Journal of Computational Physics 463 (2022) 111233

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp

Towards fast weak adversarial training to solve high dimensional parabolic partial differential equations using XNODE-WAN

Paul Valsecchi Oliva [a], Yue Wu [b,c,d], Cuiyu He [e], Hao Ni [a,c,*]

[a] The Department of Mathematics, University College London, 25 Gordon St, London, WC1H 0AY, UK
[b] Mathematical Institute, University of Oxford, Radcliffe Observatory, Andrew Wiles Building, Woodstock Rd, Oxford, OX2 6GG, UK
[c] Alan Turing Institute, 2QR, John Dodson House, 96 Euston Rd, London, NW1 2DB, UK
[d] The Department of Mathematics and Statistics, University of Strathclyde, 26 Richmond St, Glasgow, G1 1XQ, UK
[e] School of Mathematics and Statistical Science, University of Texas Rio Grande Valley, 1 West University Blvd, Brownsville, TX, 78520, USA

ARTICLE INFO

Article history:
Received 13 January 2022
Received in revised form 10 April 2022
Accepted 13 April 2022
Available online 20 April 2022

Keywords:
Parabolic partial differential equation
Generative adversarial network
Neural ordinary differential equation
Weak adversarial network

ABSTRACT

Due to the curse of dimensionality, solving high dimensional parabolic partial differential equations (PDEs) has been a challenging problem for decades. Recently, a weak adversarial network (WAN) proposed in Zang et al. (2020) [17] offered a flexible and computationally efficient approach to tackle this problem defined on arbitrary domains by leveraging the weak solution. WAN reformulates the PDE problem as a generative adversarial network, where the weak solution (primal network) and the test function (adversarial network) are parameterized by the multi-layer deep neural networks (DNNs). However, it is not yet clear whether DNNs are the most effective model for the parabolic PDE solutions as they do not take into account the fundamentally different roles played by time and spatial variables in the solution. To reinforce the difference, we design a novel so-called XNODE model for the primal network, which is built on the neural ODE (NODE) model with additional spatial dependency to incorporate the a priori information of the PDEs and serve as a universal and effective approximation to the solution. The proposed hybrid method (XNODE-WAN), by integrating the XNODE model within the WAN framework, leads to significant improvement in the performance and efficiency of training. Numerical results show that our method can reduce the training time to a fraction of that of the WAN model.

1. Introduction

Applying classical numerical methods (finite element, finite difference, finite volume, etc.) to solve high dimensional partial differential equations (PDEs) has been highly challenging due to the notorious curse of dimensionality [1,2]. Traditional numerical methods, used as PDE solvers, lead to an exponential growth of computing cost with respect to the dimension of the PDE problem. The application of neural networks on high dimensional data, including image classification [3] and natural language processing [4], has empirically proven to be successful. Recently, applying deep learning models to tackle high dimensional PDE problems, which achieved superior empirical results [1,5–8], has been a growing area of research.

* Corresponding author at: The Department of Mathematics, University College London, 25 Gordon St, London, WC1H 0AY, UK.
E-mail address: h.ni@ucl.ac.uk (H. Ni).

https://doi.org/10.1016/j.jcp.2022.111233

Relevant numerical analysis that takes into account of approximation rates, generality and optimization theory can be also found in many works, see [9,10,2,11].

As an important class of PDEs, parabolic PDEs have been studied extensively and applied to a various number of fields, including acoustic propagation [12], heat conduction [13], image denoising [14], derivative pricing [15], etc. In [16,17], multi-layer deep neural networks (DNNs) are used to approximate the solution to parabolic problems thanks to their universality. However, DNNs do not take into account the fundamental difference in the roles played by time and spatial variables in the parabolic solution, and hence may not be efficient in capturing the spatio-temporal dependence of the parabolic solution. To this end, we propose a novel so-called XNODE model for the solution to reinforce the spatio-temporal dependence. It effectively incorporates the a priori information of the PDEs through the modified neural ordinary differential equations (NODEs) [18].

More specifically, we consider the following parabolic PDE defined on an arbitrary bounded domain $\mathcal{D} \subset [0,T] \times \mathbb{R}^d$ under the mild condition,

$$\begin{cases} \partial_t u(t,\mathbf{x}) - \sum_{i=1}^{d} \partial_i (\sum_{i=1}^{d} a_{ij}(t,\mathbf{x}) \partial_j u(t,\mathbf{x})) + \sum_{i=1}^{d} b_i(t,\mathbf{x}) \partial_i u(t,\mathbf{x}) & \\ +c(u,t,\mathbf{x}) - f(t,\mathbf{x}) = 0 & \text{for } (t,\mathbf{x}) \in \mathcal{D}, \\ u(t,\mathbf{x}) = g(t,\mathbf{x}) & \text{on } \partial\mathcal{D}, \\ u(0,\mathbf{x}) - h(\mathbf{x}) = 0 & \text{on } \Omega(0), \end{cases} \tag{1}$$

where $a_{ij}, b_i, f : \mathcal{D} \to \mathbb{R}$, $c : \mathbb{R} \times \mathcal{D} \to \mathbb{R}$ and $h : \Omega(0) \to \mathbb{R}$ are given. c can be a non-linear function with respect to the first augment. Let $\Omega(t) := \{\mathbf{x} | (t,\mathbf{x}) \in \mathcal{D}\}$ denote the spatial domain of $\mathcal{D}$ when restricting time to be t. Note that $\mathcal{D}$ could be not only the time-independent domain, i.e. there exists $\Omega \subset \mathbb{R}^d$, such that $\mathcal{D} = [0,T] \times \Omega$, but also the time varying domain, i.e. $\Omega(t)$ changes over time t.

The motivation of the XNODE model lies in the key observation that for a fixed spatial variable $\mathbf{x}$, a parabolic PDE solution $\tilde{u} = u(.,\mathbf{x})$ evolves into an ODE solution, which naturally leads to employing the NODE network [18]. A NODE network builds a general model by leveraging the ODE solver and parameterizing the vector field as a neural network, which can be regarded as a continuous version of the residual neural network. The NODE model recently introduced in [18] soon became a top-pick model for (continuous) time series modeling because of its efficiency and scalability. As opposed to classical neural networks models, the gradient calculation of the NODE models via the adjoint method has constant memory cost.

Therefore, to model the parabolic PDE solution efficiently, the above observation leads to the proposed XNODE model, which is built by introducing the additional dependency on the spatial variable $\mathbf{x}$ to the NODE model. More specifically, the XNODE model takes any spatial variable $\mathbf{x}$ and h from (1) as the input, and returns the output of the NODE with vector fields dependent on spatial variable $\mathbf{x}$ to predict the solution $\tilde{u} = u(.,\mathbf{x})$. In contrast to the DNNs which treat the time and spatial variable equally, the proposed XNODE model utilizes the NODE to capture the temporal dependency of the solution path $\tilde{u}$, while the spatial dependence is embedded to the vector field of NODE model. In the XNODE model, the initial condition of PDEs h is incorporated effectively through the initial condition of the NODE. Moreover, we introduce efficient data sampling and path construction with the XNODE method to ensure efficiency for predicting the solution on both time-independent domains and time-varying domains.

The formulation of the PDE problem using deep learning can be broadly divided into two classes, (1) optimization problems and (2) min-max problems. For the first category, learning the PDE solution is translated into minimizing the loss function, which is used to quantify the performance of the estimated solution. For example, one can consider the mean square error regarding the equilibrate equations and boundary and initial conditions as the loss function, see e.g., [19,16,7]. The least square approach is known for its generality. However, the evaluation of high order derivatives are required in the cost functional if the equilibrate equation is directly penalized [16,7]. Another commonly used cost functional is based on the energy or minimal principal, [20–22]. Though the energy functional often only involves the lower order derivatives, i.e. no higher derivative is required to compute, the energy based method is restricted to certain differential operators for which a minimal principal holds. The second category is to utilize the generative adversarial networks (GANs) to solve the PDEs by using the min-max game formulation. The weak adversarial network (WAN) [17] is a typical example of this type. By leveraging the weak formulation of the PDE solution, WAN converts the task of finding the weak solution into a saddle point problem with the solution (u,ϕ), where u is the solution to the PDE, and ϕ is a test function. Compared with the least square method of the first category, WAN can be applied to solve a large general class of PDEs, including the cases where the strong solution does not exist as long as the weak solution is well defined. It also can be applied to the case where the minimal principal of the PDEs is not satisfied.

Based on the above considerations, we propose the hybrid XNODE-WAN method by incorporating the XNODE model into the WAN framework [17] as an enhancement of the WAN to learn the solutions to parabolic PDEs more efficiently. In the WAN framework, there is a primal network and an adversarial network to approximate the solution (u) and the test function (ϕ) respectively, which are both approximated by DNNs [17]. In our proposed XNODE-WAN method, we replace the DNNs with the XNODE model for the primal network of WAN, as the XNODE model is able to better model the spatio-temporal dependency of the solution and encode a priori information of the PDE, which leads to reduced training time and faster

convergence. As many of the other deep learning methods for PDE solvers [17,7,22], the proposed XNODE-WAN method is mesh-free. It can be applied to a general class of the spatio-temporal domains, whose spatial domain can be irregular and/or time-varying.

Numerical results show that our proposed method significantly outperforms WAN in terms of training efficiency for both time-independent and time-varying domains. Moreover, such improvement of the XNODE-WAN model is consistent regardless of the dimension of the spatial variable, which demonstrates a better scalability for the high dimensional PDEs.

The paper is structured as follows. Section 2 gives a brief introduction to the WAN framework [17]. It is followed by Section 3, which starts with preliminaries of NODE model and then introduces the proposed XNODE model for the primal solution and the hybrid XNODE-WAN method for parabolic PDE solution on general time-space domains. Numerical results are provided in Section 4. Section 5 concludes the paper.

2. Weak Adversarial Network (WAN) framework

In this section, we provide a brief summary of theoretical underpinning and a general framework of the weak adversarial network for solving the linear and non-linear parabolic PDEs of form (1).

2.1. The weak formulation

We consider the PDE problem (1) on a time-dependent bounded domain denoted by $\mathcal{D} \subset [0,T] \times \mathbb{R}^d$ with $\Omega(t) = \{\mathbf{x} | (t,\mathbf{x}) \in \mathcal{D}\}$, where we impose typical assumptions such as the uniformly parabolic property of the differential operator.[1] We assume further that $a_{ij} \in L^\infty(\mathcal{D})$, $f \in L^2(\mathcal{D})$ and $h \in L^2(\Omega(0))$.

When the problem (1) is linear and on a time-independent domain, i.e., $\mathcal{D} = [0,T] \times \Omega$, the well-posedness of equation (1) can be ensured with the solution $u \in L^2([0,T], H_0^1(\Omega)) := \{v : v \in L^2([0,T], H^1(\Omega), v|_{\partial\Omega} = 0\}$ and $\partial_t u \in L^2([0,T], H^{-1}(\Omega))$, given the coefficients are uniformly elliptic [23].

