NEURAL PARTIAL DIFFERENTIAL EQUATIONS WITH FUNCTIONAL CONVOLUTION

Anonymous authors
Paper under double-blind review

ABSTRACT

We developed a lightweighted neural PDE representation to discover the hidden structure and predict the solution of different nonlinear PDEs. Our key idea is to leverage the prior of "translational similarity" of numerical PDE differential operators to drastically reduce the scale of learning model and training data. We implemented three central network components, including a neural functional convolution operator, a Picard forward iterative procedure, and an adjoint backward gradient calculator. Our novel paradigm fully leverages the multifaceted priors that stem from the sparse and smooth nature of the physical PDE solution manifold and the various mature numerical techniques such as adjoint solver, linearization, and iterative procedure to accelerate the computation. We demonstrate the efficacy of our method by robustly discovering the model and accurately predicting the solutions of various types of PDEs with small-scale networks and training sets. We highlight that all the PDE examples we showed were trained with up to 8 data samples and within 325 network parameters.

1 INTRODUCTION

The advent of data-driven techniques are reforming the study of partial differential equations (PDEs). On the one hand, machine learning methods are improving the speed and accuracy of conventional forward numerical solvers to solve a given PDE (e.g. Han et al., 2016). On the other hand, data-driven paradigms are opening up new possibilities for solving inverse problems to discover the hidden structures of an unknown PDE based on limited observation (e.g. Brunton et al., 2016; Long et al., 2018a; Raissi et al., 2019). The hybridization between inductive solvers and deductive models further enables a new group of neural simulators (Kipf et al., 2017; Mrowca et al., 2018; Ajay et al., 2018; Li et al., 2019; 2020; Yang et al., 2020) that can faithfully predict the evolution of complex, differential dynamics. Among all these approaches, an *effective neural representation* of a PDE system plays an essential role to facilitate efficient data processing and pattern discovery. In particular, we expect a practical representation endorsing data-driven PDE algorithms to fuse the mathematical equation's essential features, the numerical solvers' computational efficiency, and the neural networks' expressive power. In retrospect, the design of neural PDE representations has been evolving from the raw, unstructured networks (e.g., by direct end-to-end data fitting) to various structured ones with proper mathematical priors embedded. Examples include the residual-based loss function (e.g., physics-informed networks Raissi et al., 2020; Lu et al., 2019; Raissi et al., 2019), learnable convolution kernels (e.g., PDE-Nets Long et al., 2018a;b; 2019), and hybrid of numerical stencils and MLP layers (e.g., see Amos & Kolter, 2017; Pakravan et al., 2020; Geng et al., 2020; Stevens & Colonius, 2020). The nonlinearity and the curse of dimensionality of the target PDE's solution manifold are the two main hinders for the design of a high-performance neural discretization.

In this paper, we aim to devise a novel neural PDE representation that can embed strong structural priors into a versatile learning model to uncover the hidden patterns and deliver accurate predictions of various nonlinear PDE systems from scarce observation data. In particular, we want to aggressively reduce the *scale* of both model parameters and training data to some extremal extent, while extending the *scope* of the targeted PDE systems to a broad range, encompassing equations that are both linear and nonlinear, both steady-state and dynamic. Our neural PDE representation design is motivated by the historical successes of the various *sparse*, *iterative* numerical solvers in solving nonlinear PDEs over the past decades. The key observation we have made is that the efficacy of a classical

numerical PDE solver relies on the *translational similarity* of its discretized differential operators. Namely, the form of a differential operator can be written as a functional $\mathcal{C}(p, x)$ with respect to the *local* position p and the PDE unknown x. For example, for a linear Poisson system $\nabla \cdot \nabla x = b$, $\mathcal{C}$ is a constant function; for a linear Poisson system with embedded boundary conditions, $\mathcal{C}$ is a function of position p; for a nonlinear PDE $\nabla \cdot (1 + x^2)\nabla x = b$, $\mathcal{C}$ is a function of PDE unknown x (or both x and p if it has embedded boundaries). For most numerical PDEs, these local functional operators can be parameterized and built *on-the-fly* within the solver iterations. Such operators' locality further inspired the design of a variety of computationally efficient PDE solvers, among which the most famous one is the matrix-free scheme (Bridson, 2015) that has been used widely in solving large-scale fluids or solids on GPU.

In this work, **we leverage the PDE differential operators' "translational similarity" in a reverse manner by devising a local neural representation that can uncover and describe the global structure of the target PDE**. At the heart of our approach lies in a differential procedure to simultaneously describe the spatial coupling and the temporal evolution of a local data point. Such procedure is implemented as a parameterized micro network, which is embedded in our iterative solving architecture, to learn the numerical process of converging from an initial guess to a final steady state for a PDE solution. We name these embedded micro networks "*functional convolutions*," because 1) fitting the parameters of these local embedded networks amounts to the exploration of the optimal function that best describes the observed solution of the unknown nonlinear PDE within a functional space, and 2) the local differential operators that span this functional space can be treated as numerically employing convolution kernels Hsieh et al. (2018); Lin et al. (2013). Based on these functional convolutions, we are able to devise a learning framework by embedding the micro network architecture within an iterative procedure to 1) backwardly learn the underpinning, spatially varying structures of a nonlinear PDE system by iteratively applying the adjoint linear solvers and 2) forwardly predict the steady states of a PDE system by partially observing data samples of its equilibrium. We show that our model can simultaneously discover structures and predict solutions for different types of nonlinear PDEs. We particularly focus on solving elliptic boundary value problems that were less explored in the current literature.

2 METHOD

Our neural PDE representation consists of three key components: the functional convolution operator, the Picard forward iterations, and the adjoint backward derivative calculation. We will introduce each components as follows.

2.1 FUNCTIONAL CONVOLUTION

In a conventional ML method, $\boldsymbol{C}$ can be approximated by the combination of a set of kernels Long et al. (2018a) or by solving a deconvolution problem Xu et al. (2014); Jin et al. (2017); Zhang et al. (2017). These strategies are not suited for our scenario, where the effects of $\boldsymbol{C}$ are accumulated on the final steady-state solution nonlinearly over time with the discretized kernel varies according to the system's instant states. Essentially, we want to approximate the nonlinear effects of $\boldsymbol{C}$, which characterises of the nonlinear PDEs, through the convolutional approximation of its instant linear effects (by constructing the global $\boldsymbol{A}$) over iterations.

