Quest Journals
Journal of Research in Applied Mathematics
Volume 11 ~ Issue 5 (May 2025) pp: 53-77
ISSN (Online): 2394-0743 ISSN (Print): 2394-0735
www.questjournals.org

Review Paper

Solving partial differential equations by using neural networks and operators.

Dr. Adel Ahmed Hassan Kubba[1], Ragab Mostafa Mohamed Abdelbar[2]

[1]*University Nile Vally , Department of Graduate Studies*
[2]*PhD student in mathematics at University Nile Vally*

Abstract :
Physics-Informed Neural Networks (PINNs) have enabled significant improvements in modelling physical processes described by partial differential equations (PDEs). PINNs are based on simple architectures, and learn the behavior of complex physical systems by optimizing the network parameters to minimize the residual of the underlying PDE. Current network architectures share some of the limitations of classical numerical discretization schemes when applied to non-linear differential equations in continuum mechanics. A paradigmatic example is the solution of hyperbolic conservation laws that develop highly localized nonlinear shock waves. Learning solutions of PDEs with dominant hyperbolic character is a challenge for current PINN approaches, which rely, like most grid-based numerical schemes, on adding artificial dissipation.
Physics-Informed Neural Network (PINN) has proven itself a powerful tool to obtain the numerical solutions of nonlinear partial differential equations (PDEs) leveraging the expressivity of deep neural networks and the computing power of modern heterogeneous hardware. However, its training is still time-consuming, especially in the multi-query and real-time simulation settings, and its parameterization often overly excessive. In this paper, we propose the Generative Pre-Trained PINN (GPT-PINN) to mitigate both challenges in the setting of parametric PDEs. GPT-PINN represents a brand-new meta-learning paradigm for parametric systems. As a network of networks, its outer-/meta-network is hyper-reduced with only one hidden layer having significantly reduced number of neurons. Moreover, its activation function at each hidden neuron is a (full) PINN pre-trained at a judiciously selected system configuration. The meta-network adaptively "learns" the parametric dependence of the system and "grows" this hidden layer one neuron at a time. In the end, by encompassing a very small number of networks trained at this set of adaptively-selected parameter values, the meta-network is capable of generating surrogate solutions for the parametric system across the entire parameter domain accurately and efficiently.

Keywords: *Physics-informed neural network - attention mechanism - partial differential equations - deep learning*

***R**eceived 06 May., 2025; **R**evised 15 May., 2025; **A**ccepted 17 May., 2025 © The author(s) 2025.*
P**ublished with open access at **www.questjournas.org

I. Introduction

Solving nonlinear partial differential equations (PDEs) with multiple solutions is essential in various fields, including physics, biology, and engineering. However, traditional numerical methods, such as finite element and finite difference methods, often face challenges when dealing with nonlinear solvers, particularly in the presence of multiple solutions. These methods can become computationally expensive, especially when relying on solvers like Newton's method, which may struggle with ill-posedness near bifurcation points. In this paper, we propose a novel approach, the Newton Informed Neural Operator, which learns the Newton solver for nonlinear PDEs. Our method integrates traditional numerical techniques with the Newton nonlinear solver, efficiently learning the nonlinear mapping at each iteration. This approach allows us to compute multiple solutions in a single learning process while requiring fewer supervised data points than existing neural network methods.

(1-1) Approximate solution of partial differential equations using Physics-Informed Neural Network
(1-1-1) Problem formulation

The problem of interest is that of two immiscible fluids (oil and water) flowing through a porous medium (sand). The Buckley-Leverett (BL) equation describes the evolution in time and space of the wetting-phase (water) saturation.
Let $u_M: R_0^+ \times R_0^+ \to [0,1]$

$$\frac{\partial u_M}{\partial t}(x,t) + \frac{\partial f_M}{\partial x}(x,t) = 0, \tag{1}$$

