*Chin. Ann. Math. Ser. B*
46(3), 2025, 321–340
DOI: 10.1007/s11401-025-0018-z

**Chinese Annals of Mathematics, Series B**


# Single-step Neural Operator Solver for Semilinear Evolution Equations*

Zhen LEI[1] Lei SHI[2] Xiyuan WANG[3]

**Abstract** The neural operator theory offers a promising framework for efficiently solving complex systems governed by partial differential equations (PDEs for short). However, existing neural operators still face significant challenges when applied to spatiotemporal systems that evolve over large time scales, particularly those described by evolution PDEs with time-derivative terms. This paper introduces a novel neural operator designed explicitly for solving evolution equations based on the theory of operator semigroups. The proposed approach is an iterative algorithm where each computational unit, termed the single-step neural operator solver (SSNOS for short), approximates the solution operator for the initial-boundary value problem of semilinear evolution equations over a single time step. The SSNOS consists of both linear and nonlinear components: The linear part approximates the linear operator in the solution map; in contrast, the nonlinear part captures deviations in the solution function caused by the equations nonlinearities. To evaluate the performance of the algorithm, the authors conducted numerical experiments by solving the initial-boundary value problem for a two-dimensional semilinear hyperbolic equation. The experimental results demonstrate that their neural operator can efficiently and accurately approximate the true solution operator. Moreover, the model can achieve a relatively high approximation accuracy with simple pre-training.

**Keywords** Neural operator, Evolution equations, Semigroup of operators, Predictor-corrector method
**2020 MR Subject Classification** 68T07, 65M22, 65J08

## 1 Introduction

PDEs are used to describe many systems in science and engineering, playing a vital role in modeling various problems, such as determining the evolution trends of certain key variables or inferring global information from local data. Traditional numerical methods, e.g., the finite element and finite difference methods, solve PDEs by discretizing the domain, requiring fine

Manuscript received November 3, 2024. Revised February 26, 2025.
[1]School of Mathematical Sciences, Shanghai Key Laboratory for Contemporary Applied Mathematics, Center for Applied Mathematics, Fudan University, Shanghai 200433, China. E-mail: zlei@fudan.edu.cn
[2]Corresponding author. School of Mathematical Sciences, Shanghai Key Laboratory for Contemporary Applied Mathematics, Center for Applied Mathematics, Fudan University, Shanghai 200433, China. E-mail: leishi@fudan.edu.cn
[3]School of Mathematical Sciences, Shanghai Key Laboratory for Contemporary Applied Mathematics, Fudan University, Shanghai 200433, China. E-mail: 21110180059@m.fudan.edu.cn
*This work was supported by the National Natural Science Foundation Major Program of China (No. 12494544), the National Natural Science Foundation General Program of China (No. 12171039), the New Cornerstone Science Foundation through the XPLORER PRIZE and Sino-German Center Mobility Programme (No. M-0548) and the Shanghai Science and Technology Program (No. 21JC1400600).

meshes to achieve sufficiently accurate solutions. When solving dynamic evolving systems, the modeled quantities usually depend on space and time. Solving evolving PDEs over time with traditional numerical methods requires finer spatial-temporal grids and intensively solving large-scale linear systems, making the process increasingly time-consuming. Additionally, in most applications, the relevant PDEs often depend on parameters describing the physical or geometric constraints of the equations. In some cases, the equations are solved repeatedly for different parameter values.

To address these challenges, recent efforts have focused on developing PDE solvers based on neural networks (NNs for short). Leveraging the strong approximation capabilities of NNs and the automatic differentiation technique, NN-based solvers can directly approximate the true solutions or solution mappings of PDEs without relying on mesh discretization of the domain. One type of algorithm is based on the function approximation perspective (see [3–4, 6, 11, 23]), which constructs NN approximations of the true PDE solution. Another class of algorithms, commonly referred to as neural operators, focuses on NN-based operator learning, aiming to approximate the mapping from the initial spacedescribing potential variations in geometric configurations, physical properties, initial and boundary conditions, or source terms to the solution space (see [13–16, 22]). This mapping is known as the solution operator of the PDE. Recently, various neural operators have been designed to approximate the solution operator of PDEs. A neural operator $\mathcal{G}_\theta$ is typically constructed by composing several parameterized affine operators $\{\mathcal{L}_k\}_{k=0}^N$ and nonlinear operators $\{\sigma_k\}_{k=0}^N$, often represented as

$$\begin{cases} \mathcal{G}_\theta(a) = \mathcal{L}_N \circ \sigma_N \circ \mathcal{L}_{N-1} \circ \sigma_{N-1} \circ \cdots \circ \mathcal{L}_1 \circ \sigma_1 \circ \mathcal{L}_0(a), \\ \mathcal{L}_k\eta = \mathcal{W}_k\eta + b_k, \quad \mathcal{W}_k \in \mathbb{R}^{n_{k+1}\times n_k},\ b_k \in \mathbb{R}^{n_{k+1}},\ \forall \eta \in \mathbb{R}^{n_k},\ k = 0, 1, \cdots, N, \\ \theta = \{(\mathcal{W}_k, b_k)\}_{k=0}^N, \end{cases} \tag{1.1}$$

where $\theta$ is a parametric vector in a finite-dimensional real vector space.

By selecting different nonlinear operators and linear subspaces of parameters for the neural operator, various specialized neural operator structures can be obtained. Notable examples include the graph and multi-pole graph neural operator (see [14]), the U-shaped neural operator (see [22]), DeepONets (see [16]), and the Fourier neural operator (FNO for short) (see [15]).

Unlike previous methods, neural operators directly approximate the solution operator during the training phase, while solving the equations during inference. Once these neural operators are established, they often exhibit better efficiency in solving PDE systems compared to classical methods.

Although neural operators have shown excellent performance in solving PDEs, designing more efficient neural operators that incorporate the inherent properties of the equations remains an urgent problem to be solved. Particularly when solving semilinear evolving PDEs over time, it is crucial to have reasonable control over discretization error and approximation error as the solving process progresses, and correct the nonlinear terms of the system along the continuously changing time trajectories; this is a key to designing neural operators that efficiently solve such equations. In this work, we propose a novel neural operator architecture for solving nonlinear partial differential evolution equations that can be used to perform real-time batch solving tasks. Many time-evolving systems involved in various applications exhibit specific spatiotemporal structures, where the time variables have characteristics of linear dynamics, while the spatial variables are governed by more complex and possibly highly nonlinear equations. This special

spatiotemporal decoupling is the so-called semi-linear form, which can be expressed as the following evolution equation:

$$\frac{\partial u}{\partial t} = -Au + f(u), \quad u(0) = u_0, \tag{1.2}$$

where $u(t) \in B$ represents the solution at time $t$ in some Banach space $(B, \|\cdot\|_B)$, which itself is a function of the spatial variables $x$, $A$ is a linear bounded operator on $B$. The operator $f$ represents the nonlinear term of the equation and is generally assumed to satisfy certain continuity conditions with respect to the function $u$.

The design of our neural operator is inspired by the theory of operator semigroups, which is used to solve evolution PDEs (see [8, 21]). According to semigroup theory, if the linear operator $A$ is the infinitesimal generator of a $C_0$-semigroup $\{S(t) : B \to B,\ t \geq 0\}$ on the Banach space $B$, satisfying $\|S(t)\|_B \leq M\mathrm{e}^{wt}$ for some $M \in (0, 1)$ and $w \geq 0$, then the solution $u(t) \in B$ of equation (1.2) exists and is given by:

$$u(t) = S(t)u_0 + \int_0^t S(t-\tau)f(u(\tau))\mathrm{d}\tau, \tag{1.3}$$

where the initial function $u_0$ belongs to the dense subset $D(A) \subset B$, the domain of $A$. The integral in this expression represents the Bochner integral (see [18]) of $S(t-\tau)f(u(\tau))$ over the interval $[0, t]$.