A natural choice to approximate this spatially-and-temporally varying kernel function is to devise a neural network, which takes the form of $\boldsymbol{C}(\boldsymbol{x}(\boldsymbol{p}), \boldsymbol{p}, \theta)$, with θ as the network parameters. Because we want to employ $\boldsymbol{C}$ over every discretized data point in the domain, the size of the network needs to be small. Numerically, the global matrix $\boldsymbol{A}$ can be fully parameterized by $\boldsymbol{C}(\boldsymbol{x}(\boldsymbol{p}), \boldsymbol{p}, \theta)$ and the current state of $\boldsymbol{x}$, by assuming the fact that $\boldsymbol{C}$ is a non-weight-shared convolution operator in the spatial domain. As illustrated in Figure 1, such neural network can be further incorporated into a conventional nonlinear Picard solver to obtain the forward steady-state solution. In each Picard iteration, we are solving the linear system $\boldsymbol{A}(\boldsymbol{x}_n, \theta)\boldsymbol{x}_{n+1} = \boldsymbol{b}(\boldsymbol{x}_n)$, where a black-box sparse linear solver can be used as in a conventional simulation program.

The formal definition of functional convolution written in the kernel way is

$$\boldsymbol{A}(\boldsymbol{x}(p_i), \theta) = \sum_{p_j \in \mathcal{N}(p_i)} [\boldsymbol{C}(\boldsymbol{x}(\mathcal{N}(p_i)), \mathcal{N}(p_i), \theta)](p_j)\boldsymbol{x}^t(p_j) \tag{1}$$

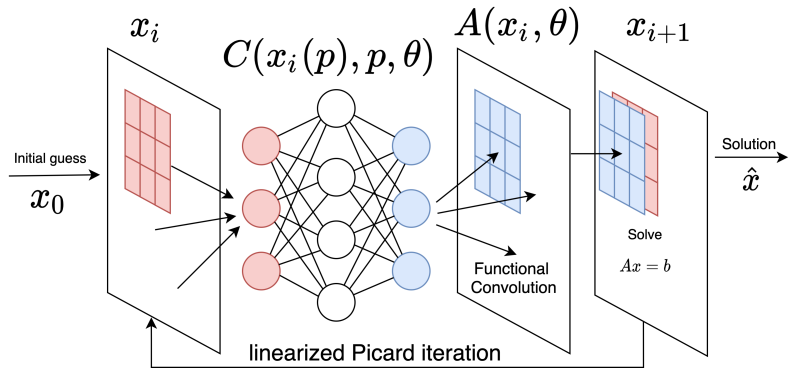


Figure 1: Overview: To obtain the solution of $A(x, \theta)x = b$, we first generate a random initial guess x_0. Then in each iteration step i, we apply the functional convolution C on x_i to obtain the convolution kernels $A(x_i, \theta)$. We obtain x_{i+1} by solving $A(x_i, \theta)x = b$. We repeat this process to obtain a converged solution.

where $\mathcal{N}(p_i)$ is the neighboring positions of p_i and $\boldsymbol{x}(\mathcal{N}(p_i))$ is all the neighboring elements (all channels) around the position p_i.

To better understand the functional convolution, we take a simple 2D example here. The traditional 3×3 weight-shared convolution is

$$A(x_{m,n}) = \sum_{i=-1}^{1} \sum_{j=-1}^{1} w_{i,j} x_{m+i,n+j} \tag{2}$$

where $x_{m,n}$ is the element that lies in row m, column n of the feature map. For simplicity, we ignore the bias and activation function, and assume the number of both input and output channel is 1. Our proposed function-shared-convolution replaces the constant w with a local function $\boldsymbol{C}$:

$$A(x_{m,n}, \theta) = \sum_{i=-1}^{1} \sum_{j=-1}^{1} [\boldsymbol{C}(\mathcal{N}(x_{m,n}), \theta)]_{i,j} x_{m+i,n+j} \tag{3}$$

where the input $\mathcal{N}(x_{m,n}) = \{x_{m+i,n+j} \text{ for } i, j = -1, 0, 1\}$ is the flatten vector of neighboring elements of pixel $x_{m,n}$. After a simple neural network $\boldsymbol{C}$ with parameters θ, we obtain the output $C(\mathcal{N}(x_{m,n}), \theta)$, which is a vector of the same length of $\mathcal{N}(x_{m,n})$. Every element in the output is a weight corresponding to a element in the input. Note that in (2), the kernel weights are constant throughout the feature map, while in 3, the kernel weights are the output of function $\boldsymbol{C}$. In this example, the functional convolution (embedded small network) $\boldsymbol{C}$ is a function of neighbor pixels.

2.2 Picard iteration for linear embedding

Let's consider the nonlinear Poisson equation with Dirichlet boundary conditions as a model problem to demonstrate the classical iterative method to obtain numerical solutions:

$$-\nabla \cdot \alpha(\boldsymbol{x}) \nabla \boldsymbol{x} = \mathbf{b}(\boldsymbol{x}) \quad \boldsymbol{x} \in \Omega \tag{4a}$$

$$\boldsymbol{x} = \hat{\boldsymbol{x}} \quad \boldsymbol{x} \in \partial\Omega \tag{4b}$$

The source of nonlinearity in this PDE includes the coefficient $\alpha(\boldsymbol{x})$ which is dependent on the solution of $\boldsymbol{x}$, e.g., $\alpha(\boldsymbol{x}) = 1 + \boldsymbol{x}^2$. A simple and effective fixed-point procedure, named Picard iteration, to discretize and solve the nonlinear Poisson equation can be sketched as follows:

$$\begin{aligned} &\text{while: } |\boldsymbol{x}_n - \boldsymbol{x}_{n-1}| > \epsilon \\ &\qquad \boldsymbol{x}_{n+1} = A^{-1}(\boldsymbol{x}_n)\boldsymbol{b}(\boldsymbol{x}_n), \end{aligned} \tag{5}$$

with the matrix $\boldsymbol{A}$ representing the current discretized nonlinear Laplacian operator approximated by the value of unknowns from the previous iteration. The key idea is to employ a linear approximation

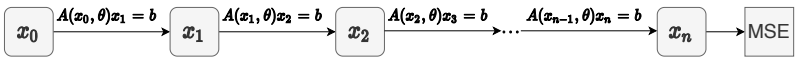


Figure 2: Forward data flow in a neural Picard solver to predict the solutions of a non-linear PDE.

of a nonlinear problem and enforce such approximation iteratively to evolve to a steady state (see Figure 2).

To uncover the underlying structure of $\boldsymbol{A}$, which can evolve both spatially and temporally, we make a prior assumption that $\boldsymbol{A}$ can be described by a kernel function $\boldsymbol{C}(\boldsymbol{x}(\boldsymbol{p}), \boldsymbol{p})$ that takes the information of $\boldsymbol{x}$ within a local region around $\boldsymbol{p}$ as its input. Such prior applies to most of the elliptic PDE systems where the spatial terms can be expressed as the combination of one or several differential operators. From a numerical perspective, $\boldsymbol{C}$ describes the local discretized interaction between an element and its neighbours. It amounts to a function that returns all the non-zero elements for each row i in $\boldsymbol{A}$ (think of $\boldsymbol{A}$ in a matrix-free way).