$$u_M(x,0) = 0, \forall x > 0, \quad \textbf{Initial condition} \tag{2}$$

$$u_M(0,t) = 1, \forall t \geq 0, \quad \textbf{Boundary condition} \tag{3}$$

where u_M usually represents the wetting-phase saturation, f_M is the fractional flow function and M is the mobility ratio of the two fluid phases.
This first-order hyperbolic equation is of interest as its solution can display both smooth solutions (rarefactions) and sharp fronts (shocks). Although the solution to this problem can be calculated analytically, the precise and stable resolution of these shocks poses well-known challenges for numerical methods .
Physics-Informed Neural Networks (PINNs) have been tested on this problem by Fuks and Tchelepi who report good performance for concave fractional flow functions. The solution of the problem in the case of a non-concave fractional flow function is, however, much more challenging and remains an open problem. We take f_M to be the non-concave flux function

$$f_M(x,t) = \frac{u_M(x,t)^2}{u_M(x,t)^2 + \frac{1}{M}\left(1 - u_M(x,t)\right)^2}, \tag{4}$$

for which we can obtain the analytical solution of the problem:

$$u_M(x,t) = \begin{cases} 0, & \frac{x}{t} > f'_M(u^*), \\ u(x/t), & f'_M(u^*) \geq \frac{x}{t} \geq f'_M(u=1), \\ 1, & f'_M(u=1) \geq \frac{x}{t}, \end{cases} \tag{5}$$

where u^* represents the shock location defined by the Rankine-Hugoniot condition [27].

(1-1-2) Methodology
Let $\mathcal{G} := \{(x_i, t_j) \in R_0^+ \times R_0^+ : i = 0, \dots\dots, N, j = 0, \dots\dots, T\}$ be a discrete version of the domain of u_M. We define our PIANN as a vector function $u_\theta: R_0^+ \times R_0^+ \to [0,1]^{N+1}$, where θ are the weights of the network to be estimated during training. The inputs for the proposed architecture are pairs of (t, M) and the output is a vector where the i-th component is the solution evaluated in x_i. Notice the different treatment applied to spatial and temporal coordinates. Where as t is a variable of the vector function u_θ , the locations where we calculated the solution $x_0, \dots\dots, x_N$ are fixed in advance. The output is a saturation map and therefore its values have to be in the interval [0,1].
For the sake of readability, we introduce the architecture of u_θ in section 4. However, we advance that in order to enforce the boundary condition, we let our PIANN learn only the components $u_\theta(t,M)_1, \dots\dots, u_\theta(t,M)_N, \forall t \neq 0$ and then we concatenate the component $u_\theta(t,M)_0 = 1$. To enforce the initial conditions, we set $u_\theta(t,M)_i = 0, i = 1, \dots\dots, N$.To enforce that the solution be in the interval [0,1] , a sigmoid activation function is applied to each component of the last layer of our PIANN.
The parameters of the PIANN are estimated according to the physics-informed learning approach, which states that θ can be estimated from the BL equation eq. (1), the initial conditions eq. (2) and boundary conditions eq. (3), or in other words, no examples of the solution are needed to train a PINN.
After utilizing the information provided by the initial and boundary conditions enforcing $u_\theta(t,M)_i = 0, i = 1, \dots\dots, N$ and $u_\theta(t,M)_0 = 1$, respectively, we now define a loss function based on the information provided by eq. (1). To calculate the first term we propose two options. The first option is a central finite difference approximation, that is,

$$\mathcal{R}_1(\theta, M)_{i,j} = \frac{\mathbf{u_\theta}(t_{j+1}, M)_i - \mathbf{u_\theta}(t_j, M)_i}{t_{j+1} - t_j}, \quad \begin{matrix} i=1,\dots,N-1 \\ j=1,\dots,T-1 \end{matrix}. \tag{6}$$

Alternatively, we can calculate the derivative of our PIANN with respect to t since we know the functional form of u_θ. It can be calculated using the automatic differentiation tools included in many machine learning libraries, such as Pytorch. Thus, we propose a second option to calculate this term as $\mathcal{R}_1(\theta, M)_{i,j} = \partial u_\theta(t, M)_i / \partial t|_{t=t_j}$.

