INTERNATIONAL JOURNAL OF
NUMERICAL ANALYSIS AND MODELING
Volume 21, Number 5, Pages 697–715

doi: 10.4208/ijnam2024-1028

DEEP SURROGATE MODEL FOR LEARNING GREEN'S FUNCTION ASSOCIATED WITH LINEAR REACTION-DIFFUSION OPERATOR

JUNQING JIA, LILI JU, AND XIAOPING ZHANG*

Abstract. In this paper, we present a deep surrogate model for learning the Green's function associated with the reaction-diffusion operator in rectangular domain. The U-Net architecture is utilized to effectively capture the mapping from source to solution of the target partial differential equations (PDEs). To enable efficient training of the model without relying on labeled data, we propose a novel loss function that draws inspiration from traditional numerical methods used for solving PDEs. Furthermore, a hard encoding mechanism is employed to ensure that the predicted Green's function is perfectly matched with the boundary conditions. Based on the learned Green's function from the trained deep surrogate model, a fast solver is developed to solve the corresponding PDEs with different sources and boundary conditions. Various numerical examples are also provided to demonstrate the effectiveness of the proposed model.

Key words. Reaction-diffusion operator, Green's function, surrogate model, deep learning, fast solver.

1. Introduction

With the rapid development and great success of deep learning technology in computer vision, natural language processing and other fields, it has also shown an increasing impact in the field of scientific computing, especially in the numerical solution of partial differential equations (PDEs) [1, 2, 3]. The use of neural networks to solve PDEs has been investigated in several early works, e.g., [4, 5], recent advances in deep learning techniques have further stimulated new explorations in this direction.

Representative methods of interest are the physics-informed neural network (PINN) [3], the deep Galerkin method (DGM) [6] and the deep Ritz method (DRM) [1]. All these methods model the mapping from space and/or time variables to the system states with a fully connected neural network. Their differences mainly lie in the construction of loss functions. The loss functions of PINN and DGM are expressed as a weighted sum of PDE residuals at randomly selected interior points as well as solution errors at initial/boundary points. This idea also has been extended to solve inverse problems [3], fractional differential equations [7], stochastic differential equations and uncertainty qualification [8, 9, 10] and other applications. DRM [1] designs loss function using the variational form of PDEs, requiring numerical integrations to train the network. Related works have subsequently emerged [11, 12, 13, 14]. Meanwhile, the solution of parameterized PDEs is also receiving increasing attention. For example, PINNs with special treatments are used to solve parameterized PDEs involving point sources in [15] and [16]; the meta-learning methods coupled with PINNs are developed to solve parameterized PDEs with different boundary conditions and domain shapes in [17] and [18].

Received by the editors on August 9, 2023 and, accepted on January 27, 2024.
2000 *Mathematics Subject Classification.* 34B27, 65N99.
*Corresponding author.

In theoretical research and engineering applications of various PDEs, including Poisson, Helmholtz and wave equations, the use of Green's function is significant. Having obtained the associated Green's function of the given differential operator, the Green's function method is used to precisely determine the solution of the corresponding PDE, which is explicitly expressed in an integral form, with the integral kernel based on the Green's function. Green's function is, in reality, a solution of the corresponding PDE with a point source subject to the homogeneous Dirichlet boundary condition. Such a problem can also be regarded as the solution of a parametrized PDE, where the location of the point source is the parameter.

However, the Green's function in a general domain typically lacks an analytic form. Therefore, we must approximate the Green's function numerically, which has led to increased attention on corresponding numerical methods in recent decades. Fortunately, the rapid development of deep learning techniques and their powerful expressive capability have introduced a potentially novel method for computing Green's function. Supervised learning methods, such as those proposed by [19] and [20], have been suggested to learn the Green's function. However, using these methods requires a considerable amount of labeled training data, which can be acquired by repeatedly solving PDEs through traditional numerical methods beforehand. The process of preparing training data consumes expensive computational resources. In addition, since these methods are purely data-driven, their generalization ability is usually restricted by the dataset coverage. In contrast, certain physics-driven models also have been proposed to compute Green's function, including GF-Net [21] and BI-GreenNet [22]. GF-Net [21] extends the PINN structure [3] to solve PDEs stipulated by Green's function. Moreover, these models utilize certain special techniques, such as the smoothness of the Dirac delta function and domain decomposition approach to optimize the network training process. BI-GreenNet [22] introduces a novel framework for computing Green's function, which leverages the fundamental solution, boundary integral method and neural networks to achieve high accuracy levels.

All of the above methods are solely based on neural networks. In the past decades, traditional numerical methods, such as finite difference, finite element and finite volume methods, have been extensively studied for solving PDEs, especially with point sources, to compute Green's function. A plausible approach is to develop a model to compute Green's function by leveraging the benefits of both traditional methods and neural networks. In this context, we propose to use the U-Net architecture to develop a deep surrogate model for learning the Green's function of the linear reaction-diffusion operator on a rectangular domain, and to design a novel loss function, inspired by traditional numerical methods, which helps train the deep surrogate model efficiently.

The remaining sections of the paper are organized as follows. In Section 1.1, we briefly introduce the problem setting, including the reaction-diffusion equation, its Green's function as well as the Green's representation formula. Section 2 presents and discusses the deep surrogate model for learning the Green's function of the linear reaction-diffusion operator on a rectangular domain. This section includes the network architecture, data generation, loss function and training strategy. In Section 3 we present a fast solver based on the proposed deep surrogate model to solve the corresponding PDEs. Extensive numerical experiments and comparisons are provided in Section 4 to demonstrate the outstanding performance of the proposed method, including some ablation studies and the application of the deep surrogate

model to the fast numerical solution of a target equation with different sources and boundary conditions.

1.1. Problem setting and Green's function. Let $\Omega \subset \mathbb{R}^d$ be a bounded Lipchitz domain, we consider the following linear reaction-diffusion operator:

$$\mathcal{L}(u)(\boldsymbol{x}) := -\nabla \cdot (a(\boldsymbol{x})\nabla u(\boldsymbol{x})) + r(\boldsymbol{x})u(\boldsymbol{x}), \quad \boldsymbol{x} \in \Omega, \tag{1}$$

where $a(\boldsymbol{x}) > 0$ is the diffusion coefficient and $r(\boldsymbol{x}) \geq 0$ is the reaction coefficient. The corresponding reaction-diffusion equation with the Dirichlet boundary condition can be represented as follows:

$$\begin{cases} \mathcal{L}(u)(\boldsymbol{x}) = f(\boldsymbol{x}), & \boldsymbol{x} \in \Omega, \\ u(\boldsymbol{x}) = g(\boldsymbol{x}), & \boldsymbol{x} \in \partial\Omega, \end{cases} \tag{2}$$

where $f(\boldsymbol{x})$ is the given source term and $g(\boldsymbol{x})$ gives the boundary value. The Green's function $G(\boldsymbol{x}, \boldsymbol{\xi})$ represents the impulse response of the PDE subject to homogeneous Dirichlet boundary condition, that is, for any impulse source point $\boldsymbol{\xi} \in \Omega$,

$$\begin{cases} \mathcal{L}(G)(\boldsymbol{x}, \boldsymbol{\xi}) &= \delta(\boldsymbol{x} - \boldsymbol{\xi}), \quad \boldsymbol{x} \in \Omega, \\ G(\boldsymbol{x}, \boldsymbol{\xi}) &= 0, \quad \boldsymbol{x} \in \partial\Omega, \end{cases} \tag{3}$$

where $\delta(\boldsymbol{x})$ denotes the Dirac delta source function satisfying

$$\delta(\boldsymbol{x}) = \begin{cases} 0, & \text{if } \boldsymbol{x} \neq \boldsymbol{0} \\ \infty, & \text{if } \boldsymbol{x} = \boldsymbol{0} \end{cases} \quad \text{and} \quad \int_{\mathbb{R}^d} \delta(\boldsymbol{x})\, d\boldsymbol{x} = 1. \tag{4}$$

If the Green's function $G(\boldsymbol{x}, \boldsymbol{\xi})$ is found, then the solution of (2) can be expressed by

$$u(\boldsymbol{x}) = \int_\Omega f(\boldsymbol{\xi}) G(\boldsymbol{x}, \boldsymbol{\xi})\, d\boldsymbol{\xi} - \int_{\partial\Omega} g(\boldsymbol{\xi}) a(\boldsymbol{\xi}) \frac{\partial G(\boldsymbol{x}, \boldsymbol{\xi})}{\partial \boldsymbol{n}_{\boldsymbol{\xi}}}\, ds_{\boldsymbol{\xi}}, \quad \forall \boldsymbol{x} \in \Omega, \tag{5}$$

where $\boldsymbol{n}_{\boldsymbol{\xi}}$ denotes the unit outer normal vector on $\partial\Omega$.