2.3 Adjoint derivatives

To train our functional convolution operator in an end-to-end fashion, we need to calculate the derivatives of the loss function L regarding the network parameters θ following the chain rule. For a neural PDE with n layers for the outer Picard loop (see Figure 1), we can calculate the sensitivity $\partial L/\partial\theta$ as:

$$\frac{\partial L}{\partial \theta} = \frac{\partial L}{\partial \boldsymbol{x}_n}\frac{\partial \boldsymbol{x}_n}{\partial \theta} + \frac{\partial L}{\partial \boldsymbol{x}_n}\frac{\partial \boldsymbol{x}_n}{\partial \boldsymbol{x}_{n-1}}\frac{\partial \boldsymbol{x}_{n-1}}{\partial \theta} + \cdots + \frac{\partial L}{\partial \boldsymbol{x}_n}\frac{\partial \boldsymbol{x}_n}{\partial \boldsymbol{x}_{n-1}}\cdots\frac{\partial \boldsymbol{x}_2}{\partial \boldsymbol{x}_1}\frac{\partial \boldsymbol{x}_1}{\partial \theta} \tag{6}$$

For layer i that maps from $\boldsymbol{x}_i \to \boldsymbol{x}_{i+1}$ by solving $\boldsymbol{A}(\boldsymbol{x}_i, \theta)\boldsymbol{x}_{i+1} = \boldsymbol{b}$, we can express its backward derivative $\boldsymbol{y}_i$ as

$$\boldsymbol{y}_i = \frac{\partial \boldsymbol{x}_{i+1}}{\partial \boldsymbol{x}_i} = -\boldsymbol{A}^{-1}(\boldsymbol{x}_i, \theta)\frac{\partial \boldsymbol{A}(\boldsymbol{x}_i, \theta)}{\partial x_i}\boldsymbol{x}_{i+1}, \tag{7}$$

which can be solved by two linear systems, including a forward equation and a backward adjoint equation:

$$\boldsymbol{A}(\boldsymbol{x}_i, \theta)\boldsymbol{x}_{i+1} = b \tag{8a}$$

$$\boldsymbol{A}(\boldsymbol{x}_i, \theta)\boldsymbol{y}_i = \frac{\partial \boldsymbol{A}(\boldsymbol{x}_i, \theta)}{\partial \boldsymbol{x}_i}\boldsymbol{x}_{i+1} \tag{8b}$$

Similarly, we can get $\partial L/\partial\theta$ as:

$$\boldsymbol{z}_i = \frac{\partial \boldsymbol{x}_{i+1}}{\partial \theta} = -\boldsymbol{A}^{-1}(\boldsymbol{x}_i, \theta)\frac{\partial \boldsymbol{A}(\boldsymbol{x}_i, \theta)}{\partial x_i}\boldsymbol{x}_{i+1} \tag{9}$$

which can be calculated by solving one additional adjoint linear system:

$$\boldsymbol{A}(\boldsymbol{x}_i, \theta)\boldsymbol{z}_i = \frac{\partial \boldsymbol{A}(\boldsymbol{x}_i, \theta)}{\partial \theta}x_{i+1} \tag{10}$$

Algorithm 1 Backward derivative of a nonlinear PDE boundary value problem

Input: $\boldsymbol{x}_1, \boldsymbol{b}, \boldsymbol{C}, L$
Output: $\partial L/\partial\theta$
//Forward linearization:
for $i = 0 \to N-1$ **do**
  Solve $\boldsymbol{A}(\boldsymbol{x}_i, \theta)x_{i+1} = \boldsymbol{b}$;
end for
//Backward adjoint:
for $i = N-2 \to 1$ **do**
  Solve adjoints (8b) and (9) for $\partial \boldsymbol{x}_{i+1}/\partial\theta$
  $\partial L/\partial\theta \mathrel{+}= (\partial L/\partial \boldsymbol{x}_{i+1})(\partial \boldsymbol{x}_{i+1}/\partial\theta)$
end for

To calculate the terms $\partial \boldsymbol{A}/\partial \boldsymbol{x}_i$ and $\partial \boldsymbol{A}/\partial\theta$ in the chain, we will take advantage of the sparse nature of $\boldsymbol{A}$ by calculating $\partial \boldsymbol{C}/\partial \boldsymbol{x}_i$ and $\partial \boldsymbol{C}/\partial\theta$ first and then distribute their values into the non-zero entries of the global matrix. Because $\boldsymbol{C}(\boldsymbol{x}(\boldsymbol{p}), \boldsymbol{p}, \theta)$ is a functional convolution operator represented by a standard neural network, its derivatives regarding the first-layer input $\boldsymbol{x}$ and the parameters θ can be calculated in a straightforward way by the auto-differentiation mechanism. The overall algorithm can be summarized as in Algorithm 1.

3 NETWORK ARCHITECTURE AND DATA TRAINING

The network we use in this study is summarized in Table 2. We use a hybrid method of IpOpt Wächter (2009) and Adam optimizer to optimize the parameters int the neural networks. Despite its fast converging rate, IpOpt optimizer typically converges to a local minimum or fail to converge due to a bad initialization. By introducing the hybrid optimization method, we are able to solve the problem. The optimized parameters from IpOpt are then oscillated and refined by Adam optimizer for a certain number of iterations. In case that the loss not converge, we then use the optimized parameters from Adam as the initial guess to warm-start the IpOpt and repeat this IpOpt-Adam process until the loss converges to the tolerance. For the Adam optimizer, we set the parameters to be $learning\ rate = 1e-3,\ \beta_1 = 0.9,\ \beta_2 = 0.999,\ \epsilon = 1e-8$. We compare the performance of our hybrid optimization method with Adam and a typical SGDM method in solving the 1D Poisson problem in Figure 6. The results show that the hybrid method not only obtains a faster converging rate, but also converges to an extremely small loss compared with other methods.

4 RESULTS

We test the efficacy of our method by using it to fit a set of PDEs, ranging from the standard linear Poisson's problem to highly nonlinear temporally evolving PDEs. We build our neural network architecture by implementing both the differentiable Picard iterations and a simple fully-connected network in C++ 17. The training process for all the examples was run on an i7-8665 four-core desktop. The PDE and boundary conditions, sample size, and test results were summarized in Table 2 in Appendix.

4.1 COMPARISON TO CNN

We evaluate our functional convolution model by comparing its performance with other naive convolutional neural network structures in solving two typical problems targeted in this study. For the first case to solve the 1D Poisson equation, we set up the baseline structure as a 5-layer network consisting of three 1D convolution layers with a ReLU layer in between each two. The second problem is the 2D time-dependent wave equation specified by Equation $\nabla \cdot \nabla \boldsymbol{x} = \frac{\partial^2 \boldsymbol{x}}{\partial t^2}$. The baseline for this problem is also set to be a 5-layer network, which is described in Table 1.For each of the baselines, the input is set to be the right hand side value and the output is the predicted solution. The results are shown in Figure 7, which demonstrates that our functional convolution model outperforms the naive convolutional neural network both in accuracy and converging rate.

