}$$

where *std* var $= \|u - \bar{u}\|_2$, $W \sim \mathrm{N}(0, 1)$, and $\mathrm{N}(0, 1)$ is a standard normal distribution. Under the condition that the order of PDEs does not exceed 2, the size of the convolution kernel used for training is 3. Then the form of the nonlinear response term is

$$F = \sum_{0 \le i+j+t \le 2} f_{ijt}(x, y, z) \frac{\partial^{i+j+t} u(x, y)}{\partial x^i \partial y^j \partial z^t}.$$

The basic module of the 3D-PDE-Net neural network can be written as

$$\hat{u}(t_{i+1}, \cdot) = D_0 u(t_i, \cdot) + \delta t \times (c_{000} D_{000} u + c_{100} D_{100} u, \ldots + c_{002} D_{002} u),$$

where $\{c_{ijt}\}$ is a matrix with size $N \times M \times L$, which is the approximation of $f_{ijt}(x, y, z)$ on $\Omega$. Use piecewise polynomial interpolation with smooth transitions at the boundaries of each piece to achieve approximation. The parameters that can be learned in the network include the interpolation parameters in $\{c_{ijt}\}$ and the convolution kernel $\{D_0, D_{ijt} : 0 \le i + j + t \le 2\}$.

The required data are generated on the fly during training or testing. Considering the calculation time and cost, the size of the convolution kernel will be $3 \times 3 \times 3$ in our experiments, and the corresponding accuracy may be lower if the size of filters is $5 \times 5 \times 5$ or $7 \times 7 \times 7$. In the training process, a layer-wise training mode that can improve training speed is adopted. It generates five samples for training each layer, and 3D-PDE-Net constructs to six layers, that is, the network trained in each batch requires a total of 30 data samples. Because we need only a small number of samples, the quasi-Newton iteration method (L-BFGS) is used for optimization with second-order precision.

**3.2 Results and Discussion.** This section introduces the numerical results of the trained 3D-PDE-Net using the data set described in the previous section and analyzes the effect of 3D-PDE-Net in solving convection problems. The predecessors Long et al. (2018) have given the influence of hyperparameters. On the whole, the larger the size of the filter in a certain range is, the better the effect of solution is, and the more network layers there are, the smaller the error of the numerical solution is. Therefore, in this section, we directly observe and analyze the three-dimensional experimental results without considering the influence of hyperparameter changes.

In order to observe the performance of the trained network, we obtain the solutions of PDEs at time intervals $T = 0.2$ and $T = 0.8$ and analyze their errors. The results are shown in Tables 1 and 2.

At different numbers of $\Delta t$, we use the network with the same number of layers to compare the errors of the solution. When $\Delta t$ is 20 and 80, we derive the results shown in Tables 3 and 4, respectively.

Table 1: Errors of the Solution to Equation 3.1 at $T = 0.2$, max_order = 1.

| $\Delta t$ | *NMSE* | $L_2$ | *MAE* | *MAPE* | $L_\infty$ | $NL_\infty$ |
|---|---|---|---|---|---|---|
| 0.000625 | 0.00040166 | 0.04008548 | 0.01079458 | 0.04247823 | 0.06605295 | 0.09340756 |
| 0.0025 | 0.00013105 | 0.02289734 | 0.00645788 | 0.04754756 | 0.03701262 | 0.0523392 |
| 0.01 | 0.00010143 | 0.02014332 | 0.00567489 | 0.01770869 | 0.02931168 | 0.04145081 |
| 0.04 | 0.00015213 | 0.02466941 | 0.00695369 | 0.02665308 | 0.03764647 | 0.05323712 |

Table 2: Errors of the Solution to Equation 3.1 at $T = 0.8$, max_order = 1.

| $\Delta t$ | *NMSE* | $L_2$ | *MAE* | *MAPE* | $L_\infty$ | $NL_\infty$ |
|---|---|---|---|---|---|---|
| 0.0025 | 0.00023583 | 0.03071465 | 0.00878196 | 0.10190985 | 0.04431477 | 0.06266827 |
| 0.01 | 0.00011312 | 0.02127255 | 0.00599603 | 0.23697056 | 0.03300223 | 0.04666945 |
| 0.04 | 0.00075804 | 0.05506779 | 0.01510641 | 0.055268575 | 0.16165133 | 0.22859792 |

Table 3: Errors of the Solution to Equation 3.1 at $20\Delta t$, max_order = 1.

| $\Delta t$ | *NMSE* | $L_2$ | *MAE* | *MAPE* | $L_\infty$ | $NL_\infty$ |
|---|---|---|---|---|---|---|
| 0.000625 | 0.00012118 | 0.02201709 | 0.00620107 | 0.01969906 | 0.03594206 | 0.05082856 |
| 0.0025 | 0.0001213 | 0.02202864 | 0.00619637 | 0.01998279 | 0.03999246 | 0.05655439 |
| 0.01 | 0.00010266 | 0.02026414 | 0.00571009 | 0.12338079 | 0.03272538 | 0.0462805 |
| 0.04 | 0.00075804 | 0.05506779 | 0.01510641 | 0.05526858 | 0.16165132 | 0.22859792 |