2. The deep surrogate model for learning Green's function

It is noteworthy that Eq. (3) is actually a parameterized PDE with the parameter $\boldsymbol{\xi}$ and the homogeneous Dirichlet boundary conditions. We will propose a deep surrogate model to solve such a parameterized PDE, which equivalently learns the Green's function associated with the linear reaction-diffusion operator (1), and then uses it to construct a fast solver for solving the problem (2) based on the formula (5). In order to represent the Green's function obeying (3), an appropriate convolutional neural network is adopted to model the mapping from the source $\delta(\boldsymbol{x} - \boldsymbol{\xi})$ to the solution $G(\boldsymbol{x}, \boldsymbol{\xi})$ of (3). In this paper, we take the two-dimensional problem for illustration and assume $\Omega = [0, L_1] \times [0, L_2]$, but the proposed method can be naturally generalized to higher-dimensional rectangular domains.

2.1. The U-Net architecture. The U-Net is a representative example of a Convolutional Neural Network (CNN), which was originally proposed for medical image segmentation, but was subsequently applied to a wide range of image processing tasks. In recent years, with the widespread application of deep learning in scientific computing, the U-Net has also been employed for regression tasks, particularly for the deep learning based method for numerical solution of PDEs, e.g. [23].

Similar to all other CNNs, the U-Net employs filter kernels for convolutional layers and pooling layers to extract features from input images. Nevertheless, the U-Net architecture is devised with a unique "U" shape, where the feature maps

from the encoding path are concatenated with those from the decoding path using skip connections. This approach enables the model to capture both high-level and low-level features. Furthermore, the U-Net is recognized for its expansive path that includes deconvolution or upsampling layers to progressively increase the spatial resolution of the output.

The input tensor $\boldsymbol{T_\xi}$ of the U-Net is designed with dimension of $n \times m \times C$, where C represents the number of input channels. The output tensor $\boldsymbol{G_\xi}$ of the U-Net is dimensioned at $n \times m$. To better suit our needs, we also slightly modify the classical architecture of the U-Net by introducing another two hyperparameters. One of these is the channels of the first hidden layer, denoted as C_1, which identifies the number of features extracted at the beginning. The other one is the depth of the encoder/decoder, denoted as D.

As depicted in Figure 1, each encoding operation in the U-Net downsamples the input size of the previous layer while simultaneously doubling the channel number of the input tensor. Conversely, each decoding operation in the U-Net doubles the input size of the previous layer and halves the number of channels. By adding more coding and decoding layers, the depth of this architecture can be easily increased. To implement the hard encoding of the homogeneous boundary condition obeyed by Green's function, we add a zero padding operation at the end of the architecture.

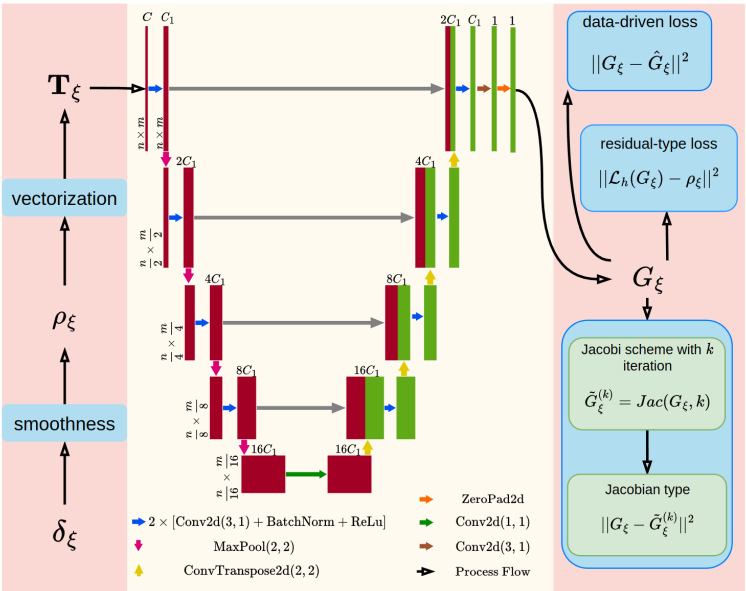


FIGURE 1. Illustration of the architecture of the U-Net for the proposed deep surrogate model for learning Green's function. Note $\boldsymbol{G_\xi} = \text{U-Net}(\boldsymbol{T_\xi}, \boldsymbol{\Theta})$ where the input tensor $\boldsymbol{T_\xi}$ of the U-Net is of dimension $n \times m \times C$ and the output tensor $\boldsymbol{T_\xi}$ is of dimension $n \times m$.

2.2. Approximation of the Dirac delta function. The Dirac delta function (4) is approximated by a multidimensional Gaussian density function

$$\delta(\boldsymbol{x} - \boldsymbol{\xi}) \approx \rho(\boldsymbol{x} - \boldsymbol{\xi}) = \frac{1}{\left(\sqrt{2\pi}\sigma\right)^2} \exp\left(-\frac{|\boldsymbol{x} - \boldsymbol{\xi}|^2}{2\sigma^2}\right), \tag{6}$$

where the parameter $\sigma > 0$ denotes the standard deviation of the distribution. As $\sigma \to 0$, the function (6) converges to the Dirac delta function pointwisely except at the point $\boldsymbol{x} = \boldsymbol{\xi}$. In practice, the standard deviation σ is set to be a value proportional to the mesh size of the problem domain Ω.

2.3. Data generation. Let us uniformly partition the domain Ω in each direction to obtain a rectangular mesh with nodes $\boldsymbol{X} = \{\boldsymbol{x}_{i,j} = ((i-1)h_1, (j-1)h_2) \,|\, i = 1, \cdots, n,\ j = 1, \cdots, m\}$, where $h_1 = L_1/(n-1)$ and $h_2 = L_2/(m-1)$. For each fixed source point $\boldsymbol{\xi}$, we first compute the distance between $\boldsymbol{\xi}$ and each node in the mesh $\boldsymbol{X}$, and then assemble them into an array

$$\boldsymbol{R}_{\boldsymbol{\xi}} = \{\|\boldsymbol{x}_{i,j} - \boldsymbol{\xi}\|_2 \mid \boldsymbol{x}_{i,j} \in \boldsymbol{X}\}. \tag{7}$$

Next a normalization is adopted to get

$$\boldsymbol{R}_{\boldsymbol{\xi}} \leftarrow \frac{\boldsymbol{R}_{\boldsymbol{\xi}} - R_{\min}}{R_{\max} - R_{\min}}, \tag{8}$$

where $R_{\min}$ and $R_{\max}$ are the minimum and maximum of $\boldsymbol{R}_{\boldsymbol{\xi}}$, respectively. The right-hand term $\delta(\boldsymbol{x} - \boldsymbol{\xi})$ of (3) is also evaluated on $\boldsymbol{X}$ for each $\boldsymbol{\xi}$, which leads to an array

$$\boldsymbol{\rho}_{\boldsymbol{\xi}} = \{\rho(\boldsymbol{x}_{i,j} - \boldsymbol{\xi}) \mid \boldsymbol{x}_{i,j} \in \boldsymbol{X}\}, \tag{9}$$

where $\rho(\cdot)$ is defined by (6).

Since the source point $\boldsymbol{\xi}$ can be randomly sampled at any location in the solution domain Ω, we can easily generate the samples. All numerical experiments in Section 4 use the training set consisting of 2000 $\boldsymbol{\xi}$ samples generated with the uniform distribution in Ω and the validation set consisting of 100 $\boldsymbol{\xi}$ samples equally spaced in Ω. The input tensors $\{\boldsymbol{T}_{\boldsymbol{\xi}}\}$ consist of three types, including 1-channel input $\boldsymbol{T}_{\boldsymbol{\xi}}^{(1)} = \boldsymbol{\rho}_{\boldsymbol{\xi}}$, 2-channels input $\boldsymbol{T}_{\boldsymbol{\xi}}^{(2)} = [\boldsymbol{R}_{\boldsymbol{\xi}}, \boldsymbol{\rho}_{\boldsymbol{\xi}}]$, and 3-channels input $\boldsymbol{T}_{\boldsymbol{\xi}}^{(3)} = [\boldsymbol{X}, \boldsymbol{\rho}_{\boldsymbol{\xi}}]$.

