SIAM J. SCI. COMPUT.
Vol. 46, No. 2, pp. C155–C185

NEURAL CONTROL OF PARAMETRIC SOLUTIONS FOR HIGH-DIMENSIONAL EVOLUTION PDEs*

NATHAN GABY†, XIAOJING YE‡, AND HAOMIN ZHOU§

Abstract. We develop a novel computational framework to approximate solution operators of evolution partial differential equations (PDEs). By employing a general nonlinear reduced-order model, such as a deep neural network, to approximate the solution of a given PDE, we realize that the evolution of the model parameters is a control problem in the parameter space. Based on this observation, we propose to approximate the solution operator of the PDE by learning the control vector field in the parameter space. From any initial value, this control field can steer the parameter to generate a trajectory such that the corresponding reduced-order model solves the PDE. This allows for substantially reduced computational cost to solve the evolution PDE with arbitrary initial conditions. We also develop comprehensive error analysis for the proposed method when solving a large class of semilinear parabolic PDEs. Numerical experiments on different high-dimensional evolution PDEs with various initial conditions demonstrate the promising results of the proposed method.

Key words. PDEs, machine learning, numerical analysis

MSC codes. 65M99, 65M15

DOI. 10.1137/23M1549870

1. Introduction.

Partial differential equations (PDEs) are ubiquitous in modeling and are vital in numerous applications from finance, engineering, and science [23]. As the solutions of many PDEs lack analytical form, it is necessary to use numerical methods to approximate the solutions [4, 23]. Traditional numerical methods such as finite difference and finite element methods rely upon the discretization of problem domains, which does not scale to high-dimensional problems due to the so-called curse of dimensionality.

In recent years, deep neural networks (DNNs), which can be thought of as a type of nonlinear reduced-order models, have emerged as powerful tools for solving high-dimensional PDEs [5, 16, 21, 32, 33, 35, 46, 73]. For example, in [5, 16, 21, 73, 93], the solution of a given PDE is parameterized as a DNN, and the network parameters are trained to minimize potential violations (in various definitions) to the PDE. These methods have shown numerous successes in solving a large variety of PDEs empirically. Their successes are partly due to the provable universal approximation power of DNNs [36, 52, 92]. On the other hand, these methods aim at solving specific instances of PDEs, and as a consequence, they need to start from scratch for the same PDE whenever the initial and/or boundary value changes.

*Submitted to the journal's Machine Learning Methods for Scientific Computing section January 30, 2023; accepted for publication (in revised form) November 9, 2023; published electronically March 6, 2024.

https://doi.org/10.1137/23M1549870

Funding: This work was supported in part by National Science Foundation grants DMS-1925263, DMS-2152960, DMS-2307465, DMS-2307466, and ONR N00014-21-1-2891.

†Department of Mathematics and Statistics, Georgia State University, Atlanta, GA 30303 USA (ngaby1@gsu.edu).

‡Corresponding author. Department of Mathematics and Statistics, Georgia State University, Atlanta, GA 30303 USA (xye@gsu.edu).

§School of Mathematics, Georgia Institute of Technology, Atlanta, GA 30332 USA (hmzhou@gatech.edu).

There have also been recent studies to find solution operators of PDEs [50, 57]. These methods aim at finding the map from the problem's parameters to the corresponding solution. Finding solution operators has substantial applications, as the same PDE may need to run many times with different initial or boundary value configurations. However, existing methods fall short in tackling high-dimensional problems, as many require spatial discretization to represent the solution operators using DNNs.

In this paper, we propose a new approach to find solution operators of high-dimensional evolution PDEs. For a given PDE, we first parameterize its solution as a general reduced-order model, such as a DNN, whose parameters, denoted as θ, are to be determined. Then we seek to find a vector field on the parameter space which describes how θ evolves in time. This vector field essentially acts as a controller on the parameter space, steering the parameters so that the induced DNN evolves and approximates the PDE solution for all time. Once such a vector field is found, we can easily change the initial conditions of the PDE by simply starting at a new point in the parameter space. Then we follow the control vector field to find the parameter trajectory, which gives an approximation of the time-evolving solution. Thus, different initial conditions can be considered for the same PDE without solving it repeatedly. Our contributions can be summarized as follows.

1. We develop a new computational framework to find the solution operator of any given initial value problem (IVP) defined by high-dimensional nonlinear evolution PDEs. This framework is purely based on the evolution PDE itself and does not require any solutions of the PDE for training. Once we find the solution operator, we can quickly compute solutions of the PDE with any initial value at a low computational cost.
2. We provide comprehensive theoretical analysis to establish error bounds for the proposed method when solving linear PDEs and some special nonlinear PDEs.
3. We conduct a series of numerical experiments to demonstrate the effectiveness of the proposed method in solving a variety of linear and nonlinear PDEs.

The remainder of this paper is organized as follows. In section 2, we provide an overview of recent neural network based numerical methods for solving PDEs. We outline the fundamentals of our proposed approach in section 3.1 and provide details of our method and its key characteristics in section 3.2. We conduct comprehensive error analysis in section 3.3. We demonstrate the performance of the proposed method on several linear and nonlinear evolution PDEs in section 4. Some variations and generalizations of the proposed approach are given in section 5. Finally, section 6 concludes this paper.

2. Related work.

2.1. Classical methods for solving PDEs. Classical numerical methods for solving PDEs, such as finite difference [84] and finite element methods [42], discretize the spatial domain using mesh or triangulation. These methods convert a PDE to its discrete counterpart, which is a system of algebraic equations with a finite number of unknowns, and solve the system to obtain approximate solution on the grid points [1, 22, 70, 83]. These methods have been significantly advanced in the past decades, and they are able to handle complicated situations such as irregular domains. However, they severely suffer the curse of dimensionality when applied to high-dimensional problems—the number of unknowns increases exponentially fast with respect to spatial dimension, which renders them computationally intractable for many problems.

2.2. Neural network based methods for solving PDEs. Early attempts using neural networks to solve PDEs can be seen in [17, 46, 47, 48]. DNNs emerged in recent years and demonstrated striking power in solving PDEs through various approaches [5, 7, 21, 65, 73, 78, 90, 93]. DNNs, which are the key machinery of deep learning, have demonstrated extraordinary potential in solving many high-dimensional nonlinear PDEs, which were considered computationally intractable using classical methods. For example, a variety of DNN based methods have been proposed based on the strong form [7, 17, 43, 61, 63, 66, 67, 73, 74], variational form [21], and weak form [5, 93] of PDEs. They are considered with adaptive collocation strategies [3], adversarial inference procedures [91], oscillatory solutions [12], and multiscale methods [13, 55, 85]. Improvements of these methods with adaptive activation functions [41], networks structures [26, 27, 38], boundary conditions [18, 60], and structure probing [38], as well as their convergence [59, 77], are also studied. Readers interested in these methods can also refer to [53, 74, 86, 87, 90, 94]. Further, there are methods that can solve inverse problems such as parameter identifications

For a class of high-dimensional PDEs which have equivalent backward stochastic differential equation (SDE) formulations due to Feynman–Kac theory, deep learning methods have been applied by leveraging such correspondences [6, 20, 25, 32, 33, 34, 39, 40, 69]. These methods are shown to be good even in high dimensions [33, 39, 69]; however, they are limited to solving the special type of evolution equations whose generator function has a corresponding SDE.

For evolution PDEs, parameter evolution algorithms [2, 10, 19] have also been considered. These methods parameterize the PDE solution as a neural network [10, 19] or an adaptively chosen ansatz as discussed in [2]. In these methods, the parameters are evolved forward in time through a time marching scheme, where at each step a linear system [10, 19] or a constrained optimization problem [2] needs to be solved.

2.3. Learning solution operator of PDEs. The aforementioned methods aim at solving a specific instance of a given PDE, and they need to be rerun from scratch when any of the problem configurations (e.g., initial value, boundary value, problem domain) changes. In contrast, the solution operator of a PDE directly maps a problem configuration to its corresponding solution. To this end, several methods have been proposed to approximate Green's functions for some linear PDEs [8, 9, 54, 82], as solutions to such PDEs have explicit expression based on their Green's functions. However, this approach only applies to a small class of linear PDEs whose solution can be represented using Green's functions. Moreover, Green's functions have singularities and it requires special care to approximate them using neural networks. For example, rational functions are used as activation functions of DNNs to address singularities in [8]. In [9], the singularities are represented with the help of fundamental solutions.

For general nonlinear PDEs, DNNs have been used for operator approximation and metalearning for PDEs [30, 50, 57, 58, 62, 76, 88, 89]. For example, the work [30] considers solving parametric PDEs in low dimension ($d \leq 3$ for the examples in the paper). Their method requires discretization of the PDE system and needs to be supplied by many full-order solutions for different combinations of time discretization points and parameter selections for their network training. Then their method applies proper orthogonal decomposition to these solutions to obtain a set of reduced basis to construct solutions for new problems. The work [76] requires a massive amount of pairs of ODE/PDE control and the corresponding system outputs, which are produced by solving the original ODE/PDE system; then the DNN is trained on such pairs to learn the mapping between these two subjects, which are discretized as vectors by evaluating the functions only at grid points in the domain.

DeepONets [57, 58, 88] seek to approximate solution mappings by use of a "branch" and "trunk" network. FNOs [50, 89] use Fourier transforms to map a neural network to a low-dimensional space and then back to the solution. In addition, several works apply spatial discretization of the problem or transform domains and use convolutional neural networks (CNNs) [31, 75, 95] or graph neural networks (GNNs) [45, 51, 56]. Interested readers may also refer to generalizations and extensions of these methods in [11, 14, 15, 24, 44, 51, 58, 62, 66]. A key similarity of all these methods is that they require certain domain discretization and often a large number of labeled pairs of IVP initial conditions (or PDE parameters) and the corresponding solution obtained through other methods for training. This limits their applicability to high-dimensional problems where such training data is unavailable or the mesh is prohibitive to generate due to cures of dimensionality.

2.4. Differences between our proposed approach and existing ones. Different from all existing approaches, we propose to approximate solution operators of evolution PDEs in a control framework in parameter spaces induced by general reduced-order models such as DNNs. Unlike the existing solution operator approximation methods (e.g., DeepONet [57] and FNO [50]), which seek to directly approximate the infinite-dimensional operator, our approach is based on the relation between evolving solutions and their projected trajectories in the parameter space. This leads us to convert the problem of finding a solution operator over infinite-dimensional function space into a control vector field optimization problem over a finite-dimensional parameter space. As a result, the problem of solving an evolution PDE in continuous space is reduced to numerically solving a system of ODEs, which can be done accurately with very low computation complexity. Moreover, our approach neither requires spatial discretization in any problem or transformed domain, nor does it need any basis function representation throughout problem formulation and computation. We provide mathematical insights into the parameter submanifold and its tangent spaces and establish their connection to the finite-dimensional parameter space. These new insights led us to the proposed approach, which approximates solution operators of PDEs by controlling network parameters in the parameter space. These new features also enable our approach to solving evolution PDEs in high-dimensional cases. This is a significant advantage over existing operator learning methods such as DeepONet or FNOs, as their spatial discretization schemes, which are used to generate the training data, hinder their application to high-dimensional cases.

3. Proposed method. The main goal of this paper is to develop a new computational framework to approximate the *solution operator* for IVPs of high-dimensional evolution PDEs. The solution operator is a procedure that, once known, can efficiently map an arbitrarily given initial value g to the solution of the IVP without solving the PDE again. We first propose to parameterize u as a *nonlinear reduced-order model,* such as a DNN, which is denoted by u_θ with parameters θ; i.e., u_θ is a parametric function determined by the value of its finite-dimensional parameters θ, and u_θ is used to approximate u.

To find the solution operator, we propose to build a control vector field V in the parameter space Θ where θ resides. Then the solution operator can be implemented as a fast numerical solver of the ODE defined by V. More precisely, we first find the parameters θ_0 such that u_{θ_0} approximates g, and then we follow the control vector field V to obtain a trajectory $\{\theta_t \,|\, 0 \le t \le T\}$ in Θ with very low computational cost, which automatically induces a trajectory u_{θ_t} to approximate the true solution u of the IVP with the initial value g. We provide details of these constructions in the following subsections.

3.1. Nonlinear reduced-order models and parameter submanifold. DNNs, which can be viewed as nonlinear reduced-order models, have emerged as powerful tools to solve high-dimensional PDEs in recent years [5, 21, 32, 71, 72, 73, 93]. Mathematically, a DNN can be expressed as the composition of a series of simple linear and nonlinear functions. In the deep learning context, a typical building block of DNNs is called a layer, which is a mapping $h : \mathbb{R}^d \to \mathbb{R}^{d'}$ for some compatible input dimension d and output dimension d':

$$h(z; W, b) := \sigma(Wz + b), \tag{3.1}$$

where $z \in \mathbb{R}^d$ is the input variable of h, the matrix $W \in \mathbb{R}^{d' \times d}$ and vector $b \in \mathbb{R}^{d'}$ are called the weight and bias, respectively, and $\sigma : \mathbb{R} \to \mathbb{R}$ is a nonlinear function that operates componentwise on its d'-dimensional argument vector $Wz + b$ (hence σ is effectively a mapping from $\mathbb{R}^{d'}$ to $\mathbb{R}^{d'}$). Common choices of activation functions include the hyperbolic tangent (tanh) and rectified linear unit (ReLU) $\sigma(z) = \max(0, z)$. We only consider smooth activation functions σ hereafter. A commonly used DNN structure u_θ, often called a feed-forward network (FFN), is defined as the composition of multiple layer functions of form (3.1) as follows:

$$\begin{aligned} &u_\theta(x) := u(x; \theta) = w^\top z_L + b, \\ &\text{where} \quad z_0 = x, \quad z_l = h_l(z_{l-1}) := h(z_{l-1}; W_l, b_l), \quad l = 1, \dots, L, \end{aligned} \tag{3.2}$$

and the lth hidden layer $h(\cdot; W_l, b_l) : \mathbb{R}^{d_{l-1}} \to \mathbb{R}^{d_l}$ is determined by its weight and bias parameters $W_l \in \mathbb{R}^{d_l \times d_{l-1}}$ and $b_l \in \mathbb{R}^{d_l}$ for $l = 1, \dots, L$ and $d_0 = d$. Here the output of u_θ is set to the affine transform of the last hidden layer $z_{NN} = h_L(z_{L-1})$ using weight $w \in \mathbb{R}^{d_L}$ and bias $b \in \mathbb{R}$. The *network parameters* θ refer to the collection of all learnable parameters (stacked as a vector in $\mathbb{R}^m$) of u_θ, i.e.,

$$\theta := (w, b, W_L, b_L, \dots, W_1, b_1) \in \mathbb{R}^m, \tag{3.3}$$

and training the network u_θ refers to finding the minimizer θ of some properly designed loss function.