Table 4: Errors of the Solution to Equation 3.1 at $80\Delta t$, max_order = 1.

| $\Delta t$ | *NMSE* | $L_2$ | *MAE* | *MAPE* | $L_\infty$ | $NL_\infty$ |
|---|---|---|---|---|---|---|
| 0.000625 | 0.00019051 | 0.0276063 | 0.00771868 | 0.01969906 | 0.05251202 | 0.06097842 |
| 0.0025 | 0.00013105 | 0.02289734 | 0.00645788 | 0.04754756 | 0.03701262 | 0.0523392 |
| 0.01 | 0.00011312 | 0.02127255 | 0.00599603 | 0.23697056 | 0.03300223 | 0.04666945 |
| 0.04 | 84602.0781 | 581.731873 | 159.800278 | 576.107727 | 1228.75745 | 1737.71594 |

By comparing the test results from Table 1 to Table 4, we can draw the following conclusions:

1. When the time interval is $T = 0.2$, the calculated solution with different numbers of $\Delta t$ has little error with the exact solution. All of *MAPE* are less than 0.1, and *NMSE* also show that the neural network fitting equation is good.

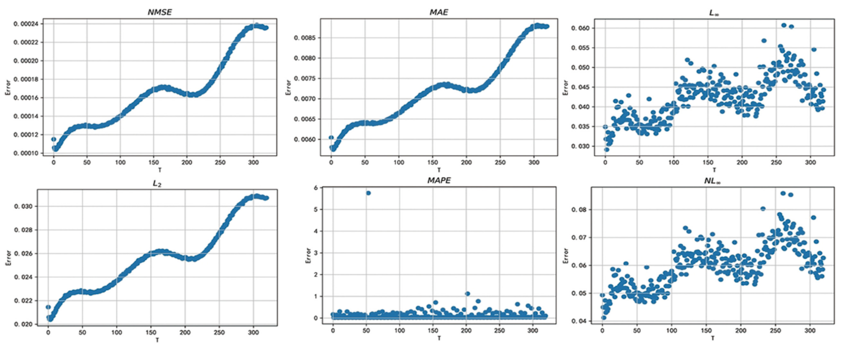

Figure 2: Errors of the solution to Equation 3.1 at $\Delta t = 0.0025$, $T = 0.8$.

2. When the time interval is $T = 0.8$, the result at $\Delta t = 0.04$ is poor because the time step is relatively too large.

Next, we compare the experimental results when the number of $\Delta t$ is the same:

3. When $\Delta t$ is 20, the overall solution error performance is very good.
4. When $\Delta t$ is 80, the effect is well under the other $\Delta t$ except for the situation of $\Delta t = 0.04$ because the time interval is relatively large.

Based on this analysis, when evaluating the error of the solution, $\Delta t$, $T$, and the number of $\Delta t$ should be taken into consideration at the same time. Comparison of the experimental results shows that both $\Delta t = 0.01$ and $\Delta t = 0.0025$ have a small error between the numerical solution with the exact solution. At $\Delta t = 0.0025$ and $T = 0.8$, the various error plots are shown in Figure 2.

When $\Delta t = 0.0025$, the number of $\Delta t$ is 80 ($T = 0.2$), and the number of $\Delta t$ is 320 ($T = 0.8$); sections of the 3D spatial solution images are shown in Figure 3.

The Douglas-Gunn scheme is a classical difference method with second-order precision summarized (Ma, 2018). We set $T = 0.2$, $dt = 0.01$, and $\Delta x = \Delta y = \Delta z = 0.05$. The $L_2$ and $L_\infty$ norm errors used for the 3D-PDE-Net method and the Douglas-Gunn scheme are depicted in Table 5. We can see that the 3D-PDE-Net provides a more accurate solution than the Douglas–Gunn ADI scheme.

## 4 Numerical Experiment 2: Diffusion Equation

Similar to the previous experiment, this one also focuses on the constant coefficient diffusion equation.