2.4. Loss function. To train the deep surrogate model in a physics-driven fashion, we need to construct a loss function based on the PDE (3). Unlike PINN and its variants, we will not use the strong form of the PDE. Instead, we discretize (3) by conventional numerical schemes. Specifically, we adopt the second-order central finite difference scheme to discretize (3) on $\boldsymbol{X}$, which leads to

$$\mathcal{L}_h(G_h)(\boldsymbol{x}_{i,j}, \boldsymbol{\xi}) = \rho(\boldsymbol{x}_{i,j} - \boldsymbol{\xi}). \tag{10}$$

where $\boldsymbol{G}_{\boldsymbol{\xi}} = \{G_h(\boldsymbol{x}_{i,j}, \boldsymbol{\xi}) \mid \boldsymbol{x}_{i,j} \in \boldsymbol{X}\}$ and $\mathcal{L}_h$ is the discrete operator for approximating the differential operator $\mathcal{L}$ given as follows:

$$\begin{aligned} \mathcal{L}_h(G_h)(\boldsymbol{x}_{i,j}, \boldsymbol{\xi}) = {} & c_{i,j} G_h(\boldsymbol{x}_{i,j}, \boldsymbol{\xi}) - c_{i+1,j} G_h(\boldsymbol{x}_{i+1,j}, \boldsymbol{\xi}) \\ & - c_{i-1,j} G_h(\boldsymbol{x}_{i-1,j}, \boldsymbol{\xi}) - c_{i,j+1} G_h(\boldsymbol{x}_{i,j+1}, \boldsymbol{\xi}) \\ & - c_{i,j-1} G_h(\boldsymbol{x}_{i,j-1}, \boldsymbol{\xi}) + r_{ij} G_h(\boldsymbol{x}_{i,j}, \boldsymbol{\xi}). \end{aligned} \tag{11}$$

where $r_{ij} = r(\boldsymbol{x}_{i,j})$ and

$$c_{i+1,j} = a(\boldsymbol{x}_{i+1/2,j})/h_1^2, \quad c_{i-1,j} = a(\boldsymbol{x}_{i-1/2,j})/h_1^2,$$

$$c_{i,j+1} = a(\boldsymbol{x}_{i,j+1/2})/h_2^2, \quad c_{i,j-1} = a(\boldsymbol{x}_{i,j-1/2})/h_2^2,$$

$$c_{i,j} = c_{i+1,j} + c_{i-1,j} + c_{i+1,j+1} + c_{i,j-1}.$$

Then a natural and common way to construct the loss function is to use the residual of (10):

$$\begin{aligned} Loss_{\mathrm{res}}(\boldsymbol{\Theta}) &= \sum_{\boldsymbol{\xi}} \|\mathcal{L}_h(\boldsymbol{G}_{\boldsymbol{\xi}}) - \boldsymbol{S}_{\boldsymbol{\xi}}\|^2 \\ &= \sum_{\boldsymbol{\xi}} \sum_{i,j} |\mathcal{L}_h(G_h)(\boldsymbol{x}_{i,j}, \boldsymbol{\xi}) - \rho(\boldsymbol{x}_{i,j} - \boldsymbol{\xi})|^2, \end{aligned} \tag{12}$$

where $\boldsymbol{S}_{\boldsymbol{\xi}} = \{\rho(\boldsymbol{x}_{i,j} - \boldsymbol{\xi}) \mid \boldsymbol{x}_{i,j} \in \boldsymbol{X}\}$, which is referred as the **residual-type loss**. It is a discrete analogue of the loss function commonly used in PINN. Unfortunately, numerical experiments in Section 4 exhibit that the use of such loss function is quite hard to train the proposed deep surrogate model and could lead to a poor performance.

Inspired by the idea of Jacobi iterative scheme for solving linear systems, we propose and test a new loss function defined by

$$Loss_{\mathrm{jac}}(\boldsymbol{\Theta}) = \sum_{\boldsymbol{\xi}} \|\boldsymbol{G}_{\boldsymbol{\xi}} - \tilde{\boldsymbol{G}}_{\boldsymbol{\xi}}^{(k)}\|^2, \tag{13}$$

where $\tilde{\boldsymbol{G}}_{\boldsymbol{\xi}}^{(k)}$ is the approximate solution of (10) obtained by using Jacobi iteration scheme with the initial value $\tilde{\boldsymbol{G}}_{\boldsymbol{\xi}}^{(0)} = \boldsymbol{G}_{\boldsymbol{\xi}}$ and k iterations, i.e.,

$$\begin{aligned} \tilde{G}_h^{(l+1)}(\boldsymbol{x}_{ij}, \boldsymbol{\xi}) = \frac{1}{c_{i,j} + r_{ij}} \Big[& \rho(\boldsymbol{x}_{i,j} - \boldsymbol{\xi}) + c_{i+1,j} \tilde{G}_h^{(l)}(\boldsymbol{x}_{i+1,j}, \boldsymbol{\xi})) \\ & + c_{i-1,j} \tilde{G}_h^{(l)}(\boldsymbol{x}_{i-1,j}, \boldsymbol{\xi})) + c_{i,j+1} \tilde{G}_h^{(l)}(\boldsymbol{x}_{i,j+1}, \boldsymbol{\xi})) \\ & + c_{i,j-1} \tilde{G}_h^{(l)}(\boldsymbol{x}_{i,j-1}, \boldsymbol{\xi})) \Big], \qquad l = 0, 1, \cdots, k-1. \end{aligned} \tag{14}$$

We will refer (13) as the **Jacobi-type loss**.

For comparison purposes, we also consider and test a **data-driven** loss function as follows:

$$Loss_{\mathrm{data}}(\boldsymbol{\Theta}) = \sum_{\boldsymbol{\xi}} \|\boldsymbol{G}_{\boldsymbol{\xi}} - \hat{\boldsymbol{G}}_{\boldsymbol{\xi}}\|^2, \tag{15}$$

where $\hat{\boldsymbol{G}}_{\boldsymbol{\xi}}$ is obtained by taking the final convergent result of the Jacobi iterative solution $\tilde{\boldsymbol{G}}_{\boldsymbol{\xi}}^{(k)}$, i.e., $\hat{\boldsymbol{G}}_{\boldsymbol{\xi}} = \lim_{k \to \infty} \tilde{\boldsymbol{G}}_{\boldsymbol{\xi}}^{(k)}$.

2.5. Training strategies. This section explores training strategies for the deep surrogate model equipped with $Loss_{\mathrm{jac}}$. The objective of the training process is to form a virtuous circle through gradually optimizing the network from the approximate solutions generated by the Jacobi iteration method. The U-Net's predictions can then be served as a potentially improved initial solutions for the Jacobi iteration in the subsequent training step.

Three options for choosing the optimal iteration number k in (13) are considered. Using a fixed k in the Jacobi iteration scheme during the training process is a conventional approach, referred as "constant strategy". In this approach selecting an optimal k is important to balance accuracy and computational complexity. The second approach is to initially set a larger value for k and then gradually decrease it during the training until it reaches a small value, referred as the "dynamic strategy". A more reasonable approach is to adaptively adjust k by comparing the validation errors observed in two successive epochs. If the error observed in the current epoch is significantly greater than that in the previous epoch, then k should be increased,

and conversely, if it is smaller, then k needs to be decreased. This approach is referred as "adaptive strategy".

3. Fast PDE solver based on the learned Green's function

Once the deep surrogate model is trained, numerical solution of the linear reaction-diffusion problem (2) can be directly computed based on the Green's formula (5) through the learned Green's function. To ensure accurate evaluation of the integrals in (5) accurately, we apply numerical quadrature on the same rectangular mesh (denoted by $\mathcal{R}_q = \{R_l\}$) as that used for training the deep surrogate model. Let us denote the intersection of the rectangle edges with the domain boundary by $\mathcal{E}_q^{\text{bdry}} = \{E_m\}$. By using the symmetry of Green's function, we have

$$
\begin{aligned}
u(\boldsymbol{\xi}) \approx & \sum_{R_l \in \mathcal{R}_q} I_{\boldsymbol{x},h}^{R_l}[f(\boldsymbol{x})G(\boldsymbol{x},\boldsymbol{\xi})] \\
& - \sum_{E_m \in \mathcal{E}_q^{\text{bdry}}} I_{\boldsymbol{x},h}^{E_m}[g(\boldsymbol{x})a(\boldsymbol{x})(\nabla_{\boldsymbol{x}} G(\boldsymbol{x},\boldsymbol{\xi}) \cdot \boldsymbol{n}_{\boldsymbol{x}})],
\end{aligned} \tag{16}
$$

where $I_{\boldsymbol{x},h}^{R_l}[\cdot]$ denotes the numerical quadrature for evaluating

$$
\int_{R_l} f(\boldsymbol{x})G(\boldsymbol{x},\boldsymbol{\xi})\, d\boldsymbol{x}
$$

and $I_{\boldsymbol{x},h}^{E_m}[\cdot]$ the numerical quadrature for evaluating

$$
\int_{E_m} g(\boldsymbol{\xi})a(\boldsymbol{\xi})(\nabla_{\boldsymbol{x}} G(\boldsymbol{x},\boldsymbol{\xi}) \cdot \boldsymbol{n}_{\boldsymbol{x}})\, ds_{\boldsymbol{x}},
$$

respectively.