Inspired by classical schemes for solving PDEs, operators that map functions to functions can be transformed into discrete operators between finite-dimensional vector spaces. Therefore, we can write the expression (1.3) above in a discrete form:

$$u_d(t) = S_d(t)u_{d,0} + \int_0^t S_d(t-\tau)f(u_d(\tau))\mathrm{d}\tau,$$

where $u_d$, $u_{d,0}$ and $S_d$ are finite-dimensional, representing the discretized solution, initial function and operator, respectively.

When the operator and the nonlinear term satisfy certain regularity requirements, it is straightforward to show that

$$\begin{aligned} u_d(t) &= S_d(t)u_{d,0} + \int_0^t S_d(t)f(u_{d,0})\mathrm{d}\tau + O(t^2) \\ &= S_d(t)(u_{d,0} + tf(u_{d,0})) + O(t^2), \end{aligned} \tag{1.4}$$

where the term $O(t^2)$ denotes terms of order $t^2$ as $t \to 0$.

An alternative approach useful in numerical integration is the trapezoidal rule. By applying this rule, we obtain another approximate solution to the evolution equation:

$$u_d(t) = S_d(t)u_{d,0} + \frac{t}{2}(S_d(t)f(u_{d,0}) + f(u_d(t))) + O(t^2). \tag{1.5}$$

Finally, the operators involved in the expressions for the approximate solutions $u_1$ and $u_2$ are approximated by neural operators of the form (1.1). We can define the neural operator structure as

$$\begin{aligned} U_1(t) &= N_d(t)(u_{d,0} + tf(u_{d,0})) + \lambda_1 N_r(u_{d,0}, t), \\ U_2(t) &= N_d(t)u_{d,0} + \frac{t}{2}(N_d(t)f(u_{d,0}) + f(U_1(t))) + \lambda_2 N_t(u_{d,0}, t), \end{aligned} \tag{1.6}$$

where $N_d(t)$ is a neural operator approximating the discretized operator corresponding to the semigroup of operators and includes only linear transformations. The operators $N_r$ and $N_t$ are nonlinear neural operators designed to approximate the infinitesimal operators in (1.4) and (1.5), respectively. The hyperparameters $\lambda_1, \lambda_2 \in \{0, 1\}$ determine whether to include the infinitesimal approximation operator in the network architecture.

It is worth noting that, for a given error tolerance $\varepsilon > 0$, the variable $t$ must be chosen sufficiently small to control the approximation error. For larger value of $t$, we can consider the above structure as recurrent units within a recurrent neural network, obtaining the inferred solution through multiple iterations.

Due to its advantages in interpretability, theoretical analysis and efficiency, which we summarize below, this model is a valuable tool for solving differential equations using neural operators.

(1) The single-step neural operator solver consists of three deep neural operators: One linear operator and two infinitesimal operators that correspond to components in the discretized equations (1.4)–(1.5) for solving a semilinear evolution equation. The functional diversity of these operators enhances their interpretability. Moreover, in practical applications, different operators can undergo targeted optimizations specific to their roles.

(2) Our proposed algorithm incorporates design principles from classical algorithms, facilitating the adaptation of their theoretical results for error analysis. The model can be seen as an extension of the predictor-corrector method (see [9–10]) from classical algorithms into the deep neural operator domain for solving evolution equations. Thus, operator convergence theory associated with the predictor-corrector method and semigroups of operators can be directly applied to the theoretical analysis of our model.

(3) Compared to traditional nonlinear numerical algorithms, the single-step neural operator solver, as a forward-solving method, skips the complex computation of repeated iterations during inference. Simultaneously, it approximates the residual error using infinitesimal neural operators, which helps to reduce errors during the training process. Since deep neural operators perform the solving during inference, the operator approximation process can be decoupled from the solving process, thereby enhancing computational efficiency.

The rest of the paper is organized as follows. In Section 2, we introduce the theory of evolution equations and semigroups of operators. Following the introduction of the theory, we provide some common examples in Section 2. These examples serve to illustrate the practical application of the methods for identifying operator semigroups. Section 3 explains the relationship between finite element methods and numerical algebra. Additionally, we analyze the characteristics of the finite element method and neural networks in solving nonlinear equations. In Section 4, we provide detailed parameters of the deep operator network and introduce the method for constructing the dataset. In Section 5, we present the results of numerical experiments and analyze the performance of the algorithm. In Section 6, we conclude the paper.

## 2 Theory of Operator Semigroups

In this section, we consider functions defined on a specific spacial domain $\Omega \subset \mathbb{R}^d$. It moreover satisfies certain regular conditions. The theory of operator semigroups is based on linear function spaces. Generally, the functions mentioned above must be selected from a complete normed space, namely, a Banach space $B$, where the functions automatically satisfy

the corresponding regularity conditions due to the definition of the space.

In PDE theory, the most common Banach spaces are the standard Lebesgue integrable function spaces $L^p(\Omega),\ p > 1$ and Sobolev spaces $W^{m,p}(\Omega),\ m \in \mathbb{N}^*,\ p > 1$. The norms associated with the above spaces are defined as

$$\|u\|_{L^p} = \Big( \int_\Omega |u(x)|^p \mathrm{d}x \Big)^{\frac{1}{p}},$$
$$\|u\|_{W^{m,p}} = \sum_{|\alpha| \le m} \|\partial^\alpha u\|_{L^p},$$

where the multi-index $\alpha \in \mathbb{N}^d$ represents the various orders of partial derivatives of the function in the sense of distributions by

$$\partial^\alpha u = \partial^{\alpha_1} \partial^{\alpha_2} \cdots \partial^{\alpha_d} u, \quad \alpha = (\alpha_i)_{i=1}^d \in \mathbb{N}^d, \quad |\alpha| = \sum_{i=1}^d \alpha_i.$$

Specifically, when $p = 2$, the Sobolev spaces become Hilbert spaces denoted by $H^m(\Omega)$. Further details related to Sobolev spaces can be found in [7, 24].

Under the setting of evolution equations, we consider mappings that map the elements in time interval $\mathcal{I} = [0, T_f],\ T_f > 0$ to Banach space $B$. Such a mapping

$$f : \mathcal{I} \to B$$

can also be regarded as a function on $\mathcal{I}$, with its values being elements in $B$. It is also convenient to view it as a generalization of vector-valued functions, namely, a function-valued function.

In this sense, we can also define the continuity and differentiability of the function above.

**Definition 2.1** *A function $f : \mathcal{I} \to B$ is continuous with respect to $t_0 \in \mathcal{I}$ if*

$$\lim_{t \to t_0} \|f(t) - f(t_0)\|_B = 0.$$

*The linear space of all continuous functions is denoted by $C(\mathcal{I}, B)$.*

**Definition 2.2** *A function $f : \mathcal{I} \to B$ is differentiable with respect to $t_0 \in \mathcal{I}$ if there exists a function $g \in B$ such that*

$$\lim_{t \to t_0} \Big\| \frac{f(t) - f(t_0)}{t - t_0} - g \Big\|_B = 0.$$

*We denote the first derivative of $f$ on $t$ by*

$$f'(t) = g.$$

*Function $f$ is continuously differentiable if*

$$f' \in C(\mathcal{I}, B).$$

*The linear space of all continuously differentiable functions is denoted by $C^1(\mathcal{I}, B)$.*

Similar definitions can be established for other regularity definitions, such as Lebesgue integrability. In the theory of solving PDEs, functions that belong to Sobolev space $W^{m,p}(\Omega)$

are typically considered, where $m$ represents the highest order of the partial derivatives and $p$ represents the parameter of the Lebesgue $p$-norm.

Using $D^k u$ to denote the spatial partial derivatives of a function $u$ of each order $k \in \mathbb{N}^*$, we can express the form of an $n$-order PDE problem that only involves spacial derivatives of the function $u$ as

$$\mathcal{F}(u, Du, D^2u, \cdots, D^n u) = 0.$$

Besides, we can represent the linear and nonlinear parts of the equation separately by

$$\mathcal{F}(u, Du, \cdots, D^n u) = \mathcal{L}(u, Du, \cdots, D^n u) + f(u, Du, \cdots, D^n u),$$

where $\mathcal{L}$ and $f$ are the linear and nonlinear operators in the PDE, respectively.