In general, we consider the time-space domain $\mathcal{D} \subset [0,T] \times \mathbb{R}^d$. We now define the following bilinear form:

$$B(u,v) = \int_0^T \int_{\Omega(t)} \partial_t u \cdot v \mathrm{d}x\mathrm{d}t + \int_0^T \int_{\Omega(t)} \left(\sum_{ij} a_{ij} \partial_j u \partial_i v + \sum_i b_i \partial_i u, v + cuv \right) \mathrm{d}x\mathrm{d}t$$

and the linear operator:

$$f(v) = \int_0^T \int_{\Omega(t)} f v \mathrm{d}x\mathrm{d}t.$$

For the ease of notation, we define $H_0^1(\mathcal{D})$, $L^2(\mathcal{D})$ and $L^2(\partial\mathcal{D})$ by

$$H_0^1(\mathcal{D}) := \left\{ f : \mathcal{D} \to \mathbb{R} \,\middle|\, \forall t \in [0,T],\, f(t,\cdot) \in H_0^1\big(\Omega(t)\big) \text{ and } \int_0^T \int_{\Omega(t)} |f(t,x)|^2 + |\nabla f(t,x)|^2 dx dt < +\infty \right\},$$

$$L^2(\mathcal{D}) := \left\{ f : \partial\mathcal{D} \to \mathbb{R} \,\middle|\, \int_0^T \int_{\Omega(t)} |f(t,x)|^2 dx dt < +\infty \right\},$$

and

$$L^2(\partial\mathcal{D}) := \left\{ f : \mathcal{D} \to \mathbb{R} \,\middle|\, \int_0^T \int_{\partial\Omega(t)} |f(t,x)|^2 dx dt < +\infty \right\},$$

where $\nabla f(t,\cdot)$ is the weak derivative of $f(t,\cdot)$.

From integration by parts, it is easy to check that the weak solution u to (1) satisfies the following variational problem:

$$B(u,v) = f(v) \quad \forall v \in H_0^1(\mathcal{D}). \tag{2}$$

[1] The uniformly parabolic property means there exists $\alpha > 0$ such that $\sum_{i,j=1}^d a_{ij}(x,t)\zeta_i\zeta_j \geq \alpha|\zeta|^2 \quad \forall \zeta \in \mathbb{R}^n, (x,t) \in [0,T] \times \Omega$.

Therefore, it leads to a new interpretation of u as a solution to the below minimization problem:

$$\inf_{w\in H_0^1(\mathcal{D})} \sup_{v\in H_0^1(\mathcal{D}), v\neq 0} \frac{|B(w,v)-f(v)|^2}{\|v\|^2_{L^2(\mathcal{D})}}, \tag{3}$$

where $\|v\|_{L^2(\mathcal{D})}$ is L^2 norm of v ([17]). Note that for a given w, the optimal test function v is the one to attain the supremum $|B(w,v)-f(v)|^2/\|v\|^2_{L^2(\mathcal{D})}$. In (3) we reformulate the original parabolic PDE into an inf-sup type optimization problem, which sets the foundation for the WAN framework.

2.2. WAN framework

In this subsection, we introduce the WAN framework initially proposed in [17] (cf. Algorithm 3 in [17]) for solving parabolic PDEs by leveraging the weak formulation in Section 2.1. The main idea is to formulate the problem as the inf-sup problem (3), where the PDE solution u and the test function ϕ are parameterized by deep neural networks u_θ and ϕ_η, respectively, with θ, η being the trainable model weights. In [17], a deep neural network is employed for both the solutions u and ϕ. The primal network will be replaced by our proposed XNODE model, which will be discussed in Section 3.

The loss function of [17], which is also used in our method, consists of the interior loss, the boundary loss, and the initial loss as follows:

$$L(\theta,\eta) = L_{\text{int}}(\theta,\eta) + \alpha L_{\text{bdry}}(\theta) + \gamma L_{\text{init}}(\theta), \tag{4}$$

where α, γ are hyperparameters as penalty terms and

$$\begin{aligned} L_{\text{int}}(\theta,\eta) &= \log\left(\frac{|B(u_\theta,\phi_\eta)-f(\phi_\eta)|^2}{||\phi_\eta||^2_{L^2(\mathcal{D})}}\right),\\ L_{\text{init}}(\theta) &= ||u_\theta(0,\mathbf{x})-h(\mathbf{x})||^2_{L^2(\Omega(0))},\\ L_{\text{bdry}}(\theta) &= ||u_\theta(t,\mathbf{x})-g(t,\mathbf{x})||^2_{L^2(\partial\mathcal{D})}. \end{aligned} \tag{5}$$

Note that the interior loss L_{int} is based on the weak formulation of the solution (3) to quantify the discrepancy between the true solution u and the estimator u_θ over the interior. Due to the different norms and log operator applied in the loss function, the coefficient α and γ should be chosen numerically by cross-validation to ensure efficiency.

The evaluation of three loss terms can be done through the Monte-Carlo simulation on the domain. We shall create a collection of points for such evaluation at every iteration of our algorithm. As shown in [17], for any bounded domain $\mathcal{D}$, at each iteration, we uniformly sample N_n points $(t_i,x_i)_{i=1}^{N_n}$ over the interior of the domain. Similarly, we conduct the uniform sampling on the boundary $\partial\mathcal{D}$ with the corresponding number of sampling points denoted by N_b. Let $\tilde{L}_{\text{int}}(\theta,\eta)$, $\tilde{L}_{\text{bdry}}(\theta)$ and $\tilde{L}_{\text{init}}(\theta)$ denote the corresponding Monte-Carlo estimators of $L_{\text{int}}(\theta,\eta)$, $L_{\text{bdry}}(\theta)$ and $L_{\text{init}}(\theta)$ based on the sampling points respectively. Then we define the empirical loss function by

$$\tilde{L}(\theta,\eta) = \tilde{L}_{\text{int}}(\theta,\eta) + \alpha\tilde{L}_{\text{bdry}}(\theta) + \gamma\tilde{L}_{\text{init}}(\theta). \tag{6}$$

Eq. (6) is the loss function of the min-max game used in the WAN to learn the PDE solution.

Despite the same loss function of our method, due to the XNODE model, we adjust the above point sampling strategies to both time-independent domain and time-dependent domain respectively. One may refer to Section 3.2.1 and 3.2.2.

The algorithm of WAN is outline in Algorithm 1, where the lines with additional comments will be modified in our algorithm in Section 3. A list of notation explanation can be found in Table B.2.

3. The XNODE-WAN method

In this section, we introduce a novel so-called XNODE model for the solution u to the parabolic PDE problem (1) on arbitrary spatio-temporal domains. It can be conveniently incorporated within the WAN framework by replacing the deep neural network by the XNODE model for the primal solution to achieve superior training efficiency. We will refer to our method as XNODE-WAN in the remaining contents.

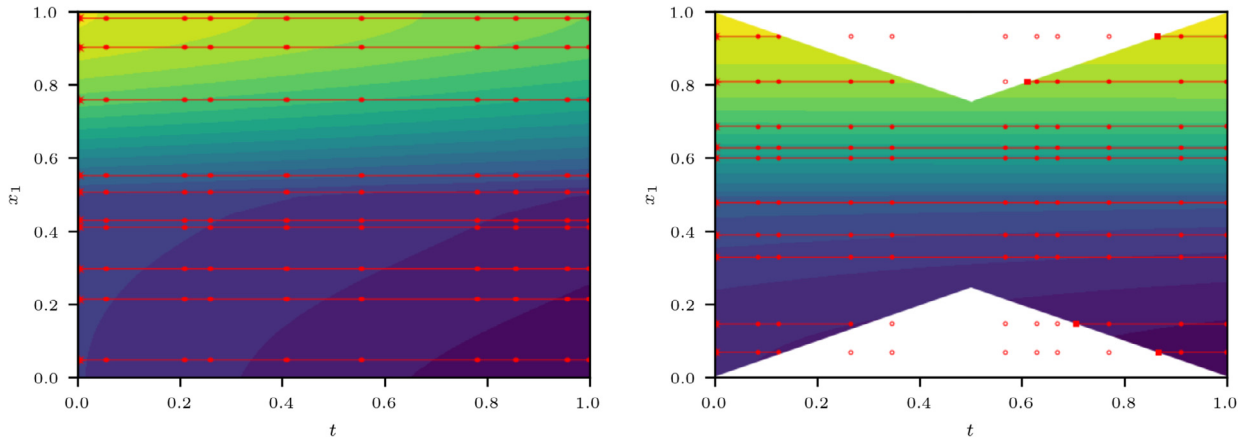
The main motivation of the proposed XNODE model comes from the observation that with the spatial variable fixed, the parabolic problem (1) reduces to an ODE problem, which can be efficiently modeled using the neural ODE model (NODE) [18]. Hence, the proposed XNODE model captures the additional dependency on spatial coordinates, incorporates the a priori information of the parabolic PDEs and initial conditions effectively. To facilitate the efficiency of XNODE to approximate the primal solution over the domain, we propose the use of multiple constant paths to traverse the domain as the point sampling strategy over the domain.

For the rest of this section, we start with a brief introduction to conventional NODE model. Then we proceed to discuss how to adapt the NODE model to approximate the parabolic solution by introducing additional spatial state variable, i.e.,

Algorithm 1 WAN algorithm.

Input: domain $\mathcal{D}$, tolerance $\epsilon > 0$, $N_{\mathcal{D}}/N_{\partial\mathcal{D}}/N_0$: number of sampled points on the domain/boundary/initial condition, K_u/K_ϕ: number of solution/adversarial network parameter updates per iteration, $\alpha/\gamma \in \mathbb{R}^+$: the weight of boundary/initial loss, and the hyperparameters for all DNNs involved

Initialize: the DNN network architectures $u_\theta, \phi_\eta : \mathcal{D} \to \mathbb{R}$
generate points sets $X_{\mathcal{D}}$, $X_{\partial\mathcal{D}}$ and X_0 ▷ random point sampling step
while $\tilde{L}(\theta, \eta) > \epsilon$ **do**
  `# update weak solution network parameter`
  for $k = 1, ..., K_u$ **do**
    compute $\nabla_\theta \tilde{L}(\theta, \eta)$; ▷ gradient calculation step
    update $\theta \leftarrow \theta - \tau_\theta \nabla_\theta \tilde{L}(\theta, \eta)$;
  end for
  `# update test network parameter`
  for $k = 1, ..., K_\phi$ **do**
    compute $\nabla_\eta \tilde{L}(\theta, \eta)$; ▷ gradient calculation step
    update $\eta \leftarrow \eta + \tau_\eta \nabla_\eta \tilde{L}(\theta, \eta)$;
  end for
  generate points sets $X_{\mathcal{D}}, X_{\partial\mathcal{D}}$ and X_0; ▷ random point sampling step for the next run
end while
Output: the weak solution $u_\theta : \mathcal{D} \to \mathbb{R}$



(a) a time-independent domain

(b) a time-dependent domain.