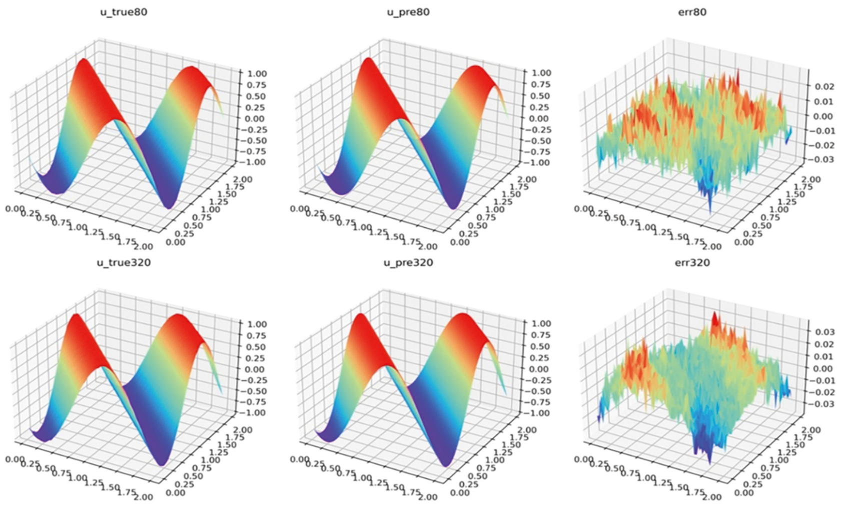

Figure 3: Sections of the 3D solution to Equation 3.1 at $\Delta t = 0.0025$ and the number of $\Delta t$ is 80 (top row) and 320 (bottom row).

Table 5: $L_2$ and $L_\infty$ Norm Errors at $T = 0.2$, $dt = 0.01$, $\delta_x = \delta_y = \delta_z = 0.05$.

| | $L_2$ | $L_\infty$ |
|---|---|---|
| Douglas-Gunn scheme (Ma, 2018) | 3.5 | 0.769 |
| 3D-PDE-Net | 0.02032988 | 0.03593109 |

**4.1 Data Preparation, Training, and Testing.** For the three-dimensional diffusion equation, 4.1, the spatial boundary is $\Omega = [0, 2\pi] \times [0, 2\pi] \times [0, 2\pi]$ :

$$\frac{\partial u}{\partial t} = c\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}\right), \tag{4.1}$$

where $(x, y, z, t) \in \Omega \times \{t > 0\}$.

In the case of $c = 0.3$, the size of the regular mesh is $50 \times 50 \times 50$ imposed on $\Omega$, and the space step is $\Delta x = \Delta y = \Delta z = 2\pi/50$. The training and testing data are given by the exact solution:

$$u = 1000e^{-0.9t} \sin(x) \sin(y) \sin(z). \tag{4.2}$$

Table 6: Errors of the Solution to Equation 4.1 at $T = 0.2$, max_order = 2.

| $\Delta t$ | *NMSE* | $L_2$ | *MAE* | *MAPE* | $L_\infty$ | $NL_\infty$ |
|---|---|---|---|---|---|---|
| 0.000625 | 0.0376337 | 572.275696 | 22.8698311 | 1.0028089 | 594.312744 | 3.17298555 |
| 0.0025 | 0.00054256 | 49.0671616 | 2.23501849 | 0.4142519 | 26.5103836 | 0.19820628 |
| 0.01 | 0.00018602 | 41.3482590 | 2.08968091 | 0.2737154 | 11.5319996 | 0.05990921 |
| 0.04 | 0.00015705 | 35.5133171 | 1.7952261 | 0.2449427 | 13.989995 | 0.07775196 |

Table 7: Errors of the Solution to Equation 4.1 at $T = 0.8$, max_order = 2.

| $\Delta t$ | *NMSE* | $L_2$ | *MAE* | *MAPE* | $L_\infty$ | $NL_\infty$ |
|---|---|---|---|---|---|---|
| 0.0025 | 0.00504207 | 87.1861191 | 3.82060003 | 0.61556971 | 34.3069229 | 0.44005689 |
| 0.01 | 0.00064279 | 44.7954369 | 2.23257637 | 0.60363084 | 22.5454159 | 0.20096888 |
| 0.04 | 0.000469653 | 35.7928543 | 1.79527438 | 0.6322729 | 32.6800957 | 0.311634362 |

The initial conditions are taken directly from this exact solution, and we use the Dirichlet boundary conditions in this experiment. For every sample, gaussian noise is added as equation 3.3.

The size of the convolution kernel, the number of network layers, and the number of samples are consistent with the parameters in the previous section of the experiment. We trained the optimum network from 30 data samples and then used the learned network for solving. It should be noted that the maximum possible order of the experiment in this section is 2.

## 4.2 Results and Discussion.

*4.2.1 Predicting Dynamics without Using the Activation Function.* For our comparison with the first-order differential equation, we observed and analyzed the three-dimensional experimental results without considering the influence of the change of hyperparameter. We calculated spatial solutions to test learned network performance at the time intervals $T = 0.2$ and $T = 0.8$. The test errors are shown in Tables 6 and 7.

We also tested the learned neural network. In the case of different $\Delta t$, the number of $\Delta t$ is used to compare the error of the numerical solution. The results for $\Delta t$ at 20 and 80 are shown in Tables 8 and 9.

The following conclusions can be drawn by comparing the test results from Tables 6 to 9:

1. When the time interval is $T = 0.2$, the errors of solution for different numbers of $\Delta t$ are small. *NMSE* shows that the overall effect is good, but *MAPE* is greater than 0.1, and the larger $NL_\infty$ also shows the larger local test error at $\Delta t = 0.00625$.