Remark 3.1. DNNs are shown to be very powerful in approximating high-dimensional functions in a vast amount of studies in recent years; see, e.g., [28, 29, 36, 37, 49, 52, 68, 92]. For example, it is shown in [28] that for any $M, \varepsilon > 0$, $k \in \mathbb{N}$, $p \in [1, \infty]$, and $\Omega = (0, 1)^d \subset \mathbb{R}^d$, we denote $\mathcal{F} := \{f \in W^{k,p}(\Omega; \mathbb{R}) \,|\, \|f\|_{W^{k,p}(\Omega)} \le M\}$, and then there exists a DNN structure u_θ of form (3.2) with sufficiently large m and L (which depend on M, ε, d, and p only), such that for any $f \in \mathcal{F}$, there is $\|u_\theta - f\|_{W^{k,p}(\Omega)} \le \varepsilon$ for some $\theta \in \mathbb{R}^m$. This result suggests that DNNs are suitable to approximate solutoins of PDEs. We note that this is one of the many error bounds of DNN approximations established in recent years, and such bounds are still being continuously improved nowadays.

Our approach relies on the key relation between the parameters θ and the reduced-order model u_θ. More specifically, we identify the finite-dimensional parameter space $\Theta \subset \mathbb{R}^m$ to which θ belongs and the submanifold $\mathcal{M}$ of functions defined by

$$\mathcal{M} := \big\{u_\theta : \Omega \to \mathbb{R} \mid \theta \in \Theta\big\}. \tag{3.4}$$

As we can see, u_θ defines a mapping from the parameter space Θ to the submanifold $\mathcal{M}$ of the infinite-dimensional function space. We call $\mathcal{M}$ the *parameter submanifold* determined by u_θ.

To approximate a time-evolving function $u^*(\cdot,t)$, e.g., the solution of an evolution PDE, over time horizon $[0,T]$ using the reduced-order model u_θ, we need to find a trajectory $\{\theta_t \in \Theta \,|\, 0 \le t \le T\}$ in the parameter space Θ so that $u_{\theta_t}(\cdot)$ is close to $u^*(\cdot,t)$ in the function space for every $t \in [0,T]$. For example, if we consider $L^2(\Omega)$ as the function space, by closeness we mean $\|u_{\theta_t} - u^*(\cdot,t)\|_{L^2(\Omega)}$ is small for all t (hereafter we denote $\|\cdot\|_p = \|\cdot\|_{L^p(\Omega)}$ for notation simplicity). Notice that $\{u_{\theta_t} \,|\, 0 \le t \le T\}$ is a trajectory on $\mathcal{M}$, whereas $u^*(\cdot,t)$ is a trajectory in the full space $L^2(\Omega)$.

3.2. Proposed methodology. Let Ω be an open bounded set in $\mathbb{R}^d$ and F be a *nonlinear differential operator* of functions $u : \Omega \to \mathbb{R}$ with necessary regularity conditions; then we consider the IVP of the evolution PDE defined by F with arbitrary initial value as follows:

$$\begin{cases} \partial_t u(x,t) = F[u](x,t), & x \in \Omega,\ t \in (0,T], \\ u(x,0) = g(x), & x \in \Omega, \end{cases} \tag{3.5}$$

where $T > 0$ is some prescribed terminal time, and $g : \mathbb{R}^d \to \mathbb{R}$ stands for an initial value. For ease of presentation, we assume zero Dirichlet boundary condition $u(x,t) = 0$ for all $x \in \bar{\Omega}$ and $t \in [0,T]$ (for compatibility, we henceforth assume $g(x)$ has zero trace on $\partial\Omega$) throughout this paper. We denote by u^g the solution to the IVP (3.5) with this initial g. The solution operator $\mathcal{S}_F$ of the IVP (3.5) is thus the mapping from the initial g to the solution u^g:

$$\mathcal{S}_F : C^2(\bar{\Omega}) \to C^{2,1}(\bar{\Omega} \times [0,T]), \quad \text{such that} \quad g \mapsto \mathcal{S}_F(g) := u^g, \tag{3.6}$$

where $C^2(\bar{\Omega}) := C(\bar{\Omega}) \cap C^2(\Omega)$ for short. Our goal is to find a numerical approximation to $\mathcal{S}_F$. Namely, *we want to find a fast computational scheme $\mathcal{S}_F$ that takes any initial g as input and accurately estimate u^g with low computation complexity.*

It is important to note the substantial difference between solving (3.5) for any given but fixed initial value g and finding the solution operator (3.6) that maps any g to the corresponding solution u^g. In the literature, most methods are developed for solving IVP (3.5) with a fixed g, such as traditional finite difference and finite element methods, as well as many state-of-the-art machine learning based methods. However, these methods are computationally expensive if (3.5) must be solved with many different initial values, and they need to start from scratch for every new g. In a sharp contrast, our goal is to find an approximation to the solution operator $\mathcal{S}_F$ which, once found, can help us to compute u^g for any given g at relatively much lower computational cost.

For ease of presentation, we use autonomous, second-order nonlinear differential operators $F[u] = F(x, u, \nabla_x u, \nabla_x^2 u)$ as an example and take $\Omega = (0,1)^d$ in (3.5) to describe our main idea below. Extensions to general nonautonomous nonlinear differential operators and PDEs defined on open bounded set $\Omega \subset \mathbb{R}^d$ with given boundary values will be discussed in section 5.

To approximate the solution operator $\mathcal{S}_F$ in (3.6), we propose *a control mechanism in the parameter space Θ of a prescribed reduced-order model u_θ.* Specifically, we first determine a reduced-order model u_θ to represent solutions of the IVP. We allow any parametric form of u_θ but only assume that $u_\theta(x) = u(x;\theta)$ is C^1 smooth with respect to θ. This is a mild condition satisfied by the commonly used reduced-order models: if u_θ is a linear combination of basis functions and θ represents the combination coefficients, then u_θ is linear and hence smooth in θ; and if u_θ is a DNN as in (3.2),

then u_θ is smooth in θ as long as all activation functions σ are smooth. Suppose there exists a trajectory $\{\theta_t \,|\, 0 \le t \le T\}$ in the parameter space Θ such that its corresponding u_{θ_t} approximates the solution of the IVP; then we must have

$$\begin{cases} \partial_t u_{\theta_t}(x) = \nabla_\theta u(x;\theta_t)\cdot \dot\theta_t = F[u_{\theta_t}](x) & \forall\, x\in\Omega,\ t\in(0,T], \\ u_{\theta_0}(x) = g(x) & \forall\, x\in\Omega. \end{cases} \tag{3.7}$$

To compute u_{θ_t}, it is sufficient to find a control vector (velocity) field $V_F : \Theta \to \mathbb{R}^m$, in the sense of $\dot\theta_t = V_F(\theta_t)$, that steers the trajectory θ_t along the correct direction starting from the initial θ_0 satisfying $u_{\theta_0}(x) = g(x)$.

This observation suggests a new approach to solving the IVP with a fixed evolution PDE but varying initial values g: for the evolution equation in (3.7) to hold, it suffices to find a vector field V_F such that

$$\nabla_\theta u_\theta \cdot V_F(\theta) = F[u_\theta] \tag{3.8}$$

for all $\theta \in \Theta$. It is important to note that V_F only depends on the nonlinear differential operator F of the original evolution PDE but not any actual initial value g of the IVP. Once this is achieved, we can effectively approximate the solution of the IVP with any initial value g: we first set $\theta_0 = \theta^g$, where θ^g denotes the parameters such that u_{θ^g} fits g, and then we numerically solve the following ODE in the parameter space Θ (which can be fast) using the control vector field V_F:

$$\begin{cases} \dot\theta_t = V_F(\theta_t) & \forall\, t\in(0,T], \\ \theta_0 = \theta^g. \end{cases} \tag{3.9}$$

The solution trajectory $\{\theta_t \,|\, 0 \le t \le T\}$ of the ODE (3.9) induces a path $\{u_{\theta_t} \,|\, 0 \le t \le T\}$ in $\mathcal{M}$ as an approximation to the solution of the IVP. The computational cost is thus composed of two parts: finding the parameters θ^g of u_θ to fit g and numerically solving the ODE (3.9), both of which are substantially cheaper than solving the IVP (3.5).

The main question is how to get the control vector field V_F in (3.9). As an explicit form of V_F is unknown, we choose to express V_F in a general parametric form V_ξ with parameters ξ to be determined. Specifically, we propose to set V_ξ as another DNN where ξ represents the set of learnable network parameters in V_ξ. A schematic plot of the pullback mechanism and the control vector field in Θ is provided in Figure 1. We call V_ξ the *neural control* field. We learn the parameters ξ by minimizing the following loss function:

$$\ell(\xi) := \int_\Theta \|\nabla_\theta u_\theta \cdot V_\xi(\theta) - F[u_\theta]\|_2^2 \, d\theta. \tag{3.10}$$

In practice, we approximate the integral in ℓ by Monte Carlo integration. We sample K points $\{\theta_k \,|\, k = 1,\dots,K\}$ uniformly from Θ (here the subscript k in θ_k stands for the kth point among the K points sampled in Θ) and form the empirical loss function

$$\hat\ell(\xi) = K^{-1}\cdot \sum_{k=1}^K \|\nabla_\theta u_{\theta_k} \cdot V_\xi(\theta_k) - F[u_{\theta_k}]\|_2^2. \tag{3.11}$$

Then we minimize $\hat\ell(\xi)$ with respect to ξ, where the L^2 norm is also approximated by Monte Carlo integration on Ω. The training of V_ξ is summarized in Algorithm 3.1.

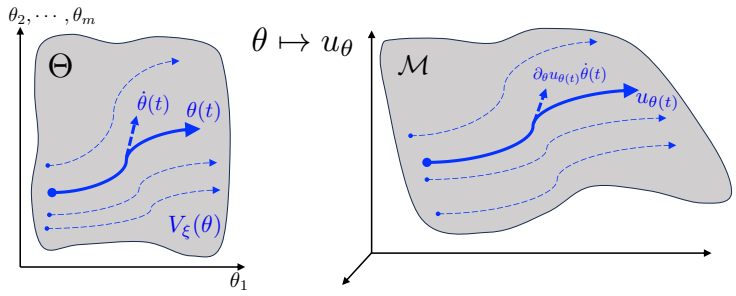

FIG. 1. *Schematic plot of pulling back trajectories (solid and dashed blue curves) in* $\mathcal{M} = \{u_\theta : \theta \in \Theta\}$ *to trajectories in the parameter space* Θ*. Here each trajectory in* $\mathcal{M}$ *represents the reduced-order model (e.g., DNN)* $u_{\theta(t)}(\cdot)$ *approximating the PDE solution* $u^*(t, \cdot)$ *starting from a given initial, and it is pulled back to the trajectory* $\theta(t)$ *(we use* $\theta(t) := \theta_t$ *as a trajectory here to avoid confusion with components* $\theta_1, \ldots, \theta_m$*) in* Θ*; and* V_ξ *is a DNN approximating the control vector field* V_F *in* Θ*.*

Algorithm 3.1. Training neural control V_ξ.

Input: Reduced-order model structure u_θ and parameter set Θ. Control vector field structure V_ξ. Error tolerance ε.

Output: Optimal control parameters ξ.

1: Sample $\{\theta_k\}_{k=1}^K$ uniformly from Θ and $\{x_n\}_{n=1}^N$ from Ω.
2: Form empirical loss $\hat\ell(\xi)$ as in (4.2).
3: Minimize $\hat\ell$ with respect to ξ using any optimizer (e.g., ADAM or AdaGrad) until $\hat\ell(\xi) \le \varepsilon$.

Algorithm 3.2. Implementation of solution operator $\mathcal{S}_F$ of the IVP (3.5) using trained control V_ξ.

Input: Initial value g and tolerance ε_0. Reduced-order model u_θ and trained neural control V_ξ.

Output: Trajectory $\hat\theta_t$ such that $u_{\hat\theta_t}$ approximate the solution $\mathcal{S}_F[g]$ of the IVP (3.5).

1: Compute initial parameters θ_0 such that $\|u_{\theta_0} - g\|_2 \le \varepsilon_0$.
2: Use any ODE solver to compute $\hat\theta_t$ to solve (3.9) with approximate field V_ξ and initial θ_0.

Once we have trained the vector field V_ξ, we can implement the solution operator $\mathcal{S}_F$ in the following two steps: we first find a θ_0 such that u_{θ_0} fits g, i.e., find a θ_0 that minimizes $\|u_\theta - g\|_2$. This can be done by sampling $\{x_n\}_{n=1}^N$ from Ω and minimizing the empirical squared L^2 norm $(1/N) \cdot \sum_{n=1}^N |u_\theta(x_n) - g(x_n)|^2$ with respect to θ. Then we solve the ODE (3.9) using any numerical ODE solver (e.g., Euler, 4th-order Runge–Kutta, predictor-corrector) with θ_0 as the initial value. Both steps can be done efficiently, and the total computational cost is substantially lower than that of solving the original IVP (3.5) again. We summarize how neural control solves IVPs in Algorithm 3.2. Further details on the practical implementation of Algorithms 3.1 and 3.2 are discussed in section 4.

3.3. Error analysis. In this subsection, we develop an error estimate of the proposed method. We first focus on the error due to projection onto the tangent space

$T_{u_\theta}\mathcal{M}$ in the L^2 space in section 3.3.1. Then we establish the solution approximation error for linear and semilinear parabolic PDEs in section 3.3.2. For ease of discussion, we again assume zero Dirichlet boundary condition $u(x,t) = 0$ for all $x \in \bar{\Omega}$ and $t \in [0,T]$, and we let $\Omega = (0,1)^d \subset \mathbb{R}^d$ be the unit open cube in $\mathbb{R}^d$ and Θ some open bounded set in $\mathbb{R}^m$ (note that our analysis below applies as long as Ω is open and bounded). We let $F[u] := F(u, \nabla u, \nabla^2 u)$ be a nonlinear differential operator with necessary regularity conditions to be specified later and which allows for a unique solution to the PDE for each initial. Additional requirements on the regularity of u_θ will be given when needed.

3.3.1. Approximation error of control vector field. We first investigate the main source of error when using a reduced-order model to approximate the time-evolving solution of the given PDE. We show that this error is due to the imperfect representation of $F[u_\theta]$ using $\nabla_\theta u_\theta$ in (3.8). Specifically, due to the approximation of reduced-order models, $T_{u_\theta}\mathcal{M}$ is only a finite-dimensional subspace of L^2, and thus we can only approximate the projection of $F[u_\theta]$ onto this tangent space. We will need the following assumptions on the regularity of u_θ and F.