Fig. 1. An illustration of *constant paths* for spatial-time domains. (a) The red stars are the initial time points and the points are all the points at which we evaluate the data. Note that they are aligned in the time dimension. (b) The red squares here represent the inserted points that act as the initial points for the paths that cannot be traced back to the initial time. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

the XNODE model. This is followed by a subsection of using the XNODE-WAN model for solving time-independent domain problems, which involves incorporating the XNODE model within the WAN framework. We then generalize the proposed methodology to the time varying domains (see Fig. 1 for illustrative examples of time-independent domains and time-varying domains respectively).

3.1. The neural ODE (NODE) model

The NODE [18] method that fuses concepts of ODEs and neural networks, is well known for offering benefits to time series and density modeling. One of its main benefits lies in the computational feasibility brought by the adjoint sensitivity method, for which one solves an ODE to compute the gradients with constant memory cost.

More specifically, NODE models a continuous function which maps $\mathbb{R}^d$ to $\mathbb{R}^d$ (or $\mathcal{C}([0, T], \mathbb{R}^d)$[2]) by defining the ODE model as follows: for every input $\mathbf{h}_0 \in \mathbb{R}^d$, the corresponding output of NODE model is $\mathbf{h}(T)$ (or $\mathbf{h} = (\mathbf{h}(t))_{t\in[0,T]}$) where $\mathbf{h}$ satisfies

$$\frac{d\mathbf{h}(t)}{dt} = \mathcal{N}_\theta^{\text{vec}}(\mathbf{h}(t), t), \quad \mathbf{h}(0) = \mathbf{h}_0. \tag{7}$$

Here, the vector field $\mathcal{N}_\theta^{\text{vec}}$ is a parameterized neural network with trainable weights θ. The generality of the NODE model mainly come from the fact that the vector field can be universally approximated by a neural network. By Picard's Theorem,

[2] $\mathcal{C}([0, T], \mathbb{R}^d)$ denotes the space of continuous functions mapping from $[0, T]$ to $\mathbb{R}^d$.

the initial value problem will yield a unique solution provided $\mathcal{N}_\theta^{\text{vec}}$ is Lipschitz continuous which is guaranteed by using finite weights and the activation functions `relu` or `tanh` [18].

The details of forward evaluation of $\mathbf{h}$ and the derivative calculation of $\mathbf{h}$ with respect to θ can be found in [18]. Note that the dependency of the NODE output $\mathbf{h}$ on the input is solely passed through the initial condition of ODE model, i.e., $\mathbf{h}_0$.

3.2. XNODE-WAN

3.2.1. Time-independent domain

Consider the time-independent domain $[0, T] \times \Omega$, where $\Omega \subset \mathbb{R}^d$ is bounded. When given a fixed spatial variable $\mathbf{x} \in \Omega$, the map $\tilde{u} : t \mapsto u(t, \mathbf{x})$ can be viewed as a solution to the following ODE problem from PDE (1):

$$\frac{\mathrm{d}\tilde{u}(t)}{\mathrm{d}t} = F(\tilde{u}, t; \mathbf{x}), \quad \tilde{u}(0) = h(\mathbf{x}), \tag{8}$$

where F is a function depending on $(\tilde{u}, t; \mathbf{x})$ defined as

$$F(\tilde{u}, t; \mathbf{x}) = \sum_{i=1}^{d} \partial_i \Big(\sum_{j=1}^{d} a_{ij} \partial_j u(t, \mathbf{x}) \Big) + \sum_{i=1}^{d} b_i \partial_i u(t, \mathbf{x}) + c(u(t, \mathbf{x})) - f(t, \mathbf{x}). \tag{9}$$

Note that although F is not observable due to the unknown partial derivative $\partial_i u(t, \mathbf{x})$, it can be universally approximated by a neural network provided that F is continuous. This observation motivates us to use the NODE model [18] for a primal solution $\tilde{u}$ for each given $\mathbf{x}$.

XNODE model. Similar to the NODE model, we will employ a deep neural network $\mathcal{N}^{\text{vec}}$ to approximate the vector field F in (8). To construct a universal model for the primal solution $\tilde{u}$, we suggest in (8) that the spatial variable $\mathbf{x}$ should be incorporated in the vector field $\mathcal{N}^{\text{vec}}$ as the additional dependent variable. More specifically, the XNODE model maps an arbitrary input $\mathbf{x} \in \mathbb{R}^d$ to the output $o_{\mathbf{x}}(t)_{t\in[0,T]} \in \mathcal{C}([0, T], \mathbb{R})$ through the $\mathbb{R}^h$-valued latent process $\mathbf{h} = (\mathbf{h}(t))_{t\in[0,T]}$ which is defined by the output of the NODE model with the $\mathbf{x}$-dependent vector field. More precisely, for every $t \in [0, T]$, we have

$$\begin{cases} \frac{\mathrm{d}\mathbf{h}(t)}{\mathrm{d}t} = \mathcal{N}_{\theta_2}^{\text{vec}}(\mathbf{h}(t), t, \mathbf{x}), \quad \mathbf{h}(0) = \mathcal{N}_{\theta_1}^{\text{init}}(h(\mathbf{x})) \in \mathbb{R}^h, \\ o_{\mathbf{x}}(t) = \mathcal{L}_{\tilde{\theta}}(\mathbf{h}(t)), \end{cases}$$

where $\mathcal{N}_{\theta_2}^{\text{vec}}$ and $\mathcal{N}_{\theta_1}^{\text{init}}$ are neural networks fully parameterized by θ_1 and θ_2 for the vector fields and initial condition of the hidden neural $\mathbf{h}$ respectively, and $\mathcal{L}_{\tilde{\theta}}$ is a linear trainable layer with the weights $\tilde{\theta}$. We denote by $\Theta = (\theta_1, \theta_2, \tilde{\theta})$ the set of all trainable model parameters of the proposed XNODE model. We note that the output of XNODE o is a function, that lives in an infinite dimensional space. However in practice, the model output has to be finite dimensional for numerical computation. Hence given any time partition $\Pi_T = (t_i)_{i=1}^{n_T}$ [3] of $[0, T]$, the corresponding output of XNODE is given by $\text{XNODE}_\Theta(\mathbf{x}, \Pi_T) := o_{\mathbf{x}}(\Pi_T)$.

In essence, our proposed XNODE model is built upon the NODE model, though it incorporates an additional spatial variable to model the high dimensional latent process of the PDE solution $\tilde{u}$. The uniqueness of XNODE follows from that of NODE, by considering $\mathbf{x}$ as the additional model parameter. Similarly, the adjoint sensitivity of XNODE is inherited from NODE, allows us to compute gradients with low memory cost. We also establish the universal approximation property of XNODE in Theorem A.2 in the Appendix.

To effectively use the initial condition of the PDE $u(0, \mathbf{x}) = h(\mathbf{x})$, the hidden neuron at the initial time $\mathbf{h}(0)$ is given by a neural network $\mathcal{N}^{\text{init}}$ of $h(\mathbf{x})$ from d-dimensional input $\mathbf{x}$ to 1-dimensional input $h(x)$, without compromising any discriminate capacity of the model. As long as $\mathcal{L}_{\tilde{\theta}} \circ \mathcal{N}^{\text{init}}$ can approximate the identity map, which is guaranteed by the universality of the neural network, it is able to achieve the satisfactory fitting performance of initial condition.

Point sampling. At each iteration as in the WAN Algorithm 1, to compute $\tilde{L}_{\text{int}}, \tilde{L}_{\text{init}}$ and $\tilde{L}_{\text{bdry}}$ defined in (5), one needs to sample points $\{(t_i, \mathbf{x}_i)\}_i$ uniformly on the interior and the boundary of the domain $[0, T] \times \Omega$ respectively and estimate the solution u evaluated at the grid. Recall that the output of the $\text{XNODE}_\Theta(\mathbf{x}, \Pi_T)$ is to approximate the solution u evaluated at the points $\Pi_T \times \{\mathbf{x}\}$. Therefore, unlike the sampling strategy of WAN, XNODE only requires sampling independently and uniformly on both time domain and spatial domain. Let us denote the sampled time partition and collocation points of the spatial domain Ω and domain boundary $\partial\Omega$ by Π_T, S^r and S^b respectively, with $S^r := \{\mathbf{x}_j^r \in \Omega\}_{j=1}^{N_r}$ and $S^b := \{\mathbf{x}_b^r \in \partial\Omega\}_{j=1}^{N_b}$. Note the proposed sampling method (e.g. see the solid red points in Fig. 1a for an illustration) implicitly generates a

[3] Π_T is a time partition of $[0, T]$ if and only if $\Pi_T = \left\{ (t_i)_1^{n_T} \middle| 0 = t_1 < t_2 < \cdots < t_{n_T} = T \right\}$.

uniformly sampled grid of $[0, T] \times \Omega$ in a more economical way by recycling the uniformly sampled spatial points at each sampled time.[4]

Weak solution prediction. For any point $\mathbf{x} \in \Omega$, we can construct the corresponding *constant spatial path* (i.e. $X(t) \equiv \mathbf{x}, \forall t \in [0, T]$, e.g. see a solid line in Fig. 1a). The output $\text{XNODE}_\Theta(\mathbf{x}, \Pi_T)$ then provides to a discrete approximation of the weak solution u evaluated at $\{(\mathbf{x}, t_i)\}_{t_i \in \Pi_T}$ along this constant spatial path. By repeating this procedure on each point $\mathbf{x}_i^r \in S^r$, we can generate a collection of constant paths $(X_i^r)_{i=1}^{N_r}$ such that $X_i^r \equiv \mathbf{x}_i^r$. Similarly we generate a collection of constant paths $(X_i^b)_{i=1}^{N_b}$. In total, we obtain the corresponding $(\text{XNODE}_\Theta(X_i, \Pi_T))_{\mathbf{x}_i \in S^r \cup S^b}$ to approximate the solution u over the grid $\Pi_T \times (S^r \cup S^b)$. The algorithm of using NODE model to approximate the weak solution network is given in Algorithm 2.

Algorithm 2 XNODE Network for modeling the weak solution u on a constant domain $[0, T] \times \Omega$.