4.2 1D EXAMPLES

Constant kernels We first test our framework by fitting the underlying structure of a standard 1D Poisson problem $\nabla \cdot \nabla \boldsymbol{x} = b$ with Dirichlet boundary conditions. Here the target true solution is set to be $x = (ap)^2$, with p as the spatial coordinate of the data sample and a as a coefficient that varies for different data sample. We use a $3 \times 4 \times 4 \times 3$ fully-connected network structured as one ReLU layer in-between two linear layers. The training dataset consists of four sample points, observed from the two solutions of the equations parameterized with different values of a and boundary conditions. The 4-size training data is sampled on a 128×1 grid with two layers of picard network. After training, we run 16 PDE tests with varying a and boundary conditions and obtain an averaged MSE loss as 8.3e-20. The predicted solutions in all 16 test cases match the analytical curves with the maximum MSE 1.5e-14, as shown in in Figure 8a.

Next, we test the same framework on another two Poisson problems. The first problem is $\nabla \cdot \nabla \boldsymbol{x} = 0$ with Neumann boundary conditions which does not have an analytical solution (see Figure 8b). The prediction results from our model are shown in Figure 8b with an averaged MSE 5.0e-29. The other problem is with the target solution as $x = sin(ap)$. The results are shown in Figure 3 with the 9.8e-4 MSE loss compared with the true solution. We further conducted a stability test by adding noise values within the scale of $[-.1, .1]$ (10% of the extreme values) and showed that our method can still predict the PDE solution with an MSE error around 2.4e-3.

Spatially varying coefficients Next we test our model by predicting the solutions of a Poisson PDE with spatially varying coefficients. The PDE has the analytical formula $\nabla \cdot (1 + |\pi p|)\nabla \boldsymbol{x} = 0$, with p as the spatial coordinate of the data sample. We build the training dataset by solving the equation with randomly set boundary conditions on a 32×1 grid. The data sample size is 4 and each data sample has the full observation of the solution space. We build our functional convolution operator as a $6 \times 6 \times 6 \times 6 \times 6 \times 3$ fully-connected network with 5 layers. The input of the network is the current values of $\boldsymbol{x}$ and the sample positions $\boldsymbol{p}$. We show that our model can precisely uncover the distribution of the hidden latent space behind the Laplacian operator with spatially varying coefficients by fitting a functional convolution operator that predicts the solutions of the PDE in the forward process (see Figure 1). The averaged MSE loss is 1.3e-06 for this case.

Non-linear equations In this example, we demonstrate the ability of our neural PDE solver by solving a non-linear PDE with the form: $\nabla \cdot (1 + |x| + sin(|x| * .001))\nabla x = 0$ Dirichlet boundary conditions are set on the two endpoints of the domain. The training dataset is generated by solving the PDE with standard Picard iterations. The number of the neural PDE network layers is set to be 5. The functional convolution operator as a $3 \times 6 \times 6 \times 3$ fully-connected network with 3 layers. We employ 4 solution samples on a 32×1 discretization for training. As shown in Figure 9, our model can precisely uncover the hidden nonlinear structures of the PDE kernel and predicts the numerical solution by employing the learned functional convolution through the Picard iterations.

4.3 2D EXAMPLES

Poisson equation We then expand our model to solve 2D problems. Similarly, we start from employing our model to predict the solutions for two standard 2D Poisson problems $\nabla \cdot \nabla \boldsymbol{x} = b$ with Dirichlet boundary conditions, whose target true solutions are set to be $\boldsymbol{x} = (ap_u)^3 + ap_v^2$ and $\boldsymbol{x} = sin(ap_u + ap_v)$ respectively. Here p_u and p_v refer to the x axis and y axis coordinates of the data sample. The neural network used is a $3 \times 14 \times 14 \times 14 \times 14 \times 3$ fully-connected structure with one ReLU layer in-between two linear layers for the first problem, and $3 \times 10 \times 10 \times 10 \times 10 \times 10 \times 10 \times 3$ for the second problem. We use 4-size data samples for both the two cases which are sampled on a 32*32 grid. After training, we run 16 test cases for both the two problems and the results are shown in Figure 3. We obtain an averaged MSE loss at 5.4e-14 for the first problem and 3.7e-3 for the second problem.

Helmholtz equation In this example, we test our model's performance by predicting the solution of Helmholtz equation $\nabla \cdot \nabla \boldsymbol{x} + \boldsymbol{x} = 0$. We set two different types of varying Dirichlet boundary conditions for this problem, one as $\boldsymbol{x} = a/(p_u^2 + p_v^2)$ and another as $\boldsymbol{x} = a * sin((0.02p_u + 0.02p_v))$ with a varies across the data. We use a $3 \times 5 \times 5 \times 5 \times 5 \times 3$ fully connected 5-layer network for this example, which is trained by dataset of 4 data sample according to the two types of boundary conditions with varying coefficients. For each type of boundary conditions, the training data is sampled on a 32×32 grid in two different domains respectively. The results are exhibited in Figure 4. In predicting solution for the first type of boundary condition, we achieve an averaged MSE of 5.25e-27 and 9.3e-29 in the two specific domains. For the second type of boundary condition, our predicted solution achieves an averaged MSE of 3.5e-25 and 3.9e-18 in the two specific domains.

Time-dependent wave equation Here in this example, we demonstrate the ability of our Neural PDE solver by solving a time-dependent wave equation: $\nabla \cdot \nabla \boldsymbol{x} = \frac{\partial^2 \boldsymbol{x}}{\partial t^2}$ We use a standard finite difference method to generate the training data, which is 6-size data sample with each training sample indicating the target solution at the n time step ($1 < n < 6$). Our model is trained to map $\boldsymbol{x}$ from the $n - 1$ frame to the n frame. The training data is sampled on a 49×49 grid with a source of $\boldsymbol{x} = sin(60(n * dt))$ at the center of the grid, where dt is the time interval. We build the functional convolution network to be $3 \times 5 \times 5 \times 5 \times 5 \times 3$ fully-connected 5-layer network, whose input is the current value of $\boldsymbol{x}$. With this observation of previous frames of a time-related wave function, our model is able to predict the following frames. The model's performance is tested by predicting the following 42 frames after the first 6 training frames. The training data and the predicting results are showed in Figure10 and Figure 5. With an averaged MSE loss of 6.9e-4, we show that our model can precisely uncover the intrinsic structures of the kernel with sparse observation and can well predict the numerical solution of it in a long-period of time.