An evolution equation is a PDE that involves the time variable $t$. Similar to the expression above, the evolution equation can be expressed as

$$\mathcal{F}\Big(u, \frac{\partial u}{\partial t}, \cdots, \frac{\partial^{n_t} u}{\partial t^{n_t}}, Du, \cdots, D^n u\Big) = 0.$$

Specifically, the equations considered in the theory of semigroups are mainly semilinear equations, which are often expressed as

$$\frac{\partial u}{\partial t} = -Au + f(u),$$

where $A$ is a linear operator that maps the function $u$ to the differential form $Au$ and $f$ is the nonlinear operator that depends only on $u$.

Generally speaking, the domain and range of the operator $A$ are not the same. Considering the Laplace operator on $H^2(\Omega)$ as an example, it is easy to see that its range is $L^2(\Omega)$. The linear differential operators considered in the theory of operator semigroups map functions to a Banach space $B$. We denote the domain by $D(A)$, a subspace of $B$.

On the other hand, a semigroup $\{S(t) : B \to B,\ t \geq 0\}$ of operators that is able to solve a linear problem

$$\frac{\partial u}{\partial t} = -Au$$

is called a strongly continuous semigroup of contractions or a $C_0$-semigroup in a Banach space. If the initial function is denoted by $u_0$ and belongs to $D(A)$, the solution to the equation can be expressed as

$$u(t) = S(t)u_0, \quad t \geq 0,$$

and satisfies

$$\frac{\partial u}{\partial t}(t) = -AS(t)u_0 = -S(t)Au_0, \quad t \geq 0.$$

For an equation that can be solved using a semigroup of operators, the corresponding linear operator $A$ is related to the semigroup. Specifically, linear operator $A$ and the semigroup $\{S(t),\ t \geq 0\}$ satisfy the condition:

$$-Au = \lim_{t \to 0} \frac{S(t)u - u}{t}, \quad u \in \Big\{u \in B \mid \exists v \in B,\ \lim_{t \to 0} \Big\| \frac{S(t)u - u}{t} - v \Big\|_B = 0\Big\}.$$

In this sense, we have

$$D(A) = \Big\{u \in B \mid \exists v \in B,\ \lim_{t\to 0}\Big\|\frac{S(t)u-u}{t} - v\Big\|_B = 0\Big\}. \tag{2.1}$$

The linear operator $A$ is called the infinitesimal generator of the $C_0$-semigroup of operators $\{S(t) : B \to B,\ t \geq 0\}$. The remarkable Hille-Yosida Theorem provides the necessary and sufficient conditions for a linear operator $A$ to be the infinitesimal generator of a semigroup.

**Theorem 2.1** (Hille-Yosida Theorem) (see [8, 21]) *Assume that $B$ is a Banach space and $A$ is a linear operator*

$$A : D(A) \subset B \to B.$$

*The necessary and sufficient conditions for $A$ to be an infinitesimal generator of a linear semigroup are*

(1) *$A$ is a densely defined operator in $B$, i.e., $D(A)$ is a dense subset of $B$,*

(2) *for all $\lambda > 0$, $\lambda I + A$ is a one-to-one and onto mapping, and*

$$\|\lambda(\lambda I + A)^{-1}\| \leq 1.$$

Furthermore, if $B$ is a Hilbert space, the necessary and sufficient conditions for $A$ to be an infinitesimal generator of a semigroup becomes

$$\begin{aligned} &\mathrm{Re}\langle Ax, x\rangle \geq 0, \quad \forall u \in D(A),\\ &R(I + A) = B, \end{aligned} \tag{2.2}$$

where the notation $\mathrm{Re}\langle Ax, x\rangle$ represents the real part of the inner product $\langle Ax, x\rangle$ and the notation $R(I + A)$ is the range of the operator $I + A$.

In the case of semilinear evolution equations, semigroup theory also provides an existence and uniqueness theorem for the solution. This theorem can be regarded as the theoretical foundation for constructing our deep neural solution operator framework.

**Theorem 2.2** (Existence and uniqueness theorem of $C_0$-semigroups) (see [8]) *Suppose that $A$ is an infinitesimal generator of an $C_0$-semigroup of linear operators $\{S(t) : B \to B,\ t \geq 0\}$ on a reflexive space $B$, and $f$ is a nonlinear operator from $D(A)$ to $D(A)$, satisfying the local Lipschitz condition*

$$\|f(u) - f(v)\| \leq L_M\|u - v\|, \quad \forall u, v \in B \text{ s.t. } \|u\|, \|v\| \leq M,\ \forall M > 0.$$

*Considering the initial value problem of the semilinear evolution equation*

$$\begin{cases} \dfrac{\partial u}{\partial t} = -Au + f(u),\\ u(0) = u_0 \in B \end{cases} \tag{2.3}$$

*for each $u_0 \in D(A)$, there is a positive constant $T > 0$ depending on $u_0$ such that the initial value problem has a unique Lipschitz continuous classical solution $u$ in $[0, T]$, satisfying*

$$u \in C([0, T], D(A)) \cap C^1([0, T], B).$$

*The solution $u$ also satisfies the condition*

$$u(t) = S(t)u_0 + \int_0^t S(t - \tau)f(u(\tau))\mathrm{d}\tau, \quad \forall t \in [0, T].$$

Since Hilbert spaces are reflexive Banach spaces, there is a straightforward corollary about the existence and uniqueness of the solution of problem (2.3) defined on a Hilbert space $H$.

We provide an example of the initial-boundary value problem of wave equation to illustrate how the Banach space $B$ and linear operator $A$ are determined. The wave function is a hyperbolic equation that involves the second derivative with respect to time $t$,

$$\frac{\partial^2 u}{\partial t^2} = \Delta u,$$

where $\Delta$ represents the Laplace operator that for a sufficiently smooth function $u$,

$$\Delta u = \sum_{i=1}^{d} \frac{\partial^2}{\partial x_i^2} u.$$

To use the solution expression provided by semigroup theory, the wave equation must be transformed into a form that only involves the first derivative with respect to time $t$. The technique is to invite a new variable

$$v = \frac{\partial u}{\partial t}.$$

By doing this, it is easy to see

$$\begin{cases} \dfrac{\partial u}{\partial t} = v, \\ \dfrac{\partial v}{\partial t} = \Delta u, \end{cases}$$

or in a matrix expression formally as

$$\frac{\partial}{\partial t} \begin{pmatrix} u \\ v \end{pmatrix} = - \begin{pmatrix} 0 & -I \\ -\Delta & 0 \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix}.$$

By observing the above expression, we can identify the linear operator $A$ that generates the semigroup as

$$A = \begin{pmatrix} 0 & -I \\ -\Delta & 0 \end{pmatrix}.$$

Furthermore, to determine the space $B$ and the domain of $A$, it is needed to consider the boundary conditions of the initial-boundary value problem. Taking the Dirichlet initial-boundary value problem as an example, we have

$$\begin{cases} \dfrac{\partial^2 u(t,x)}{\partial t^2} = \Delta u(t,x), & \forall t > 0,\ x \in \Omega, \\ u(0,x) = u_0(x),\ u_t(0,x) = u_{t,0}(x), & \forall x \in \Omega, \\ u(t,x) = 0, & \forall t > 0,\ x \in \partial\Omega \end{cases}$$

and a natural selection of spaces for $u$ and $v$ are $H_0^1(\Omega)$ and $L^2(\Omega)$, respectively. Based on the above spaces and the energy estimates of wave functions, we can establish a Hilbert space $H$ by

$$H = H_0^1(\Omega) \times L^2(\Omega),$$

$$\langle (u_1, v_1), (u_2, v_2) \rangle_H = \langle \nabla u_1, \nabla u_2 \rangle_{L^2} + \langle v_1, v_2 \rangle_{L^2},$$

where $\langle \cdot, \cdot \rangle$ represents an inner product on a certain Hilbert space.