Input: sampled sets of points $S^r = \{\mathbf{x}_j^r \in \Omega\}_{j=1}^{N_r}$ and $S^b = \{\mathbf{x}_j^b \in \partial\Omega\}_{j=1}^{N_b}$; $\Pi_T = (t_i)_{i=0}^{n_T}$ time partition of $[0, T]$; XNODE parameter set $\Theta = (\theta_1, \theta_2, \tilde{\theta})$
Output: O ▷ approximate u at $S^r \times \Pi_T$ and $S^b \times \Pi_T$
Algorithm:
1: initialize O as an empty vector
2: **for** $\mathbf{x}$ in $S^r \cup S^b$ **do**
3: compute $o_{\mathbf{x}} = \text{XNODE}_\Theta(\mathbf{x}, \Pi_T)$; ▷ approximate u at $\{x\} \times \Pi_T$
4: set $O = \text{concatenate}(O, o_{\mathbf{x}})$;
5: **end for**
6: **return** output O

XNODE-WAN algorithm. Finally, we obtain the XNODE-WAN model by replacing the DNN network by the XNODE network for the primal solution in the WAN algorithm (see Algorithm 1). In particular, we adapt the WAN algorithm in the following steps:

- In line 1, the DNN network for the primary solution u_θ is replaced by the XNODE network.
- In line 2, the point sampling strategy is updated as described before. For each iteration, our algorithm implements such sampling to generate a set of random collocation data points over the domain.
- In line 6 and line 12, the adjoint method is used to compute the derivative of XNODE model instead of the gradient descent method of DNN network in WAN.

The full XNODE-WAN algorithm is outlined in Algorithm 5 in Appendix.

3.2.2. Time-dependent domain

In this subsection, we consider a more general case $\mathcal{D} \subset [0, T] \times \mathbb{R}^d$ which allows that the spatial domain is time-dependent. Let $\Omega(t)$ denote the spatial domain of $\mathcal{D}$ restricted to any $t \in [0, T]$, i.e. $\Omega(t) = \{\mathbf{x} | (t, \mathbf{x}) \in \mathcal{D}\}$. When $\Omega(t)$ changes over $t \in [0, T]$, we call $\mathcal{D}$ a time-varying domain. The proposed XNODE model in Section 3.2.1 can not be directly applied to a time varying domain as the constant path $X \equiv \mathbf{x} \in \Omega(0)$ might go outside the domain $\mathcal{D}$ at some time $s \in [0, T]$, i.e. $(s, \mathbf{x}) \notin \mathcal{D}$ (see e.g., the sample point marked in $\circ$ in Fig. 1b).

To circumvent the problem, we shall divide each constant path $X \equiv \mathbf{x}$ into multiple sub-paths and only keep the sub-paths remaining within the domain $\mathcal{D}$. To be more specific, we firstly define the entry and exit points on a constant path X.

Definition 3.1. Let $\mathcal{D}$ be a bounded domain in $[0, T] \times \mathbb{R}^d$ and fix $\mathbf{x} \in \mathbb{R}^d$. Let X denote a constant path taking value in $\mathbf{x}$, i.e. $X(t) \equiv \mathbf{x}, \forall t \in [0, T]$. For $i = 1, 2, \cdots$, the i^{th} entry point and exit point of the constant path X, denoted by $\underline{\tau}_X^{(i)}$ and $\overline{\tau}_X^{(i)}$ are defined as follows:

$$\underline{\tau}_X^{(1)} = \inf_{t \in [0,T]} \{t | (t, \mathbf{x}) \in \mathcal{D}\};$$
$$\overline{\tau}_X^{(i)} = \inf_{t \in [0,T]} \{t | (t, \mathbf{x}) \notin \mathcal{D}, t \geq \underline{\tau}_X^{(i)}\}; \quad \underline{\tau}_X^{(i+1)} = \inf_{t \in [0,T]} \{t | (t, \mathbf{x}) \in \mathcal{D}, t \geq \overline{\tau}_X^{(i)}\},$$

where $\inf \emptyset = +\infty$ by convention. Let $N_{\mathbf{x}}$ denote the total number of finite $\overline{\tau}_X^{(i)}$, i.e.

$$N_{\mathbf{x}} = \sup_{i \in \{1,2,\cdots\}} \{i | \overline{\tau}_X^{(i)} < +\infty\},$$

where $\sup \emptyset = -\infty$ by convention.

[4] The collocation of points on the interior (resp. boundary) of the domain $[0, T] \times \Omega$ (resp. $[0, T] \times \partial\Omega$) is given by $\Pi_T \times S^r$ (resp. $\Pi_T \times S^b$). To generate $N_T \times N_r$ points over the interior of the domain $[0, T] \times \Omega$, the number of random points generated by the WAN is $N_T \times (N_r + N_b)$ while that of our point sampling strategy is $N_T + N_r + N_b$.

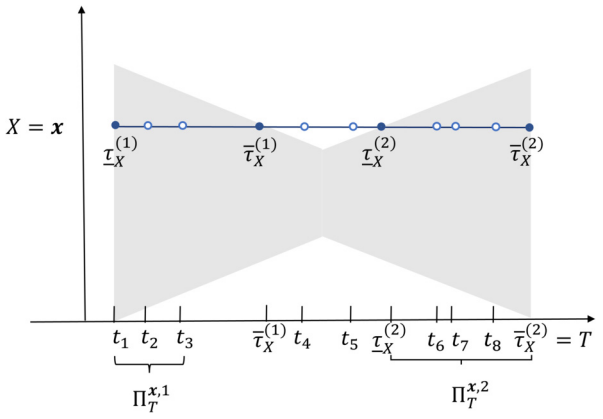


Fig. 2. An illustration for entry and exit points of a constant path (see Definition 3.1) and the corresponding subpaths $\Pi_T^{\mathbf{x},1}$ and $\Pi_T^{\mathbf{x},2}$ for the given domain presented as a shaded region. The solid line presents the constant path $X \equiv \mathbf{x}$. The time coordinates of the blue solid dots are the entry and exit points, while the blue circles represent $\mathbf{x} \times \Pi_T$ the samples along the path. Clearly there are two points $(t_4, \mathbf{x})$ and $(t_5, \mathbf{x})$ along the path but not in the shaded domain, which are filtered out when constructing sub-paths within the domain.

An illustration example is given in Fig. 2, with the blue solid dots along the blue lines indicating where the entry and exit points locate for the given constant path X.

For the ease of notation, when there is no ambiguity, we omit the subscript X in $\underline{\tau}_X^{(i)}$ and $\overline{\tau}_X^{(i)}$ for the rest of the paper. When restricting the constant path $X \equiv \mathbf{x}$ to the time interval from the i^{th} entry point $\underline{\tau}^{(i)}$ to the i^{th} exit point $\overline{\tau}^{(i)}$, we obtain a constant sub-path denoted by $X^i = X_{[\underline{\tau}^{(i)}, \overline{\tau}^{(i)}]}$, whose time augmented path[5] takes value in the closure of $\mathcal{D}$ (e.g. see the solid red lines in Fig. 1b).

We assume that for the given domain $\mathcal{D}$, the entry and exit points of every constant path $X \equiv \mathbf{x}$ can be computed and $N_{\mathbf{x}}$ is finite. For any given time partition Π_T and the spatial point $\mathbf{x}$, we propose the following way to assign the collocation of time points from Π_T to each sub-path X^i for the XNODE model. We define a set $\Pi_T^{\mathbf{x},i}$, which is composed with the i^{th} entry starting point $\underline{\tau}^{(i)}$ and all the elements of Π_T belonging to the i^{th} time interval $[\underline{\tau}^{(i)}, \overline{\tau}^{(i)}]$ for $i \in \{1, \cdots, N_x\}$. Hence $\Pi_T^{\mathbf{x},i}$ is the time collocation points associated with the sub-path X^i. Fig. 2 presents an example of computing the entry and exit points and constructing time collection points of each constant subpaths.

Example 3.1. In Fig. 2, a shaded grey region represents the domain $\mathcal{D}$. For the given constant path $X = \mathbf{x}$ marked as the solid blue line, one can compute two pairs of finite entry and exit points, denoted by $\{(\underline{\tau}^{(i)}, \overline{\tau}^{(i)})\}_{i=1}^2$, which corresponds to two constant sub-paths within the domain $\mathcal{D}$. Let $\Pi_T = (t_i)_{i=1}^8$ denote a time partition of length 8 in Fig. 2. The time collocation points of the first and second sub-paths are $\Pi_T^{\mathbf{x},1} = (\underline{\tau}_1, t_2, t_3)$ and $\Pi_T^{\mathbf{x},2} = (\underline{\tau}_X^{(2)}, t_6, t_7, t_8, \overline{\tau}_X^{(2)})$ respectively.[6]

Now we are ready to outline the proposed modification of Algorithm 2 for the time-varying domain as follows.

Point sampling. First of all, we find the minimum time-independent domain to cover $\mathcal{D}$, i.e. $\tilde{D} = [0, T] \times \Omega_{\max}$, where $\Omega_{\max} = \cup_{t \in [0,T]} \Omega(t)$. We adopt the point sampling method in Section 3.2.1 to generate the uniform grid to cover $[0, T] \times \Omega_{\max}$, i.e. $\Pi_T \times S^r$, where S^r is sampled uniformly from $\Omega_{\max}$. Secondly, for every $\mathbf{x} \in S^r$, we compute $\{(\underline{\tau}^{(i)}, \overline{\tau}^{(i)})\}_{i=1}^{N_{\mathbf{x}}}$. We then construct $\mathbf{x}$-valued constant sub-paths and determine the time collocations points associated with each sub-path as described above. The algorithm of computing $\Pi_T^{\mathbf{x},i}$ is given in Algorithm 3. To sample of the boundary points of the general domain $\mathcal{D}$, we use the same method of WAN [17], i.e. uniformly sample a number of points from the boundary $\partial\mathcal{D}$.

Weak solution prediction. For each $\mathbf{x} \in S^r$, we have $N_{\mathbf{x}}$ many constant sub-paths defined on the sub-time interval $[\underline{\tau}^i, \overline{\tau}^i]$ with corresponding discrete time collocation points $\Pi_T^{\mathbf{x},i}$. Therefore we can estimate the weak solution u evaluated at $\{\mathbf{x}\} \times \Pi_{\mathbf{x},i}^T$ by XNODE$(\mathbf{x}, \Pi_T^{\mathbf{x},i})$ for $i \in \{1, \cdots N_{\mathbf{x}}\}$. Note that the initial condition of XNODE model for each constant sub-path X^i is well defined as $u(\underline{\tau}_i, \mathbf{x})$ is given by either the initial condition or boundary condition, i.e.

$$\mathbf{h}(\underline{\tau}^{(i)}) = \begin{cases} \mathcal{N}_{\theta_1}^{\text{init}}(h(\mathbf{x})), & \text{if } i = 0 \text{ and } \underline{\tau}^{(i)} = 0; \\ \mathcal{N}_{\theta_1}^{\text{init}}(g(\mathbf{x}, \underline{\tau}^{(i)})), & \text{otherwise.} \end{cases} \tag{10}$$

We now present the modified algorithm of XNODE for the time-dependent domain in Algorithm 4, on top of Algorithm 3. Note training the XNODE-WAN network relies on the ODE solver to update the model parameters of the XNODE model. The

[5] Let $f : [0, T] \to \mathbb{R}^d$. The time augmented path of f is defined as a map from $[0, T] \to [0, T] \times \mathbb{R}^d : t \mapsto (t, f(t))$.