Assumption 1. The reduced-order model $u_\theta(\cdot) \in C^3(\Omega) \cap C(\bar{\Omega})$ for every $\theta \in \bar{\Theta}$ and $u(x;\cdot) \in C^2(\Theta) \cap C(\bar{\Theta})$. Moreover, there exists $L > 0$ such that for all $\theta \in \bar{\Theta}$

$$F[u_\theta] \in \mathcal{F}^L := \{f \in C^1(\Omega) \cap C(\bar{\Omega}) : \|f\|_\infty \le L,\ \|\nabla f\|_\infty \le L\}. \tag{3.12}$$

Assumption 1 provides some sufficient regularity conditions on the reduced-order model u_θ and boundedness of $F[u_\theta]$ and its gradient to be used in our error estimates. Notice that we consider F as second-order differential operator here and therefore the assumption $u_\theta \in C^3(\Omega)$ ensures that $u_\theta(x), \nabla u_\theta(x), \nabla^2 u_\theta(x)$ are all sufficiently smooth. The regularity condition on F in Assumption 1 requires that the mapping $F[u_\theta](x)$ is a C^1 function and has magnitudes and gradients bounded by L over $\bar{\Omega}$. These assumptions are generally mild, as we will use reduced-order models smooth in (x,θ), e.g., a DNN with smooth activation functions, and the operator F is sufficiently regular.

Assumption 2. For any $\bar{\varepsilon} > 0$, there exist a reduced-order model u_θ and a bounded open set $\Theta \subset \mathbb{R}^m$, such that for every $\theta \in \bar{\Theta}$ there exists a vector $\alpha_\theta \in \mathbb{R}^m$ satisfying

$$\|\alpha_\theta \cdot \nabla_\theta u_\theta - F[u_\theta]\|_2 \le \bar{\varepsilon}.$$

Assumption 2 provides an upper bound on the error when projecting $F[u_\theta]$ onto the tangent space $T_{u_\theta}\mathcal{M}$, which is spanned by the functions in $\nabla_\theta u_\theta$. This error bound is determined by the choice of the reduced-order model u_θ and the parameter set Θ. As will be demonstrated in our numerical experiments, a small projection error can be achieved by using a standard DNN as reduced-order model u_θ. As such, an error is difficult to analyze due to the complex structures of general DNNs. We provide an example reduced-order model with special structure to justify the reasonableness of Assumption 2.

Example 3.2. Let $\bar{\varepsilon} > 0$, and let $\{\varphi_j\}_{j=1}^\infty$ be a complete smooth orthonormal basis (e.g., generalized Fourier basis) for $L^2(\Omega)$. Suppose there exist $C > 0$, $\gamma > 1$, and $C_0 > 0$ such that for all $u \in C^3(\Omega) \cap C(\bar{\Omega})$ and $\|u\|_2^2 \le C_0$ we have

$$F[u] \in \mathcal{G}^{C,\gamma} := \left\{f \in C^1(\Omega) \cap C(\bar{\Omega}) : |\langle f, \varphi_j\rangle|^2 \le C j^{-\gamma}\ \forall\, j \ge 1\right\}. \tag{3.13}$$

Then there exists $m = m(\bar{\varepsilon}, C, \gamma) \in \mathbb{N}$ such that $\sum_{j=m+1}^{\infty} Cj^{-\gamma} < \bar{\varepsilon}^2$. Consider $u_\theta = \theta \cdot \varphi = \sum_{j=1}^{m} \theta_j \varphi_j$. We denote $f_\theta := F[u_\theta]$ for short. Then $\nabla_\theta u_\theta = \varphi = (\varphi_1, \dots, \varphi_m)$ and for $\alpha^{f_\theta} = (\alpha_1^{f_\theta}, \dots, \alpha_m^{f_\theta})$ with $\alpha_j^{f_\theta} := \langle f_\theta, \varphi_j \rangle$, there is

$$\|\alpha^{f_\theta} \cdot \nabla_\theta u_\theta - F[u_\theta]\|_2^2 = \left\| \sum_{j=1}^{m} \alpha_j^{f_\theta} \varphi_j - f_\theta \right\|_2^2 = \sum_{j=m+1}^{\infty} |\langle f_\theta, \varphi_j \rangle|^2 \le \bar{\varepsilon}^2.$$

Therefore, the reduced-order model $u_\theta = \theta \cdot \varphi$ with $\Theta = \{\alpha \in \mathbb{R}^m : |\alpha|^2 < C_0\}$ and $\alpha_\theta = \alpha^{f_\theta}$ satisfies Assumption 2.

This example can be modified to use a more general form of reduced-order model u_θ, such as a DNN. To see this, we first repeat the procedure above but with $\bar{\varepsilon}$ replaced by $\bar{\varepsilon}/2$. Then the universal approximation theorem [36, 92] and the continuity of DNNs in its parameters imply that there exist DNNs $\{\hat{\varphi}_j : 1 \le j \le m\}$, whose network parameters are collectively denoted by $\eta \in \mathbb{R}^{m'}$, which satisfy $\|\hat{\varphi}_j - \varphi_j\|_\infty \le \bar{\varepsilon}/(2\sqrt{mC_0|\Omega|})$ and hence $\|\hat{\varphi}_j - \varphi_j\|_2 \le \bar{\varepsilon}/(2\sqrt{mC_0})$ for all η in an open set $H \subset \mathbb{R}^{m'}$. Consider the DNN $u_\theta = c \cdot \hat{\varphi}$ with parameters $\theta = (c, \eta) \in \mathbb{R}^n$, where $n = m + m'$. Then $\nabla_c u_\theta(x) = (\hat{\varphi}_1, \dots, \hat{\varphi}_m)$. Using the example above, we know that for any $f_\theta := F[u_\theta] \in \mathcal{G}^{C,\gamma}$, there exists $\alpha^{f_\theta} \in \mathbb{R}^m$ such that $\|\alpha^{f_\theta} \cdot \varphi - F[u_\theta]\|_2 \le \bar{\varepsilon}/2$. Therefore, we use $(\alpha^{f_\theta}, 0)$, which concatenates α^{f_θ} and $0 \in \mathbb{R}^{m'}$ as the combination coefficients of $\nabla_\theta u_\theta$ to obtain

$$\begin{aligned}
\|(\alpha^{f_\theta}, 0) \cdot \nabla_\theta u_\theta - F[u_\theta]\|_2 &= \|\alpha^{f_\theta} \cdot \nabla_c u_\theta - F[u_\theta]\|_2 \\
&\le \|\alpha^{f_\theta} \cdot \hat{\varphi} - \alpha^{f_\theta} \cdot \varphi\|_2 + \|\alpha^{f_\theta} \cdot \varphi - F[u_\theta]\|_\infty \\
&\le \sum_{j=1}^{m} |\alpha_j^{f_\theta}| \|\hat{\varphi}_j - \varphi_j\|_2 + \frac{\bar{\varepsilon}}{2} \\
&\le \sqrt{mC_0} \cdot \frac{\bar{\varepsilon}}{2\sqrt{mC_0}} + \frac{\bar{\varepsilon}}{2} \\
&= \bar{\varepsilon}.
\end{aligned}$$

Therefore, the DNN $u_\theta = c \cdot \hat{\varphi}$ with $\Theta = \{(c, \eta) : |c_j|^2 < C_0,\ \eta \in H\}$ and $\alpha_\theta = (\alpha^{f_\theta}, 0)$ satisfies Assumption 2.

Before proving the main proposition of this section, we will need the following lemma.

LEMMA 3.3. *Suppose Assumptions* 1 *and* 2 *are satisfied. For all* $\varepsilon > \bar{\varepsilon}$, *there exists* $v : \bar{\Theta} \to \mathbb{R}^m$ *such that* v *is bounded over* $\bar{\Theta}$ *and the value of* v *at* θ, *denoted by* v_θ, *satisfies*

$$\|v_\theta \cdot \nabla_\theta u_\theta - F[u_\theta]\|_2 \le \varepsilon \qquad \forall \theta \in \bar{\Theta}.$$

Proof. Let $\varepsilon > \bar{\varepsilon}$, and let $\delta \in (0, \varepsilon - \bar{\varepsilon})$. By Assumption 2, for all $\theta \in \Theta$ there exists an $\alpha_\theta \in \mathbb{R}^m$ coefficient such that

$$\|\alpha_\theta \nabla_\theta u_\theta - F[u_\theta]\|_2 \le \bar{\varepsilon}.$$

As $F[u_\theta]$ and $\nabla_\theta u_\theta$ are continuous in θ and Ω is bounded, we associate to each θ and each coefficient α_θ the open set U_θ containing θ, small enough, such that for all $\theta' \in U_\theta$ we have

$$\|\alpha_\theta \nabla_\theta u_{\theta'} - \alpha_\theta \nabla_\theta u_\theta\|_2 + \|F[u_\theta] - F[u_{\theta'}]\|_2 \le \delta \tag{3.14}$$

and hence

$$\begin{aligned}\|\alpha_\theta \cdot \nabla_\theta u_{\theta'} - F[u_{\theta'}]\|_2 \le \|\alpha_\theta \nabla_\theta u_{\theta'} - \alpha_\theta \nabla_\theta u_\theta\|_2 + \|\alpha_\theta \nabla_\theta u_\theta - F[u_\theta]\|_2 \\ + \|F[u_\theta] - F[u_{\theta'}]\|_2 \le \delta + \bar{\varepsilon}.\end{aligned} \tag{3.15}$$

Therefore, $\cup_{\theta\in\bar{\Theta}} U_\theta$ is an open cover of $\bar{\Theta}$. As $\bar{\Theta}$ is compact, this open cover has a finite subcover $\cup_{i=1}^N U_{\theta_i}$ for particular θ_i's. Define $v : \bar{\Theta} \to \mathbb{R}^m$ such that $v_\theta := v(\theta) = \alpha_{\theta_i}$ if $\theta \in U_{\theta_i}$ (if θ is in the intersection of multiple U_{θ_i}'s, we choose a single α_{θ_i} arbitrarily). We see from this construction that v_θ is uniformly bounded over $\bar{\Theta}$ as the range of v_θ is finite. From (3.15) we have

$$\|v_\theta \cdot \nabla_\theta u_\theta - F[u_\theta]\|_2 \le \delta + \bar{\varepsilon} \le \varepsilon. \qquad \square$$

With Assumptions 1 and 2 and Lemma 3.3 we can prove the existence of an accurate neural control field V_ξ parameterized as a neural network, as shown in the next proposition.

PROPOSITION 3.4. *Suppose Assumptions* 1 *and* 2 *hold. Then, for any $\varepsilon > 0$, there exists a differentiable vector field parameterized as a neural network $V_\xi : \bar{\Theta} \to \mathbb{R}^m$ with parameters ξ, such that*

$$\|V_\xi(\theta) \cdot \nabla_\theta u_\theta - F[u_\theta]\|_2 \le \varepsilon$$

for all $\theta \in \bar{\Theta}$.

Proof. We first show that there exists a differentiable vector-valued function $V : \bar{\Theta} \to \mathbb{R}^d$ such that

$$\|V(\theta) \cdot \nabla_\theta u_\theta - F[u_\theta]\|_2 \le \frac{\varepsilon}{2} \tag{3.16}$$

for all $\theta \in \bar{\Theta}$. To this end, we choose $\bar{\varepsilon}_0 \in (0, \varepsilon/2)$ and $\bar{\varepsilon} \in (\bar{\varepsilon}_0, \varepsilon/2)$; then by Assumption 2 and Lemma 3.3 we know that there exist a reduced-order model u_θ, a bounded open set $\Theta \subset \mathbb{R}^m$, and $M_v > 0$ such that there exists a vector-valued function $\theta \mapsto v_\theta$, where for any $\theta \in \bar{\Theta}$ we have $|v_\theta| < M_v$ and

$$\|v_\theta \cdot \nabla_\theta u_\theta - F[u_\theta]\|_2 \le \bar{\varepsilon}.$$

Note that v_θ is not necessarily differentiable with respect to θ. To obtain a differentiable vector field $V(\theta)$, for each $\theta \in \bar{\Theta}$ we define the function ψ_θ by

$$\psi_\theta(w) := \|w \cdot \nabla_\theta u_\theta - F[u_\theta]\|_2^2 = w^\top G(\theta) w - 2w^\top p(\theta) + q(\theta),$$

where

$$\begin{aligned} G(\theta) &:= \int_\Omega \nabla_\theta u_\theta(x) \nabla_\theta u_\theta(x)^\top \, dx, \quad p(\theta) := \int_\Omega \nabla_\theta u_\theta(x) F[u_\theta](x) \, dx, \\ q(\theta) &:= \int_\Omega F[u_\theta](x)^2 \, dx. \end{aligned} \tag{3.17}$$

Then we know that

$$\psi_\theta^* := \psi_\theta(v_\theta) = \|v_\theta \cdot \nabla_\theta u_\theta - F[u_\theta]\|_2^2 \le \bar{\varepsilon}^2. \tag{3.18}$$

It is also clear that $G(\theta)$ is symmetric and positive semidefinite. Moreover, due to the compactness of $\bar{\Omega}$ and $\bar{\Theta}$, as well as the fact that $\nabla_\theta u \in C(\bar{\Omega} \times \bar{\Theta})$, we know there exists $\lambda_G > 0$ such that

$$\|G(\theta)\|_2 \le \lambda_G$$

for all $\theta \in \bar{\Theta}$. Therefore, ψ_θ is a convex function and the Lipschitz constant of $\nabla\psi_\theta$ is uniformly upper bounded by λ_G over $\bar{\Theta}$. Now, for any $w \in \mathbb{R}^m$, $h > 0$, and $K \in \mathbb{N}$ (we reuse the letter K as the iteration counter instead of the number of sampling points in this proof), we define

$$\mathcal{O}_\theta^{K,h}(w) := w_K, \quad \text{where} \quad w_k = w_{k-1} - h\nabla\psi_\theta(w_{k-1}), \quad w_0 = w, \quad k = 1, \dots, K.$$

Namely, $\mathcal{O}_\theta^{K,h}$ is the oracle of executing the gradient descent optimization scheme on ψ_θ with step size $h > 0$ for K iterations.

Next, we slightly modify the standard convergence result of gradient descent in convex optimization [64, Theorem 2.1.14] and obtain Lemma A.1 in Appendix A. Notice that ψ_θ is convex and differentiable and that $\nabla\psi_\theta$ is Lipschitz continuous with its Lipschitz constant upper bounded by λ_G. Therefore, applying Lemma A.1 with $y = v_\theta$, $f = \psi_\theta$, and the gradient descent scheme for K iterations (K to be determined soon) with initial 0 and any fixed step size $h \in (0, 1/\lambda_G)$ to ψ_θ directly yields an error bound for $\psi_\theta(\mathcal{O}_\theta^{K,h}(0))$:

$$\psi_\theta(\mathcal{O}_\theta^{K,h}(0)) - \psi_\theta^* \le \frac{|0 - v_\theta|^2}{2Kh}. \tag{3.19}$$

Combining this with the bound $|v_\theta| < M_v$, we choose any

$$K \ge \frac{M_v^2}{2h((\varepsilon/2)^2 - \bar{\varepsilon}^2)},$$

and there exists

$$\psi_\theta(\mathcal{O}_\theta^{K,h}(0)) - \psi_\theta^* \le \frac{M_v^2}{2Kh} \le \left(\frac{\varepsilon}{2}\right)^2 - \bar{\varepsilon}^2. \tag{3.20}$$

Notice that $\mathcal{O}_\theta^{K,h}$ is a differentiable vector-valued function of θ because K and h are fixed. Therefore, combining (3.18) and (3.20) yields

$$0 \le \psi_\theta(\mathcal{O}_\theta^{K,h}(0)) = (\psi_\theta(\mathcal{O}_\theta^{K,h}(0)) - \psi_\theta^*) + \psi_\theta^* \le (\varepsilon/2)^2 - \bar{\varepsilon}^2 + \bar{\varepsilon}^2 = (\varepsilon/2)^2.$$

As this inequality holds for all $\theta \in \bar{\Theta}$, we set $V(\theta) = \mathcal{O}_\theta^{K,h}(0)$, which is a differentiable function of θ satisfying (3.16).