The second term of eq. (1), the derivative of the flux with respect to the spatial coordinate, is approximated using central finite difference as

$$\mathcal{R}_2(\theta, M)_{i,j} = \frac{\mathbf{f}(t_j, M)_{i+1} - \mathbf{f}(t_j, M)_{i-1}}{x_{i+1} - x_{i-1}}, \quad \begin{matrix} i=1,\dots,N-1 \\ j=1,\dots,T-1 \end{matrix}, \tag{7}$$

where the vector of fluxes at the $i-th$ location x_i is calculated as

$$\mathbf{f}_{\boldsymbol{\theta}}(t, M)_i = \frac{\mathbf{u}_{\boldsymbol{\theta}}(t, M)_i^2}{\mathbf{u}_{\boldsymbol{\theta}}(t, M)_i^2 - \frac{(1-\mathbf{u}_{\boldsymbol{\theta}}(t,M)_i)^2}{M}}, i = 0, \dots N. \tag{8}$$

The spatial coordinate x is included as a fixed parameter in our architecture.

The loss function to estimate the parameters of the PINN is given as

$$\mathcal{L}(\theta) = \sum_M \|\mathcal{R}_1(\theta, M) + \mathcal{R}_2(\theta, M)\|_F^2, \tag{9}$$

where $\|\cdot\|_f$ is the Fröbenius norm.

It should be noted that unlike all previous physics-informed learning works in the literature (recall section 1), the initial and boundary conditions are not included in the loss function; they are already enforced in the architecture. This has three direct consequences. First, we are enforcing a stronger constraint that does not allow any error on the initial and boundary conditions. Second, the PIANN does not need to learn these conditions by itself, and it can concentrate only on learning the parameters that minimize the residuals of the BL equation. Third, since we only have the term of the residuals, there are no weights to be tuned to control the effect of the initial and boundary conditions in the final solution.

Finally, the parameters of the PIANN are estimated using ADAM optimizer to minimize eq. (9) with respect to θ .

(1-1-3) PIANN architecture [2,32,33]

Although it has been demonstrated that neural networks are universal function approximators, certain challenging problems (e.g. solving non-linear PDEs) may require more specific architectures to capture all their properties. For that reason, we have proposed a new architecture, inspired by , to solve non-linear PDEs with discontinuities under two assumptions.

First, to automatically detect discontinuities we need an architecture that can exploit the correlations between the values of the solution for all spatial locations $x_1, \dots \dots, x_N$. Second, the architecture has to be flexible enough to capture different behaviors of the solution at different regions of the domain. To this end, we propose the use of encoder-decoder GRUs for predicting the solution at all locations at once, with the use of a recent machine learning tool known as attention mechanisms .

Our approach presents several advantages compared to traditional simulators: i) Instead of using just neighboring cells' information to calculate u as in numerical methods, our architecture uses the complete encoded sequence input of the grid to obtain u_i, allowing us to capture non-local relationships that numerical methods struggle to identify. ii) the computer time for the forward pass of neural networks models is linear with respect to the number of cells in our grid. In other words, our method is a faster alternative with respect to traditional methods of solving PDEs.

Figure 1 shows an outline of the proposed architecture. We start feeding the input pair (t, M) to a single fully connected layer. Thus, we obtain h^0 the initial hidden state of a sequence of N GRU blocks (yellow). Each of them corresponds to a spatial coordinate x_i which is combined with the previous hidden state h^{i-1} inside the block.

This generates a set of vectors $y^1, \ldots\ldots, y^N$ which can be understood as a representation of the input in a latent space. The definitive solution u (we omit the subindex $u\theta$ for simplicity) is reached after a new sequence of GRU blocks (blue) whose initial hidden state d^0 is initialized as h^N to preserve the memory of the system.