The domain can be determined by definition (2.1),

$$\begin{cases} H_1 = H_0^1(\Omega) \cap H^2(\Omega), \\ H_2 = H_0^1(\Omega), \\ D(A) = H_1 \times H_2. \end{cases} \tag{2.4}$$

## 3 Hybrid Numerical Solving Algorithm for Evolution Equations

The main idea behind using deep neural networks for operator learning methods is to approximate the solution operator of PDEs through the network structure. In the above process, the input and output of the dataset are the initial-boundary conditions of the equation and the solution to the equation, respectively. Therefore, it is necessary to generate a series of initial-boundary conditions in advance and solve them using traditional numerical methods, finally forming the dataset.

The finite element method is an important approach for solving PDEs on irregular domains, while the finite difference method is effective for equations on regular domains [2, 5, 19]. By combining the two methods, evolution equations on irregular domains can be solved using a temporal finite difference method together with a spatial finite element method.

Assume that $A$ is a linear operator defined in a Hilbert space $H$ and the infinitesimal generator of a $C_0$-semigroup of operators $\{S(t) : H \to H,\ t \geq 0\}$. $f : D(A) \to D(A)$ is a nonlinear operator satisfying the local Lipschitz condition. The goal is to find an approximate solution of the classical solution $u \in C^1([0, T], H) \cap C([0, T], D(A))$ for some $T > 0$ that satisfies

$$\begin{cases} \dfrac{\partial u}{\partial t} = -Au + f(u), \\ u(0) = u_0 \in D(A). \end{cases} \tag{3.1}$$

Further, assume that both the trial space and the test space of the finite element method are a finite-dimensional subspace $H_h$ of $D(A)$, and the finite-dimensional approximate solution is $u_h \in H_h$. The weak form of the finite-dimensional problem can be expressed as

$$\begin{cases} \left\langle \dfrac{\partial u_h}{\partial t}, v \right\rangle = \langle -Au_h + f(u_h), v \rangle, & \forall v \in H_h, \\ \langle u_h(0), v \rangle = \langle u_0, v \rangle, & \forall v \in H_h. \end{cases}$$

Next, the function is discretized with respect to time. Fix a series of time for discretization

$$0 = t_0 < t_1 < \cdots < t_N = T, \quad N \in \mathbb{N}^*.$$

The discretized solution can be represented by finite-dimensional vectors, denoted by $(u_k)_{k=1}^N$, where $u_k = (u_{k,j})_{j=1}^d \in \mathbb{R}^d$ represents the linear coefficients of the approximate solution at time $t_k \in [0, T]$.

Assume that a basis of $H_h$ is $\{\varphi_j \in H_h\}_{j=1}^d$. The linear operator that maps the linear coefficients to the corresponding function can be denoted by

$$\Pi : \mathbb{R}^d \to H_h,$$

$$\beta = (\beta_j)_{j=1}^d \mapsto \sum_{j=1}^d \beta_j \varphi_j.$$

Furthermore, by applying the Crank-Nicolson scheme for time discretization of the function, the solution vector satisfies that for all $v \in \mathbb{R}^d$,

$$\Big\langle \frac{\Pi(u_{k+1}-u_k)}{\delta_{t_{k+1}}}, \Pi v \Big\rangle = \Big\langle -A\frac{\Pi(u_{k+1}+u_k)}{2} + \frac{(f(\Pi u_{k+1})+f(\Pi u_k))}{2}, \Pi v \Big\rangle,$$

where

$$\delta_{t_{k+1}} = t_{k+1} - t_k.$$

Equivalently, we can rewrite it in the form of an operator equation,

$$\begin{aligned} &\Pi^*\Big(I + \frac{\delta_{t_{k+1}}}{2}A\Big)\Pi u_{k+1} \\ &= \Pi^*\Big(I - \frac{\delta_{t_{k+1}}}{2}A\Big)\Pi u_k + \frac{\delta_{t_{k+1}}}{2}\Pi^*(f(\Pi u_{k+1}) + f(\Pi u_k)). \end{aligned} \tag{3.2}$$

Assume that $\lambda$ is an eigenvalue of the matrix and there exists a non-zero vector $\alpha \in \mathbb{R}^d$ such that

$$\Pi^*\Big(I + \frac{\delta_{t_{k+1}}}{2}A\Big)\Pi\alpha = \lambda\alpha.$$

By (2.2),

$$\begin{aligned} \lambda\|\alpha\|^2 = \lambda\alpha^*\alpha &= \alpha^*\Pi^*\Big(I + \frac{\delta_{t_{k+1}}}{2}A\Big)\Pi\alpha \\ &= \|\Pi\alpha\|^2 + \langle A\Pi\alpha,\ \Pi\alpha\rangle \geq \|\Pi\alpha\|^2 \geq \lambda_{\min}(\Pi^*\Pi)\|\alpha\|^2. \end{aligned}$$

Since $\Pi$ is degenerate by definition, it can be shown that $\lambda_{\min}(\Pi^*\Pi) > 0$. Therefore, matrix $\Pi^*\big(I + \frac{\delta_{t_{k+1}}}{2}A\big)\Pi$ is invertible.

Up to this point, the approximate solution satisfies the nonlinear equation (3.2). The remaining step is to linearize these nonlinear equations at each step. For a fixed $v \in \mathbb{R}^d$ and some sufficiently small $\delta_t > 0$, assume $\mathcal{T}_{\delta_t,v} : \mathbb{R}^d \to \mathbb{R}^d$ satisfies

$$\mathcal{T}_{\delta_t,v}(\alpha) = \Phi^{-1}\Big(\Pi^*\Big(I - \frac{\delta_t}{2}A\Big)\Pi v + \frac{\delta_t}{2}\Pi^*(f(\Pi\alpha) + f(\Pi v))\Big),$$

where

$$\Phi = \Pi^*\Big(I + \frac{\delta_t}{2}A\Big)\Pi.$$

The goal is to prove that $\mathcal{T}_{\delta_t,v}$ is a contraction mapping on $\mathbb{R}^d$. Let $v_0 \in \mathbb{R}^d$ and satisfy

$$v_0 = \Phi^{-1}\Big(\Pi^*\Big(I - \frac{\delta_t}{2}A\Big)\Pi v + \frac{\delta_t}{2}\Pi^* f(\Pi v)\Big).$$

Let $\mathcal{B}$ be a subset of $H_h$,

$$\mathcal{B} = B(v_0, 1) = \{w \in \mathbb{R}^d \mid \|w - v_0\| \leq 1\},$$

and a local upper bound $L_{\mathcal{B}} > 0$ be established to satisfy

$$\|f(\Pi w)\| \leq L_{\mathcal{B}}, \quad \forall w \in \mathcal{B}.$$

Therefore,

$$\|\mathcal{T}_{\delta_t,v}(\alpha) - v_0\| = \left\|\frac{\delta_t}{2}\Phi^{-1}\Pi^* f(\Pi\alpha)\right\| \leq \frac{\|\Phi^{-1}\Pi^*\|L_{\mathcal{B}}}{2}\delta_t.$$

Let $\delta_t \leq \frac{1}{\|\Phi^{-1}\Pi^*\|L_{\mathcal{B}}}$, it holds that

$$\mathcal{T}_{\delta_t,v}(\alpha) \in \mathcal{B}, \quad \forall \alpha \in \mathcal{B}.$$

On the other hand, for $\alpha_1, \alpha_2 \in \mathcal{B}$,

$$\|\mathcal{T}_{\delta_t,v}(\alpha_1) - \mathcal{T}_{\delta_t,v}(\alpha_2)\| = \frac{\delta_t}{2}\|\Phi^{-1}\Pi^*(f(\Pi\alpha_1) - f(\Pi\alpha_2))\| \leq \frac{1}{2}.$$

It is now proved that if $\delta_t$ is sufficiently small at each step $k = 1, 2, \cdots, N$, the sequence $\{u_k^j\}_{j=1}^{\infty}$ that satisfies

$$\begin{cases} u_k^0 = u_{k-1}, \\ u_k^j = \mathcal{T}_{\delta_t,u_{k-1}}(u_k^{j-1}), \quad j \in \mathbb{N}^* \end{cases}$$

will converge to $u_k^*$ such that

$$\mathcal{T}_{\delta_t,u_{k-1}}(u_k^*) = u_k^*.$$

Clearly, if $u_{k-1}$ is the approximate solution at time $t_{k-1}$, $u_k^*$ will become the approximate solution $u_k$ at time $t_k$.