By the universal approximation theorem of neural networks [28] (see also Remark 3.1), we know there exists a differentiable vector-valued function parameterized as a neural network V_ξ with parameters ξ such that

$$|V_\xi(\theta) - V(\theta)|_\infty \le \varepsilon/(2B)$$

for all $\theta \in \bar{\Theta}$, where $B := \max_{\theta\in\bar{\Theta}} \|\nabla_\theta u_\theta\|_2 < \infty$ and $|\cdot|_\infty$ stands for the ∞ norm of vectors. Hence, we know that

$$\begin{aligned}\|V_\xi(\theta)\cdot\nabla_\theta u_\theta - F[u_\theta]\|_2 &\le \|V_\xi(\theta)\cdot\nabla_\theta u_\theta - V(\theta)\cdot\nabla_\theta u_\theta\|_2 + \|V(\theta)\cdot\nabla_\theta u_\theta - F[u_\theta]\|_2 \\ &\le B\cdot\frac{\varepsilon}{2B} + \frac{\varepsilon}{2} = \varepsilon.\end{aligned}$$

This completes the proof. ∎

Remark 3.5. It is important to note that the geometry of $\mathcal{M}$, especially its dimensionality, is complex and highly dependent on the structure of u_θ and the parameter space Θ. In particular, we can show that the tangent space $T_{u_\theta}\mathcal{M} = \mathrm{span}(\nabla_\theta u_\theta)$ at any $u_\theta \in \mathcal{M}$ is in the L^2 space, where $\nabla_\theta u_\theta = (\partial_{\theta_1} u_\theta, \dots, \partial_{\theta_m} u_\theta)$ for $\theta = (\theta_1, \dots, \theta_m)$. (Here we use discrete indices $1, \dots, m$ as subscripts of θ to indicate its components for notation simplicity. This is to be distinguished from the subscript t in θ_t which stands for time of the trajectory θ_t.) However, $\dim(T_{u_\theta}\mathcal{M})$ may vary across different u_θ on $\mathcal{M}$. For example, consider the reduced-order model u_θ parameterized as a DNN as in (3.2): when $w = 0$, we have $\theta = (0, b, \dots)$, and hence $\partial_{W_l} u_\theta = 0$ and $\partial_{b_l} u_\theta = 0$ for all $l = 1, \dots, L$. In this case, the m components of $\nabla_\theta u_\theta$ are *not* linearly independent, and $\dim(T_{u_\theta}\mathcal{M}) < m$ for such θ's. This distinguishes our parameter submanifold from existing ones, such as in [2], which assumes that the tangent space is always of full dimension m at any point of the submanifold. In our case, however, challenges and complications in dealing with the parameter submanifold $\mathcal{M}$ could be avoided if we made such an assumption, but it will lead to incorrect analysis and error estimation, which poses a major technical challenge for the proposed framework. Specifically, we note that the rank of $G(\theta)$ varies across Θ, and therefore the pseudoinverse $G(\theta)^+$ may be discontinuous. A major theoretical merit of Proposition 3.4 is that we can still ensure the existence of a differentiable control vector field in such a case.

3.3.2. Error analysis in solving (semi-)linear parabolic PDEs. Now we are ready to provide error bounds of our method in solving a large class of linear and semilinear parabolic PDEs. This class of PDEs covers many types of reaction-diffusion equations, such as heat equations, Fisher's equation, or the Allen–Cahn equation. The differential operator F in linear and semilinear parabolic PDEs has the form

$$F[u] = \nabla \cdot (A \nabla u) + b \cdot \nabla u + f(u),$$

where $A : \Omega \to \mathbb{R}^{d \times d}$ and $b : \Omega \to \mathbb{R}^d$ are continuous, and $f : \mathbb{R} \to \mathbb{R}$ is L_f-Lipschitz and acts on $u(x)$ for each x. Moreover, we assume that there exist $\lambda \geq 0$ and $B \geq 0$ such that

$$z^\top A(x) z \geq \lambda |z|^2 \quad \forall\, z \in \mathbb{R}^d,\ x \in \Omega, \tag{3.21}$$

and

$$\|\nabla \cdot b\|_\infty \leq B. \tag{3.22}$$

Furthermore, due to the smoothness of V_ξ and compactness of $\bar{\Theta}$, we know there exist $M_V > 0$ and $L_V > 0$ such that

$$\max_{\theta \in \bar{\Theta}} |V_\xi(\theta)| \leq M_V \qquad \text{and} \qquad \max_{\theta \in \bar{\Theta}} |\nabla_\theta V_\xi(\theta)| \leq L_V. \tag{3.23}$$

THEOREM 3.6. *Suppose Assumptions* 1 *and* 2 *hold. Then there exists control field* V_ξ *such that for any* u^* *satisfying the evolution PDE in* (3.5) *there exists*

$$\|u_{\theta_t}(\cdot) - u^*(\cdot, t)\|_2 \leq e^{(L_f + B/2 - \lambda/C_p)t}(\varepsilon_0 + \varepsilon t) \tag{3.24}$$

for all t *as long as* $\theta_t \in \bar{\Theta}$, *where* θ_t *is solved from the ODE* (3.9) *with* V_ξ *and initial* θ_0 *satisfying* $\|u_{\theta_0}(\cdot) - u^*(\cdot, 0)\|_2 \leq \varepsilon_0$. *Here* C_p *is a constant depending only on* Ω.

Proof. We denote the residual

$$r(x,t) := \nabla_\theta u_{\theta_t}(x) \cdot V_\xi(\theta_t) - F[u_{\theta_t}](x).$$

Then by Proposition 3.4 we have $\|r(\cdot,t)\|_2 \le \varepsilon$ for all t. Furthermore, we denote

$$\delta(x,t) := u_{\theta_t}(x) - u^*(x,t)$$

for all $(x,t) \in \bar{\Omega} \times [0,T]$ and $D(t) := \|\delta(\cdot,t)\|_2$, and then there exists

$$D'(t) = \left\langle \frac{\delta(\cdot,t)}{\|\delta(\cdot,t)\|_2}, \partial_t \delta(\cdot,t) \right\rangle. \tag{3.25}$$

Here we use the convention that $\delta(\cdot,t)/\|\delta(\cdot,t)\|_2 = 0$ if $\delta(\cdot,t) = 0$ a.e. By the definition of δ, we have

$$\begin{aligned}
\partial_t \delta(x,t) &= \partial_t u_{\theta_t}(x) - \partial_t u^*(x,t) \\
&= \nabla_\theta u_{\theta_t}(x) \cdot \dot{\theta}_t - F[u^*](x,t) \\
&= \nabla_\theta u_{\theta_t}(x) \cdot V_\xi(\theta_t) - F[u^*](x,t) \\
&= F[u_{\theta_t}](x) - F[u^*](x,t) + r(x,t) \\
&= \nabla \cdot (A(x) \nabla \delta(x,t)) + b(x) \cdot \nabla \delta(x,t) + f(u_{\theta_t}(x)) - f(u^*(x,t)) + r(x,t).
\end{aligned}$$

Therefore, we have

$$\begin{aligned}
\langle \delta(\cdot,t), \partial_t \delta(\cdot,t) \rangle &= \int_\Omega \delta(x,t) \left(\nabla \cdot (A(x) \nabla \delta(x,t)) + b(x) \cdot \nabla \delta(x,t) \right) dx \\
&\quad + \int_\Omega \delta(x,t) (f(u_{\theta_t}(x)) - f(u^*(x,t)) + r(x,t))\, dx \\
&=: I(t) + J(t).
\end{aligned} \tag{3.26}$$

Because $u_{\theta_t}(\cdot)|_{\partial\Omega} = u^*(\cdot,t)|_{\partial\Omega} = 0$, we know that $\delta(\cdot,t)|_{\partial\Omega} = 0$. Thus, we have

$$\begin{aligned}
I(t) &= \int_\Omega \delta(x,t) \left(\nabla \cdot (A(x) \nabla \delta(x,t)) + b(x) \cdot \nabla \delta(x,t) \right) dx \\
&= -\int_\Omega \nabla \delta(x,t)^\top A(x) \nabla \delta(x,t)\, dx - \frac{1}{2} \int_\Omega (\nabla \cdot b(x)) \delta(x,t)^2\, dx \\
&\le -\lambda \int_\Omega |\nabla \delta(x,t)|^2\, dx - \frac{1}{2} \int_\Omega (\nabla \cdot b(x)) \delta(x,t)^2\, dx \\
&\le -\frac{\lambda}{C_p} \int_\Omega |\delta(x,t)|^2\, dx + \frac{B}{2} \int_\Omega |\delta(x,t)|^2\, dx,
\end{aligned} \tag{3.27}$$

where the first equality is just by the definition of $I(t)$, the second equality is obtained by integrating by parts on both terms and using $\delta(\cdot,t)|_{\partial\Omega} = 0$, the first inequality is due to (3.21), and the last inequality is due to Poincare's inequality

$$\|\delta(\cdot,t)\|_2 \le C_p \|\nabla \delta(\cdot,t)\|_2$$

as $\delta(\cdot,t)|_{\partial\Omega}=0$ for all t (here C_p is Poincare's constant depending on Ω only) and the bound (3.22). We can also obtain

$$\begin{aligned} J(t) &= \int_\Omega \delta(x,t)(f(u_{\theta_t}(x)) - f(u^*(x,t)) - r(x,t))\,dx \\ &\le \int_\Omega |\delta(x,t)| \cdot |f(u_{\theta_t}(x)) - f(u^*(x,t)) - r(x,t)|\,dx \\ &\le \int_\Omega |\delta(x,t)| \cdot (L_f|\delta(x,t)| + |r(x,t)|)\,dx \\ &\le L_f\|\delta(x,t)\|_2^2 + \|r(\cdot,t)\|_2\|\delta(\cdot,t)\|_2 \\ &\le L_f\|\delta(x,t)\|_2^2 + \varepsilon\|\delta(\cdot,t)\|_2, \end{aligned} \tag{3.28}$$

where the first identity is by the definition of $J(t)$, and the second inequality is due to the Lipschitz condition of f. Combining (3.25), (3.26), (3.27), and (3.28), we obtain

$$D'(t) \le \left(L_f + \frac{B}{2} - \frac{\lambda}{C_p}\right) D(t) + \varepsilon.$$

By Grönwall's inequality, we deduce that

$$D(t) \le e^{(L_f + B/2 - \lambda/C_p)t}(D(0) + \varepsilon t).$$

Recalling that

$$D(0) = \|\delta(\cdot,0)\|_2 = \|u_{\theta_0}(\cdot) - u^*(\cdot,0)\|_2 = \|u_{\theta_0}(\cdot) - g(\cdot)\|_2 \le \varepsilon_0,$$

we thus have

$$\|u(\cdot,\theta(t)) - u(\cdot,t)\|_2 = D(t) \le e^{(L_f + B/2 - \lambda/C_p)t}(\varepsilon_0 + \varepsilon t)$$

for all time t, which completes the proof. ∎

The error estimate in Theorem 3.6 indicates that the approximation error is determined by three factors: (i) the approximation error ε_0 of the reduced-order model to the initial value g, (ii) the local approximation error ε of the projection of $F[u_\theta]$ onto the tangent space of $\mathcal{M}$ at u_θ; and (iii) the irregularity of the differential operator F itself. While the error from (iii) is determined by the given PDE, we can make an effort to suppress (i) and (ii) in practice by robust architecture of u_θ and the training of V_ξ. We note the error estimate provided in Theorem 3.6 is an upper bound of the approximation error.

Remark 3.7. While we assumed f to be globally Lipschitz, the result in Theorem 3.6 still holds locally with the local Lipschitz condition of f. For example, in the case of the Allen–Cahn example, we know that if our initial function is bounded by 1, the true trajectories will remain bounded, allowing the results of Theorem 3.6 to apply.

COROLLARY 3.8. *Suppose the conditions in Theorem* 3.6 *hold. Let* $\hat{\theta}_t$ *be the numerical solution to the ODE* (3.9) *obtained by using the Euler scheme with step size* $h>0$. *Then*

$$\|u_{\hat{\theta}_t}(\cdot) - u^*(\cdot,t)\|_2 \le \frac{L_V M_V |\Omega| h}{2}(e^{L_V t} - 1) + e^{(L_f - B/2 + \eta/C_p)t}(\varepsilon_0 + \bar{\varepsilon} t) \tag{3.29}$$

for all t *as long as* $\theta_t \in \bar{\Theta}$.

Proof. Given the estimate provided in Theorem 3.6, we only need to show that

$$\|u_{\hat{\theta}_t}(\cdot) - u_{\theta_t}(\cdot)\|_2 \leq \frac{L_V M_V |\Omega| h}{2}(e^{L_V t} - 1), \tag{3.30}$$

since combined with (3.24) it yields the claimed estimate (3.29). To show (3.30), we notice that

$$\ddot{\theta}_t = \frac{d}{dt} V_\xi(\theta_t) = \nabla_\theta V_\xi(\theta_t) \cdot \dot{\theta}_t = \nabla_\theta V_\xi(\theta_t) \cdot V_\xi(\theta_t).$$

Therefore, we have

$$|\ddot{\theta}_t| = |\nabla_\theta V_\xi(\theta_t) \cdot V_\xi(\theta_t)| \leq L_V M_V,$$

where L_V and M_V are defined in (3.23). Hence, by the standard results for Euler's method [4, p. 346]), we know the numerical solution $\hat{\theta}_t$ satisfies

$$|\hat{\theta}_t - \theta_t| \leq \frac{h M_V}{2}\left(e^{L_V t} - 1\right) \tag{3.31}$$

for all t. Therefore, we obtain

$$\begin{aligned}\|u_{\hat{\theta}_t} - u_{\theta_t}\|_2 &= \left(\int_\Omega |u_{\hat{\theta}_t}(x) - u_{\theta_t}(x)|^2 \, dx\right)^{1/2} = \left(\int_\Omega |\nabla_\theta u_{\tilde{\theta}_t}(x) \cdot (\hat{\theta}_t - \theta_t)|^2 \, dx\right)^{1/2} \\ &\leq L_V |\Omega| |\hat{\theta}_t - \theta_t| \leq \frac{L_V M_V |\Omega| h}{2}(e^{L_V t} - 1),\end{aligned}$$

where the second equality is due to the fact that u_θ is C^1 in θ and hence the mean value theorem applies to u_θ (here $\tilde{\theta}_t$ is some point on the line segment between $\hat{\theta}_t$ and θ_t). ∎

The proof above can be modified if a different numerical ODE solver is employed. In that case, one can obtain an improved upper bound and order in step size h in (3.31).