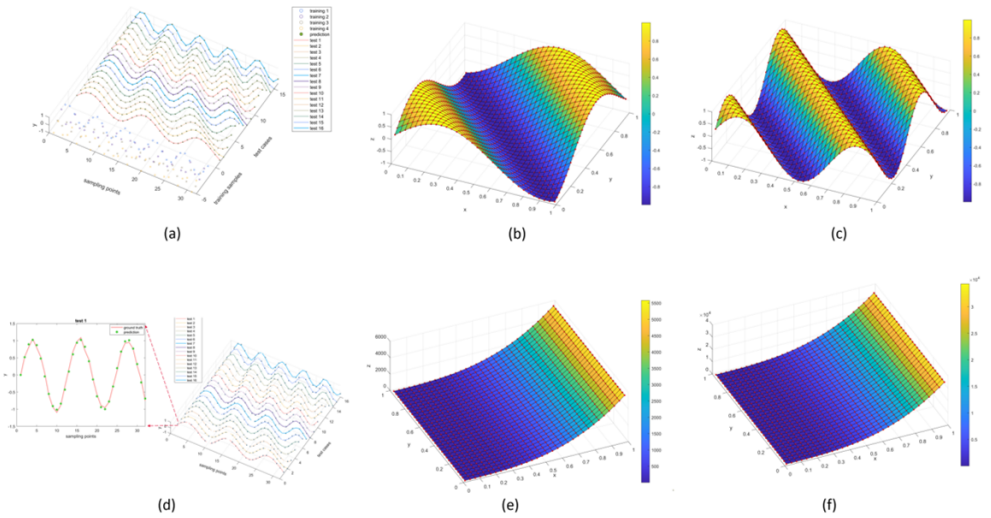


Figure 3: Test cases of 1D and 2D Poisson equations. (a)Test cases for predicting a sine function. (d)Test cases for predicting a sine function with noise. (b-f)Test cases in predicting the 2D Poisson equation with analytical solution (b,e) $\boldsymbol{x} = sin(a * p - u + a * p_v)$, (c,f) $\boldsymbol{x} = (ap_u)^3 + ap_v^2$.

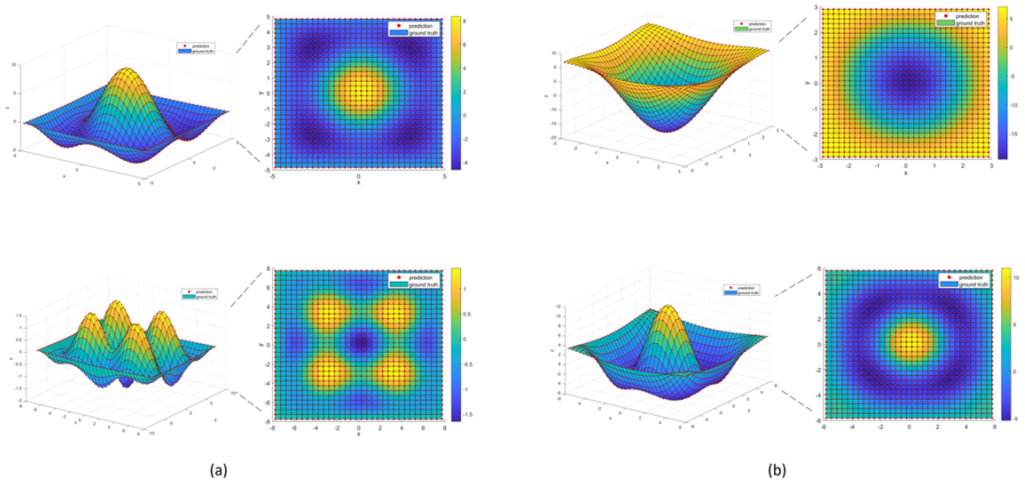


Figure 4: Test cases to predict the solution of a 2D Helmholtz equation with different boundary conditions in different domains. Right figures show the top view of the solution.

5 Related Works

PDE networks Long et al. (2018a; 2019) explore using neural networks to solve the Partial differential equations (PDEs). Li & Shi (2017) formulate the ResNet as a control problem of PDEs on manifold. Raissi et al. (2019) embeds physical priors in neural networks to solve the nonlinear PDEs. Han et al. (2018) handles general high-dimensional parabolic PDEs. Brunton et al. (2020) gives an overview of neural networks in turbulence applications. Wu et al. (2020) train generative adversarial networks to model chaotic dynamical systems. Han et al. (2016) solves high-dimensional stochastic control problems based on Monte-Carlo sampling. Raissi (2018) approximate a deterministic function of time and space by a deep neural network for backward stochastic differential equations with random terminal condition to be satisfied. Wang et al. (2019) first propose to use reinforcement learning (RL)

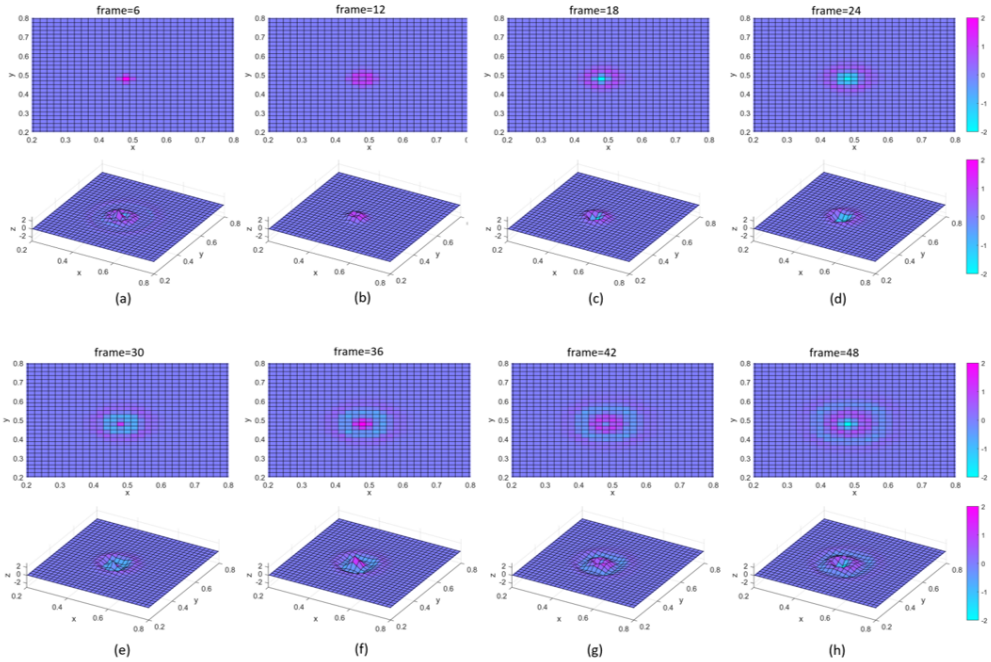


Figure 5: The predicted solution of the time-dependent wave equation. (a-h) The predicted solution at timestep 6, 12, 18, 24, 30, 36, 42, 48 with top view and 3D view. The equation is solved in $domain = [0, 1] \times [0, 1]$, here we only show the plots in $[0.2, 0.8] \times [0.2, 0.8]$.

to aid the process of solving the conservation laws. Jagtap et al. (2020b) propose a neural network on discrete domains for nonlinear conservation laws. Jagtap et al. (2020a) employ adaptive activation functions to solve PDEs and approximate smooth and discontinuous functions. Pang et al. (2019) combines neural networks and Gaussian process to solve PDEs.