To conclude, the local approximate solution of the semilinear evolution problem (3.1) can be obtained step by step. To enhance precision, multiple iterations are needed at each step of numerical computation.

## 4 Structure of Single-step Neural Operator Solver

Traditional numerical methods for solving differential equations often require repeatedly solving systems of linear equations. When dealing with nonlinear equations, the situation can become more complicated. As shown in the last section, at each step of time, the numerical scheme requires multiple iterations, and each iteration involves solving systems of linear equations, significantly reducing the algorithm's computational efficiency.

Neural network frameworks show great potential in handling nonlinear problems. Using all kinds of nonlinear activation functions, neural networks can effectively approximate nonlinear operators.

The goal of the single-step neural operator solver proposed in this paper is to approximate the expression for solutions of evolution equations provided by the theory of semigroups of operators. The term single-step is in contrast to the overall solver of an evolution equation.

Moreover, the approximated overall solution of the equation can be obtained by repeatedly applying the single-step method.

According to Theorem 2.2, for $\delta_t > 0$, the solution of a semilinear evolution equation in the form of (3.1) at time $t + \delta_t,\ t > 0$ can be expressed as

$$u(t + \delta_t) = S(\delta_t)u(t) + \int_0^{\delta_t} S(\delta_t - s)f(u(t + s))\mathrm{d}s,$$

if the solution at $t + \delta_t$ exists.

Based on the properties of the semigroup of operators (see [8, 21]) and Theorem 2.2, it holds that

$$\begin{aligned}
&\limsup_{s\to 0} \frac{\|S(\delta_t - s)f(u(t + s)) - S(\delta_t)f(u(t))\|}{s} < +\infty,\\
&\limsup_{s\to 0} \frac{\|S(\delta_t - s)f(u(t + s)) - f(u(t + \delta_t))\|}{s} < +\infty,
\end{aligned}$$

or equivalently

$$\begin{aligned}
&\|S(\delta_t - s)f(u(t + s)) - S(\delta_t)f(u(t))\| \sim O(s),\\
&\|S(\delta_t - s)f(u(t + s)) - f(u(t + \delta_t))\| \sim O(s).
\end{aligned} \tag{4.1}$$

Thus, we can discretize the integral term in equation (1.4) using the trapezoidal rule and obtain a numerical scheme

$$u(t + \delta_t) = S(\delta_t)u(t) + \frac{\delta_t}{2}(S(\delta_t)f(u(t)) + f(u(t + \delta_t))) + \mathcal{G}_t(u(t), \delta_t) \tag{4.2}$$

for an approximate solution of the semilinear evolution equation. Furthermore, it is straightforward to show that the discretized error of numerical integration is $O(\delta_t^2)$ and $\mathcal{G}_t(u(t), \delta_t)$ converges to 0 on a fixed interval $[0, T], T > 0$, while $\delta_t$ tends to 0.

In order to construct a structure in the form of forward propagation, it is also necessary to preprocess the function $u_h(t + \delta_t)$ on the right side of the equation. An explicit-form method is applied to obtain the preprocessed function.

By (4.1), it holds that

$$\begin{cases}
u(t + \delta_t) = S(\delta_t)u(t) + \delta_t S(\delta_t)f(u(t)) + \mathcal{G}_r(u(t), \delta_t),\\
\mathcal{G}_r(u_h(t), \delta_t) \sim O(\delta_t^2).
\end{cases} \tag{4.3}$$

Operators appeared in the above schemes are further approximated by different neural operators (see Table 1) to construct the final neural network structure. The linear neural operator $N_{d,\delta_t}, \delta_t > 0$ is used to approximate the operator $S(\delta_t)$ in the semigroup of operators. To reduce the dimensionality of neural network parameters and accelerate model inference, the linear operator can be represented by the multiplication of several low-rank matrices. And for infinitesimal operators, $N_r$ and $N_t$, various nonlinear deep operator frameworks can be used as approximation modules, such as DeepONet, Fourier Neural Operators, or a fully connected network. To improve the flexibility of the network structure, the nonlinear terms $N_r$ and $N_t$ are optional for model training and inference.

Table 1 Notations of neural operators and the task of each neural operator.

| notation | task |
|---|---|
| $N_d : \mathbb{R}^d \to \mathbb{R}^d$ | approximating the linear operator $S(\delta_t)$ |
| $N_t : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$ | approximating the infinitesimal operator $\mathcal{G}_t(u_h(t), \delta_t)$ |
| $N_r : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$ | approximating the infinitesimal operator $\mathcal{G}_r(u_h(t), \delta_t)$ |

The network structure to solve a semilinear evolution equation with time-stepping with respect to a sufficiently small $\delta_t$ can be expressed as

$$U_1(\delta_t) = N_d(\delta_t)(u_{d,0} + \delta_t f(u_{d,0})) + \lambda_1 N_r(u_{d,0}, \delta_t),$$
$$U_2(\delta_t) = N_d(\delta_t)u_{d,0} + \frac{\delta_t}{2}(N_d(\delta_t)f(u_{d,0}) + f(U_1(\delta_t))) + \lambda_2 N_t(u_{d,0}, \delta_t),$$

where $\lambda_1, \lambda_2 \in \{0, 1\}$ are hyperparameters representing the optionality of infinitesimal operators.

The single-step neural operator solver takes discretized initial functions or solutions at the previous time as inputs, and takes discretized solutions as outputs at the next step. Since there are two approximate solutions $U_1$ and $U_2$ in the network structure, the loss function $l$ for model training is also adjusted so that both approximate solutions can converge to the real solution,

$$l : \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$$
$$(U_1, U_2, u) \mapsto \frac{1}{2}(\mathrm{MSE}(U_1, u) + \mathrm{MSE}(U_2, u)),$$

where MSE stands for the mean squared error.

One advantage of using a neural operator solver is that a significant amount of computation can be completed during the offline training process, and there is no need to solve many equations repeatedly for real-time massive solving tasks. Methods for reducing the dimensionality of neural network parameters can also accelerate the model's speed for inference.

## 5 Numerical Analysis

The single-step neural operator solver is applied to a test problem to evaluate its performance. SSNOS is also compared with DeepONet and FNO in an experimental study, and the results indicate that the model demonstrated competitive training performance compared to other models.

Due to the model's strong interpretability, we also split the training process in experiments: first, training the linear operator $N_d$, followed by training the other operators. This training approach demonstrated even better approximation results compared to other models.

On the other hand, due to the computational characteristics of neural operator networks, SSNOS can simultaneously handle large-scale batch-solving tasks. As a result, the model boasts a significantly faster average inference speed when compared to classical algorithms, a testament to its efficiency and speed.

### 5.1 Problem description

The problem is an initial-boundary problem of a two-dimensional semilinear evolution PDE

$$\frac{\partial^2 u}{\partial t^2} = \Delta u + u - u^3.$$

The initial value $(u_0, u_{t,0})$ of the problem is sampled from function space $D(A) = (H_0^1(\Omega) \cap H^2(\Omega)) \times H_0^1(\Omega)$, and the boundary condition of the problem is Dirichlet boundary condition. By (2.4), the initial value problem is at least locally solvable with respect to time $t$.

### 5.2 Data generation

Training and test data are obtained by applying the hybrid numerical solving algorithm for PDEs mentioned in Section 3, which solves spatial equations by a finite element method at each step and solves the temporal equation by a finite difference method, involving multiple iterations at each step.