4. Numerical experiments

This section presents various numerical experiments. We first conduct ablation studies for the deep surrogate model used to learn the Green's function of the Laplacian operator. Then, we test more examples on the learned Green's functions of the reaction-diffusion operator and corresponding fast solver. In the following examples, the solution domain is chosen to be $[-1,1]\times[-1,1]$ and partitioned into a uniform rectangular mesh of 64×64 uniform nodes, i.e., $n = m = 64$. In all experiments, the maximum number of epochs and the batch size are to 150 and 6, respectively. All experiments are implemented using the PyTorch framework and run on the GTX 2080Ti card.

4.1. Ablation study of deep surrogate model. To simplify the matter, we use the deep surrogate model for learning the Green's function of the Laplacian operator (i.e., $a(\boldsymbol{x}) \equiv 1$ and $r(\boldsymbol{x}) \equiv 0$) as an example. We conduct a series of ablation studies to measure the influence of the model's performance, including the impact of network architecture, loss functions, input forms and the number of Jacobi iterations. In this subsection, the number of Jacobi iterations remains fixed at $k = 20$ (constant strategy) for $Loss_{\text{jac}}$ except for the experiments in subsections 4.1.4 and 4.1.5.

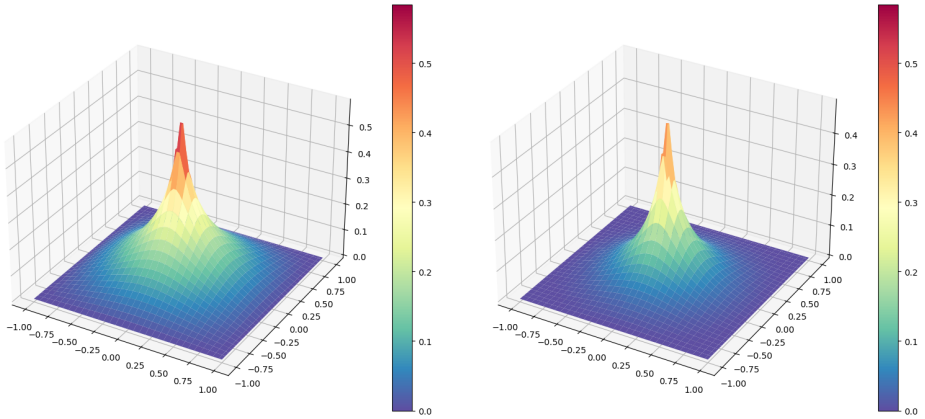
4.1.1. Effect of the U-Net architecture. The U-Net architecture used for the proposed deep surrogate model is determined by the number of channels of the first hidden layer (C_1) and the depth of its encoder/decoder (D), as already explained in subsection 2.1. We carefully investigate its effect on the performance of the model, and report the corresponding test results on the model sizes and the three training MSE losses (i.e., the residual-type loss $Loss_{\text{res}}$, the Jacobi-type loss $Loss_{\text{jac}}$, and the data-driven loss $Loss_{\text{data}}$) for the U-Net architecture under various values of C_1 and D in Table 1. Our observations include: 1) the prediction of the model equipped with the residual-type loss ($Loss_{\text{res}}$) is always unsatisfactory regardless of the choice of the U-Net architecture; 2) for a fixed depth D, the performance of the model will gradually improve as the number of channels C_1 increases; 3) for the model equipped with the Jacobi-type loss $Loss_{\text{jac}}$, the performance improvement of the model does not continue when the depth D increases up to a certain level. To balance the size and performance of the proposed deep surrogate model, we will use the U-Net architecture with $C_1 = 32$ and $D = 4$ in the subsequent analysis, which appears to perform the best in all cases based on Table 1.

TABLE 1. Results on the model sizes and the training MSE losses for the U-Net architecture under various values of C_1 and D.

C_1	D	Model Size	$Loss_{\text{res}}$	$Loss_{\text{jac}}$	$Loss_{\text{data}}$
4	3	15.1K	3.69e-3	2.29e-5	1.85e-4
8	3	59.1K	2.60e-3	1.16e-5	7.93e-6
16	3	234K	3.10e-3	8.25e-6	3.41e-6
32	3	930K	2.88e-3	7.79e-6	2.65e-6
4	4	59.7K	3.25e-3	2.39e-6	8.09e-6
8	4	236K	2.67e-3	1.49e-6	2.16e-6
16	4	940K	2.90e-3	1.61e-6	1.40e-6
32	4	3.8M	3.56e-3	1.15e-6	1.67e-6
4	5	237K	3.83e-3	2.58e-6	6.74e-6
8	5	943K	3.33e-3	6.85e-6	2.10e-6
16	5	3.8M	2.86e-3	1.61e-6	1.49e-6
32	5	15M	2.52e-3	1.22e-6	1.16e-6

4.1.2. Effect of the loss functions. The key of training the proposed deep surrogate model often lies in the choice of loss functions. Figure 2 presents a visual comparison of the Green's function computed by the finite difference method (as the reference solution) with those predicted by the proposed deep surrogate model equipped with the three different loss functions. The following observations are made: 1) the model equipped with the residual-type loss learns the rough shape of the Green's function but its detailed values are almost completely inaccurate; 2) the results predicted by the model equipped with the Jacobi-type loss and data-driven loss are very similar and both are quite accurate.

Comparisons of the contour maps of the reference solution and the predicted solutions with $Loss_{\text{jac}}$ and $Loss_{\text{data}}$ are provided in Figure 3, together with the corresponding L^2 errors. It is observed that for the the models equipped with $Loss_{\text{jac}}$ and $Loss_{\text{data}}$, the contour lines (indicating the gradient information) of the predicted solutions overlap well with those of the reference solution. Figure 4 presents the heat maps of the errors for the predicted solutions by using $Loss_{\text{jac}}$



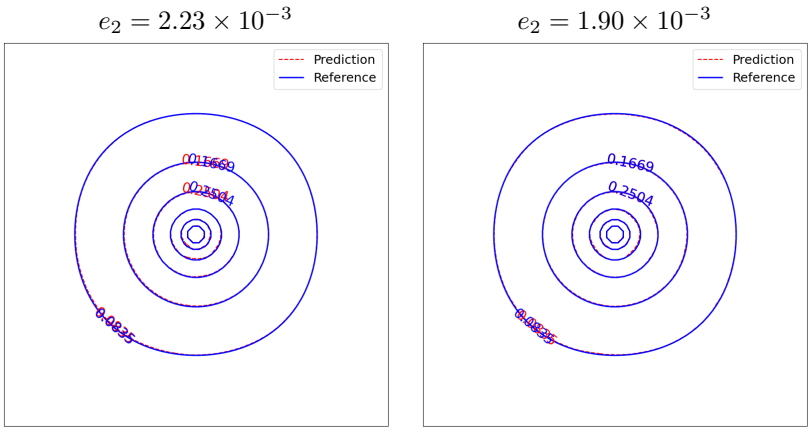
(a) Reference (b) $Loss_{\text{res}}$

(c) $Loss_{\text{jac}}$ (d) $Loss_{\text{data}}$

FIGURE 2. Plots of Green's functions $G(\boldsymbol{x}, \boldsymbol{\xi})$ at $\boldsymbol{\xi} = (0, 0)$ computed by the finite difference method (as the reference solution) and predicted by the proposed deep surrogate model equipped with the three different loss functions ($Loss_{\text{res}}$, $Loss_{\text{res}}$, $Loss_{\text{data}}$) respectively.