Prior-embedded neural simulators Many recent learning physics works are based on building networks to describe interactions among objects or components (see Battaglia et al. (2018) for a survey). The pioneering works done by Battaglia et al. (2016) and Chang et al. (2017) predict different particle dynamics such as N-body by learning the pairwise interactions. Following this, the interaction networks are enhanced by a graph network by Kipf et al. (2017) for different applications. Specialized hierarchical representations by Mrowca et al. (2018), residual corrections by Ajay et al. (2018), propagation mechanisms by Li et al. (2019), linear transitions by Li et al. (2020) were employed to reason various physical systems. On another front, modeling neural networks under a dynamic system's perspective has drawn increasingly attention. In 2018 Chen et al. (2018) solves the neural networks as ordinary differential equations (ODEs).

6 Conclusion

In this paper, we introduced neural PDE, a machine learning approach to learn the intrinsic properties of PDEs by learning and employing a novel functional convolution and adjoint equations to enable the end-to-end training. Our model resents strong robustness against the arbitrary noise. The main limitation of our work is that we assume the PDE systems are sparse. That is, the relation is restricted locally. To enable the constrain in larger area, one can enlarge the kernel size, but this can potentially cost much more computation and memory. For the future applications, we plan to apply the method to solve real-world 3D applications, such as incompressible fluids and nonlinear soft materials. We also plan to scale up the system by leveraging the sparse iterative linear solver for solving the linearized PDEs in each iteration.

REFERENCES

Anurag Ajay, Jiajun Wu, Nima Fazeli, Maria Bauza, Leslie P Kaelbling, Joshua B Tenenbaum, and Alberto Rodriguez. Augmenting physical simulators with stochastic neural networks: Case study of planar pushing and bouncing. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3066–3073. IEEE, 2018.

Brandon Amos and J Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. *arXiv preprint arXiv:1703.00443*, 2017.

Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. Interaction networks for learning about objects, relations and physics. In *Advances in neural information processing systems*, pp. 4502–4510, 2016.

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Robert Bridson. *Fluid simulation for computer graphics*. CRC press, 2015.

Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the national academy of sciences*, 113(15):3932–3937, 2016.

Steven L Brunton, Bernd R Noack, and Petros Koumoutsakos. Machine learning for fluid mechanics. *Annual Review of Fluid Mechanics*, 52:477–508, 2020.

Michael B Chang, Tomer Ullman, Antonio Torralba, and Joshua B Tenenbaum. A compositional object-based approach to learning physical dynamics. *International Conference on Learning Representations*, 2017.

Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in neural information processing systems*, pp. 6571–6583, 2018.

Zhenglin Geng, Daniel Johnson, and Ronald Fedkiw. Coercing machine learning to output physically accurate results. *Journal of Computational Physics*, 406:109099, 2020.

Jiequn Han, Arnulf Jentzen, and E Weinan. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115(34):8505–8510, 2018.

Jiequn Han et al. Deep learning approximation for stochastic control problems. *arXiv preprint arXiv:1611.07422*, 2016.

Jun-Ting Hsieh, Shengjia Zhao, Stephan Eismann, Lucia Mirabella, and Stefano Ermon. Learning neural pde solvers with convergence guarantees. In *International Conference on Learning Representations (ICLR)*, 2018.

Ameya D. Jagtap, Kenji Kawaguchi, and George Em Karniadakis. Adaptive activation functions accelerate convergence in deep and physics-informed neural networks. *Journal of Computational Physics*, 404:109136, 2020a.

Ameya D. Jagtap, Ehsan Kharazmi, and George Em Karniadakis. Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems. *Computer Methods in Applied Mechanics and Engineering*, 365:113028, 2020b.

Kyong Hwan Jin, Michael T McCann, Emmanuel Froustey, and Michael Unser. Deep convolutional neural network for inverse problems in imaging. *IEEE Transactions on Image Processing*, 26(9): 4509–4522, 2017.

Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. Neural relational inference for interacting systems. *International Conference on Learning Representations*, 2017.

Yunzhu Li, Jiajun Wu, Jun-Yan Zhu, Joshua B Tenenbaum, Antonio Torralba, and Russ Tedrake. Propagation networks for model-based control under partial observation. In *International Conference on Robotics and Automation (ICRA)*, pp. 1205–1211. IEEE, 2019.

Yunzhu Li, Hao He, Jiajun Wu, Dina Katabi, and Antonio Torralba. Learning compositional koopman operators for model-based control. In *International Conference on Learning Representations (ICLR)*, 2020.

Zhen Li and Zuoqiang Shi. Deep residual learning and pdes on manifold. *arXiv preprint arXiv:1708.05115*, 2017.

Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

Zichao Long, Yiping Lu, Xianzhong Ma, and Bin Dong. Pde-net: Learning pdes from data. In *International Conference on Machine Learning (ICML)*, pp. 3208–3216, 2018a.

Zichao Long, Yiping Lu, Xianzhong Ma, and Bin Dong. Pde-net: Learning pdes from data. In *International Conference on Machine Learning*, pp. 3208–3216, 2018b.

Zichao Long, Yiping Lu, and Bin Dong. Pde-net 2.0: Learning pdes from data with a numeric-symbolic hybrid deep network. *Journal of Computational Physics*, 399:108925, 2019.

Lu Lu, Xuhui Meng, Zhiping Mao, and George E Karniadakis. Deepxde: A deep learning library for solving differential equations. *arXiv preprint arXiv:1907.04502*, 2019.

Damian Mrowca, Chengxu Zhuang, Elias Wang, Nick Haber, Li F Fei-Fei, Josh Tenenbaum, and Daniel L Yamins. Flexible neural representation for physics prediction. In *Advances in neural information processing systems*, pp. 8799–8810, 2018.

Samira Pakravan, Pouria A. Mistani, Miguel Angel Aragon-Calvo, and Frederic Gibou. Solving inverse-pde problems with physics-aware neural networks, 2020.

Guofei Pang, Liu Yang, and George Em Karniadakis. Neural-net-induced gaussian process regression for function approximation and pde solution. *Journal of Computational Physics*, 384:270 – 288, 2019.

Maziar Raissi. Forward-backward stochastic neural networks: Deep learning of high-dimensional partial differential equations. *arXiv preprint arXiv:1804.07010*, 2018.