Table 8: Errors of the Solution to Equation 4.1 at $20\Delta t$, max_order = 2.

| $\Delta t$ | *NMSE* | $L_2$ | *MAE* | *MAPE* | $L_\infty$ | $NL_\infty$ |
|---|---|---|---|---|---|---|
| 0.000625 | 0.00028515 | 58.9685478 | 2.79729247 | 0.3049064 | 30.8024178 | 0.13892281 |
| 0.0025 | 0.00021219 | 35.1178742 | 1.74150527 | 0.27727368 | 14.9238386 | 0.09749512 |
| 0.01 | 0.00018602 | 41.3482589 | 2.08968091 | 0.27371544 | 11.5319996 | 0.05990921 |
| 0.04 | 0.00046965 | 35.7928543 | 1.79527438 | 0.6322729 | 32.6800957 | 0.31163436 |

Table 9: Errors of the Solution to Equation 4.1 at $80\Delta t$, max_order = 2.

| $\Delta t$ | *NMSE* | $L_2$ | *MAE* | *MAPE* | $L_\infty$ | $NL_\infty$ |
|---|---|---|---|---|---|---|
| 0.000625 | 0.0014779 | 129.799072 | 5.07427359 | 0.77636933 | 69.9301459 | 0.32619828 |
| 0.0025 | 0.0005426 | 49.0671616 | 2.23501849 | 0.41425186 | 26.5103836 | 0.19820628 |
| 0.01 | 0.0006428 | 44.7954369 | 2.23257637 | 0.60363084 | 22.5454159 | 0.20096888 |
| 0.04 | 271004.31 | 100707.023 | 347.044799 | 13.3790703 | 423570.312 | 34484.4766 |

2. When the time interval is $T = 0.8$, $NL_\infty$ indicates that the testing effect at $\Delta t = 0.01$ is better than at other $\Delta t$.

Next, we compare the scenario with the same number of $\Delta t$ at different time step $\Delta t$:

3. When $\Delta t$ is 20, *NMSE* indicates a great overall solution, and the scenario at $\Delta t = 0.01$ has the smallest $L_\infty$ error.
4. When $\Delta t$ is 80, the test effect of the scenario at $\Delta t = 0.0025$ is the best.

Based on this analysis, with time step $\Delta t$, total time interval $T$, and number of $\Delta t$ as evaluation indexes, the solution in the $\Delta t = 0.01$ case performs very well. Various error plots at $\Delta t = 0.01$ and the time interval $T = 0.8$ are shown in Figure 4.

When $\Delta t = 0.01$ and the $\Delta t$ is 20 ($T = 0.2$ ) and 80 ($T = 0.8$), sections of the 3D spatial solution images are shown in Figure 5.

*4.2.2 Predicting Dynamics Using Adaptive Activation Functions.* For common neural networks, the activation functions can accelerate the neural network training and improve the accuracy of solution. We tried to add an activation layer in 3D-PDE-Net. The LAAF mentioned in this letter is to accelerate network convergence by altering the loss landscape. Scenario 1 represents the numerical solution using the layer-wise adaptive activation function (tanh); scenario 2 represents the solution of the partial differential equation without using activation function. The errors are shown in Table 10.

The 3D-PDE-Net is made up of basic modules that are cyclically stacked, so L-LAAF is converted to GAAF. When $\Delta t = 0.01$ and the number of $\Delta t$

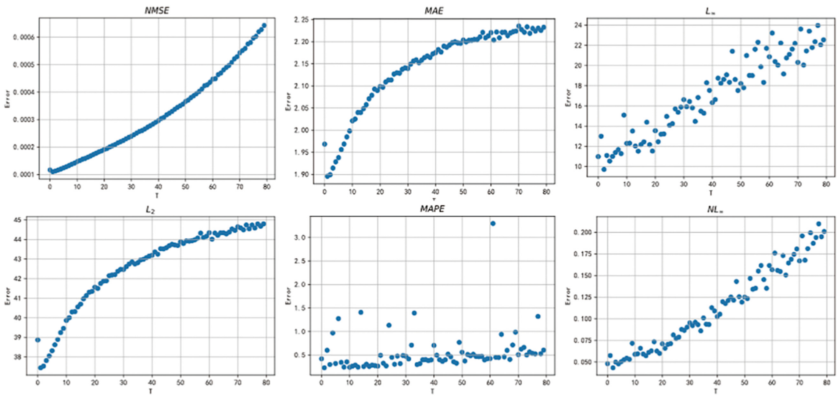

Figure 4: Errors of the Solution to Equation 4.1 at $\Delta t = 0.01$, $T = 0.8$.