[6] Note $t_1 = \underline{\tau}_X^{(1)}$ and hence $\Pi_T^{\mathbf{x},1} = (\underline{\tau}^{(1)}) \cup (t_1, t_2, t_3) = (\underline{\tau}^{(1)}, t_2, t_3)$.

Algorithm 3 Compute the constant sub-paths and the corresponding time collocation points.

Input: domain $\mathcal{D}$, $\mathbf{x} \in \mathbb{R}^d$, Π_T
Output: $(\Pi_T^{\mathbf{x},i})_{i=1}^{N_\mathbf{x}}$
Algorithm:
compute the entry and exit points $(\underline{\tau}^{(i)}, \overline{\tau}^{(i)})_{i=1}^{N_\mathbf{x}}$
for i from 1 to $N_\mathbf{x}$ **do**
  initialize $\Pi_T^{\mathbf{x},i}$ as an empty list;
  add $\underline{\tau}^{(i)}$ to $\Pi_T^{\mathbf{x},i}$;
  for t in Π_T and t in $(\underline{\tau}^{(i)}, \overline{\tau}^{(i)}]$ **do**
    add t to $\Pi_T^{\mathbf{x},i}$;
  end for
end for

Algorithm 4 XNODE Network for modeling the weak solution u on a time-dependent domain $\mathcal{D}$.

Input: sampled sets of points $S^r = \{\mathbf{x}_j^r \in \Omega_{\max}\}_{j=1}^{N_r}$ and $\Pi_T = (t_i)_{i=0}^{n_T}$ time partition of $[0, T]$; XNODE parameter set $\Theta = (\theta_1, \theta_2, \tilde{\theta})$
Output: O ▷ approximate u at $S^r \times \Pi_T$
Algorithm:
initialize O as an empty vector
for $\mathbf{x}$ in S^r **do**
  compute $(\Pi_T^{\mathbf{x},i})_{i=1}^{N_\mathbf{x}}$ with the input arguments $\mathcal{D}$, x and Π_T through Algorithm 3;
  initialize $o_\mathbf{x}$ as a zero vector of length $N_\mathbf{x}$;
  for i from 1 to N_x **do**
    compute $o_{\mathbf{x},i} = \text{XNODE}_\Theta(\mathbf{x}, \Pi_{\mathbf{x},i})$; ▷ approximate u at $\{\mathbf{x}\} \times \Pi_{\mathbf{x},i}^T$
    add $o_{\mathbf{x},i}$ to $o_\mathbf{x}$ in the corresponding locations based on $\Pi_{\mathbf{x},i}^T$;
  end for
  set $O = \text{concatenate}(O, o_\mathbf{x})$;
end for
return output O

supplementary code implementation of the proposed XNODE-WAN method is flexible enough to incorporate any ODE solver (e.g., Euler method or higher-order Runge-Kutta method). The choice of a suitable ODE solver used in XNODE-WAN can be determined based on cross-validation numerical results.

Lastly, we highlight that the proposed XNODE-WAN is intrinsically different from classical numerical methods (e.g., Euler or other time advanced methods) of the parabolic PDEs in the following two main aspects. First, the XNODE network is a family of *universal* models for continuous spatial-temporal functionals, which does not depend on the specific parabolic PDE. However, the estimated solution by classical numerical methods is specific to the given parabolic PDEs and computed directly through discretizing the PDE in the temporal domain. Second, in stark contrast to the classical numerical methods, the vector field of the XNODE model is optimized and updated iteratively during the training process. Hence, the XNODE model is trained by optimizing the loss function, which incorporates the PDE information.

4. Numerical results

In this section, we conduct several numerical experiments of parabolic PDEs defined on time independent domains or time varying domains, respectively. We compare the performance of the proposed XNODE-WAN algorithms (Algorithm 2 and Algorithm 4) with the baseline WAN algorithm [17]. In particular, following the numerical example in [17], we consider PDE examples in the form of a d-dimensional nonlinear diffusion-reaction equation (Eq. (11)) defined on a bounded domain $\mathcal{D} \subset [0, 1] \times \mathbb{R}^d$:

$$\begin{cases} \partial_t u - \triangle u - u^2 - f = 0 & \text{for } (t, \mathbf{x}) \in \mathcal{D} \\ u - g = 0 & \text{on } \partial\mathcal{D} \\ u(0, \mathbf{x}) - h(\mathbf{x}) = 0 & \text{on } \Omega(0), \end{cases} \tag{11}$$

where f, g and h may vary from example to example. In Section 4.2, we consider PDE (11) on two time independent domains $\mathcal{D} = [0, 1] \times \Omega$ where Ω is a hyper cube or a unit ball. Note our example where Ω is a hyper cube is the same as that in Section 4.2.6 of [17]. We then investigate the scalability of our proposed method with respect to high spatial dimension in Section 4.3 and follow by a sensitivity analysis in Section 4.4. At the end of this section, we provide an example of a time-varying domain. Numerical implementation of our work is available at https://github.com/paulvoliva/XNODE-WAN-PDE-solver.git.

4.1. Experimental setup

To quantitatively access the accuracy and efficiency of the method, we consider the following test metrics:

Table 1
Performance comparison between XNODE-WAN and WAN on Example 4.1.

	Relative error (± SE)	Time per epoch (s)	N_ε	T_ε (s)
WAN	2.6% ± 0.0144%	0.38	15,463	5865
XNODE-WAN	1.7% ± 0.0114%	0.35	211	74

1. The relative error: $||u_\theta - u||_{L^2}/||u||_{L^2}$, where u_θ and u are the approximation and exact solution of the problem with $||u||_{L^2} = \left(\int_0^T \int_{\Omega(t)} |u(t,\mathbf{x})|^2 \mathrm{d}\mathbf{x}\mathrm{d}t\right)^{\frac{1}{2}}$.
2. Time per epoch: the average time elapsed in each epoch in the training process.
3. N_ϵ: for any error tolerance level $\epsilon > 0$, the minimum number of epochs such that the relative error of the trained model is no more than ϵ.
4. T_ϵ: given any error tolerance level $\epsilon > 0$, the minimum of time (seconds) such that the relative error of the trained model is no more than ϵ; we would also refer to it the time at convergence.

For a fair comparison, we choose the following hyperparameters, which are the same for both WAN and XNODE-WAN: $N_r = N_b = 400$, $n_T = 20$, $K_u = 2$, $K_\phi = 1$, $\alpha = \gamma = 400,000 \times d^2$ and the learning rate of the adversarial network $\tau_\eta = 0.04$ (see Table B.2 for the summary of notations of variables used in models and algorithms). Note that we set $N_{\mathcal{D}} = N_r * n_T$ and $N_{\partial\mathcal{D}} = N_b * n_T$ for WAN to have a fair comparison. Besides we keep the same network architecture of the adversarial network (test function ϕ_η) the same for both WAN and XNODE-WAN. However, as the training performance of each method may be sensitive to the learning rate, the learning rate of the primal network should be adjusted to the XNODE or DNN separately. By hyper-parameter tuning, the learning rate of the XNODE-WAN and WAN for the primal network are chosen to be 0.015 and 0.00005 respectively. Note when increasing the learning rate of the WAN algorithm from 0.00005, the WAN may suffer from either divergence or slower training. One may refer the effects of the learning rate in Section 4.4. We use the above hyper-parameter setting for all the following numerical examples. In addition, here we use the explicit midpoint method as the ODE solver of the XNODE-WAN model.

4.2. Time-independent domains

In this subsection, we investigate the performance of the proposed method on a time-independent domain $\mathcal{D} = [0,1] \times \Omega$. To demonstrate that our method is applicable to a general spatial domain Ω, we consider two examples of Ω, i.e. (1) a hyper-cube $\Omega = [0,1]^d$ and (2) a unit ball $\Omega = \{x \in \mathbb{R}^d | |x| \leq 1\}$ for $d = 5$.

Example 4.1. We consider the same example in Section 4.2.6 in [17] with f, g, h specified as follows:

$$\begin{cases} f(t,\mathbf{x}) = (\pi^2 - 2)\sin\left(\frac{\pi}{2}\right)\cos\left(\frac{\pi}{2}e^{-t}\right) - 4\sin^2(\frac{\pi}{2}x_1)\cos(\frac{\pi}{2}x_2)e^{-2t}; \\ g(t,\mathbf{x}) = 2\sin(\frac{\pi}{2}x_1)\cos(\frac{\pi}{2}x_2)e^{-t}; \\ h(\mathbf{x}) = 2\sin(\frac{\pi}{2}x_1)\cos(\frac{\pi}{2}x_2), \end{cases} \tag{12}$$

where $\Omega = [0,1]^5$. The exact solution is given by

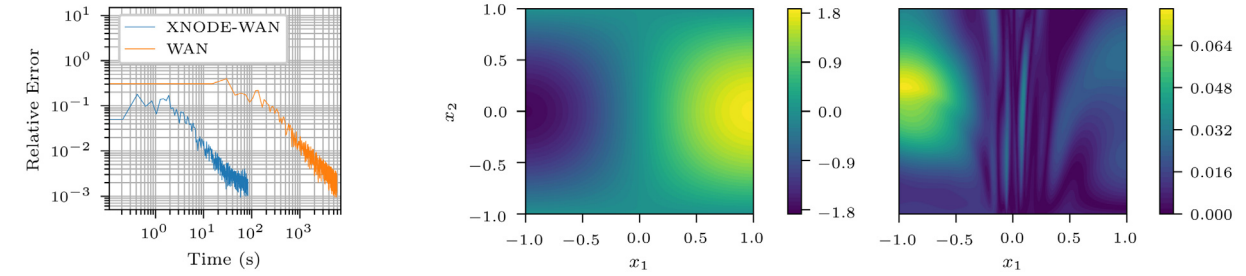
$$u(t,\mathbf{x}) = 2\sin\left(\frac{\pi}{2}x_1\right)\cos\left(\frac{\pi}{2}x_2\right)e^{-t}. \tag{13}$$

The stopping criteria for both algorithms is set to achieve a relative error tolerance level $\epsilon = 10^{-3}$. We also randomly test 50 trials on randomly selected data to measure the relative L^2 error on the obtained models. We report the mean and the standard deviation of relative error of the XNODE-WAN and WAN in Table C.3, which demonstrates that XNODE-WAN model significantly outperforms in terms of computational time and iteration number. In particular, to reach the same stopping criteria, XNODE-WAN model saves 99.8% in time and only takes less than 1.5 minutes for training in this case. Furthermore, XNODE-WAN and WAN need 165 and 15,463 iterations respectively whereas the time used for each iteration is comparable. In addition, Fig. 3a gives the evolution of relative errors over time for both models. Clearly the proposed model converges much faster than that of WAN. See Table 1.