Maziar Raissi, Paris Perdikaris, and George E Karniadakis. 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, 2019.

Maziar Raissi, Alireza Yazdani, and George Em Karniadakis. Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. *Science*, 367(6481):1026–1030, 2020.

Ben Stevens and Tim Colonius. Finitenet: A fully convolutional lstm network architecture for time-dependent partial differential equations. *arXiv preprint arXiv:2002.03014*, 2020.

Andreas Wächter. Short tutorial: getting started with ipopt in 90 minutes. In *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2009.

Yufei Wang, Ziju Shen, Zichao Long, and Bin Dong. Learning to discretize: Solving 1d scalar conservation laws via deep reinforcement learning, 2019.

Jin-Long Wu, Karthik Kashinath, Adrian Albert, Dragos Chirila, Heng Xiao, et al. Enforcing statistical constraints in generative adversarial networks for modeling chaotic dynamical systems. *Journal of Computational Physics*, 406:109209, 2020.

Li Xu, Jimmy SJ Ren, Ce Liu, and Jiaya Jia. Deep convolutional neural network for image deconvolution. In *Advances in neural information processing systems*, pp. 1790–1798, 2014.

Shuqi Yang, Xingzhe He, and Bo Zhu. Learning physical constraints with neural projections. *arXiv preprint arXiv:2006.12745*, 2020.

Kai Zhang, Wangmeng Zuo, Shuhang Gu, and Lei Zhang. Learning deep cnn denoiser prior for image restoration. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3929–3938, 2017.

A COMPARISON OF OPTIMIZATION METHODS

The neural networks are trained by a hybrid method of IpOpt and Adam optimizer. Here we demonstrate our hybrid optimizer's capability to outperform a typical Adam optimizer and a typical SGDM optimizer, which is shown in Figure 6.

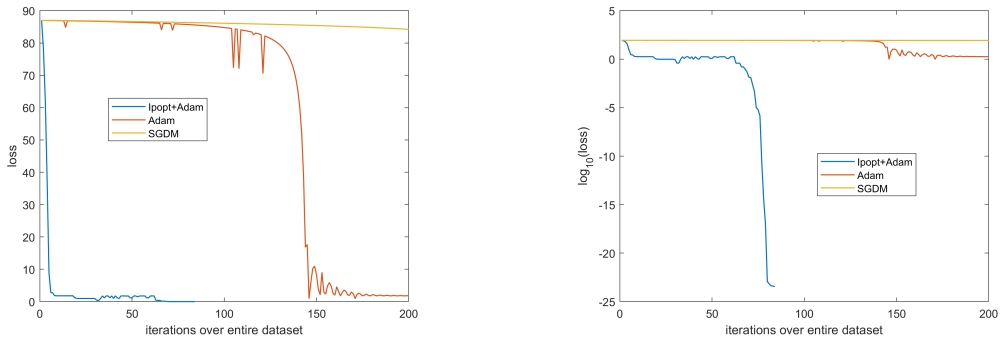
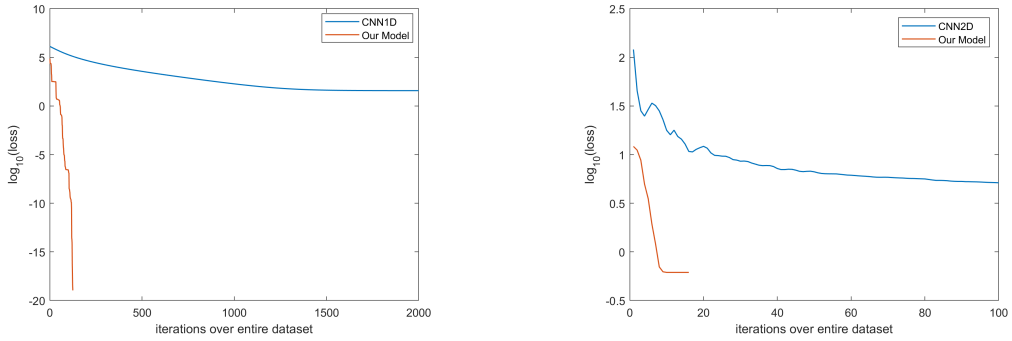


Figure 6: Training our model to predict the solution for the 1D Poisson problem $\nabla \cdot \nabla \boldsymbol{x} = 0$ using different optimization methods. The IpOpt+Adam optimizer stops at early converge with the total loss of 3.43e-24.

B COMPARISON WITH NAIVE CNN STRUCTURE

We compare our functional convolution model with two naive convolutional neural networks by testing their performances in solving the 1D Poisson problem and 2D time-dependent wave equation respectively. The results are shown in Figure 7 to demonstrate our model's merits both in converging rate and accuracy.



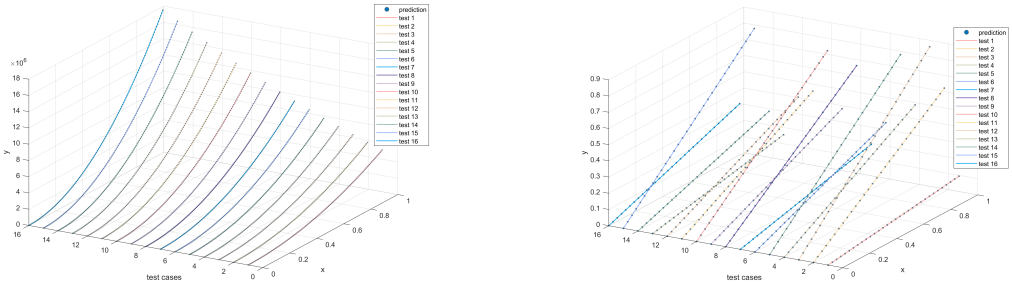
(a) (b)

Figure 7: comparison between predictions from Naive CNN networks and our model to predict the solution for (a) the 1D Poisson problem $\nabla \cdot \nabla \boldsymbol{x} = 0$ and (b) the time-dependent wave equation.

C 1D EXAMPLES

C.1 1D POISSON EXAMPLE

We employ our functional convolutional model with the aim to solve typical 1D Poisson problems with constant kernels, the testing results of which are shown in Figure 8.



(a) (b)

Figure 8: Two test cases for predicting the solution of a 1D Poisson equation $\nabla \cdot \nabla \boldsymbol{x} = b$ with (a) target solution $x = (ap)^2$ and (b) $b = 0$. The dots represent the predicted values and the curves indicate the true values.

C.2 1D SPATIAL VARYING AND NONLINEAR EXAMPLE

We also train the model to predict the solution of $\nabla \cdot (1 + |\pi p|)\nabla \boldsymbol{x} = 0$ and $\nabla \cdot (1 + |x| + sin(|x| * .001))\nabla x = 0$. The results are shown in Figure 9.