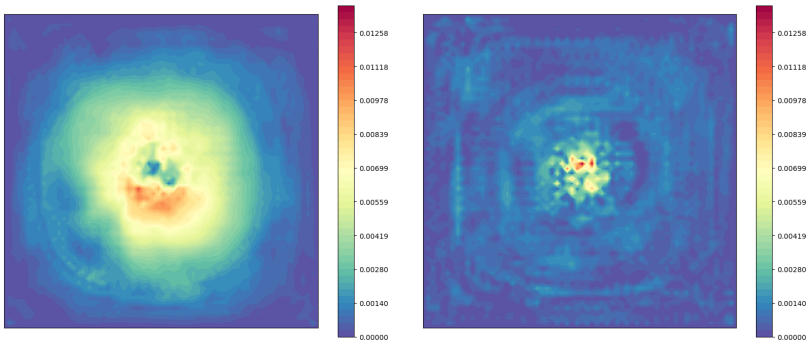
and $Loss_{\text{data}}$, from which, we find that the errors of the proposed model equipped with $Loss_{\text{jac}}$ are comparable to that of the model equipped with $Loss_{\text{data}}$.

4.1.3. Effect of input forms. Here we investigate the effect of different input forms on the the learned Green's function. As mentioned in subsection 2.3, we provide three types of input tensors, including 1-channel input $\boldsymbol{T}_{\boldsymbol{\xi}}^{(1)} = \boldsymbol{\rho}_{\boldsymbol{\xi}}$, 2-channels input $\boldsymbol{T}_{\boldsymbol{\xi}}^{(2)} = [\boldsymbol{R}_{\boldsymbol{\xi}}, \boldsymbol{\rho}_{\boldsymbol{\xi}}]$, and 3-channels input $\boldsymbol{T}_{\boldsymbol{\xi}}^{(3)} = [\boldsymbol{X}, \boldsymbol{\rho}_{\boldsymbol{\xi}}]$. Figure 5 shows the contour maps of the Green's functions $G(\boldsymbol{x}, \boldsymbol{\xi})$ at $\boldsymbol{\xi} = (0, 0)$ for the reference solution and the predicted solutions by the proposed deep surrogate model with different input forms, where $Loss_{\text{jac}}$ is used. We find that the results caused by these three input forms are almost the same, indicating that the performance of the deep surrogate model is mainly determined by the point source information and extra spatial location information isn't necessary. Therefore, we will use the first input form $\boldsymbol{T}_{\boldsymbol{\xi}}^{(1)}$ in following experiments.



(a) $Loss_{\text{jac}}$ (b) $Loss_{\text{data}}$

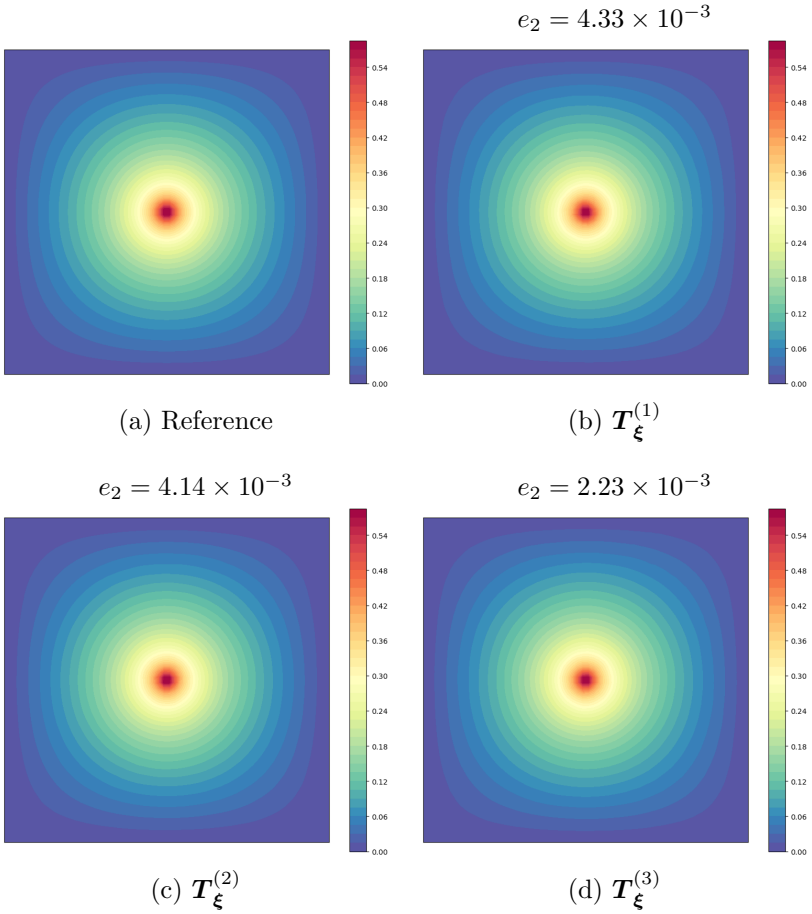
FIGURE 3. Comparisons of the contour maps of Green's functions $G(\boldsymbol{x}, \boldsymbol{\xi})$ at $\boldsymbol{\xi} = (0, 0)$ between the reference solution and the predicted solutions by the proposed deep surrogate model equipped with $Loss_{\text{jac}}$ and $Loss_{\text{data}}$ respectively. The corresponding L^2 errors (denoted as e_2) are also provided.



(a) $Loss_{\text{jac}}$ (b) $Loss_{\text{data}}$

FIGURE 4. Heat maps of the errors of Green's functions $G(\boldsymbol{x}, \boldsymbol{\xi})$ at $\boldsymbol{\xi} = (0, 0)$ for the predicted solutions by using $Loss_{\text{jac}}$ and $Loss_{\text{data}}$ respectively.

4.1.4. Effect of the number of Jacobi iterations in the constant strategy. The Jacobi scheme is a simple but important iterative method for solving large-scale linear systems. Here we study the effect of the number of Jacobi iterations in the constant strategy on the performance of the proposed deep surrogate model. Experimental results with a fixed number of Jacobi iterations k are shown in Fig. 6. We find that k has a significant impact on the model's performance: 1) when k is set to be relatively small, the difference between the predicted solution and the reference solution is significant (see Fig. 6-(a) and Fig. 6-(b)); 2) as k increases, the predicted solution gradually matches the reference solution (see Fig. 6-(c)), and subsequently, the number of iterations tends to be saturate, which means that further increase in k may not improve the predicted solution, and may even lead to



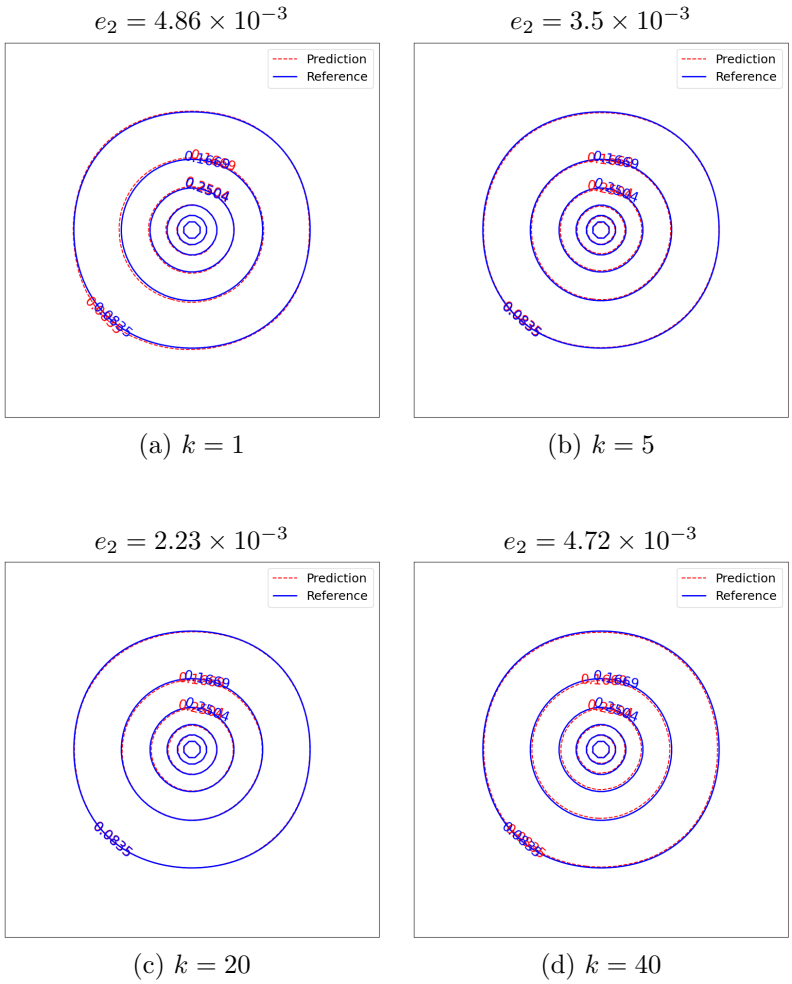
(a) Reference (b) $\boldsymbol{T}_{\boldsymbol{\xi}}^{(1)}$

(c) $\boldsymbol{T}_{\boldsymbol{\xi}}^{(2)}$ (d) $\boldsymbol{T}_{\boldsymbol{\xi}}^{(3)}$

FIGURE 5. Contour maps of the Green's functions $G(\boldsymbol{x}, \boldsymbol{\xi})$ at $\boldsymbol{\xi} = (0, 0)$ for the reference solution and the predicted solutions by the proposed deep surrogate model with different input forms. The corresponding L^2 errors (denoted as e_2) are also provided.

a poor predictive performance of the model (see Fig. 6-(d)). In fact, if k is set to be large enough, the approximate solution produced by the Jacobi iteration scheme is almost the exact solution, and our model then could be regarded as the data-driven model. As mentioned earlier, the predicted solution generated by the data-driven surrogate model lacks some regularized constraints, which partially explains the phenomenon in Fig 6-(d). That is to say, there is no need to choose large number for k in practice. Although the incomplete Jacobi iterations may produce imperfect approximate solutions, it still provides a good estimate (label) for the training of the surrogate model, then this estimate is somehow corrected by the back-propagation algorithm. As the training progresses, a more accurate regularized solution will be generated in the end.