The domain $\Omega$ of the functions is the unit square $[0,1]^2 \subset \mathbb{R}^2$, and is represented by $33 \times 33 = 1089$ degrees of freedom after discretization.

The discretized inputs, targets and outputs of the single-step neural operator solver are real vectors representing $(\delta_t, u_0, u_{t,0})$, $(u(\delta_t),\ u_t(\delta_t))$ and $(U_1(\delta_t),\ U_2(\delta_t))$, respectively, where

$$\delta_t > 0, \quad u_0, u_{t,0}, u(\delta_t),\ u_t(\delta_t) \in \mathbb{R}^{1089}, \quad U_1(\delta_t),\ U_2(\delta_t) \in \mathbb{R}^{2178}.$$

To generate initial vectors for solving the evolution problems, a conditioned Gaussian field

$$X|BX = b, \quad X \sim \mathcal{N}(\mu, \Sigma)$$

is selected for sampling, where $\mathcal{N}(\mu,\ \Sigma)$ is a Gaussian field defined on $\mathbb{R}^{2178}$ and $BX = b$ represents the Dirichlet boundary condition.

The training data consists of 60000 samples generated by solving problems using FENICS (see [1]) with initial functions sampled from $X$ and the testing data consists of 13200 samples generated in the same way. The data is further batched in groups of 16 entries each for training and testing.

### 5.3 Operator structure and parameter selection

The linear operator is selected to be the product of two lower-rank matrices

$$N_d = A_1 A_2, \quad A_1 \in \mathbb{R}^{2178 \times 700}, \quad A_2 \in \mathbb{R}^{700 \times 2178}.$$

The infinitesimal operator $N_t$ is removed, and $N_r$ is selected to be a fully-connected neural network with 3 hidden layers, where the dimensions of each hidden layer are 600, 400 and 300, respectively. The robustness of our model is further enhanced by the use of LeakyReLU activation in each layer, including the final layer, a choice that ensures the stability and reliability of our system.

In the sense of distributions, the nonlinear operator of the equation can be expressed as

$$\langle f(u), v\rangle = \langle g(\mathcal{T}_{(\cdot)}(u)), v\rangle, \quad \forall u \in D(A),\ v \in D(A)^*,$$

where

$$\begin{aligned} g : \mathbb{R} &\to \mathbb{R} \\ y &\mapsto y - y^3, \\ \mathcal{T}_x : D(A) &\to \mathbb{R}, \quad \forall x \in \Omega \\ u &\mapsto u(x). \end{aligned}$$

To approximate the Dirac measures $\mathcal{T}_{(\cdot)}$ more accurately, we use a neural linear transformation $N_{\mathcal{T}}$ to represent the approximate operator of the measure once again. Similar to $N_d$, we use two matrices $A_3 \in \mathbb{R}^{2178 \times 600}$ and $A_4 \in \mathbb{R}^{600 \times 2178}$ to compose this linear transformation.

## 5.4 Experiments and results

### 5.4.1 Comparison with DeepONet and FNO

Taking a batch size as 16, each training epoch requires completing 3750 iterations. The comparison experiment between DeepONet, FNO and our model was conducted over 30 epochs, resulting in 112500 iterations. To compare the model's training efficiency, we used a learning rate of $10^{-3}$ and the SGD optimization method to train both models. In addition, the models' number of parameters is adjusted to achieve a consistent magnitude. The training code for training SSNOS and DeepONet is written using PyTorch (see [20]) and DeepXDE (see [17]). And the training code for training FNO is written using PyTorch (see [20]) and NeuralOperator (see [12–13]).

The error results for model training are shown in Figure 1. The results show that the SSNOS model, with 10310156 parameters, performs comparably to DeepONet, which has 7376301 parameters regarding training efficiency.

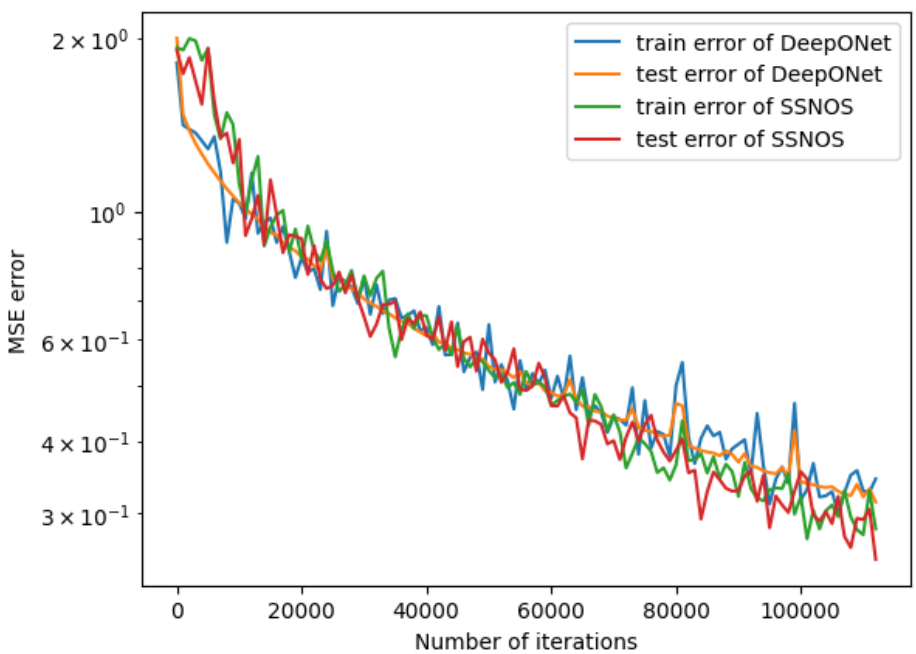


Figure 1 Error trend chart for SSNOS and DeepONet. The error recording interval is set to once every 1000 iterations. It is worth noting that the 112500 iterations of training are equivalent to 30 epochs of training for SSNOS.

With more epochs of training, SSNOS's fitting performance gradually improves. Figure 2 shows the model's inference performance after training for different epochs. By observing the changes in the predicted solution with respect to the number of training epochs, as shown in Figure 2, we can see that the model efficiently captures the contour information of the actual solution during training and subsequently learns the finer local details. This demonstrates the strong approximation ability of our model.