Fig. 3b and Fig. 3d demonstrate the comparison between the XNODE-WAN and WAN estimator for the PDE solution $u(T,\mathbf{x})$ at the convergence and their respective absolute point-wise error on the (x_1, x_2) domain. It shows that the out-of-sample point-wise absolution error of our method has less magnitude than that of WAN. The superior performance of our method over WAN is also verified by a smaller relative error (1.7%±0.0114% v.s. 2.6%±0.0144%) with statistical significance.

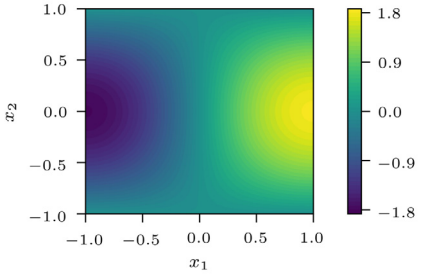
Finally, in Fig. 3e, we show that when XNODE-WAN reaches the stopping criteria at 74 s, the WAN solution is far from being close to the true solution with large error.

Example 4.2. We consider the same problem as in Example 4.1 but on a 5-dimensional hypersphere domain, i.e., $\mathcal{D} = [0,1] \times B(\mathbf{x}_c, 0.5)$, where $\mathbf{x}_c = [0.5, 0.5, 0.5, 0.5, 0.5]^T$. The exact solution is given in (13) on $\mathcal{D}$.

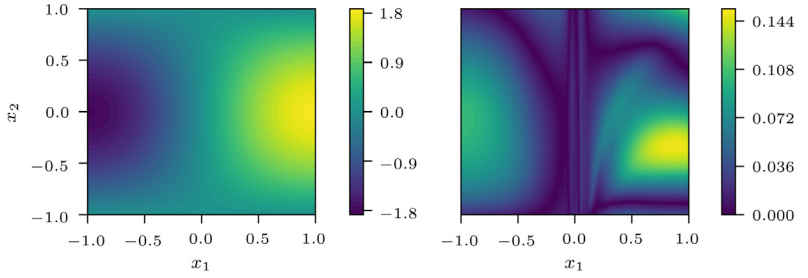


(a) The relative error over time

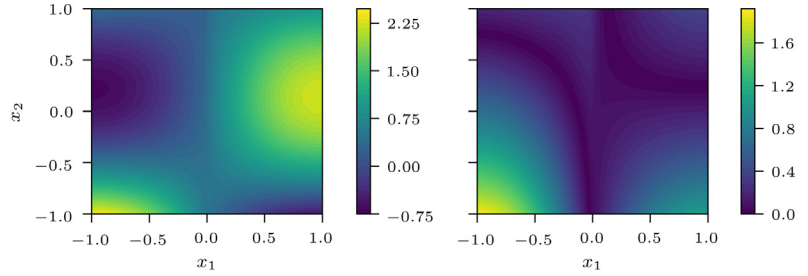
(b) XNODE-WAN solution and its error at convergence (74s)



(c) True solution $u(T, \mathbf{x})$



(d) WAN solution and its error at convergence (5865s)



(e) WAN solution and its error at 74s

Fig. 3. Performance comparison between XNODE-WAN and WAN in Example 4.1. In the right subplots of (b), (d) and (e), the error is the pointwise absolute error $|u(T, \mathbf{x}) - \hat{u}(T, \mathbf{x})|$, where $\hat{u}$ is the model estimated solution. All the images (b)-(e) only show 2D slice of $x_3 = x_4 = x_5 = 0$ as the true solution u of this example only depends on the first two coordinates.

We again implement both models and the results is shown in Fig. 4 with the same stopping criteria, i.e., the relative L^2 error $\epsilon = 10^{-3}$. The evolution of the relative L^2 error for the hypersphere domain is similar to that in Example 4.1. In particular, the XNODE-WAN model achieves the targeted error tolerance within 100 s while it takes longer than 3,000 s for WAN to reach the stopping criteria.

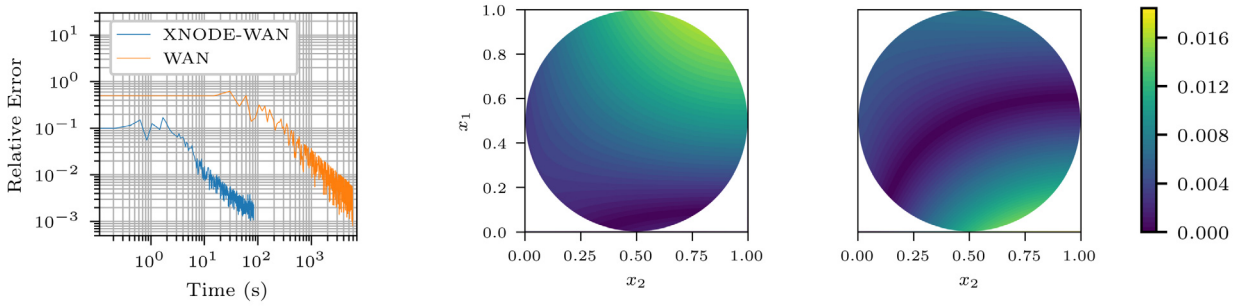
This example justifies the advantage of using XNODE-WAN for solving parabolic PDE with an irregular spatial domain Ω.

4.3. Scalability analysis

In this subsection, we investigate the scalability of the proposed XNODE-WAN model with respect to the spatial dimension.

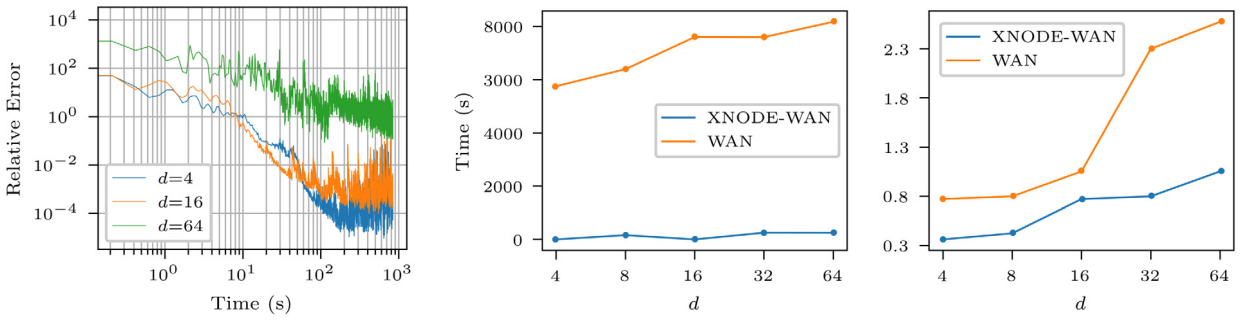
Example 4.3. We consider Eq. (11) on different spatial dimension $d \in \{4, 8, 16, 32, 64\}$, where f, g are defined on $[0, 1] \times \Omega$ and h are defined on Ω as follows:

$$\begin{cases} f(t, \mathbf{x}) = \frac{2e}{1-e} \left(\frac{\pi}{2}\right)^d \left(\left(\frac{\pi}{2} - 2\right) e^{-t} \Pi_{i=1}^d \sin\left(\frac{\pi}{2} x_i + \frac{\pi i}{2}\right) - 4e^{-2t} \Pi_{i=1}^d \sin^2\left(\frac{\pi}{2} x_i + \frac{\pi i}{2}\right) \right), \\ g(t, \mathbf{x}) = \frac{2e}{1-e} \left(\frac{\pi}{2}\right)^d 2e^{-t} \Pi_{i=1}^d \sin\left(\frac{\pi}{2} x_i + \frac{\pi i}{2}\right), \\ h(\mathbf{x}) = \frac{2e}{1-e} \left(\frac{\pi}{2}\right)^d 2 \Pi_{i=1}^d \sin\left(\frac{\pi}{2} x_i + \frac{\pi i}{2}\right). \end{cases} \tag{14}$$



(a) The relative error over time (b) Error at convergence for XNODE-WAN (left) and WAN (right)

Fig. 4. Performance comparison of XNODE-WAN and WAN in Example 4.2, where Ω is a 5 dimensional unit ball. (a) the evolution of relative L^2 error of both XNODE-WAN and WAN method over time; (b) The plot of pointwise absolute error $|u(T,\mathbf{x}) - \hat{u}(T,\mathbf{x})|$, where $\hat{u}$ is the trained model of each method until hitting the stopping criteria. All the images only show 2D slice of $x_3 = x_4 = x_5 = 0$ as the true solution u of this example only depends on the first two coordinates.



(a) The relative error over time (b) Total computation time and training time per epoch

Fig. 5. Scalability results of Example 4.3 in terms of spatial dimension d. (a) The evolution of relative L^2 error of the XNODE-WAN method over time (0 to 10^3 s) for $d \in \{4, 16, 64\}$; (b) the computational time costs (Left) and average training time per epoch (Right) of XNODE-WAN and WAN until they achieve error tolerance respectively. Here we use $\epsilon = 10^{-2}$ for each spatial dimension $d \in \{4, 8, 16, 32, 64\}$.

It is easy to verify that the solution to (11) is given by

$$u(t,\mathbf{x}) = \left(\frac{\pi}{2}\right)^d 2e^{-t}\Pi_{i=1}^d \sin\left(\frac{\pi}{2}x_i + \frac{\pi i}{2}\right).$$

Note in Example 4.3, $||u||_{L^2} = 1$ regardless of the problem dimension d. For both methods, we choose the number of points $N_r, N_b = 800d$ and $\epsilon = 10^{-2}$.

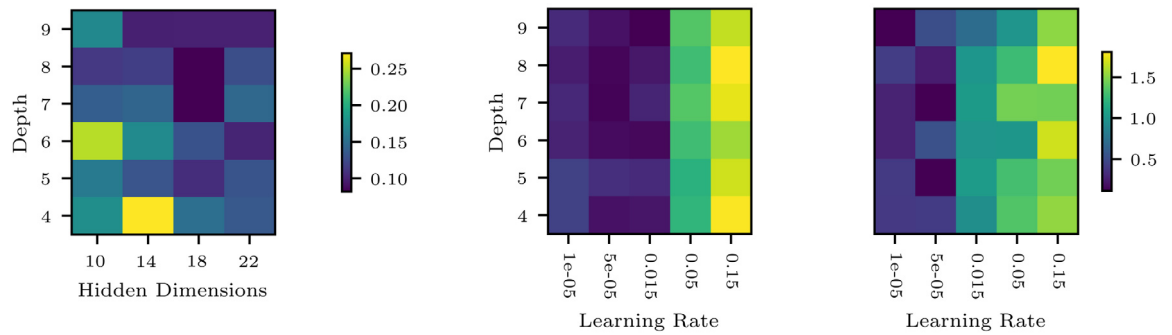
We investigate the performance of our proposed XNODE-WAN method with different spatial dimension d in terms of the relative error evolution over the training, total training time and training time per epoch, which is visualized in Fig. 5. Fig. 5a shows that even for a high dimensional $d = 64$, our method is able to converge to the relative error tolerance $\epsilon = 10^{-2}$ within 10^2 seconds.