4.1.5. Effect of the number of Jacobi iterations in the dynamic and adaptive strategies. Now we investigate the effect of the other two training strategies, including the dynamic and adaptive ways to adjust the number of Jacobi iterations during the training process, on the model's performance. In our setting, the



(a) $k = 1$ (b) $k = 5$

(c) $k = 20$ (d) $k = 40$

FIGURE 6. Comparisons of contour maps of the Green's functions $G(\boldsymbol{x}, \boldsymbol{\xi})$ at $\boldsymbol{\xi} = (0, 0)$ for the reference solution and the predicted solutions by the proposed deep surrogate model with different number of Jacobi iterations in the constant strategy. The corresponding L^2 errors (denoted as e_2) are also provided.

dynamic strategy is to initially set $k = 40$, and then reduce k by 10 for every 20 epochs until it is ultimately maintained at 10. The adaptive strategy is to initially set $k = 40$ for the first epoch, and then let k be adaptively adjusted in [0,20] the remaining process. Specifically, if $Loss_{\text{cur}} > 1.2Loss_{\text{pre}}$, multiply k by 2, and if $Loss_{\text{cur}} < 0.8Loss_{\text{pre}}$, divide k by 2, where $Loss_{\text{cur}}$ and $Loss_{\text{pre}}$ are the validation losses at the current and previous epochs, respectively.

The evolution of the number of Jacobi iterations k and the validation loss during the training process for three training strategies are shown in Fig. 7 and Fig. 8, respectively, where we fix $k = 20$ for the constant strategy. Compared with the constant strategy, the other two strategies, especially the adaptive strategy, significantly reduce the total number of Jacobi iterations, while the downtrend of the loss function remains similar for all three strategies. We present the contour

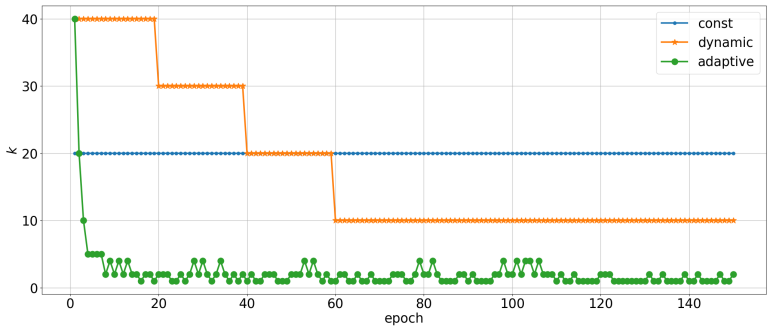


FIGURE 7. Evolution of the number of Jacobi iterations k during the training process for three training strategies, where we fix $k = 20$ for the constant strategy.

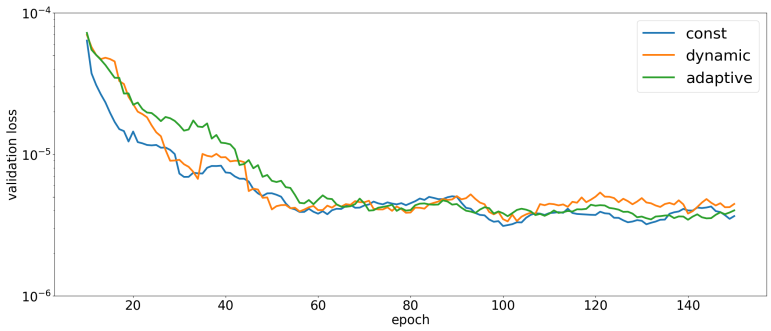


FIGURE 8. Evolution of the validation loss during the training process for three training strategies.

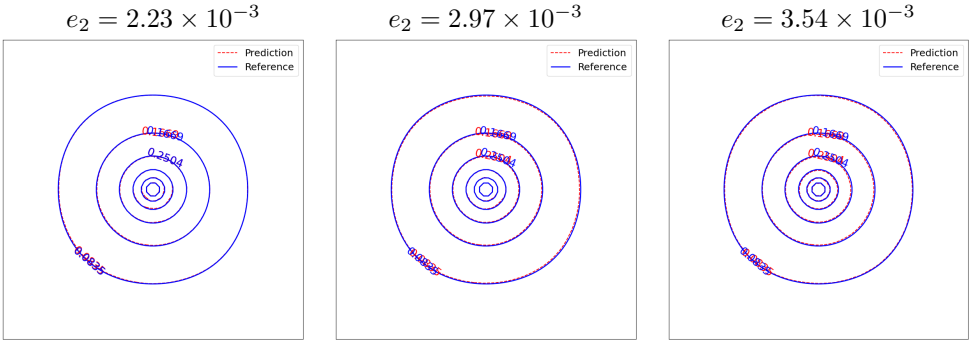
maps of the predicted Green's functions at $\boldsymbol{\xi} = (0, 0)$ by the proposed deep surrogate model with the three different training strategies in Fig. 9, from which it can be seen that the dynamic and adaptive strategies significantly reduce computational complexity, while the predicted results are comparable to those of the constant strategy. Therefore we will always use the adaptive strategy for all the remaining experiments.

4.2. More examples for learning Green's functions. More experimental results about the learned Green's functions for different linear reaction-differential operators, produced by the proposed deep surrogate model, are provided in this subsection. Comparisons of contour maps (and corresponding errors) shown in Fig. 10 verify that our model can produce accurate prediction results for Green's function of Laplacian operator at different source positions, even near the boundary or corner of the domain. We also investigate the deep surrogate model for learning Green's function of the reaction-diffusion operator (1) with the variable coefficients

$$a(\boldsymbol{x}) = 1 + 2x_2^2, \quad r(\boldsymbol{x}) = 1 + x_1^2, \tag{17}$$

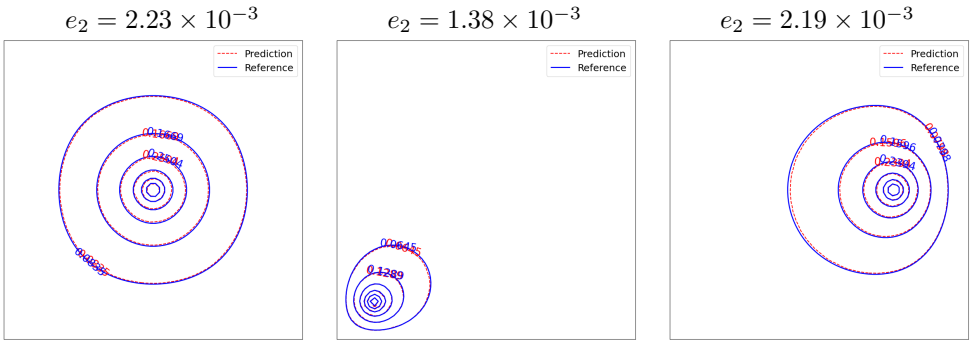
and the corresponding results are shown in Figure 11, which demonstrate our model again works very well.

4.3. Fast solvers for solving PDEs. In this subsection, we conduct some experiments for investigating performance of the fast solvers (16) based on the learned



(a) constant (b) dynamic (c) adaptive

FIGURE 9. Comparisons of contour maps of the Green's functions $G(\boldsymbol{x}, \boldsymbol{\xi})$ at $\boldsymbol{\xi} = (0, 0)$ for the reference solution and the predicted solutions by the proposed deep surrogate model with the three different training strategy. The corresponding L^2 errors (denoted as e_2) are also provided.