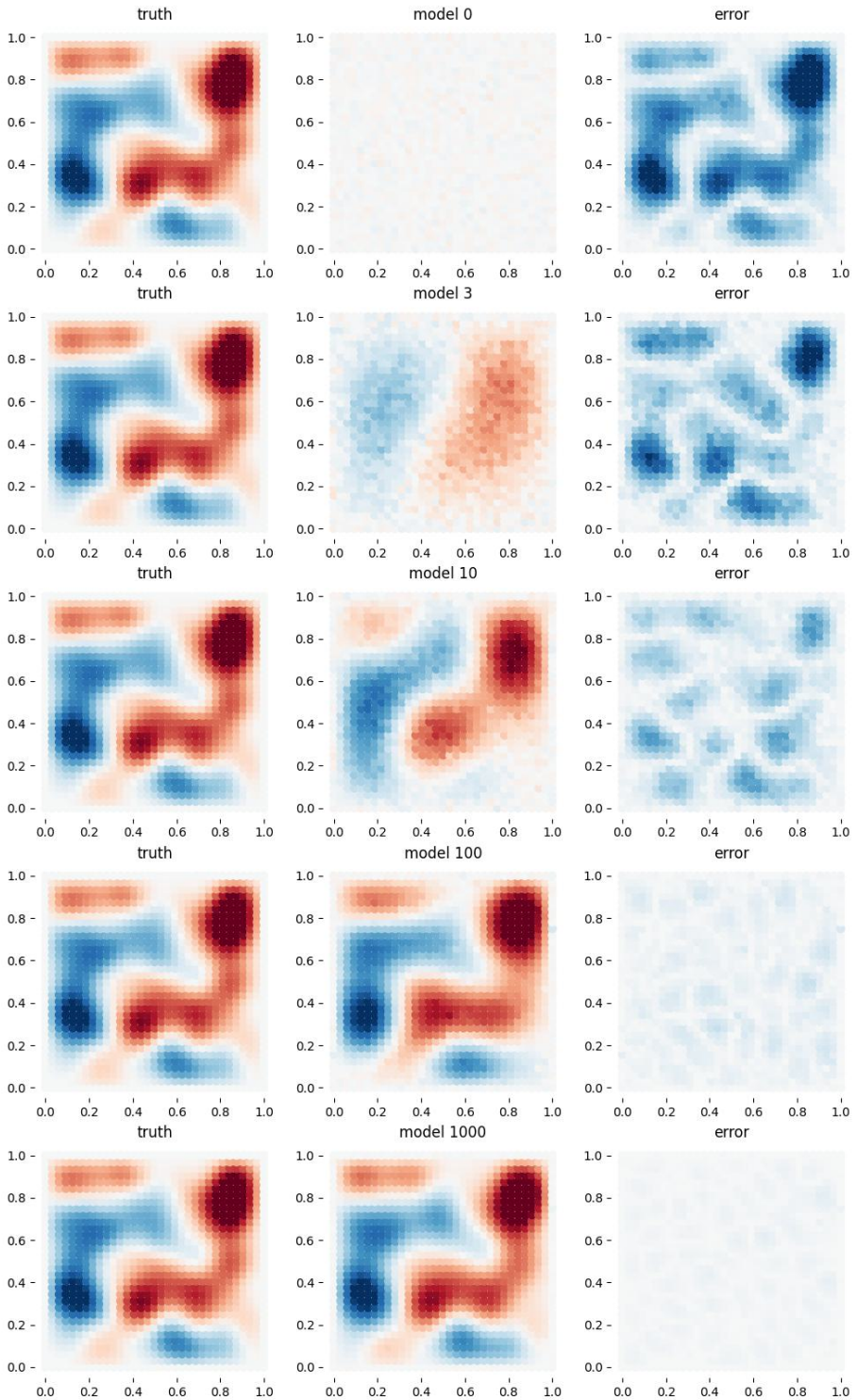


Figure 2 Comparison chart of model prediction performance at various stages of training, with data taken from the 1023rd sample in the test dataset.

### 5.4.2 Pre-training techniques for linear operators

One of the critical advantages of SSNOS is its clear division of roles among its components, which enhances its interpretability and operational flexibility. Based on the structure of the model, we know that $N_d$ is used to approximate the linear part of the solution operator $S(\delta_t)$ for the differential equation. On the other hand, $S_d$ is the solution operator to the linear equation

$$\begin{cases} \dfrac{\partial u}{\partial t} = -Au, \\ u(0) = u_0 \in D(A). \end{cases}$$

Therefore, we can pre-train $N_d$ separately by constructing linear equation solutions as a dataset, and then embed $N_d$ into the SSNOS model for further training. The pre-trained version of SSNOS and the standard version of the model are trained for 1000 epochs, and a performance comparison is conducted. As shown in Figure 3, the model with the pre-trained module starts with a significantly smaller error at the beginning of training and can further reduce the error throughout the training process.

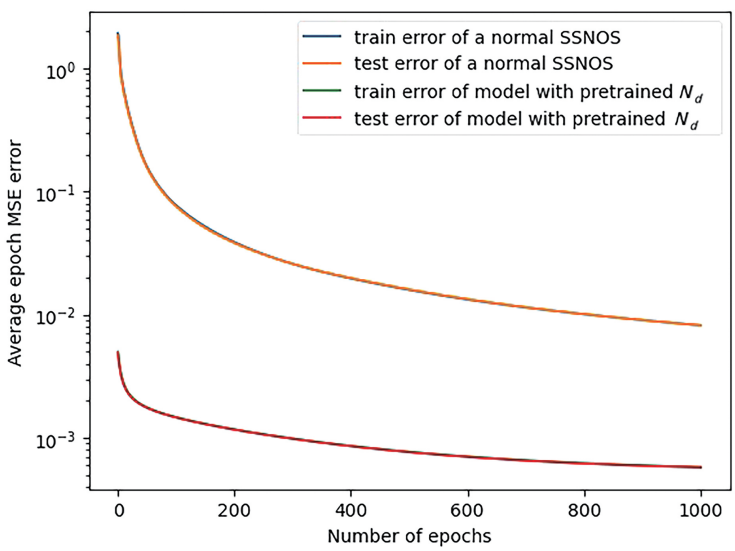


Figure 3 Error trend chart of model training. The statistical error is the average MSE error calculated on the test dataset after each completed epoch.

According to the relationship between batch size and dataset size, the two models trained in the first experiment are equivalent to models that have completed 30 training epochs. Figure 4 illustrates the inference performance of the two models mentioned above and the pre-trained model after training for 30 epochs. It can be observed that the performance of the model with simple pre-training significantly surpasses that of the models above under the same training conditions. A detailed statistical Table 2 compares each model's inference performance.

Based on the experimental data, it can be observed that the model is capable of quickly and effectively approximating the solution operator through training, gradually capturing detailed information as the training progresses. In addition, SSNOS boasts a significantly faster inference speed, far surpassing that of traditional algorithms.

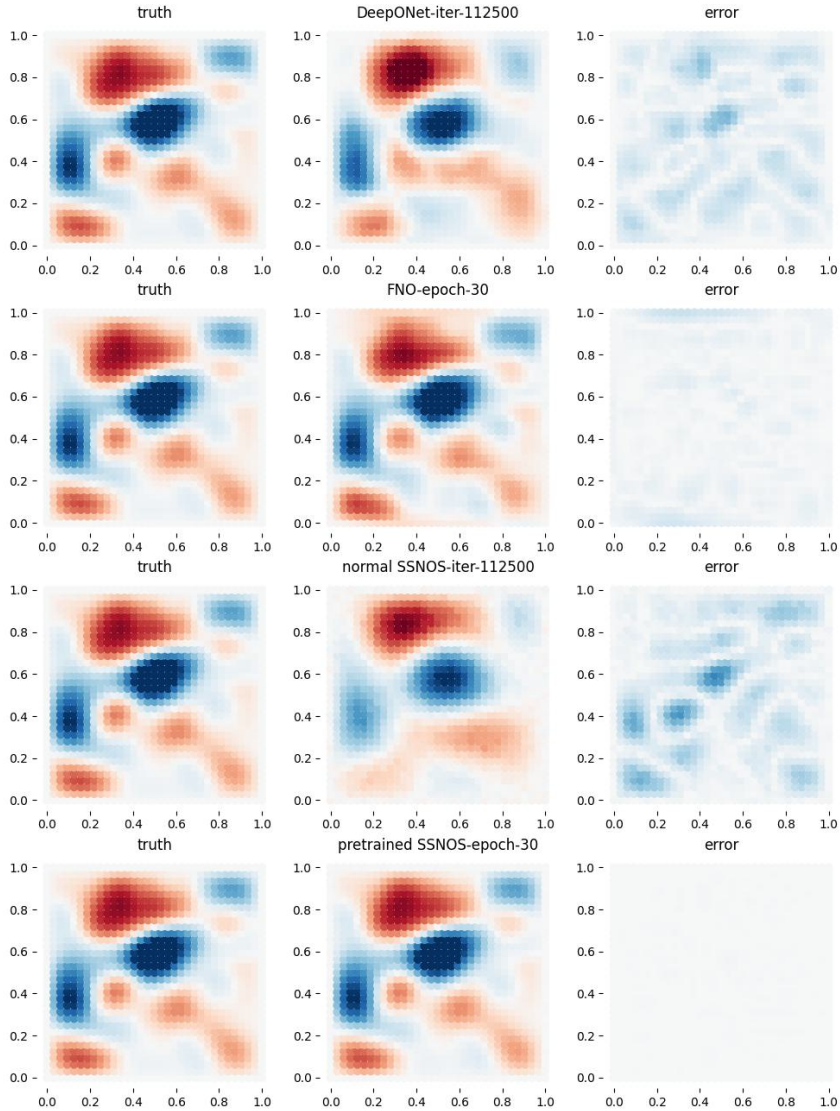


Figure 4 Comparison chart of actual and predicted solutions, with data taken from the 217th sample in the test dataset.