4. Numerical results.

4.1. Implementation of the training process of control field V_ξ. In section 3.2, we have shown that the neural control field V_ξ is parameterized as a deep network, and its parameters ξ can be learned by solving

$$\min_\xi \left\{ \ell(\xi) := \int_\Theta \|V_\xi(\theta) \cdot \nabla_\theta u_\theta - F[u_\theta]\|_2^2 \, d\theta \right\}.$$

The first-order optimality condition of this minimization problem is given by $G(\theta)V_\xi(\theta) = p(\theta)$, where $G(\theta)$ and $p(\theta)$ are defined in (3.17). The objective function $\ell(\xi)$ above shares the same minimizers as the following one:

$$\bar{\ell}(\xi) := \int_\Theta |G(\theta)V_\xi(\theta) - p(\theta)|^2 \, d\theta. \tag{4.1}$$

In our numerical experiments, we use $\bar{\ell}$ defined in (4.1), as we can train towards the optimal solution $V_\xi = G^+(\theta)p(\theta)$ as the optimal value, which seems to produce lower error empirically. Moreover, we know the minimum loss value of (4.1) is 0, which is in contrast to (3.10), where the minimum loss value is often unknown.

In practice, as the dimension of θ and Ω could be large, we have to approximate (4.1) using techniques such as Monte Carlo integration. This leads to the approximate forms

$$\tilde{G}(\theta) = \frac{1}{N_x}\sum_{i=1}^{N_x} \nabla_\theta u_\theta(x_i)\nabla_\theta u_\theta(x_i)^\top, \quad \tilde{p}(\theta) = \frac{1}{N_x}\sum_{i=1}^{N_x} \nabla_\theta u_\theta(x_i) F[u_\theta](x_i),$$

where x_i, $i = 1, \ldots, N_x$, are sampled from Ω. By also drawing samples from Θ, we arrive at our empirical loss function defined by

$$\ell_1(\xi) := \frac{1}{N_\theta}\sum_{j=1}^{N_\theta} |\tilde{G}(\theta_j)\cdot V_\xi(\theta_j) - \tilde{p}(\theta_j)|^2. \tag{4.2}$$

To improve the training of V_ξ, we also augment the loss function ℓ_1 in (4.2) with an additional term following a data-driven approach. Specifically, we follow the methods in [10, 19] to generate multiple sample trajectories starting from randomly sampled initial values $\{\theta_0^{(i)} : i \in [M]\}$ in Θ. For the ith trajectory, a sequence of directions $\{v_j^{(i)} : j = 0, 1 \ldots, N_t\}$ is solved from linear systems $\tilde{G}(\theta_j^{(i)})v_j^{(i)} = \tilde{p}(\theta_j^{(i)})$ and the discrete-time points on the trajectory are obtained by $\theta_{j+1}^{(i)} = \theta_j^{(i)} + hv_j^{(i)}$ for $j = 0, 1, \ldots, N_t - 1$. We add the augment loss term

$$\ell_2(\xi) := \frac{1}{N_t M}\sum_{i=1}^{M}\sum_{j=1}^{N_t} |V_\xi(\theta_j^{(i)}) - v_j^{(i)}|^2. \tag{4.3}$$

Combining with (4.2), we obtain our final loss function

$$\ell_{\text{total}}(\xi) = \ell_1(\xi) + \zeta\ell_2(\xi), \tag{4.4}$$

where ζ is a weight parameter. In our experience, parabolic linear PDEs using only ℓ_1 is sufficient to generate a good result. For the nonlinear case, adding ℓ_2 substantially improves training results empirically, as network parameters may move far away from those we sampled near the initial parameters.

4.2. Experimental setting. To demonstrate the performance of the proposed method, we test it on three different PDEs: a 10-dimensional (10D) transport equation, a 10D heat equation, and a 2D Allen–Cahn equation. Both the transport equation and the heat equations are linear PDEs, while the Allen–Cahn one is a highly nonlinear PDE. In fact, we also tested a 10D Allen–Cahn equation but only present the result of the 2D one here. This is because the true solution of the Allen–Cahn equation does not have closed form, and we have to employ a classical finite difference method, which does not scale to the 10D case, to produce a reference solution for comparison. In contrast, we have closed-form solutions of the IVPs with transport and heat equations, and hence we can use them as the true solution for direct comparison. In our tests, we employ the following structure of our reduced-order model:

$$u_\theta(x) = \alpha(x) z_L(x, \theta) \tag{4.5}$$

for the heat equation and Allen–Cahn equation. We use the following network structure:

$$u_\theta(x) = z_L(\beta(x), \theta) \tag{4.6}$$

TABLE 1
Problem settings, network structures, and the number of training trajectories/samples in numerical experiments. Here M is the number of trajectories used from Θ and N_θ is the total number of samples from Θ.

Problem	Dim. d	u_θ width/depth	V_ξ width/depth	M	N_θ
Transport Equation	10	12/4	1,500/4	0	160,000
Heat Equation	10	12/5	2,000/10	600	200,000
Allen–Cahn Equation	2	10/3	2,000/5	200	200,000

for the transport equation. In (4.5), $\alpha(x)$ is a distance function of $\partial\Omega$ such that it satisfies the zero boundary condition, and in (4.6), $\beta(x)$ is a function chosen to satisfy a periodic boundary condition as in [19]. This aligns with our choice of u_θ in (4.5) and (4.6), as the IVP with heat and Allen–Cahn equations has zero boundary value, whereas the IVP with a transport equation has a periodic boundary value in our experiments. In both (4.5) and (4.6), z_L is the neural network and is defined by

$$z_L = w_L z_{L-1}, \quad z_l = z_{l-1} + \sigma(W_l z_{l-1} + b_l), \quad l = 1, \dots, L-1, \tag{4.7}$$

and $z_0 = \sigma(W_0 x + b_0)$. Here σ is a user-chosen activation function (we use tanh or ReLU in our experiments), where $W_l \in \mathbb{R}^{d' \times d'}$ are the weight matrices and $b_l \in \mathbb{R}^{d'}$ are the bias vectors, and $W_0 \in \mathbb{R}^{d' \times d}$ and $w_L \in \mathbb{R}^{1 \times d}$; all of these matrices and vectors make up the parameter vector θ. Networks such as in (4.7) are often called *residual neural networks* (ResNet) and have been shown to perform better than FFNs in function approximation [80]. The values of L and d' in our experiments are shown in Table 1. They are selected manually to balance the depth L and width d' so that u_θ does not have too many neurons but still remains expressive. We use a similar structure for the vector field V_ξ but adjust the layers to be $\eta_l = \eta_{l-1} + \mathrm{GeLU}(\bar{U}_l\theta + \bar{b}_l)\tanh(U_l\eta_{l-1} + b_l)$. Here $\mathrm{GeLU}(x) = x\Phi(x)$, where $\Phi(x)$ is the standard Gaussian cdf. This is a slight modification of the network architecture proposed in [79] for improved effectiveness in training by gradient descent. We selected this network structure by starting with a ResNet with small width and depth and ReLU activation, and then we gradually increased the width and depth until the improvement in the final loss value became insignificant. Finally, we attempted a few different activation functions and network architectures for this width and depth and selected the aforementioned structure, which appeared to perform slightly better. This process was by no means exhaustive.

Other network architectures can be used as well. The width and depth of our network are reported in Table 1. Information about the number of trajectories used for (4.3) is also collected in Table 1. For all of the experiments, we set the weight $\zeta = 0.1$ in (4.4) to reflect the scale difference of the two loss terms and use the standard ADAM optimizer with learning rate 0.001, $\beta_1 = 0.9$, $\beta_2 = 0.999$. We terminate the training process when the empirical loss $\ell_{\text{total}}(\xi) < 0.1$ or when the percent decrease of the empirical loss is less than 0.1% averaged over the past 100 steps. Once V_ξ is learned, we use the 4th-order Runge–Kutta method with a step size of $T/200$ (T is determined from the problem) to solve θ_t from the ODE in (3.9) in Algorithm 3.2 and compare the corresponding u_θ with the reference solutions. All the implementations and experiments are performed using PyTorch in Python 3.9 in Windows 10 OS on a desktop computer with an AMD Ryzen 7 3800X 8-Core Processor at 3.90 GHz, 16 GB of system memory, and an Nvidia GeForce RTX 2080 Super GPU with 8 GB of graphics memory. The total computational time is split between three unique

activities: (i) the generation of N_θ samples for ℓ_1 in (4.2); (ii) the generation of the M trajectories for ℓ_2 in (4.3); and (iii) the training of the network V_ξ. Parts (i) and (ii) can be parallelized offline to speed up the process. We discuss the specific time cost of the implementation of our method in the examples below.

We also provide a few remarks on the sampling strategy in Θ. While one can draw θ uniformly from Θ, adding samples θ corresponding to some example solutions to the PDE may further improve training efficiency. In practice, we use both uniformly sampled θ's and those close to the θ's corresponding to some randomly chosen initial functions. These initial functions are only used to help the loss function weigh more on the regions that are potentially more important than others in Θ; but they are not among the randomly chosen initial functions used for any testing. Details on samplings are given below.

4.3. Numerical results on transport equation. We first consider the IVP defined by a 10D transport equation with periodic boundary conditions as follows:

$$\begin{cases} \partial_t u(x,t) = -\mathbf{1} \cdot \nabla_x u(x,t) & \forall\, x \in \Omega,\ t \in [0,T], \\ u(x,0) = g(x) & \forall\, x \in \bar{\Omega}, \end{cases} \tag{4.8}$$

where $\Omega = (0,1)^{10}$, $T = 1$, $\mathbf{1}$ is the vector whose components are all ones, and the boundary value $u(x,t) = 0$ for all $x \in \partial\Omega$ and $t \in [0,T]$. This IVP has the true solution $u^*(x,t) = g(x - \mathbf{1} \cdot t)$. We obtain the solution operator of the IVP (4.8), and we use (4.6) as the reduced-order model u_θ. Although our error analysis requires certain regularity on the initial and solution of PDEs, we test on the case where both are only Lipschitz continuous but not differentiable for this transport equation. To this end, we set the activation of u_θ to ReLU. Further, define $\beta(x) = (\cos(2\pi(x-b)), \sin(2\pi(x-b)))^\top$, where $b \in \mathbb{R}^{10}$ is a trainable parameter with sin and cos acting componentwise to x. This means that the first hidden layer uses $W_0 \in \mathbb{R}^{12\times 20}$. For this example, we shall set $\Theta = [-1,1]^m$, where m are the number of parameters in u_θ. Then we train the neural control vector field V_ξ by minimizing (3.10) with the number of sampled θ drawn uniformly from Θ shown in Table 1. We note that this equation performed equally well with or without the loss ℓ_2 in (4.3). As such, we need not generate any trajectories and this is reflected in Table 1.

After the control V_ξ is obtained, we test the performance of V_ξ on a variety of initial values g by uniformly sampling $\theta_0 \in \Theta$. We emphasize that the θ_0's corresponding to these initial values are not used in the training process. We show three approximate solutions for three random initials in Figure 2. For the first random initial g_1 determined by the random θ_0, we plot the corresponding true solution $u^*(\cdot,t)$, the approximate solution $u_{\theta_t}(\cdot)$ obtained by Algorithm 3.2, and their pointwise absolute difference $|u_{\theta_t}(x) - u^*(x,t)|$ from row 1 to row 3 in Figure 2, respectively, for $t = 0, 0.15, 0.5, 0.85, 1$. The plots for the second and third g_2 and g_3 random initials are shown in rows 4–6 and 7–9 in Figure 2, respectively. From Figure 2, we can see that the reduced-order model u_{θ_t} with θ_t controlled by the trained vector field V_ξ closely approximates the true solution $u^*(\cdot,t)$ with low absolute errors (note that the scale of the error is different from that of $u^*(\cdot,t)$ and $u_{\theta_t}(\cdot)$). Figures 3(a) and 3(b) plot the mean of the absolute error $\|u^*(\cdot,t) - u_{\theta_t}(\cdot)\|_2^2$ and the relative error $\|u^*(\cdot,t) - u_{\theta_t}(\cdot)\|_2^2 / \|u^*(\cdot,t)\|_2^2$, respectively, over 100 randomly chosen initials, while the standard deviation is shaded in. We see mean errors $< 1\%$, even though the initial functions considered are not smooth. This suggests that the proposed model can generalize to the case where the initial and solution of the PDEs are not sufficiently smooth.

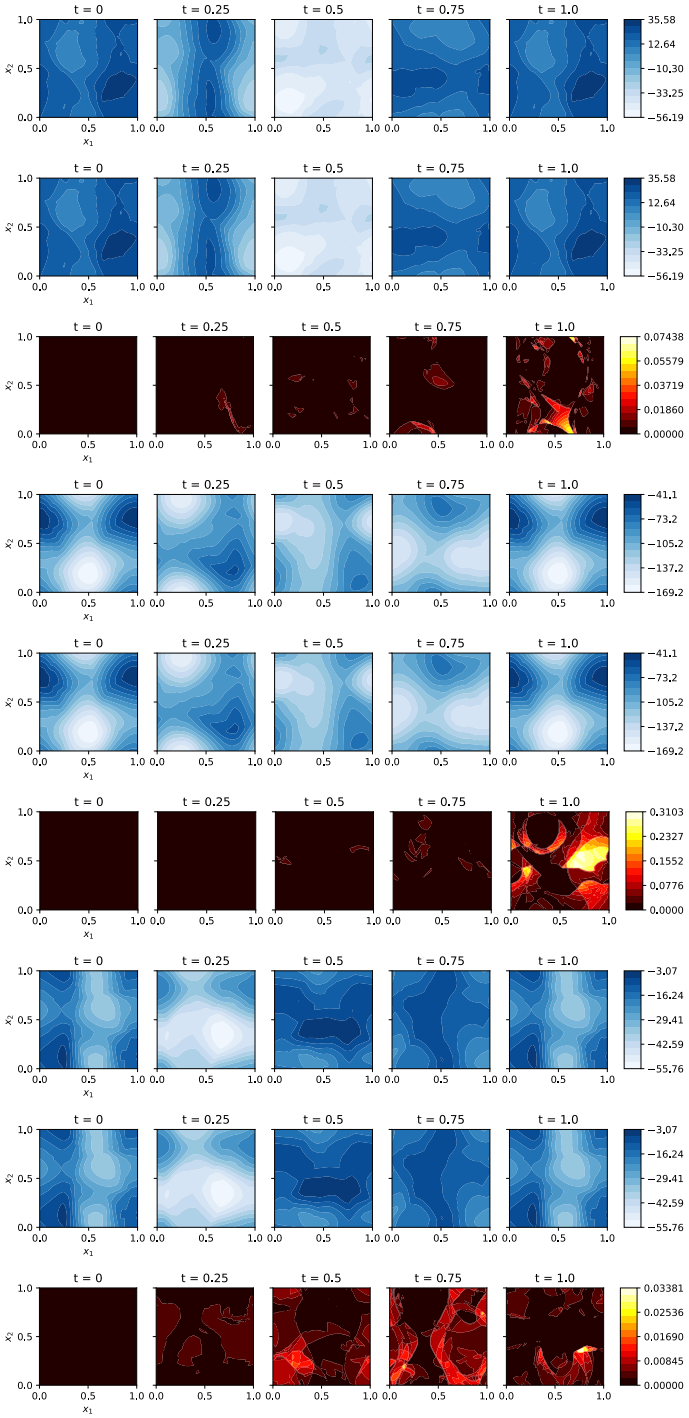

FIG. 2. *(Transport equation). Comparison between the true solution* $u^*(\cdot, t)$*, the approximation* $u_{\theta_t}(\cdot)$*, and their pointwise absolute difference* $|u_{\theta_t}(x) - u^*(x,t)|$ *for times* $t = 0, 0.15, 0.5, 0.85, 1$ *for IVPs with the first initial (rows 1–3), second initial (rows 4–6), and third initial (rows 7–9) given by* u_θ *with* θ *randomly drawn from* $[-1,1]^m$*.*