(a) $\boldsymbol{\xi} = (0, 0)$ (b) $\boldsymbol{\xi} = (-0.75, -0.75)$ (c) $\boldsymbol{\xi} = (0.5, 0)$

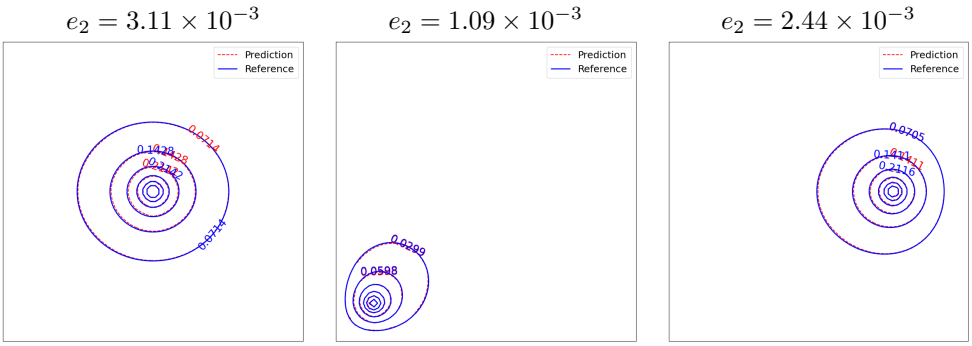
FIGURE 10. Comparisons of contour maps of the Green's functions $G(\boldsymbol{x}, \boldsymbol{\xi})$ for the reference solution and the predicted solutions by the proposed deep surrogate model for the Laplacian operator with different source positions. The corresponding L^2 errors (denoted as e_2) are also provided.

Green's function in solving the linear reaction-diffusion equation (2). For the choice of numerical quadratures, we adopt $I^{R_l}_{\boldsymbol{x},h}$ as the 2D Simpson's rule and $I^{E_m}_{\boldsymbol{x},h}$ as the 1D Simpson's rule, and the gradient $\nabla_{\boldsymbol{x}} G(\boldsymbol{x}, \boldsymbol{\xi})$ on the boundary is approximated by a first order difference scheme. For the visualization and the computation of L^2 error between the exact solution and the approximate soution provided by fast solvers, we approximately compute $u(\boldsymbol{\xi})$ by (16) on a 64×64 uniform mesh.

4.3.1. Laplacian equation. Let us consider the Laplacian equation. First, we choose the exact solution as

$$u(x_1, x_2) = \sin(\lambda \pi x_1) \sin(\lambda \pi x_2) \tag{18}$$

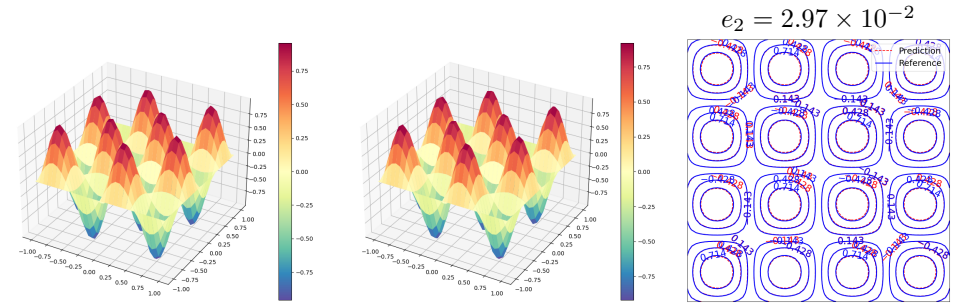
and the source term is determined accordingly, where λ is used to determine the frequency of the solution. In this case, the boundary condition is homogeneous,



(a) $\boldsymbol{\xi} = (0, 0)$ (b) $\boldsymbol{\xi} = (-0.75, -0.75)$ (c) $\boldsymbol{\xi} = (0.5, 0)$

FIGURE 11. Comparisons of contour maps of the Green's functions $G(\boldsymbol{x}, \boldsymbol{\xi})$ for the reference solution and the predicted solutions by the proposed deep surrogate model for the reaction-diffusion operator (17) with different source positions. The corresponding L^2 errors (denoted as e_2) are also provided.

i.e., $g = 0$, and the second integral in (5) vanishes. Plots of the exact solution, the numerical solution and their contour maps are shown in Figure 12 for the case of $\lambda = 2$ and Figure 13 for the case of $\lambda = 4$. We see that our fast solver produces good numerical solutions, and the L^2 errors are within acceptable ranges.



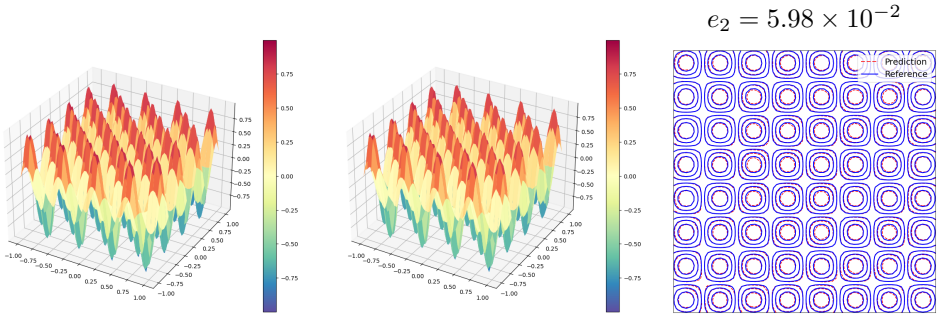
(a) reference (b) prediction (c) contour

FIGURE 12. Plots of the exact solution (left) and the numerical solution (middle) produced by the fast solver based on the learned Green's function for the Poisson equation with the solution $\sin(2\pi x_1)\sin(2\pi x_2)$. The comparison of their contour maps (right) is also provided with the corresponding L^2 errors (denoted as e_2).

Second, we choose the exact solution as

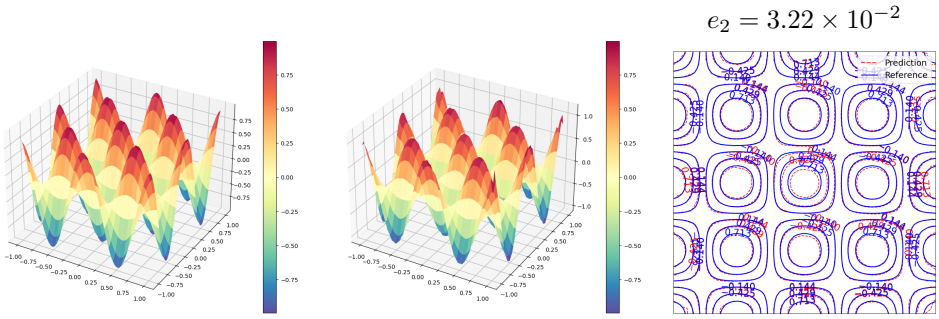
$$u(x_1, x_2) = \cos(\lambda \pi x_1)\cos(\lambda \pi x_2) \tag{19}$$

and the boundary condition (inhomogeneous now) and source term are determined accordingly. Plots of the exact solution, the numerical solution and their contour maps are shown in Figure 14 for the case of $\lambda = 2$ and Figure 15 for the case of $\lambda = 4$, from which we see that the fast solver is also suitable for solving the Laplacian equation with inhomogeneous boundary condition.



(a) reference (b) prediction (c) contour

FIGURE 13. Plots of the exact solution (left) and the numerical solution (middle) produced by the fast solver based on the learned Green's function for the Poisson equation with the solution $\sin(4\pi x_1)\sin(4\pi x_2)$. The comparison of their contour maps (right) is also provided with the corresponding L^2 errors (denoted as e_2).



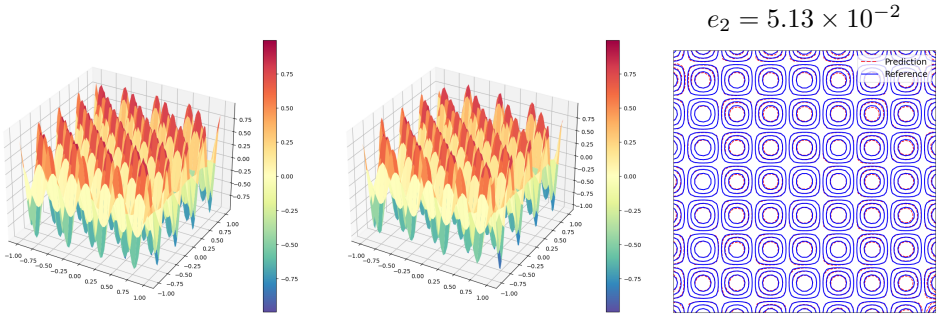
(a) reference (b) prediction (c) contour

FIGURE 14. Plots of the exact solution (left) and the numerical solution (middle) produced by the fast solver based on the learned Green's function for the Poisson equation with the solution $\cos(2\pi x_1)\cos(2\pi x_2)$. The comparison of their contour maps (right) is also provided with the corresponding L^2 errors (denoted as e_2).