Shown in Fig. 5b, for a dimension as high as 64, to achieve the error tolerance 10^{-2}, the training time of WAN is 4,000 s, roughly 40 times of that of XNODE-WAN. It appears that, as N_r and N_b are chosen proportional to d, the average training time increases approximately linearly in dimension d for XNODE-WAN. Clearly for the same dimension d, the training time per epoch from WAN is longer than that from XNODE-WAN, especially for large values of d. In total, Fig. 5 shows that XNODE-WAN has greater potential in scalability for higher dimensional PDEs.

4.4. Sensitivity analysis

In this subsection, we aim to investigate the sensitivity of our proposed XNODE model in terms of the network architecture of the hidden field $\mathcal{N}_{\theta_2}^{\text{vec}}$ and learning rate τ_θ. Here we choose the same PDE setting defined in Section 4.3 and set $d = 5$. We follow [17] to choose the error tolerance level $\epsilon = 0.005$.

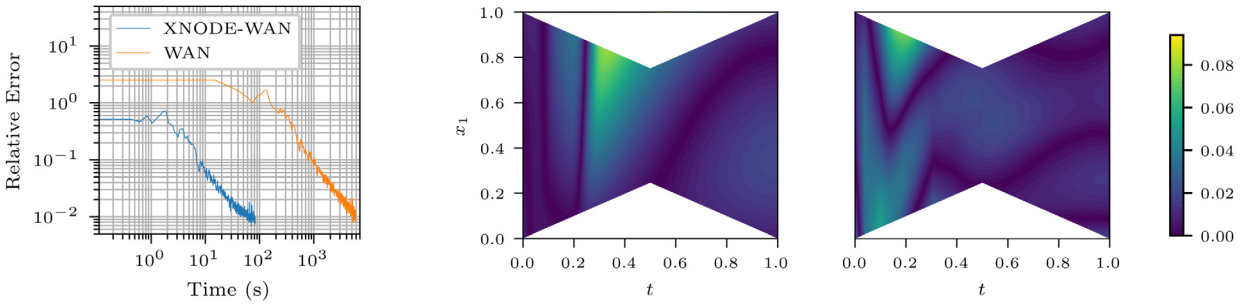
To investigate the effects of the neural network architecture of the hidden field $\mathcal{N}_{\theta_2}^{\text{vec}}$ on the predictive performance of the XNODE-WAN, we consider different number of neurons per layer (hidden dimension) and number of layers (depth). When the hidden dimension is high as 18 or 22, our method can achieve the satisfactory results. It can be observed, when



(a) XNODE-WAN: depths against hidden dimensions

(b) XNODE-WAN (right) and WAN (left): the heatmap of relative error against the network depth and learning rate

Fig. 6. The relative error plots for sensitivity analysis. All these experiments are implemented under the same setting of hyperparameters as in the Section 4.3.



(a) The relative error over time

(b) Error at convergence for XNODE-WAN and WAN solutions

Fig. 7. Performance comparison of XNODE-WAN and WAN on an time-dependent domain in Example 4.4. (a) The evolution of relative L^2 error of both XNODE-WAN and WAN method over time; (b) The plot of pointwise absolute error $|u(t, x_1) - \hat{u}(t, x_1)|$, where $\hat{u}$ is the trained model of each method until hitting the stopping criteria.

increasing the depth to 7 or above, the predictive performance of our method is improving. It suggests that our method usually works well as long as the neural network architecture is sufficiently rich. However, the over-complicated network should be discouraged as it may prolong the training time and cause the overfitting.

Furthermore, we consider different values of learning rate for both our method and WAN, and the corresponding performance. Fig. 6b shows that, independently of the depth of the hidden field, a learning rate on the magnitude of 10^{-5} is needed to achieve a reasonable loss for WAN while XNODE-WAN allows for a much wider range of learning rate, i.e., from 10^{-5} to 10^{-2}. In practice, we conduct the grid search to select the optimal learning rate for each method, i.e. to achieve the satisfactory accuracy, the learning rate is chosen to be the one corresponding to the least training time. That is the justification of different learning rate used for XNODE-WAN and WAN.

4.5. Time-varying domain

Example 4.4. We now consider the PDE problem on the time-varying domain $\mathcal{D} \subset [0, 1] \times [-1, 1]$ defined as follows (see Fig. 7a for its shape),

$$\mathcal{D} = \{(t, x_1) | t \in [0, 0.5], |x_1 - 0.5| \leq 0.5(1 - t)\} \bigcup \{(t, x_1) | t \in [0.5, 1], |x_1 - 0.5| \leq 0.5t\}.$$

Note that to illustrate the idea in this example we only consider $d = 1$.

In this example, we choose the error tolerance level $\epsilon = 10^{-2}$ to train both XNODE-WAN and WAN models. The performance comparison results are shown in Fig. 7. In particular, from Fig. 7b we observe that both models are able to learn the solution on the time-varying domain to a satisfactory level. The heatmap plot of Fig. 7b shows that they both achieve an error below 0.04 for most of the region at their convergence. However, as shown in Fig. 7a, similarly to the time-independent

case, the convergence of XNODE-WAN in terms of the relative error is much faster than that of WAN, which is reflected by the enlarged relative error gap between two models over time until the end of XNODE-WAN training.

To solve the PDE problem on the time-varying domain, both WAN and XNODE-WAN may require more computational time to train compared with their time-independent counterpart as expected. It is evident by that the computational cost of each model on low-dimensional time varying domain example (Example 4.4) is even higher than that for the higher dimensional time-invariant domain (see Fig. 3a). For our method, it is mainly a result of additional complexity of constant sub-path construction in Algorithm 4.

However, more importantly, for this time-varying domain example, our proposed model consistently outperforms in terms of efficiency: WAN requires significantly longer time (5,000 s) to converge compared to XNODE-WAN (less than 100 s) in Fig. 7a. In part this can be explained by the fact that XNODE-WAN model incorporates the boundary condition to the model (Eq. (10)) and its weak solution estimators evaluated at the entry points on the boundary have little error. This anchors our model at more places than the WAN model, making it more suited to a time-varying domain. This example justifies the advantage of using XNODE-WAN for solving PDE on an time-varying domain.

5. Conclusion

In this paper, we propose the novel XNODE model as a universal and effective network for the parabolic PDE solution by leveraging the priori information of the PDE though the NODEs. To tackle the high dimensional parabolic PDE problems on arbitrary bounded domain, incorporating the XNODE model to the WAN as a primal network leads to significantly faster training and better scalability. Our XNODE-WAN method consistently outperforms WAN in terms of computational time on both time-independent domain and time-dependent domain. However, quantitative analysis of the impact of the primal network architecture on the training efficiency of WAN for the parabolic PDE solution remains open and merits further investigation in future work.

CRediT authorship contribution statement

Paul Valsecchi Oliva: Conceptualization, Investigation, Methodology, Software, Writing – original draft, Writing – review & editing. **Yue Wu:** Conceptualization, Formal analysis, Methodology, Software, Validation, Writing – original draft, Writing – review & editing. **Cuiyu He:** Conceptualization, Formal analysis, Methodology, Validation, Writing – original draft, Writing – review & editing. **Hao Ni:** Conceptualization, Formal analysis, Methodology, Supervision, Validation, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

Funding: HN and YW are supported by the Engineering and Physical Sciences Research Council (EPSRC) [grant number EP/S026347/1] and the Alan Turing Institute under the EPSRC grant [grant number EP/N510129/1].

Appendix A. Universality of XNODE model

Let us recall the definition of XNODE model, which is a map from an input $\mathbf{x} \in \mathbb{R}^d$ to the output $o = (o_t)_{t\in[0,T]} \in \mathcal{C}([0,T],\mathbb{R})$ through the $\mathbb{R}^h$-valued latent process $\mathbf{h} = (\mathbf{h}(t))_{t\in[0,T]}$, i.e. for every $t \in [0,T]$,

$$\begin{cases} \frac{\mathrm{d}\mathbf{h}(t)}{\mathrm{d}t} = \mathcal{N}^{\mathrm{vec}}_{\theta_1}(\mathbf{h}(t), t, x), \quad \mathbf{h}(0) = \mathcal{N}^{\mathrm{init}}_{\theta_2}(h(x)) \in \mathbb{R}^h, \\ o_t = \mathcal{L}_{\tilde{\theta}}(\mathbf{h}(t)), \end{cases} \tag{A.1}$$

where $\mathcal{N}^{\mathrm{vec}}_{\theta_1}$ and $\mathcal{N}^{\mathrm{init}}_{\theta_2}$ are neural networks fully parameterized by θ_1 and θ_2 for the vector fields and initial condition of the hidden neural $\mathbf{h}$ respectively, and $\mathcal{L}_{\tilde{\theta}}$ is a linear trainable layer with the weights $\tilde{\theta}$.

We will establish the universal approximation theorem of the proposed XNODE model for the parabolic PDE solution. Our proof mainly replies on the stability estimates of the ODE theory.

Theorem A.1 *(Theorem 2.8 in [24]). Suppose $V_1, V_2 \in \mathcal{C}(U, \mathbb{R}^n)$ where U is an compact subset in $[0,T] \times \mathbb{R}^n$ and V_1 is Lipschitz continuous in the first argument with*

$$L = \sup_{\substack{(\mathbf{y}_1,t),(\mathbf{y}_2,t)\in U \\ \mathbf{y}_1\neq\mathbf{y}_2}} \frac{|V_1(\mathbf{y}_1,t) - V_1(\mathbf{y}_2,t)|}{|\mathbf{y}_1 - \mathbf{y}_2|}.$$

If F_1 and F_2 are the solutions to the two initial value problems defined as follows:

$$F_1'(t) = V_1\big(F_1(t), t\big), \quad F_1(0) = f_1; \tag{A.2}$$

$$F_2'(t) = V_2\big(F_2(t), t\big), \quad F_2(0) = f_2. \tag{A.3}$$

Then for all $t \in [0, T]$,

$$|F_1(t) - F_2(t)| \le |f_1 - f_2| \exp(Lt) + \frac{M}{L}\big(\exp(Lt) - 1\big),$$

where

$$M = \sup_{(\mathbf{y},t)\in U} |V_1(\mathbf{y}, t) - V_2(\mathbf{y}, t)|.$$

Thanks to the universality of neural network and the above theorem, we now establish the universality of the XNODE model in the following theorem.