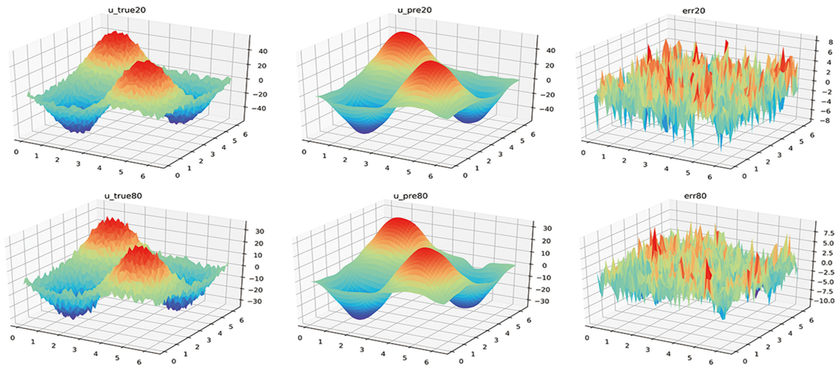

Figure 5: Sections of the 3D spatial solution to equation 4.1 at $\Delta t = 0.01$ with the number of $\Delta t$ is 20 (top row) and 80 (bottom row).

Table 10: Numerical Errors of the Solution to Equation 4.1 in Scenarios 1 and 2.

| | | Scenario 1 | | Scenario 2 | |
|---|---|---|---|---|---|
| $\Delta t$ | Number of $\Delta t$ | *NMSE* | *MAPE* | *NMSE* | *MAPE* |
| 0.0025 | 1 | 0.00012862 | 0.27094537 | 0.00011708 | 0.2836791 |
| | 20 | 0.00048785 | 0.41643637 | 0.00021219 | 0.2772737 |
| | 80 | 0.00343868 | 0.79955858 | 0.00054256 | 0.4142519 |
| 0.01 | 1 | 0.00011903 | 0.23753765 | 0.00011676 | 0.419004 |
| | 20 | 0.00040321 | 0.63613981 | 0.00018602 | 0.273715 |
| | 80 | 0.00946921 | 1.04254532 | 0.00064279 | 0.603631 |
| 0.04 | 1 | 0.0001397 | 0.27728489 | 0.000121 | 0.225869 |
| | 20 | 0.0075582 | 1.08468592 | 0.000470 | 0.632273 |
| | 80 | 0.2126092 | 2.7424283 | 2.7100E+05 | 13.379070 |

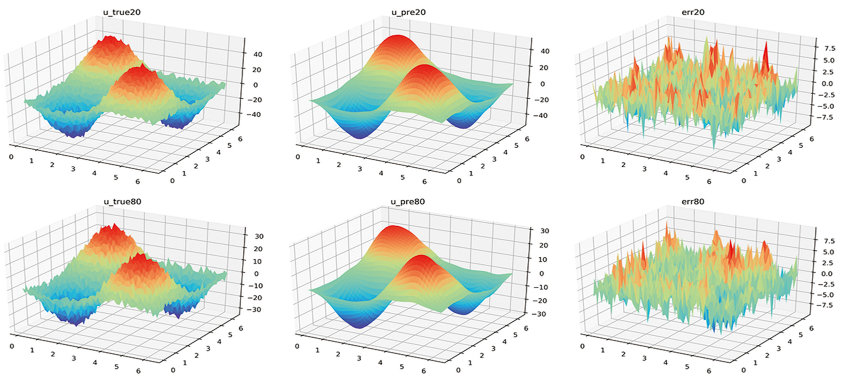

Figure 6: Scenario 1: Sections of the 3D spatial solution to equation 4.1 at $\Delta t = 0.01$ and the number of $\Delta t$ is 20 (top row) and 80 (bottom row).

is 20 ($T = 0.2$) and 80 ($T = 0.8$), sections of the 3D spatial solution images (scenario 1) are shown in Figure 6.

The following conclusions can be drawn from the above solution results. First, the activation function significantly shortens the training period by nearly 1/10. Similarly good experimental performance and smaller margin of error can be achieved in less time after adding the activation function. Figures 5 and 6 show good solution images. And when the number of $\Delta t$ is 80, the prediction error interval of scenario 2 is $(-10.0, 7.5)$, while the error interval of scenario 1 is $(-7.5, 7.5)$, which shows that the activation function has its own advantages.

Another advantage is that L-LAAF's method has better prediction precision when the time step is large. When $\Delta t = 0.04$, the prediction error obtained at $80\Delta t$ in scenario 1 is 0.2126092, while the prediction error is 2.7100E+05 in scenario 2, which shows the excellent effect of the activation function.

However, activation functions will increase the solution error when the time step is small. Because the network in this letter is different from the traditional CNN, the total *NMSE* errors in scenario 1 are slightly larger than in scenario 2 at $\Delta t = 0.0025$ and $\Delta t = 0.01$. The activation function selection and model construction have a great influence on model training and solution results.