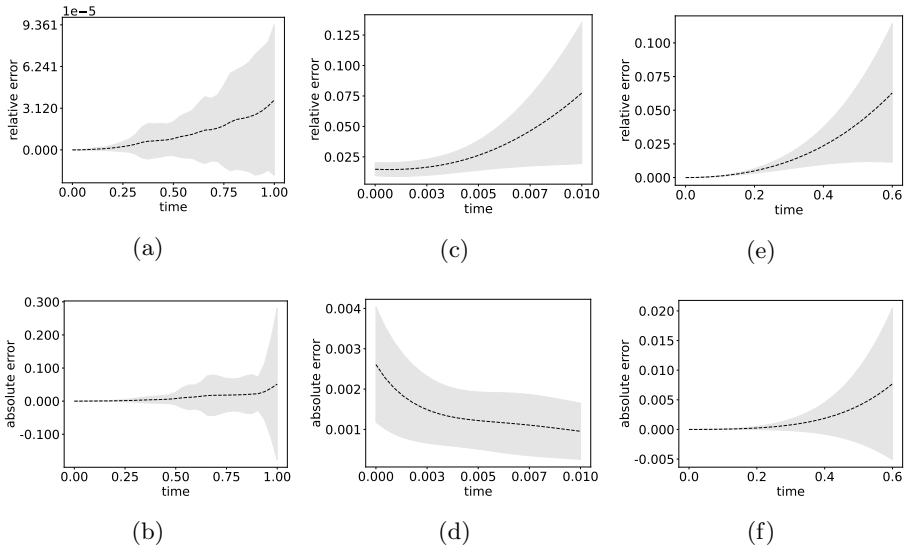

FIG. 3. *Comparison of the mean relative error* $\|u^*(\cdot,t) - u_{\theta_t}(\cdot)\|_2^2/\|u^*(\cdot,t)\|_2^2$ *(top) and the mean absolute* $\|u^*(\cdot,t) - u_{\theta_t}(\cdot)\|_2^2$ *(bottom) versus time* t *for* 100 *different initial conditions of the transport* (a)–(b)*, heat* (c)–(d)*, and Allen–Cahn* (e)–(f) *equations. Shaded areas indicate the standard deviation over the* 100 *results.*

We now discuss the computational cost of the method. In our tests, it took 1.78 hours to generate $\tilde{G}$ and $\tilde{p}$ from the samples in Θ used for training. Once generated, the training of V_ξ (i.e., minimizing the loss function ℓ_{total} in (4.4)) took 5 minutes to complete. Testing each initial condition by solving (3.9) using a 4th-order Runge–Kutta (RK4) solver with step size 0.005 took an average of 2.1 seconds per initial. We note that no time is needed in this case to fit an initial, as θ_0 is chosen randomly.

The proposed method has evident improvement on computational cost over existing methods that only solve specific instances of the PDEs. In this test, we compare the computational cost with PINN [73] and a time marching (TM) [19] method. We use the same structure of u_θ for PINN and time marching as used by our method. We sample 10,000 points $(x,t) \in (0,1)^{10} \times [0,1]$ for PINN and 10,000 points $x \in (0,1)^{10}$ for each step of the time marching method. We train PINN using its default parameters until convergence. For TM, we use RK4 with the same step size 0.005 and its default linear system solver for each step. We follow all other implementation steps of both PINN and TM as described in their original papers. For a single initial g, PINN, TM, and the proposed method took 116.5s, 16.7s, and 2.1s, respectively, to obtain the solution. This significant time reduction is due to the fact that the proposed method has learned the control field in the parameter space and thus can compute the solution of the PDE by solving an ODE which has very low computation complexity. The improvement is more significant for higher-order PDEs because PINN and TM require more time to compute the differential operator, whereas the computation complexity of the proposed method remains the same.

The proposed method is capable of approximating solution operators of high-dimensional PDEs, whereas existing methods cannot. This is because existing solution operator learning methods, such as DeepONet, require spatial discretization, and thus the network size and sampling amounts grow exponentially fast in problem dimensions. For example, for a one-dimensional ($d = 1$) evolution PDE, DeepONet

[57] requires 100 sample solutions (which must be generated by another numerical method), each evaluated at 10^4 grid points in the (x,t) domain in $\mathbb{R}\times\mathbb{R}_+$. Thus, the size of their trunk network alone is already 10 times larger than our V_ξ in the 10D case. When the problem dimension d becomes over 3, DeepONet will be infeasible computationally. In addition, our method does not require sample solutions which could be unavailable or difficult to obtain in practice.

4.4. Heat equation. Next, we consider an IVP with heat equation in 10 dimensions:

$$\begin{cases} \partial_t u(x,t) = \Delta u(x,t) & \forall\, x\in\Omega,\, t\in[0,T], \\ u(x,0) = g(x) & \forall\, x\in\bar{\Omega}, \end{cases} \tag{4.9}$$

where $\Omega=(0,1)^{10}$ and the boundary value $u(x,t)=0$ for all $x\in\partial\Omega$ and $t\in[0,T]$. As most of the initial conditions we consider have rapid evolution in a short time, we use $T=0.01$ in this test. For neural networks, we use (4.5), with $\alpha(x)=\Pi_{i=1}^{10}4(x_i-x_i^2)$ and tanh activation.

In order to have a class of analytical examples against which to compare, we use the base functions

$$\begin{aligned} g_1(x) &= \Pi_{i=1}^{10}\sin(\pi x_i), \\ g_2(x) &= \sin(2\pi x_1)\Pi_{i=1}^{10}\sin(\pi x_i), \\ g_3(x) &= \sin(2\pi x_2)\Pi_{i\neq 2}^{10}\sin(\pi x_i), \\ g_4(x) &= \sin(2\pi x_1)\sin(2\pi x_2)\Pi_{i=3}^{10}\sin(\pi x_i) \end{aligned} \tag{4.10}$$

to generate a class of initial conditions $\mathcal{G}:=\{\sum_{i=1}^4 c_i g_i \ :\ c_i\in[-1,1]\}$. To train our method, we drew 600 samples from $\mathcal{G}$ and found a corresponding $\theta_0^{(j)}$ for each sample. We set the parameter space to be $\Theta:=\{\theta_0^{(j)}+\delta \ :\ |\delta|\le 3,\ j=1,\dots,600\}$. We then uniformly sampled 200,000 points from this set Θ and generated paths for (4.2) from the 600 centers to train V_ξ. We then tested the method on a new set of 100 initials randomly drawn from $\mathcal{G}$ by following the method outlined in Algorithm 3.2. We randomly select three from the test set containing the 100 initials and plot the result using our method in Figure 4. In addition, Figures 3(c) and 3(d) show the mean and standard deviations of the relative and absolute errors versus time t. We notice that the relative error increases while absolute error decreases: this is because the true solution $u^*(t,\cdot)$ gradually vanishes in time, and hence it is easy to cause large relative error even when the absolute error is small.

In this test, it took 2.64 hours to generate $\tilde{G}$ and $\tilde{p}$ for (4.2) and 1.33 hours to generate the trajectories for (4.3). This time cost is significantly higher than the transport equation, as the heat equation requires the computation of the Laplacian which is second-order. Once the samples were generated, training V_ξ took approximately 10 minutes. For testing, it took an average of 25 seconds to train a θ_0 to a sampled g and an average of 2.6 seconds to then solve (3.9) using a 4th-order Runge–Kutta solver with step size 0.0001. This amounts to less than 30 seconds in time per initial for the testing stage.

4.5. Allen–Cahn equation. In this test, we consider the IVP with the nonlinear Allen–Cahn equation given by

$$\begin{cases} \partial_t u(x,t) = \epsilon\Delta u(x,t) + \frac{3}{2}\left(u(x,t)-u(x,t)^3\right) & \forall\, x\in\Omega,\, t\in(0,T], \\ u(x,0) = g(x) & \forall\, x\in\bar{\Omega}, \end{cases} \tag{4.11}$$

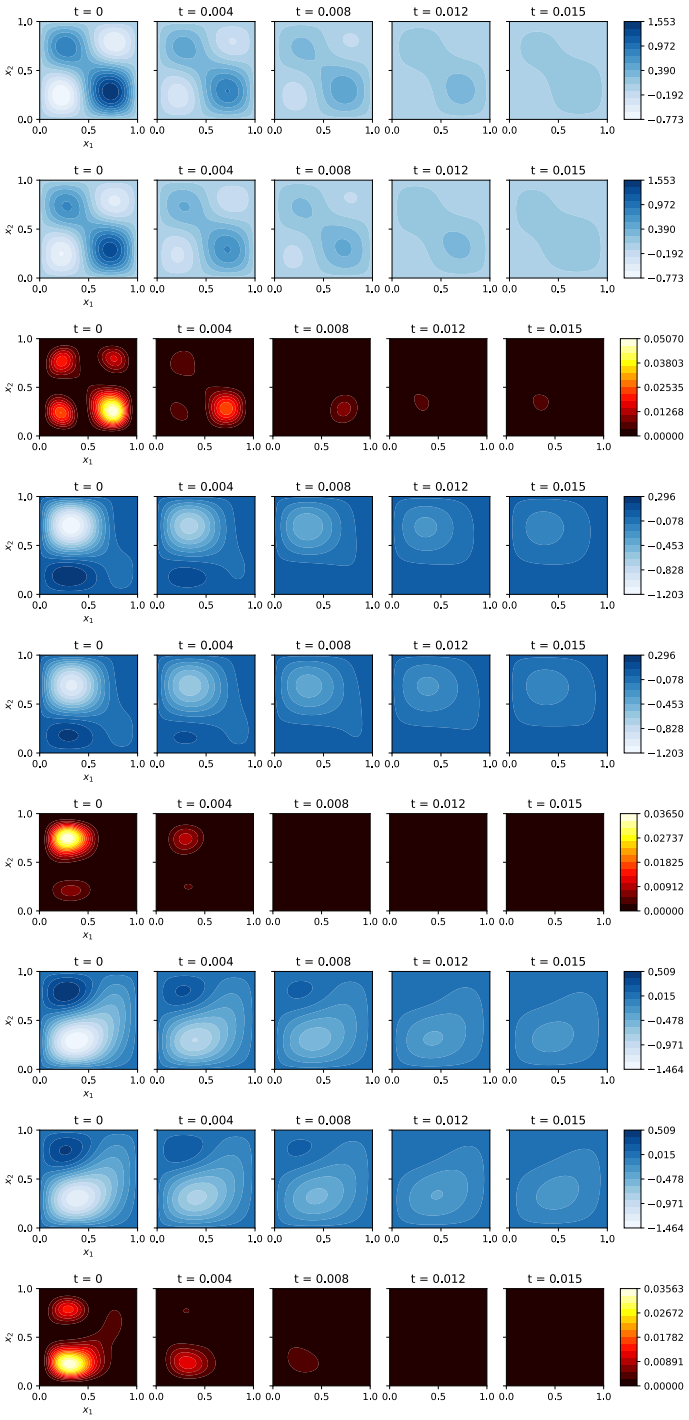

FIG. 4. *(Heat equation). Comparison between the true solution* $u^*(\cdot, t)$*, the approximation* $u_{\theta_t}(\cdot)$*, and their pointwise absolute difference* $|u_{\theta_t}(x) - u^*(x,t)|$ *for times* $t = 0, 0.004, 0.008, 0.012, 0.015$ *for IVPs with the first (rows 1–3), second (rows 4–6), and third initial (rows 7–9) drawn from the set* $\mathcal{G} := \{\sum_{i=1}^{4} c_i g_i \ : \ c_i \in [-1, 1]\}$*, where* g_i *is defined in* (4.10).

where $\Omega = (-1,1)^2$, $\epsilon = 0.0001$, and the boundary value $u(x,t) = 0$ for all $x \in \partial\Omega$ and $t \in [0,T]$. As the Allen–Cahn PDE does not have an analytical solution against which to compare, we resort to the classical implicit-explicit scheme (see, e.g., [81]) with a 100×100 grid and 2000 time points to generate a reference solution for comparison in the 2D case only, despite the fact that our method can be applied to a higher-dimensional case. In this test, we use (4.5) with $\alpha(x) = (1-x_1^2)(1-x_2^2)$ as our neural network. We let $T_i : \mathbb{R} \to \mathbb{R}$ represent the ith-order Chebyshev polynomial. We generate a class of initial conditions

(4.12)
$$\mathcal{G} := \left\{ (1-x_1^2)(1-x_2^2) \sum_{k=1}^{m} c_k T_{i_k}(x_1) T_{j_k}(x_2) \ : \ i_k, j_k \in \{0,\dots,6\},\ m \le 36,\ |c_k| \le 1 \right\}.$$

We see that $\mathcal{G}$ is a space of all combinations of Chebyshev polynomials up to degree 6 multiplied by a boundary function. This set is chosen to represent a diverse spread of initials that can be approximated by our neural network from (4.5). We drew 200 samples from $\mathcal{G}$ and found a corresponding $\theta_0^{(j)}$ for each sample. Then, as in the case of heat equations above, we generated the parameter set $\Theta := \{\theta_0^{(j)} + \delta \ : \ |\delta| \le 3,\ j = 1,\dots,200\}$. We sampled from Θ uniformly and generated paths from the 200 centers to train V_ξ. We again tested the method on a new set of 100 initials from $\mathcal{G}$. The results of the proposed method at times $t = 0, 0.15, 0.3, 0.45, 0.6$ for three random initials are shown in Figure 5. In Figures 3(e) and 3(f), we again plot the mean relative and absolute errors versus time, which demonstrate the promising approximation performance of our method.

Figure 3(e) shows some challenges in the relative error as time advances. This is because the solution to the Allen–Cahn equation for this initial value has fast-increasing derivatives as time progresses, which poses a challenge to all numerical methods, including ours in solving Allen–Cahn equations in general. Specifically, such large derivatives force the parameters θ of the neural network to blow up quickly, and hence the trajectory θ_t may rapidly escape from the prescribed Θ over which we trained the vector field V_ξ. This is a challenge that remains to be overcome by using more adaptive training methods and sampling strategies.