Theorem A.2. *Let $\Omega \subset \mathbb{R}^d$ be a bounded domain and $F : [0, T] \times \Omega \to \mathbb{R}$ be a solution to the following ODE with the initial condition:*

$$\frac{\mathrm{d}F(t, \mathbf{x})}{\mathrm{d}t} = V(F(t, \mathbf{x}), t; \mathbf{x}), \quad F(0, \mathbf{x}) = h(\mathbf{x}), \tag{A.4}$$

where V is Lipschitz continuous in the first argument for any $\mathbf{x} \in \Omega$ and there exists a bounded constant $L > 0$ such that

$$L = \sup_{\mathbf{x}\in\Omega} \sup_{\substack{(\mathbf{y}_1,t),(\mathbf{y}_2,t)\in U \\ \mathbf{y}_1\neq\mathbf{y}_2}} \frac{|V(\mathbf{y}_1, t; \mathbf{x}) - V(\mathbf{y}_2, t; \mathbf{x})|}{|\mathbf{y}_1 - \mathbf{y}_2|}.$$

Then for any given error tolerance level $\epsilon > 0$, there exists a XNODE solution $\tilde{F}$ defined in (A.1) such that

$$\sup_{(t,\mathbf{x})\in[0,T]\times\Omega} |F(t, \mathbf{x}) - \tilde{F}(t, \mathbf{x})| \le \epsilon. \tag{A.5}$$

Proof. In this proof, we constrain the XNODE models to the ones which has the hidden dimension 1 and identity map being their output layer. Therefore $\tilde{F}$ satisfies the below ODE similar to (A.4):

$$\frac{\mathrm{d}\tilde{F}(t, x)}{\mathrm{d}t} = \mathcal{N}^{\mathrm{vec}}_{\theta_1}(\tilde{F}, t; x), \quad \tilde{F}(0, x) = \mathcal{N}^{\mathrm{init}}_{\theta_2}(h(x)), \tag{A.6}$$

where the activation functions (e.g. ReLu) are chosen such that the vector field is uniformly Lipschitz continuous. The wellposeness of the XNODE model is a directly consequence of Picard's Theorem. So is the function F due to the Lipschitz continuity assumption of its vector field F.

Thanks to the universality of neural network, for every $\epsilon > 0$, there exist two neural networks with sufficiently large number of hidden neurons and number of layers, denoted by $\mathcal{N}^{\mathrm{vec}}_{\theta_1}$ and $\mathcal{N}^{\mathrm{init}}_{\theta_2}$ such that

$$\sup_{\mathbf{y},t,\mathbf{x}} |\mathcal{N}^{\mathrm{vec}}_{\theta_1}(\mathbf{y}, t; \mathbf{x}) - V(\mathbf{y}, t; \mathbf{x})| + \sup_{\mathbf{x}} |\mathcal{N}^{\mathrm{init}}_{\theta_2}(h(\mathbf{x})) - h(\mathbf{x})| \le C^{-1}\epsilon,$$

where C can be pre-determined through

$$C = \exp(LT) + \exp(LT)/L.$$

The stability result (A.5) is then a direct consequence of Theorem A.1. □

Appendix B. Pseudocode of XNODE-WAN algorithm

We present in the Table B.2 the explanation for notations in algorithms. The following algorithm (Algorithm 5) is the full pseudocode of the XNODE-WAN algorithm for a time-independent domain.

Table B.2
List of algorithm parameters.

Notation	Meaning
$X_{\mathcal{D}}$	Sampled collocation points of the whole domain $\mathcal{D}$
$X_{\partial\mathcal{D}}$	Sampled collocation points on the domain boundary $\partial\mathcal{D}$
X_0	Sampled collocation points at the initial condition

(*continued on next page*)

Table B.2 (*continued*)

Notation	Meaning
S^r	Sampled collocation points of the spatial domain
S^b	Collocation points of the spatial domain boundary
Π_T	Sampled time partition
$N_{\mathcal{D}}$	Number of sampled collocation points of the space-time domain $\mathcal{D}$
$N_{\partial\mathcal{D}}$	Number of sampled collocation points of domain boundary $\partial\mathcal{D}$
N_0	Number of sampled collocation points of the initial condition
N_r	Number of sampled collocation points of the spatial domain
N_b	Number of sampled collocation points of the spatial domain boundary
n_T	Number of sampled time partition
K_u	Inner iteration to update weak solution u_θ or u_Θ
K_ϕ	Inner iteration to update test function ϕ_η
α	Weight parameter of boundary loss $\tilde{L}_{\text{bdry}}$
γ	Weight parameter of initial loss $\tilde{L}_{\text{init}}$
ϵ	error tolerance
τ_θ or τ_Θ	Learning rate for the primal network
τ_η	Learning rate for network parameter η of test function ϕ_η

Algorithm 5 XNODE-WAN algorithm for time-independent domains.

Input: domain $\mathcal{D} = \Omega \times [0, T]$, tolerance $\epsilon > 0$, $N_r/N_b/n_T$: number of sampled points on spatial domain/spatial boundary/temporal domain, K_u/K_ϕ: number of solution/adversarial network parameter update per iteration, $\alpha/\gamma \in \mathbb{R}^+$: the weight of boundary/initial loss

Initialize: the XNODE network $\phi_\Theta : \mathcal{D} \to \mathbb{R}$ and the DNN network $\phi_\eta : \mathcal{D} \to \mathbb{R}$
generate the sets of collection points S^r, S^b and Π_T ▷ random point sampling step
while $\tilde{L}(\Theta, \eta) > \epsilon$ **do**
  `# update weak solution network parameter`
  for $k = 1, ..., K_u$ **do**
    compute $\nabla_\theta \tilde{L}(\Theta, \eta)$; ▷ gradient calculation step
    update $\Theta \leftarrow \Theta - \tau_\Theta \nabla_\Theta \tilde{L}(\Theta, \eta)$;
  end for
  `# update test network parameter`
  for $k = 1, ..., K_\phi$ **do**
    compute $\nabla_\eta \tilde{L}(\Theta, \eta)$ using the points X^r;
    update $\eta \leftarrow \eta + \tau_\eta \nabla_\eta \tilde{L}(\Theta, \eta)$;
  end for
  generate the sets of collocation points S^r, S^b and Π_T; ▷ random point sampling step
end while
Output: $\{\text{XNODE}_\Theta(\mathbf{x}_i, \Pi_T)\}_{\mathbf{x}_i \in S^r \cup S^b}$ ▷ the weak solution estimator evaluated at $\{x\} \times \Pi_T$

Appendix C. Numerical results summary

Below we present some results from the comparisons of the XNODE-WAN and WAN models presented in Example 4.2 and Example 4.4.

Table C.3
Performance comparison between XNODE-WAN and WAN on various domains for Example 4.2 and Example 4.4.

Example 4.2: Cylinder domain				
Model	Relative error	Time per epoch (s)	N_ε	T_ε (s)
WAN	1.0%	0.38	15,278	5806
XNODE-WAN	1.1%	0.35	271	88
Example 4.4: Hourglass domain				
WAN	8.3%	0.38	21,594	8206
XNODE-WAN	7.1%	0.43	221	95

References

[1] J. Han, A. Jentzen, E. Weinan, Solving high-dimensional partial differential equations using deep learning, Proc. Natl. Acad. Sci. 115 (2018) 8505–8510.
[2] M. Hutzenthaler, A. Jentzen, T. Kruse, T. Anh Nguyen, P. von Wurstemberger, Overcoming the curse of dimensionality in the numerical approximation of semilinear parabolic partial differential equations, Proc. R. Soc. A 476 (2020) 20190630.
[3] P. Druzhkov, V. Kustikova, A survey of deep learning methods and software tools for image classification and object detection, Pattern Recognit. Image Anal. 26 (2016) 9–15.
[4] Y. Goldberg, A primer on neural network models for natural language processing, J. Artif. Intell. Res. 57 (2016) 345–420.
[5] M. Raissi, P. Perdikaris, 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) 686–707.

[6] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, A. Anandkumar, Multipole graph neural operator for parametric partial differential equations, preprint, arXiv:2006.09535, 2020.
[7] C. He, X. Hu, L. Mu, A mesh-free method using piecewise deep neural network for elliptic interface problems, preprint, arXiv:2005.04847, 2020.
[8] L. Lu, X. Meng, Z. Mao, G.E. Karniadakis, Deepxde: a deep learning library for solving differential equations, SIAM Rev. 63 (2021) 208–228.
[9] J. Berner, P. Grohs, A. Jentzen, Analysis of the generalization error: empirical risk minimization over deep artificial neural networks overcomes the curse of dimensionality in the numerical approximation of Black–Scholes partial differential equations, SIAM J. Math. Data Sci. 2 (2020) 631–657.
[10] C. Duan, Y. Jiao, Y. Lai, X. Lu, Z. Yang, Convergence rate analysis for deep Ritz method, preprint, arXiv:2103.13330, 2021.
[11] T. Luo, H. Yang, Two-layer neural networks for partial differential equations: optimization and generalization theory, preprint, arXiv:2006.15733, 2020.
[12] K.E. Gilbert, M.J. White, Application of the parabolic equation to sound propagation in a refracting atmosphere, J. Acoust. Soc. Am. 85 (1989) 630–637.
[13] J. Douglas, H.H. Rachford, On the numerical solution of heat conduction problems in two and three space variables, Trans. Am. Math. Soc. 82 (1956) 421–439.
[14] J. Hahn, X.-C. Tai, S. Borok, A.M. Bruckstein, Orientation-matching minimization for image denoising and inpainting, Int. J. Comput. Vis. 92 (2011) 308–324.
[15] Y. Achdou, O. Pironneau, Computational Methods for Option Pricing, SIAM, 2005.
[16] J. Sirignano, K. Spiliopoulos, Dgm: a deep learning algorithm for solving partial differential equations, J. Comput. Phys. 375 (2018) 1339–1364.
[17] Y. Zang, G. Bao, X. Ye, H. Zhou, Weak adversarial networks for high-dimensional partial differential equations, J. Comput. Phys. 411 (2020) 109409.
[18] R.T. Chen, Y. Rubanova, J. Bettencourt, D. Duvenaud, Neural ordinary differential equations, preprint, arXiv:1806.07366, 2018.
[19] Z. Cai, J. Chen, M. Liu, X. Liu, Deep least-squares methods: an unsupervised learning-based numerical method for solving elliptic PDEs, J. Comput. Phys. 420 (2020) 109707.
[20] E. Weinan, B. Yu, The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems, Commun. Math. Stat. 6 (2018) 1–12.
[21] E. Samaniego, C. Anitescu, S. Goswami, V.M. Nguyen-Thanh, H. Guo, K. Hamdia, X. Zhuang, T. Rabczuk, An energy approach to the solution of partial differential equations in computational mechanics via machine learning: concepts, implementation and applications, Comput. Methods Appl. Mech. Eng. 362 (2020) 112790.
[22] Q. Hong, J.W. Siegel, J. Xu, A priori analysis of stable neural network solutions to numerical PDEs, preprint, arXiv:2104.02903, 2021.
[23] M. Renardy, R.C. Rogers, An Introduction to Partial Differential Equations, vol. 13, Springer Science & Business Media, 2006.
[24] G. Teschl, Ordinary Differential Equations and Dynamical Systems, vol. 140, American Mathematical Soc., 2012.