More important is that the added activation function will weaken the interpretability of the network. Without the activation function, the 3D-PDE-Net can be expressed by mathematical formula, and the spatial coefficients formula can be obtained from the learnable parameters. When the activation function is added, the coefficients cannot be explicitly calculated and 3D-PDE-Net cannot be expressed by mathematical formula directly.

## 5 Numerical Experiment 3: Burgers Equation

Sections 3 and 4 dealt mainly with the problem of linear homogeneous equations. In this section, we introduce a forced Burgers equation with constant coefficients.

**5.1 Data Preparation, Training, and Testing.** For the three-dimensional diffusion, equation 5.1, the spatial boundary is $\Omega = [0, \pi] \times [0, \pi] \times [0, \pi]$ :

$$\frac{\partial u}{\partial t} + u\left(\frac{\partial u}{\partial x} + \frac{\partial u}{\partial y} + \frac{\partial u}{\partial z}\right) = \frac{1}{2}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}\right) + x + y + z, \quad (5.1)$$

where $(x, y, z, t) \in \Omega \times \{t > 0\}$ and $c = 0.5$, the size of the regular mesh is $50 \times 50 \times 50$ imposed on $\Omega$; the space step is $\Delta x = \Delta y = \Delta z = \pi/49$; and the training and testing data are high-precision solutions simulated by Mathematica, including the initial condition. The boundary condition is set to periodic in this experiment. For every sample, gaussian noise is added as in equation 3.3.

Hyperparameters are consistent with the parameters in the section 4. We trained the optimum network from 30 data samples and then used the well-trained network for solving the problem. The maximum possible order of the experiment in this section is also 2. The corresponding nonlinear network structure is adjusted as

$$\begin{aligned}\hat{u}(t_{i+1}, \cdot) = D_0 u(t_i, \cdot) + \delta t \times (&c_{000} D_{000} u + c_{100} u D_{100} u + c_{010} u D_{010} u, \ldots \\ &+ c_{002} D_{002} u + \tilde{f}(x, y, z)).\end{aligned}$$

$\tilde{f}(x, y, z)$ can be interpolated by $x, y, z$ or learned directly, and three-dimensional Lagrange interpolation is used in this experiment.

**5.2. Results and Discussion.** The second-order equation may get better solution results at $\Delta t = 0.0025$. Considering the complexity of the equation and computational cost, this section mainly focuses on $\Delta t = 0.0025$ and compares the errors of numerical solutions with different numbers of $\Delta t$. Also, we do not consider the influence of other superparameter changes. The solution errors are shown in Table 11.

From Table 11 and Figure 8, we get the following obvious results:

1. On the whole, *NMSE* shows that the 3D-PDE-Net model can obtain high precision results in solving equation 5.1. $L_2$ and *MAE* show that the error of the numerical solution solved by the network is relatively small—for example, the mean absolute error is less than 0.02. Also, the larger the number of $\Delta t$, the greater errors there are.

Table 11: Errors of the Solution to Equation 5.1 at $\Delta t = 0.0025$, max_order = 2.

| Number of $\Delta t$ | *NMSE* | $L_2$ | *MAE* | *MAPE* |
|---|---|---|---|---|
| 5 | 0.00014521 | 0.05129173 | 0.00712171 | 0.55777174 |
| 10 | 0.00021479 | 0.06304602 | 0.00869071 | 0.9814496 |
| 20 | 0.00032384 | 0.07930905 | 0.01082481 | 1.19216216 |
| 80 | 0.00066307 | 0.13017096 | 0.01607719 | 4.0829854 |

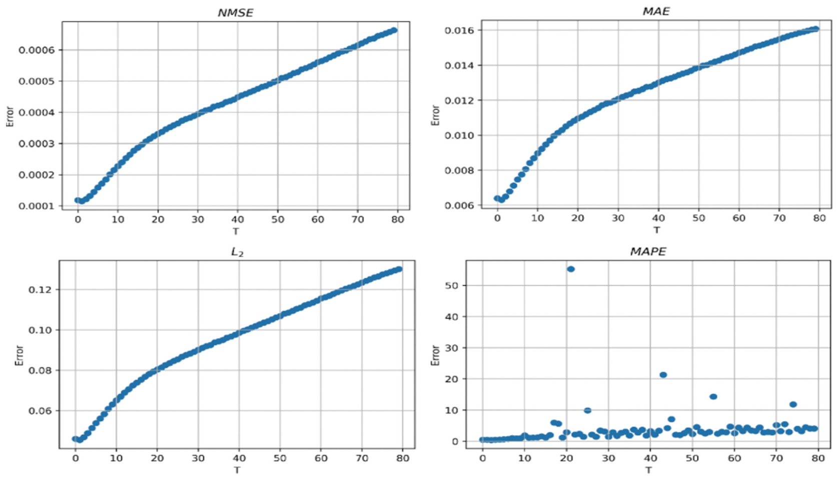

Figure 7: Errors of the solution to equation 5.1 at $\Delta t = 0.0025$, $T = 0.2$.