For this experiment, generating $\tilde{G}$ and $\tilde{p}$ for (4.2) took 1.04 hours, while the generation of the trajectories for (4.3) took only 15 minutes. The much lower dimension of this problem compared to the transport and heat equation examples accounted for the speedup in the generation of samples. Similar to the transport equation, training V_ξ took only 7 minutes. For testing, it took an average of 21 seconds to train a θ_0 to a sampled g and an average of 2.1 seconds to then solve (3.9) using a 4th-order Runge–Kutta solver with step size 0.002. This amounts to less than 24 seconds in total time per initial for the testing stage.

5. Variations and generalizations. In this section, we briefly discuss modifications of the proposed approach so that it can be applied to some other problems involving evolution PDEs. In particular, we consider the following two cases: general time-dependent PDEs and IVPs with time-varying boundary conditions.

Applications to general time-dependent PDEs. Our approach can be readily applied to a large variety of time-dependent PDEs. The reason is that these PDEs can be converted to the exact form of (3.5) for which our method is designed. To avoid overloading the bracket notation, we temporarily use $F(u)$ and $F(t,u)$ to represent

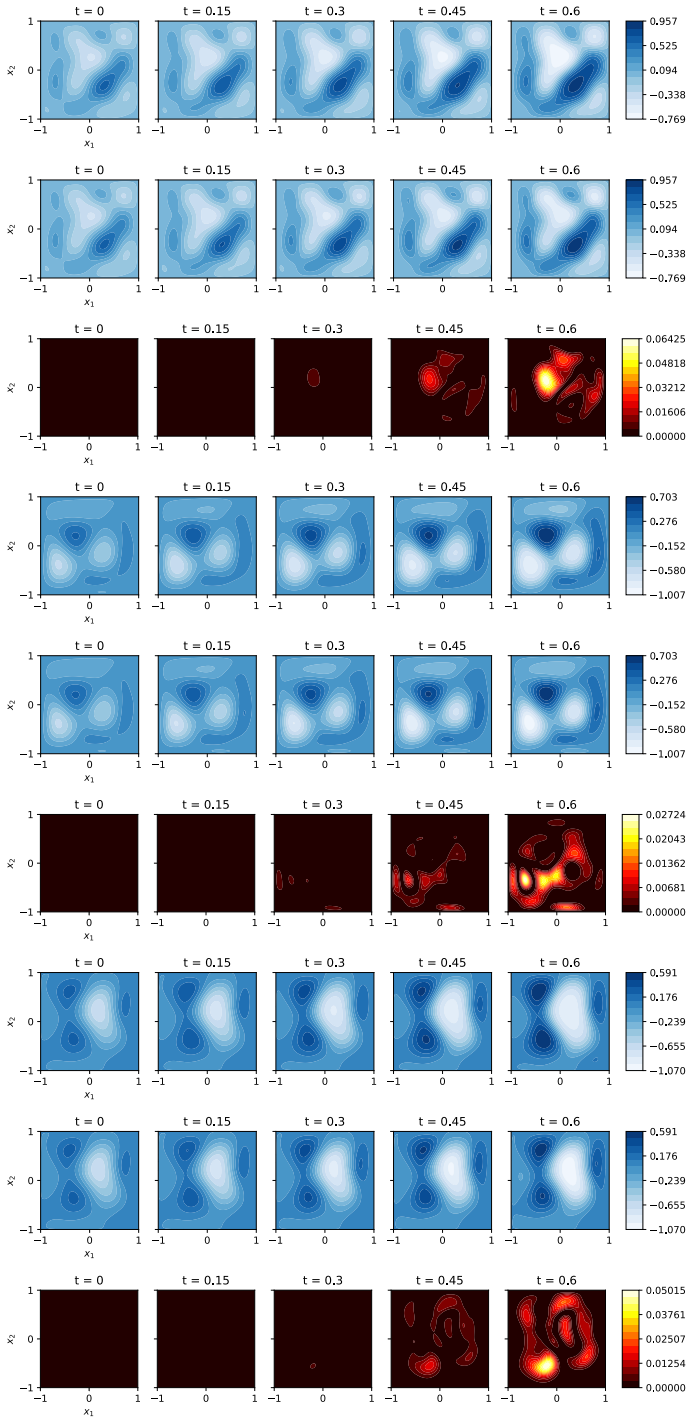

FIG. 5. *(Allen–Cahn equation). Comparison between the true solution* $u^*(\cdot,t)$*, the approximation* $u_{\theta_t}(\cdot)$*, and their pointwise absolute difference* $|u_{\theta_t}(x) - u^*(x,t)|$ *for times* $t = 0, 0.004, 0.008, 0.012, 0.015$ *for IVPs with the first (rows* 1–3*), second (rows* 4–6*), and third initial (rows* 7–9*) drawn from the set* $\mathcal{G}$ *defined in* (4.12).

$F[u]$ and $F_t[u]$ (differential operator that explicitly depends on time t). We first note that one can convert any nonautonomous evolution PDE into an autonomous one:

$$\partial_t u = F(t,u) \quad \Longleftrightarrow \quad \partial_t \tilde{u} = \tilde{F}(\tilde{u}), \quad \text{where} \quad \tilde{u} := [t;u], \quad \tilde{F}(\tilde{u}) := \big[1; F(u)\big], \tag{5.1}$$

and $[\cdot\,;\cdot]$ means to stack the two arguments vertically to form a single one. We can also consider PDEs involving higher-order time derivatives and convert them to first-order PDE systems by noticing equivalency as follows:

$$\partial_{tt} u = F(u) \quad \Longleftrightarrow \quad \partial_t \tilde{u} = \tilde{F}(\tilde{u}), \quad \text{where} \quad \tilde{u} := [u;v], \quad \tilde{F}(\tilde{u}) := \big[v; F(u)\big]. \tag{5.2}$$

History-dependent PDEs can also be considered: denote $H_u(t) := \{u(\cdot,s)\,|\,0 \le s \le t\}$ as the trajectory recording the path of u up to time t and F a nonlinear operator on path H_u; then we can set $H_u(t)$ as an auxiliary variable and convert the problem $\partial_t u = F[H_u]$ to an autonomous evolution PDE of $[u; H_u]$.

Evolution PDEs with boundary conditions. We can also modify our method to solve IVPs with different boundary conditions. Let (g,ϕ) be the pair of initial and boundary values of the IVP. That is, $u(x,0)|_{\bar{\Omega}} = g$ and $u(x,t)|_{\partial\Omega\times[0,T]} = \phi$. In this case, we can parameterize $u_\theta(x) = \varphi_\eta(x) + \alpha(x)\psi_\zeta(x)$ with $\theta = (\eta,\zeta)$, where φ_η and ψ_ζ are two reduced-order models (e.g., neural nets) with parameters η and ζ, respectively, and $\alpha(x)$ is a prescribed smooth function such that $\alpha(x) > 0$ if $x \in \Omega$ and $\alpha(x) = 0$ if $x \in \partial\Omega$. Here φ_η is to fit the boundary value ϕ without interference from $\alpha\psi_\zeta$ as the latter vanishes on the boundary $\partial\Omega$.

6. Conclusion and future work. We have shown a novel strategy for solving linear and nonlinear evolution PDEs numerically. Specifically, we propose to use DNNs as nonlinear reduced-order models to represent PDE solutions and learn a control vector field to steer the network parameters so that the induced time-evolving neural network can approximate the solution accurately. The proposed method allows a user to quickly solve an evolution PDE with different initial values without the need to retrain the neural network. Error estimates of the proposed approach are also provided.

We implemented the nonlinear reduced-order models as generic deep networks which yield promising results. We expect that the accuracy and effectiveness can be further improved by incorporating structural information and prior knowledge about the PDE and its solutions into the design of these networks. Training of control vector fields can also be made more efficient by integrating informative sample trajectories of θ_t. These improvements can potentially make the proposed method very effective in solving evolution PDEs in specified application domains.

Appendix A. Proof of (3.19). In the proof of Proposition 3.4, we need (3.19). This can be obtained by applying the lemma below, whose proof is a slight modification of the proof of [64, Theorem 2.1.14].

LEMMA A.1. *Let $f : \mathbb{R}^d \to \mathbb{R}$ be a differentiable convex function and ∇f be L-Lipschitz continuous for some $L > 0$. Define the gradient descent iterates by*

$$x_i = x_{i-1} - h\nabla f(x_{i-1})$$

with $x_0 \in \mathbb{R}^d$. Let $y \in \mathbb{R}^d$ and $0 < h < \frac{1}{L}$; then for any $k \ge 1$ there exists

$$f(x_k) - f(y) \le \frac{|x_0 - y|^2}{2kh}.$$

Proof. Following the standard steps in the proof of [64, Theorem 2.1.14] and using $0 < h < 1/L$, we can derive the bound

$$f(x_i) - f(x_{i-1}) \le -h\left(1 - \frac{1}{2}hL\right)|\nabla f(x_{i-1})|^2 \le -\frac{h}{2}|\nabla f(x_{i-1})|^2. \tag{A.1}$$

Since f is convex, there exists

$$f(x) \le f(y) + \nabla f(x)^\top (x - y) \quad \forall x \in \mathbb{R}^d.$$

Combining this with $x = x_{i-1}$ and (A.1), we derive

$$\begin{aligned} f(x_i) - f(y) &\le \nabla f(x_{i-1})^\top (x_{i-1} - y) - \frac{h}{2}|\nabla f(x_{i-1})|^2 \\ &= \frac{1}{2h}\left(2h\nabla f(x_{i-1})^\top (x_{i-1} - y) - h^2|\nabla f(x_{i-1})|^2\right. \\ &\quad \left. + |x_{i-1} - y|^2 - |x_{i-1} - y|^2\right) \\ &= \frac{1}{2h}\left(|x_{i-1} - y|^2 - |x_{i-1} - h\nabla f(x_{i-1}) - y|^2\right) \\ &= \frac{1}{2h}\left(|x_{i-1} - y|^2 - |x_i - y|^2\right). \end{aligned}$$

We can now bound the telescoping sum

$$\sum_{i=1}^{k}(f(x_i) - f(y)) \le \frac{1}{2h}\sum_{i=1}^{k}(|x_{i-1} - y|^2 - |x_i - y|^2) \le \frac{1}{2h}|x_0 - y|^2.$$

By (A.1), we know that $f(x_k) \le f(x_{k-1}) \le \cdots \le f(x_0)$ and therefore

$$f(x_k) - f(y) \le \frac{1}{k}\sum_{i=1}^{k}(f(x_i) - f(y)) \le \frac{|x_0 - y|^2}{2hk}.$$

∎

REFERENCES

[1] W. F. Ames, *Numerical Methods for Partial Differential Equations*, Academic Press, New York, 2014.

[2] W. Anderson and M. Farazmand, *Evolution of nonlinear reduced-order solutions for PDEs with conserved quantities*, SIAM J. Sci. Comput., 44 (2022), pp. A176–A197, https://doi.org/10.1137/21M1415972.

[3] C. Anitescu, E. Atroshchenko, N. Alajlan, and T. Rabczuk, *Artificial neural network methods for the solution of second order boundary value problems*, Comput. Mater. Continua, 59 (2019), pp. 345–359.

[4] K. Atkinson, *An Introduction to Numerical Analysis*, 2nd ed., John Wiley & Sons, New York, 1989.

[5] G. Bao, X. Ye, Y. Zang, and H. Zhou, *Numerical solution of inverse problems by weak adversarial networks*, Inverse Problems, 36 (2020), 115003, https://doi.org/10.1088/1361-6420/abb447.

[6] C. Beck, W. E, and A. Jentzen, *Machine learning approximation algorithms for high-dimensional fully nonlinear partial differential equations and second-order backward stochastic differential equations*, J. Nonlinear Sci., 29 (2019), pp. 1563–1619.

[7] J. Berg and K. Nyström, *A unified deep artificial neural network approach to partial differential equations in complex geometries*, Neurocomputing, 317 (2018), pp. 28–41.

[8] N. Boullé, C. Earls, and A. Townsend, *Data-driven discovery of Green's functions with human-understandable deep learning*, Sci. Rep., 12 (2022), 4824, https://doi.org/10.1038/s41598-022-08745-5.

[9] N. Boullé, S. Kim, T. Shi, and A. Townsend, *Learning Green's functions associated with time-dependent partial differential equations*, J. Mach. Learn. Res., 23 (2022), 218.

[10] J. Bruna, B. Pherstorfer, and E. Vanden-Eijnden, *Neural Galerkin Scheme with Active Learning for High-Dimensional Evolution Equations*, preprint, arXiv:2203.01360, 2022.

[11] S. Cai, Z. Wang, L. Lu, T. A. Zaki, and G. E. Karniadakis, *DeepM&Mnet: Inferring the Electroconvection Multiphysics Fields Based on Operator Approximation by Neural Networks*, preprint, arXiv:2009.12935, 2020.

[12] W. Cai, X. Li, and L. Liu, *A phase shift deep neural network for high frequency approximation and wave problems*, SIAM J. Sci. Comput., 42 (2020), pp. A3285–A3312, https://doi.org/10.1137/19M1310050.

[13] W. Cai and Z.-Q. J. Xu, *Multi-scale Deep Neural Networks for Solving High Dimensional PDEs*, preprint, arXiv:1910.11710, 2019.

[14] Y. Chen, B. Dong, and J. Xu, *Meta-MgNet: Meta Multigrid Networks for Solving Parameterized Partial Differential Equations*, preprint, arXiv:2010.14088, 2020.

[15] P. Clark Di Leoni, C. Meneveau, G. Karniadakis, and T. Zaki, *Deep operator neural networks (DeepONets) for Prediction of Instability Waves in High-Speed Boundary Layers*, Bulletin of the American Physical Society, College Park, MD, 2020.

[16] S. Cuomo, V. S. Di Cola, F. Giampaolo, G. Rozza, M. Raissi, and F. Piccialli, *Scientific Machine Learning through Physics-Informed Neural Networks: Where We Are and What's Next*, preprint, arXiv:2201.05624, 2022.

[17] M. Dissanayake and N. Phan-Thien, *Neural-network-based approximations for solving partial differential equations*, Commun. Numer. Methods Engrg., 10 (1994), pp. 195–201.

[18] S. Dong and N. Ni, *A Method for Representing Periodic Functions and Enforcing Exactly Periodic Boundary Conditions with Deep Neural Networks*, preprint, arXiv:2007.07442, 2020.

[19] Y. Du and T. A. Zaki, *Evolutional deep neural network*, Phys. Rev. E (3), 104 (2021), 045303, https://doi.org/10.1103/PhysRevE.104.045303.

[20] W. E, J. Han, and A. Jentzen, *Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations*, Commun. Math. Stat., 5 (2017), pp. 349–380.

[21] W. E and B. Yu, *The deep Ritz method: A deep learning-based numerical algorithm for solving variational problems*, Commun. Math. Stat., 6 (2018), pp. 1–12.