Table 2 Model inference capability comparison table. The pre-trained models correspond to those trained for 30 and 1000 epochs, respectively. The normal SSNOS model and DeepONet are trained for 112500 iterations. The FNO model is trained for 30 epochs.

| Model | Number of parameters | Training error | Test error | Inference time |
|---|---|---|---|---|
| FEM+FDM | - | - | - | $60000 \times 0.02451$s |
| Pretrained-1000 | 10310156 | 0.00060 | 0.00059 | 0.00141s |
| Pretrained-30 | 10310156 | 0.00206 | 0.00204 | 0.00164s |
| Normal | 10310156 | 0.27739 | 0.27343 | 0.00160s |
| Deeponet | 7376301 | 0.32061 | 0.31763 | 1.06322s |
| FNO | 10369490 | 0.01218 | 0.01224 | 0.00037s |

## 6 Conclusions

This work introduces a unique neural operator framework, specifically designed to solve initial boundary problems of semilinear evolution PDEs. The framework's novelty lies in its ability to separate the operator training process from the operator inference process, thereby accelerating the model's inference speed. This framework is based on the theory of semigroups of operators, leading to a class of neural network operator solvers. As a neural operator, this model can separate the operator training process from the operator inference process, thereby accelerating the model's inference speed.

In the context of the increasing demand for large-scale high-dimensional problem solving, the limitations of classical numerical algorithms based on numerical algebra, including the finite element method, in terms of solving speed are becoming increasingly pronounced. To overcome this challenge, designing solution schemes that leverage the characteristics of neural operators is a crucial optimization approach. Compared to traditional methods, neural operator methods often exhibit a certain decrease in solution accuracy but achieve significantly higher inference speed. Therefore, when performing small-scale high-precision tasks, traditional methods still have an advantage in solution accuracy and are better suited for such tasks. However, when handling large-scale batch-solving tasks, the parallel-solving capability of neural operator methods makes them more suitable for these scenarios. In order to combine the advantages of both and create a high-precision large-scale solver, a feasible approach is to generate as much high-quality data as possible using traditional methods during the model training phase, allowing the neural operator to better approximate the solution operator of the equation. The experiments in the article strongly demonstrate the feasibility and potential of this integrated approach.

The simulation results show that the single-step neural operator solver can efficiently approximate the operator, allowing it to quickly and accurately predict the function solutions in future instances. Even more noteworthy is that the model can efficiently achieve high-precision training through a straightforward pre-training phase and attain a strong approximation capability. Furthermore, the SSNOS model, designed based on the theory of semigroups of operator, can naturally incorporate a pre-training stage during training. According to the experimental results, the pre-trained model achieves the highest accuracy among all compared models. These qualities make the model a very promising neural operator framework.

In addition to the model's current potential for operator approximation, the flexibility of its architecture can be further explored. Currently, the solution framework accomplishes operator approximation through supervised training. The model's data is obtained through numerical solutions using traditional methods such as the finite element method. Using unsupervised methods, such as incorporating the principles of physics-informed neural networks, to train the model is also feasible. Besides, to enhance the fitting efficiency of the model, adopting network structures that are better suited for representing function operators, as opposed to the original fully connected networks, is also a viable optimization approach.

## Declarations

**Conflicts of interest** Zhen LEI is a deputy editor-in-chief for Chinese Annals of Mathematics Series B and was not involved in the editorial review or the decision to publish this article. All authors declare that there are no conflicts of interest.

## References


[1] Alnæs, M., Blechta, J., Hake, J., et al., The FEniCS project version 1.5, Archive of numerical software, **3**, 2015, http://api.semanticsholar.org/CorpusID:61220975.

[2] Ames, M. F., Numerical Methods for Partial Differential Equations, Academic press, InC., Boston, MA, 1992.

[3] Cai, S. Z., Mao, Z. P., Wang, Z. C., et al., Physics-informed neural networks (PINNs) for fluid mechanics: A review, *Acta Mechanica Sinica*, **37**(12), 2021, 1727–1738.

[4] Cuomo, S., Schiano, D. C., Vincenzo, F., et al., Scientific machine learning through physics–informed neural networks: Where we are and whats next, *Journal of Scientific Computing*, **92**(3), 2022, 88, 62.

[5] Dhatt, G., Lefrançois, E. and Touzot, G., Finite Element Method, John Wily & Sons, 2012.

[6] E, W. N., Han, J. Q. and Jentzen, A., Algorithms for solving high dimensional PDEs: From nonlinear Monte Carlo to machine learning, *Nonlinearity*, **35**(1), 2022.

[7] Evans, L. C., Partial Differential Equations, **19**, American Mathematical Society, Providence, RI, 2010.

[8] Goldstein, J. A., Semigroups of Linear Operators and Applications, Dover Publications, Inc. Mineola, NY, 2017.

[9] Gragg, W. B. and Stetter, H. J., Generalized multistep predictor-corrector methods, *Journal of the ACM* (*JACM*), **11**(2), 1964, 188–209.

[10] Hamming, R. W., Stable predictor-corrector methods for ordinary differential equations, *Journal of the ACM* (*JACM*), **6**, 1959, 37–47.

[11] Han, J. Q., Jentzen, A. and E, W. N., Solving high-dimensional partial differential equations using deep learning, *Proceedings of the National Academy of Sciences of the United states of America*, **115**(34), 2018, 8505–8510.

[12] Kossaifi, J., Kovachki, N., Li, Z. Y., et al., A library for learning neural operators, 2024, arXiv: 2412.10354.

[13] Kovachki, N., Lanthaler, S. and Mishra, S., On universal approximation and error bounds for Fourier neural operators, *Journal of Machine Learning Research*, **22**, 2021, 290.

[14] Kovachki, N., Li, Z. Y., Liu, B., et al., Neural operator: Learning maps between function spaces with applications to pdes, *Journal of Machine Learning Research*, **24**, 2023, 89.

[15] Li, Z. Y., Kovachki, N., Azizzadenesheli, K., et al., Fourier neural operator for parametric partial differential equations, 2020, arXiv: 2010.08895.

[16] Lu, L., Jin, P. Z., Pang, G. F., et al., Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators, *Nature machine intelligence*, **3**, 2021, 218–229.

[17] Lu, L., Meng, X. H., Mao, Z. P. and Karniadakis, G. E., DeepXDE: A deep learning library for solving differential equations, *SIAM Review*, **63**(1), 2021, 208–228.

[18] Mikusiński, J., The Bochner Integral, Birkhäuser Verlag, Basel-Stuttgart, 1978.

[19] Nikishkov, G. P., Introduction to the finite element method, University of Aizu, Aizu-wakamatsu, 2004.

[20] Paszke, A., Gross, S., Massa, F., et al., Pytorch: An imperative style, high-performance deep learning library, Advances in neural information processing systems, 2019, arXiv: 1912.01703, **32**.

[21] Pazy, A., Semigroups of Linear Operators and Applications to Partial Differential Equations, **44**, Springer-Verlag, New York, 1983.

[22] Rahman, M. A., Ross, Z., E. and Azizzadenesheli, K., U-no: U-shaped neural operators, 2022, arXiv: 2204.11127.

[23] Sirignano, J., and Spiliopoulos, K., DGM: A deep learning algorithm for solving partial differential equations, *Journal of computational physics*, **375**, 2018, 1339–1364.

[24] Sloan, D., Süli, E. and Vandewalle, S., Partial Differential Equations, Numerical Analysis 2000, **7**, Elsevier Science Publisher, B. V. Amsterdam, 2001.