2. All *MAPE* are greater than 0.1, which also shows that the local errors of solutions are relatively large. Figure 8 also shows large errors, mainly at the boundary.

Figure 7 shows the error within the interval $(0, 0.2]$ after giving initial solution $u_0$.

When $\Delta t = 0.0025$, the number of $\Delta t$ is 20 ($T = 0.05$) and 80 ($T = 0.2$), sections of the 3D spatial solution images are shown in Figure 8.

## 6 Conclusion

We have proposed a three-dimensional network, 3D-PDE-Net, to solve 3D unsteady PDEs, based on a convolutional neural network. This method mainly learns differential operators by learning convolution kernels, which is guaranteed by the mathematical theoretical justification from Taylor's expansion. The 3D-PDE-Net does not need any prior knowledge except the highest possible order.

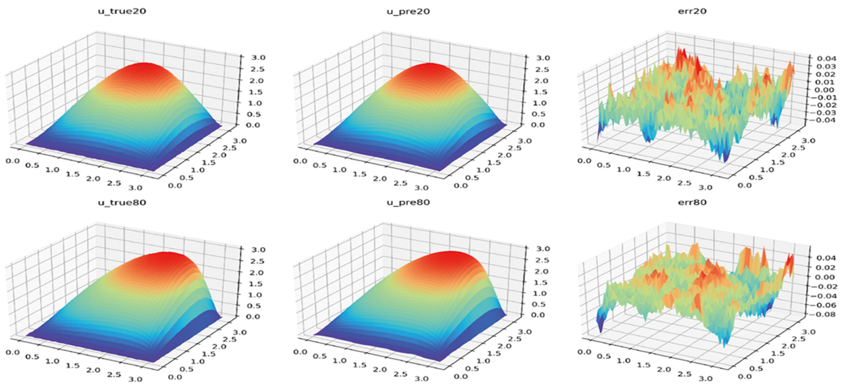

Figure 8: Sections of the 3D spatial solution to equation 5.1 at $\Delta t = 0.0025$, when the number of $\Delta t$ is 20 (top row) and 80 (bottom row).

The 3D-PDE-Net as a PDE solver has good performance. Numerical experiment shows that the solution of the method outperforms that of a Douglas-Gunn scheme. We carried out convection and diffusion equations as numerical experiments to support our theoretical analysis. Although 3D-PDE-Net is not as effective in nonlinear Burgers equation problems as in solving homogeneous convection or diffusion equations, it still has good performance and potential for development potential. We are adding activation functions to accelerate the neural network training and will be conducting research on that in the future.

## Acknowledgments

This work was supported by the National Natural Science Foundation of China (grant 1217020361).

## References

Anger, G. (1990). *Inverse problems in differential equations*. New York: Plenum Press.

Brunton, S. L., Proctor, J. L., & Kutz, J. N. (2016). Discovering governing equations from data by sparse identification of nonlinear dynamical systems. In *Proceedings of the National Academy of Sciences*, *113*, 3932–3937. 10.1073/pnas.1517384113

Cai, J. F., Dong, B., Osher, S., & Shen, Z. W. (2012). Image restoration: Total variation, wavelet frames, and beyond. *Journal of the American Mathematical Society*, *25*, 1033–1089. 10.1090/S0894-0347-2012-00740-1

Cao, F., & Ge, Y. (2011). A high-order compact ADI scheme for the three-dimensional unsteady convection-diffusion equation. In *Proceedings of the 2011 International Conference on Computational and Information Sciences* (pp. 1087–1090). Los Alamitos, CA: IEEE Computer Society.

Chang, H. B., & Zhang, D. X. (2019). Identification of physical processes via combined data-driven and data-assimilation methods. *Journal of Computational Physics*, *393*, 337–350. 10.1016/j.jcp.2019.05.008

Dai, W., & Nassar, R. (1998). A second-order ADI scheme for three-dimensional parabolic differential equations. *Numerical Methods for Partial Differential Equations*, *14*, 159–168. 10.1002/(SICI)1098-2426(199803)14:2⟨159::AID-NUM2⟩3.0.CO;2-N

Dong, B., Jiang, Q. T., & Shen, Z. W. (2017). Image restoration: Wavelet frame shrinkage, nonlinear evolution PDEs, and beyond. *Multiscale Modeling and Simulation*, *15*, 606–660. 10.1137/15M1037457

Douglas, J. , & Gunn, J. E. (1964). A general formulation of alternating direction methods, I. Parabolic and hyperbolic problems. *Numer. Math.*, *6*, 428–453. 10.1007/BF01386093

Ge, Y., Tian, Z. F., & Zhang, J. (2013). An exponential high-order compact ADI method for three-dimensional unsteady convection–diffusion problems. *Numerical Methods for Partial Differential Equations*, *29*, 186–205. 10.1002/num.21705

Hayden, S. (2017). Learning partial differential equations via data discovery and sparse optimization. In *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences, 473*, 0446.