4.3.2. The reaction-diffusion equation. We consider the reaction-diffusion equation with the coefficients defined in (17) and the exact solution is set to be

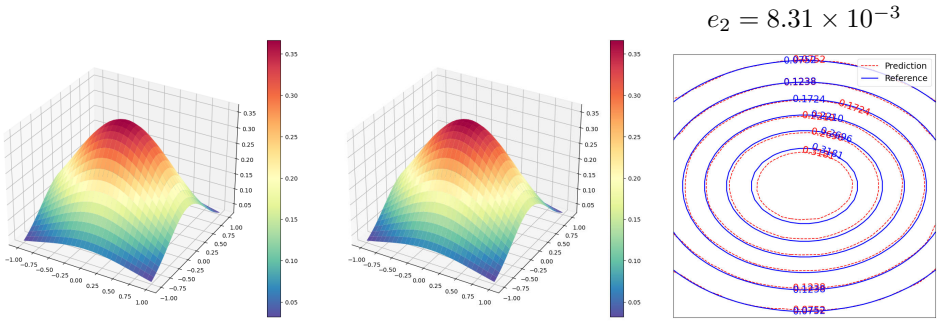
$$u(x_1, x_2) = 10^{-(x_1^2+2x_2^2+1)}. \tag{20}$$

The boundary condition and source term are then determined accordingly. Plots of the exact solution, the numerical solution and their contour maps are shown in Figure 16, from which we see that numerical solution produced by the fast solver (16) based on the learned Green's function again matches the exact solution very well.



(a) reference (b) prediction (c) contour

FIGURE 15. Plots of the exact solution (left) and the numerical solution (middle) produced by the fast solver based on the learned Green's function for the Poisson equation with the solution $\cos(4\pi x_1)\cos(4\pi x_2)$. The comparison of their contour maps (right) is also provided with the corresponding L^2 errors (denoted as e_2).



(a) reference (b) prediction (c) contour

FIGURE 16. Plots of the exact solution (left) and the numerical solution (middle) produced by the fast solver based on the learned Green's function for the linear reaction-diffusion equation with the coefficients (17) and the solution (20). The comparison of their contour maps (right) is also provided with the corresponding L^2 errors (denoted as e_2).

5. Conclusion

In this paper we propose and numerically study a deep surrogate model for learning Green's function of linear reaction-diffusion operator based on the U-Net architecture. Inspired by the Jacobi iteration scheme for solving linear systems, a novel Jacobi-type loss function and corresponding training strategies are designed and demonstrated to be very effective. In addition, a fast solver is tested and shown to be effective for numerically solving linear reaction-diffusion equations based on the learned Green's function. The proposed model is a beneficial attempt to integrate deep learning with traditional numerical methods. It fully utilizes the powerful expressive capability of neural networks, and on the other hand, it also combines advantages of traditional numerical methods.

Acknowledgments

X. Zhang's research is partially supported by Hubei Provincial Natural Science Foundation of China under grant number 2024AFB749 and the National Key Research and Development Program of China under grant number 2021YFD1900805-02. L. Ju's research is partially supported by US Department of Energy under grant number DE-SC0022254.

References

[1] Weinan E and Bing Yu. The deep ritz method: A deep learning-based numerical algorithm for solving variational problems. Communications in Mathematics and Statistics, 6(1):1–12, 2018.

[2] Justin A Sirignano and Konstantinos Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. Journal of Computational Physics, 375:1339–1364, 2018.

[3] 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.

[4] M. Dissanayake and N. Phan-Thien. Neural-network-based approximations for solving partial differential equations. Communications in Numerical Methods in Engineering, 10(3):195–201, 1994.

[5] Isaac E Lagaris, Aristidis Likas, and Dimitrios I Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. IEEE transactions on neural networks, 9(5):987–1000, 1998.

[6] Justin Sirignano and Konstantinos Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. Journal of Computational Physics, 375:1339–1354, 2018.

[7] Guofei Pang, Lu Lu, and George Em Karniadakis. fpinns: Fractional physics-informed neural networks. SIAM Journal on Scientific Computing, 41(4):A2603–A2626, 2019.

[8] Liu Yang, Dongkun Zhang, and George Em Karniadakis. Physics-informed generative adversarial networks for stochastic differential equations. SIAM Journal on Scientific Computing, 42(1):A292–A317, 2020.

[9] Dongkun Zhang, Lu Lu, Ling Guo, and George Em Karniadakis. Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic problems. Journal of Computational Physics, 39:108850, 2019.

[10] Dongkuan Zhang, Ling Guo, and George Em Karniadakis. Learning in modal space: Solving time- dependent stochastic pdes using physics-informed neural networks. Journal of Computational Physics, 42(2):A639–A665, 2020.

[11] Yulei Liao and Pingbing Ming. Deep nitsche method: Deep ritz method with essential boundary conditions, 2019.

[12] Johannes Mller and Marius Zeinhofer. Deep ritz revisited, 2019.

[13] Zhongjian Wang and Zhiwen Zhang. A mesh-free method for interface problems using the deep learning approach. Journal of Computational Physics, 400:108963, 2020.

[14] Jingrun Chen, Rui Du, and Keke Wu. A comparison study of deep galerkin method and deep ritz method for elliptic problems with different boundary conditions. Communications in Mathematical Research, 36(3):354–376, 2020.

[15] Xiaoping Zhang, Yan Zhu, Jing Wang, Lili Ju, Yingzhi Qian, Ming Ye, and Jinzhong Yang. Gw-pinn: A deep learning algorithm for solving groundwater flow equations. Advances in Water Resources, 165:104243, 2022.

[16] Xiang Huang, Hongsheng Liu, Beiji Shi, Zidong Wang, Kang Yang, Yang Li, Bingya Weng, Min Wang, Haotian Chu, Jing Zhou, Fan Yu, Bei Hua, Lei Chen, and Bin Dong. Solving partial differential equations with point source based on physics-informed neural networks, 2021.

[17] Xiang Huang, Zhanhong Ye, Hongsheng Liu, Beiji Shi, Zidong Wang, Kang Yang, Yang Li, Bingya Weng, Min Wang, Haotian Chu, Fan Yu, Bei Hua, Lei Chen, and Bin Dong. Meta-auto-decoder for solving parametric partial differential equations, 2022.

[18] Zhanhong Ye, Xiang Huang, Hongsheng Liu, and Bin Dong. Meta-auto-decoder: A meta-learning based reduced order model for solving parametric partial differential equations, 2023.

[19] Craig R. Gin, Daniel E. Shea, Steven L. Brunton, and J. Nathan Kutz. Deepgreen: deep learning of green's functions for nonlinear boundary value problems. Scientific reports, 11:21614, 2021.

[20] Nicolas Boulle, Christopher J. Earls, and Alex Townsend. Data-driven discovery of green's functions with human-understandable deep learning. Scientific reports, 12:4824, 2022.
[21] Yuankai Teng, Xiaoping Zhang, Zhu Wang, and Lili Ju. Learning green's functions of linear reaction-diffusion equations with application to fast numerical solver. Proceedings of Machine Learning Research, 3rd Annual Conference on Mathematics and Scientific Machine Learning, 145:1–22, 2022.
[22] Guochang Lin, Fukai Chen, Pipi Hu, Xiang Chen, Junqing Chen, Jun Wang, and Zuoqiang Shi. Bi-greennet: Learning green's functions by boundary integral network. Communications in Mathematics and Statistics, 11:103–129, 2023.
[23] Xiaoyu Zhao, Zhiqiang Gong, Yunyang Zhang, Wen Yao, and Xiaoqian Chen. Physics-informed convolutional neural networks for temperature field prediction of heat source layout without labeled data. Engineering Applications of Artificial Intelligence, 117:105516, 2023.

School of Mathematics and Statistics, Wuhan University, Wuhan, 430072, PR China
E-mail: whujjq@whu.edu.cn

Department of Mathematics,. University of South Carolina, Columbia, SC 29208, USA
E-mail: ju@math.sc.edu

School of Mathematics and Statistics, Wuhan University, Wuhan, 430072, PR China
E-mail: xpzhang.math@whu.edu.cn