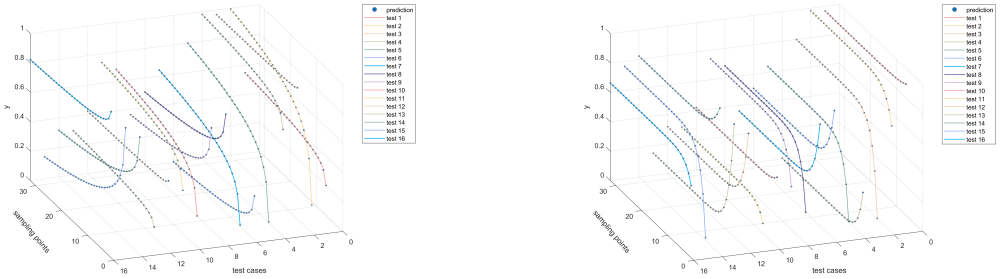


Figure 9: Test cases for predicting the numerical solution of an unknown Poisson PDE. The left figure shows function domain of all tests to predict the Poisson PDE with with spatially varying coefficients. The right figure shows tests to predict the solution of a nonlinear PDE specified by $\nabla \cdot (1 + |x| + sin(|x| * .001))\nabla x = 0$.

D TRAINING DATA FOR WAVE EQUATION

We refer the readers to Figure 10 for the training data of the wave equation.

E DETAILS ON NEURAL NETWORK STRUCTURES

We show the details of the naive CNN structures in Table 1 which are trained as baseline compared to our convolution model. We also refer readers to Table 2 for the details of neural network structure and training parameters across different examples.

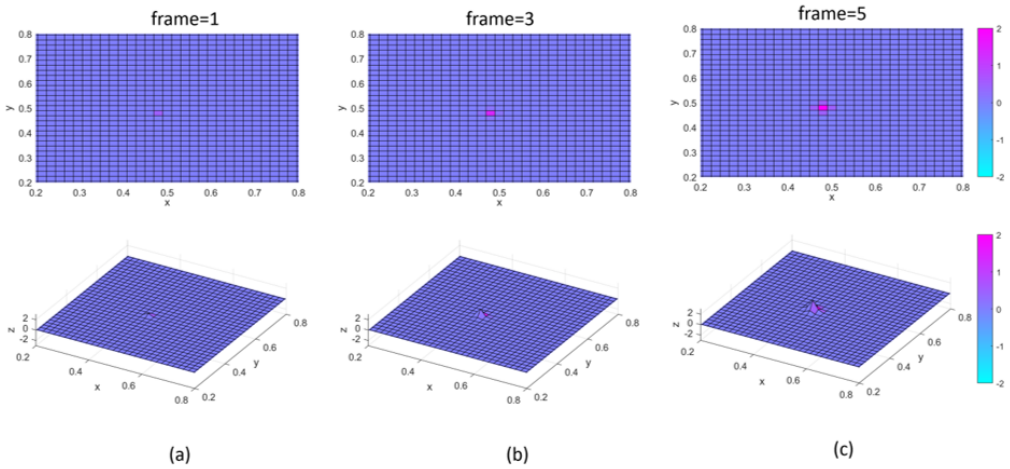


Figure 10: The training data to solve the time-dependent wave equation specified by Equation $\nabla \cdot \nabla \boldsymbol{x} = \frac{\partial^2 \boldsymbol{x}}{\partial t^2}$. (a-c) The target solution at timestep 1, 3, 5, with top view and 3D view. The equation is solved in $domain = [0, 1] \times [0, 1]$, here we only show the plots in $[0.2, 0.8] \times [0.2, 0.8]$.

Table 1: Naive CNN baselines

Baseline	neural network structure	grid	# training smaples
1D CNN structure	cnn1d(in=1,out=1,kernel=3), ReLU, cnn(in=1,out=1,kernel=3),ReLU, cnn1d(in=1,out=1,kernel=3)	16×1	4
2D CNN structure	cnn1d(in=1,out=2,kernel=3), ReLU, cnn(in=2,out=2,kernel=3),ReLU, cnn1d(in=2,out=1,kernel=3)	20×20	8

Table 2: Neural network structure and training parameters

Equation	Dimension	Network	# Parameters	# Training sample	# Test sample	Test MSE
$\nabla \cdot \nabla \boldsymbol{x} = b$ with target solution $x = (ap)^2$	128×1	$3 \times 4 \times 4 \times 3$	31	4	16	8.3e-20
$\nabla \cdot \nabla \boldsymbol{x} = 0$	32×1	$3 \times 4 \times 4 \times 3$	31	4	16	5.0e-29
$\nabla \cdot \nabla \boldsymbol{x} = b$ with target solution $x = sin(ap)$	32×1	$3 \times 4 \times 4 \times 3$	31	2	16	9.8e-4
$\nabla \cdot \nabla \boldsymbol{x} = b$ with target solution $x = sin(ap)$ with 10 % noise	32×1	$3 \times 4 \times 4 \times 3$	31	2	16	2.4e-3
$\nabla \cdot (1 + \lvert \pi p \rvert) \nabla \boldsymbol{x} = 0$	32×1	$6 \times 6 \times 6 \times 6 \times 6 \times 3$	31	4	16	1.3e-6
$\nabla \cdot (1 + \lvert x \rvert + sin(\lvert x \rvert * .001)) \nabla x = 0$	32×1	$3 \times 6 \times 6 \times 3$	45	4	16	2.1e-5
$\nabla \cdot \nabla \boldsymbol{x} = b$ with target solution $x = (ap_u)^3 + ap_v^2$	32×32	$3 \times 14 \times 14 \times 14 \times 14 \times 3$	325	4	16	5.4e-14
$\nabla \cdot \nabla \boldsymbol{x} = b$ with target solution $x = sin(ap_u + ap_v)$	32×32	$3 \times 10 \times 10 \times 10 \times 10 \times 10 \times 10 \times 3$	293	4	16	3.7e-3
$\nabla \cdot \nabla \boldsymbol{x} + \boldsymbol{x} = 0$ with boundary $\boldsymbol{x} = a/(p_u^2 + p_v^2)$	32×32	$3 \times 5 \times 5 \times 5 \times 5 \times 5 \times 3$	68	4	16	5.3e-27
$\nabla \cdot \nabla \boldsymbol{x} + \boldsymbol{x} = 0$ with boundary, $\boldsymbol{x} = a * sin((0.02p_u + 0.02p_v))$	32×32	$3 \times 5 \times 5 \times 5 \times 5 \times 5 \times 3$	68	4	16	9.3e-29
$\nabla \cdot \nabla \boldsymbol{x} = \frac{\partial^2 \boldsymbol{x}}{\partial t^2}$ with center source $\boldsymbol{x} = sin(60(n * dt))$	49×49	$3 \times 5 \times 5 \times 5 \times 5 \times 5 \times 3$	68	6	42	6.9e-4