[22] G. Evans, J. Blackledge, and P. Yardley, *Numerical Methods for Partial Differential Equations*, Springer, New York, 2012.

[23] L. C. Evans, *Partial Differential Equations*, Grad. Stud. Math. 19, American Mathematical Society, Providence, RI, 1998.

[24] Y. Fan, C. O. Bohorquez, and L. Ying, *BCR-Net: A neural network based on the nonstandard wavelet form*, J. Comput. Phys., 384 (2019), pp. 1–15.

[25] M. Fujii, A. Takahashi, and M. Takahashi, *Asymptotic expansion as prior knowledge in deep learning method for high dimensional BSDEs*, Asia-Pac. Financ. Mark., 26 (2019), pp. 391–408.

[26] Y. Gu, C. Wang, and H. Yang, *Structure Probing Neural Network Deflation*, preprint, arXiv:2007.03609, 2020.

[27] Y. Gu, H. Yang, and C. Zhou, *SelectNet: Self-Paced Learning for High-Dimensional Partial Differential Equations*, preprint, arXiv:2001.04860, 2020.

[28] I. Guhring, G. Kutyniok, and P. Peterson, *Error bounds for approximations with deep ReLU neural networks in $W^{s,p}$ norms*, Anal. Appl. (Singap.), 18 (2020), pp. 803–859.

[29] I. Gühring and M. Raslan, *Approximation rates for neural networks with encodable weights in smoothness spaces*, Neural Netw., 134 (2020), pp. 107–130, https://doi.org/10.1016/j.neunet.2020.11.010.

[30] M. Guo and J. S. Hesthaven, *Data-driven reduced order modeling for time-dependent problems*, Comput. Methods Appl. Mech. Engrg., 345 (2019), pp. 75–99, https://doi.org/10.1016/j.cma.2018.10.029.

[31] X. Guo, W. Li, and F. Iorio, *Convolutional neural networks for steady flow approximation*, in Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016, pp. 481–490.

[32] J. Han, A. Jentzen, and W. E, *Overcoming the Curse of Dimensionality: Solving High-Dimensional Partial Differential Equations Using Deep Learning*, preprint, arXiv:1707.02568v1, 2017.

[33] J. Han, A. Jentzen, and W. E, *Solving high-dimensional partial differential equations using deep learning*, Proc. Natl. Acad. Sci. USA, 115 (2018), pp. 8505–8510, https://doi.org/10.1073/pnas.1718942115.

[34] J. Han, J. Lu, and M. Zhou, *Solving high-dimensional eigenvalue problems using deep neural networks: A diffusion Monte Carlo like approach*, J. Comput. Phys., 423 (2020), 109792.

[35] Y. Han, J. Yoo, H. H. Kim, H. J. Shin, K. Sung, and J. C. Ye, *Deep learning with domain adaptation for accelerated projection-reconstruction MR*, Magn. Reson. Med., 80 (2018), pp. 1189–1205.

[36] K. Hornik, *Approximation capabilities of multilayer feedforward networks*, Neural Netw., 4 (1991), pp. 251–257.

[37] K. Hornik, M. Stinchcombe, and H. White, *Multilayer feedforward networks are universal approximators*, Neural Netw., 2 (1989), pp. 359–366.

[38] J. Huang, H. Wang, and H. Yang, *Int-Deep: A deep learning initialized iterative method for nonlinear problems*, J. Comput. Phys., 419 (2020), 109675.

[39] C. Huré, H. Pham, and X. Warin, *Deep backward schemes for high-dimensional nonlinear PDEs*, Math. Comp., 89 (2020), pp. 1547–1579.

[40] M. Hutzenthaler, A. Jentzen, T. Kruse, and T. A. Nguyen, *A proof that rectified deep neural networks overcome the curse of dimensionality in the numerical approximation of semilinear heat equations*, Partial Differ. Equ. Appl., 1 (2020), 10.

[41] A. D. Jagtap, K. Kawaguchi, and G. E. Karniadakis, *Adaptive activation functions accelerate convergence in deep and physics-informed neural networks*, J. Comput. Phys., 404 (2020), 109136.

[42] C. Johnson, *Numerical Solution of Partial Differential Equations by the Finite Element Method*, Courier Corporation, North Chelmsford, MA, 2012.

[43] E. Kharazmi, Z. Zhang, and G. E. Karniadakis, *hp-VPINNs: Variational Physics-Informed Neural Networks with Domain Decomposition*, preprint, arXiv:2003.05385, 2020.

[44] N. Kovachki, S. Lanthaler, and S. Mishra, *On universal approximation and error bounds for Fourier neural operators*, J. Mach. Learn. Res., 22 (2021), 290.

[45] N. B. Kovachki, Z. Li, B. Liu, K. Azizzadenesheli, K. Bhattacharya, A. M. Stuart, and A. Anandkumar, *Neural operator: Learning maps between function spaces with applications to PDEs*, J. Mach. Learn. Res., 24 (2023), 89.

[46] M. Kumar and N. Yadav, *Multilayer perceptrons and radial basis function neural network methods for the solution of differential equations: A survey*, Comput. Math. Appl., 62 (2011), pp. 3796–3811.

[47] I. E. Lagaris, A. Likas, and D. I. Fotiadis, *Artificial neural networks for solving ordinary and partial differential equations*, IEEE Trans. Neural Netw., 9 (1998), pp. 987–1000.

[48] H. Lee and I. S. Kang, *Neural algorithm for solving differential equations*, J. Comput. Phys., 91 (1990), pp. 110–131.

[49] B. Li, S. Tang, and H. Yu, *Better approximations of high dimensional smooth functions by deep neural networks with rectified power units*, Commun. Comput. Phys., 27 (2020), pp. 379–411, https://doi.org/10.4208/cicp.OA-2019-0168.

[50] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, *Fourier Neural Operator for Parametric Partial Differential Equations*, preprint, arXiv:2010.08895, 2020.

[51] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, *Neural Operator: Graph Kernel Network for Partial Differential Equations*, preprint, arXiv:2003.03485, 2020.

[52] S. Liang and R. Srikant, *Why deep neural networks for function approximation?*, in Proceedings of the International Conference on Learning Representations (ICLR), 2017.

[53] S. Liang and H. Yang, *Finite Expression Method for Solving High-Dimensional Partial Differential Equations*, preprint, arXiv:2206.10121, 2022.

[54] G. Lin, F. Chen, P. Hu, X. Chen, J. Chen, J. Wang, and Z. Shi, *BI-GreenNet: Learning Green's Functions by Boundary Integral Network*, preprint, arXiv:2204.13247, 2022.

[55] Z. Liu, W. Cai, and Z.-Q. J. Xu, *Multi-scale Deep Neural Network (MscaleDNN) for Solving Poisson-Boltzmann Equation in Complex Domains*, preprint, arXiv:2007.11207, 2020.

[56] W. Lötzsch, S. Ohler, and J. S. Otterbach, *Learning the Solution Operator of Boundary Value Problems Using Graph Neural Networks*, preprint, arXiv:2206.14092, 2022.

[57] L. Lu, P. Jin, and G. E. Karniadakis, *DeepONet: Learning Nonlinear Operators for Identifying Differential Equations Based on the Universal Approximation Theorem of Operators*, preprint, arXiv:1910.03193, 2019.

[58] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis, *DeepXDE: A Deep Learning Library for Solving Differential Equations*, preprint, arXiv:1907.04502, 2019.

[59] T. Luo and H. Yang, *Two-Layer Neural Networks for Partial Differential Equations: Optimization and Generalization Theory*, preprint, arXiv:2006.15733, 2020.

[60] L. Lyu, K. Wu, R. Du, and J. Chen, *Enforcing Exact Boundary and Initial Conditions in the Deep Mixed Residual Method*, preprint, arXiv:2008.01491, 2020.
[61] M. Magill, F. Qureshi, and H. de Haan, *Neural networks trained to solve differential equations learn general representations*, in Advances in Neural Information Processing Systems, MIT Press, Cambridge, MA, 2018, pp. 4071–4081.
[62] Z. Mao, L. Lu, O. Marxen, T. A. Zaki, and G. E. Karniadakis, *DeepM&Mnet for Hypersonics: Predicting the Coupled Flow and Finite-Rate Chemistry Behind a Normal Shock Using Neural-Network Approximation of Operators*, preprint, arXiv:2011.03349, 2020.
[63] M. A. Nabian and H. Meidani, *A Deep Neural Network Surrogate for High-Dimensional Random Partial Differential Equations*, preprint, arXiv:1806.02957, 2018.
[64] Y. Nesterov, *Introductory Lectures on Convex Programming: Basic Course*, Vol. 1, 1998, pp. 119–120.
[65] N. Nüsken and L. Richter, *Solving high-dimensional Hamilton–Jacobi–Bellman PDEs using neural networks: Perspectives from the theory of controlled diffusions and measures on path space*, Partial Differ. Equ. Appl., 2 (2021), 48.
[66] G. Pang, M. D'Elia, M. Parks, and G. E. Karniadakis, *nPINNs: Nonlocal Physics-Informed Neural Networks for a Parametrized Nonlocal Universal Laplacian Operator. Algorithms and Applications*, preprint, arXiv:2004.04276, 2020.
[67] G. Pang, L. Lu, and G. E. Karniadakis, *fpinns: Fractional physics-informed neural networks*, SIAM J. Sci. Comput., 41 (2019), pp. A2603–A2626, https://doi.org/10.1137/18M1229845.
[68] P. Petersen and F. Voigtlaender, *Optimal approximation of piecewise smooth functions using deep ReLU neural networks*, Neural Netw., 108 (2018), pp. 296–330.
[69] H. Pham, X. Warin, and M. Germain, *Neural networks-based backward scheme for fully nonlinear PDEs*, Partial Differ. Equ. Appl., 2 (2021), 16, https://doi.org/10.1007/s42985-020-00062-8.
[70] A. Quarteroni and A. Valli, *Numerical Approximation of Partial Differential Equations*, Springer Ser. Comput. Math. 23, Springer-Verlag, Berlin, 2008.
[71] M. Raissi and G. E. Karniadakis, *Machine Learning of Linear Differential Equations Using Gaussian Processes*, preprint, arXiv:1701.02440, 2017.
[72] M. Raissi, P. Perdikaris, and G. E. Karniadakis, *Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations*, preprint, arXiv:1711.10561, 2017.
[73] M. Raissi, P. Perdikaris, and G. E. Karniadakis, *Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations*, J. Comput. Phys., 378 (2019), pp. 686–707.
[74] A. A. Ramabathiran and P. Ramachandran, *SPINN: Sparse, physics-based, and partially interpretable neural networks for PDEs*, J. Comput. Phys., 445 (2021), 110600.
[75] B. Raonić, R. Molinaro, T. Rohner, S. Mishra, and E. de Bezenac, *Convolutional Neural Operators*, preprint, arXiv:2302.01178, 2023.
[76] F. Regazzoni, L. Dedè, and A. Quarteroni, *Machine learning for fast and reliable solution of time-dependent differential equations*, J. Comput. Phys., 397 (2019), 108852, https://doi.org/10.1016/j.jcp.2019.07.050.
[77] Y. Shin, J. Darbon, and G. E. Karniadakis, *On the Convergence and Generalization of Physics Informed Neural Networks*, preprint, arXiv:2004.01806, 2020.
[78] J. Sirignano and K. Spiliopoulos, *DGM: A deep learning algorithm for solving partial differential equations*, J. Comput. Phys., 375 (2018), pp. 1339–1364.
[79] R. K. Srivastava, K. Greff, and J. Schmidhuber, *Training very deep networks*, in Advances in Neural Information Processing Systems, Vol. 28, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, eds., Curran Associates, Red Hook, NY, 2015, pp. 2377–2385.
[80] P. Tabuada and B. Gharesifard, *Universal approximation power of deep residual neural networks through the lens of control*, IEEE Trans. Automat. Control, 68 (2023), pp. 2715–2728, https://doi.org/10.1109/TAC.2022.3190051.
[81] T. Tang and J. Yang, *Implicit-explicit scheme for the Allen-Cahn equation preserves the maximum principle*, J. Comput. Math., 34 (2016), pp. 471–481, https://doi.org/10.4208/jcm.1603-m2014-0017.
[82] Y. Teng, X. Zhang, Z. Wang, and L. Ju, *Learning Green's functions of linear reaction-diffusion equations with application to fast numerical solver*, in Proceedings of Mathematical and Scientific Machine Learning, Proc. Mach. Learn. Res. 190, 2022, pp. 1–16.
[83] J. W. Thomas, *Numerical Partial Differential Equations: Conservation Laws and Elliptic Equations*, Texts Appl. Math. 33, Springer, New York, 2013.

[84] J. W. Thomas, *Numerical Partial Differential Equations: Finite Difference Methods*, Texts Appl. 22, Springer, New York, 2013.
[85] B. Wang, W. Zhang, and W. Cai, *Multi-scale Deep Neural Network (MscaleDNN) Methods for Oscillatory Stokes Flows in Complex Domains*, preprint, arXiv:2009.12729, 2020.
[86] C. Wang, S. Li, D. He, and L. Wang, *Is L^2 Physics-Informed Loss Always Suitable for Training Physics-Informed Neural Network?*, preprint, arXiv:2206.02016, 2022.
[87] S. Wang, S. Sankaran, and P. Perdikaris, *Respecting Causality Is All You Need for Training Physics-Informed Neural Networks*, preprint, arXiv:2203.07404, 2022.
[88] S. Wang, H. Wang, and P. Perdikaris, *Learning the solution operator of parametric partial differential equations with physics-informed DeepONets*, Sci. Adv., 7 (2021), eabi8605.
[89] G. Wen, Z. Li, K. Azizzadenesheli, A. Anandkumar, and S. M. Benson, *U-FNO—an enhanced Fourier neural operator-based deep-learning model for multiphase flow*, Adv. Water Resour., 163 (2022), 104180.
[90] L. Yang, D. Zhang, and G. E. Karniadakis, *Physics-informed generative adversarial networks for stochastic differential equations*, SIAM J. Sci. Comput., 42 (2020), pp. A292–A317, https://doi.org/10.1137/18M1225409.
[91] Y. Yang and P. Perdikaris, *Adversarial uncertainty quantification in physics-informed neural networks*, J. Comput. Phys., 394 (2019), pp. 136–152.
[92] D. Yarotsky, *Error bounds for approximations with deep ReLU networks*, Neural Netw., 94 (2017), pp. 103–114.
[93] Y. Zang, G. Bao, X. Ye, and H. Zhou, *Weak adversarial networks for high-dimensional partial differential equations*, J. Comput. Phys., 411 (2020), 109409.
[94] E. Zhang, M. Yin, and G. E. Karniadakis, *Physics-Informed Neural Networks for Nonhomogeneous Material Identification in Elasticity Imaging*, preprint, arXiv:2009.04525, 2020.
[95] Y. Zhu and N. Zabaras, *Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification*, J. Comput. Phys., 366 (2018), pp. 415–447.