Jagtap A. D., Kawaguchi, K., & Karniadakis G. E. (2020a). Locally adaptive activation functions with slope recovery for deep and physics-informed neural networks. In *Proc. R. Soc. A*, *476*, 20200334. 10.1098/rspa.2020.0334

Jagtap, A. D., Kawaguchi K., & Karniadakis G. E. (2020b). Adaptive activation functions accelerate convergence in deep and physics-informed neural networks. *J. Comput. Phys.*, *404*, 109136. 10.1016/j.jcp.2019.109136

Karaa, S. (2006). A high-order compact ADI method for solving three-dimensional unsteady convection-diffusion problems. *Numerical Methods for Partial Differential Equations*, *22*, 983–993. 10.1002/num.20134

Karaa, S., & Zhang, J. (2004). High order ADI method for solving unsteady convection-diffusion problems. *Journal of Computational Physics*, *198*, 1–9. 10.1016/j.jcp.2004.01.002

Lagaris, I. E., & Likas, A. (1998). Artificial neural networks for solving ordinary and partial differential equations. *IEEE Trans. Neural Netw.*, *9*, 987–1000. 10.1109/72.712178, PubMed: 18255782

Liewa, K. M., & Cheng, Y. M. (2009). Complex variable boundary element-free method for two-dimensional elastodynamic problems. *Computer Methods in Applied Mechanics and Engineering*, *198*, 3925–3933. 10.1016/j.cma.2009.08.020

Liu, D., & Cheng, Y. M. (2019). The interpolating element-free Galerkin (IEFG) method for three-dimensional potential problems. *Engineering Analysis with Boundary Elements*, *108*, 115–123.

Long, Z. C., Lu, Y. P., & Dong, B. (2019). PDENet 2.0: Learning PDEs from data with a numeric-symbolic hybrid deep network. *Journal of Computational Physics*, *399*, 108925. 10.1016/j.jcp.2019.108925

Long, Z. C., Lu, Y. P., Ma, X. Z., & Dong, B. (2018). PDE-Net: Learning PDEs from data. In *Proceedings of the International Conference on Machine Learning* (pp. 3208–3216).

Ma, T. L. (2018). *Software development for numerical solutions of partial differential equations with finite difference method.* Master's thesis, Ningxia University, China.

Mai-Duy, N., & Tran-Cong, T. (2001). Numerical solution of differential equations using multiquadric radial basis function networks. *Neural Netw.*, *14*, 185–199. 10.1016/S0893-6080(00)00095-2, PubMed: 11316233

Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2017a). *Physics informed deep learning (part I): Data-driven solutions of nonlinear partial differential equations*. arXiv:1711.10561.

Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2017b). *Physics informed deep learning (part II): Data-driven discovery of nonlinear partial differential equations*. arXiv:1711.10566.

Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2018). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, *378*, 686–707. 10.1016/j.jcp.2018.10.045

Reinbold Patrick, A. K., & Grigoriev Roman, O. (2019). Data-driven discovery of partial differential equation models with latent variables. *Physical Review E*, *100*, 022219. 10.1103/PhysRevE.100.022219, PubMed: 31574616

Rudy, S. H., Brunton, S. L., Proctor, J. L., & Kutz, J. N. (2017). Data-driven discovery of partial differential equations. *Science Advances*, *3*, 1602614. 10.1126/sciadv.1602614

Sun, H. L., Hou, M. Z., Yang, Y. L., Zhang, T., Weng, F., & Han, F. (2019). Solving partial differential equation based on Bernstein neural network and extreme learning machine algorithm. *Neural Processing Letters*, *50*, 1153–1172. 10.1007/s11063-018-9911-8

Verwer, J. G. (1996). Explicit Runge-Kutta methods for parabolic partial differential equations. *Applied Numerical Mathematics*, *22*, 359–379. 10.1016/S0168-9274(96)00022-0

Wang, Z. C., Li, X. Y., Shi, H. B., Li, W. P., Yang, W. H., & Qin, Y. M. (2020). Estimating the water characteristic curve for soil containing residual plastic film based on an improved pore-size distribution. *Geoderma*, *370*, 114341. 10.1016/j.geoderma.2020.114341

Winovich, N., Ramani, K., & Lin, G. (2019). ConvPDE-UQ: Convolutional neural networks with quantified uncertainty for heterogeneous elliptic partial differential equations on varied domains. *Journal of Computational Physics*, *394*, 263–279. 10.1016/j.jcp.2019.05.026

Wu, B., & White, R. E. (2004). One implementation variant of finite difference method for solving ODEs/DAEs, *Computers and Chemical Engineering*, *28*, 303–309. 10.1016/j.compchemeng.2003.06.002

Wu, K. L., & Xiu, D. B. (2020). Data-driven deep learning of partial differential equations in modal space. *Journal of Computational Physics*, *408*, 109307. 10.1016/j.jcp.2020.109307

Received May 16, 2021; accepted August 31, 2021.