In addition to the hidden state d^i, the $i-th$ block g_i is fed with a concatenation of the solution at the previous location and a context vector, that is

$$\mathbf{u}_i = g_i([\mathbf{u}_{i-1}, \mathbf{c}^i], \mathbf{d}^{i-1}). \tag{10}$$

How the context vector is obtained is one of the key aspects of our architecture, since it will provide the PINN with enough flexibility to fit to the different behaviors of the solution depending on the region of the domain. Inspired by , we introduce an attention mechanism between both GRU block sequences. Our attention mechanism is a single fully connected layer, a, that learns the relationship between each component of yj and the hidden states of the (blue) GRU sequence,

$$\mathcal{E}_{i,j} = a(\mathbf{d}^{i-1}, \mathbf{y}^j). \tag{11}$$

Then, the rows of matrix ε are normalized using a softmax function as

$$\boldsymbol{\alpha}_{i,j} = \frac{\exp(\mathcal{E}_{i,j})}{\sum_{j=1}^{N} \exp(\mathcal{E}_{i,j})}, \tag{12}$$

and the context vectors are calculated as

$$\mathbf{c}^i = \sum_{j=1}^{N} \boldsymbol{\alpha}_{i,j} \mathbf{y}^j, i = 1, \ldots, N. \tag{13}$$

The coefficients $\alpha_{i,j}$ can be understood as the degree of influence of the component y^j in the output u_i. This is one of the main innovations of our work to solve hyperbolic equations with discontinuities. The attention mechanism automatically determines the most relevant encoded information of the full sequence of the input data to predict the u_i. In other words, attention mechanism is a new method that allows one to determine the location of the shock automatically and provide more accurate behavior of the PIANN model around this location. This new methodology breaks the limitations explored by other authors since is able to capture the discontinuity without specific prior information or the regularization term of the residual. This is the first paper to use attention mechanisms to solve non-linear PDEs for hyperbolic problems with discontinuity.

(1-1-4) Results

In this section we perform a set of experiments that support the proposed methodology. The goal of our experiments is to demonstrate that our PIANN is indeed able to approximate the analytical solution given in eq. (5).

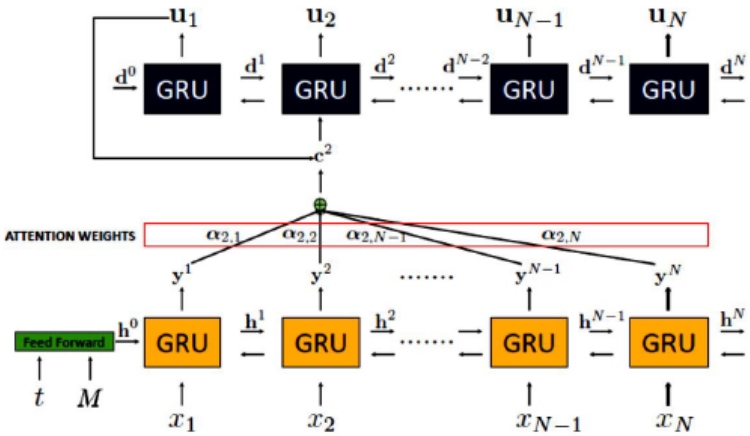


Figure 1: Architecture of physical attention neural network for the prediction of the variable u_2

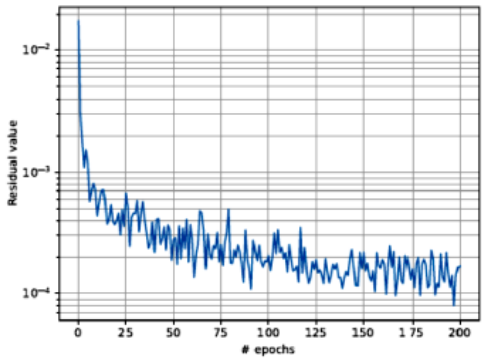


Figure 2: Residual values for each epoch for $M = 4.5$ values in a semilog scale.

The training set is given by a grid $\mathcal{G} := \{(x_i, t_j) \in R_0^+ \times R_0^+ : x_i \in \{0,0.01,\dots,0.99,1\}, t_j \in \{0,0.01,\dots,0.49,0.5\}$, and a set of values of $M \in \{2,4,6,\dots,100\}$, which produces $N = 101, T = 51$, and a total of 257,550 points. We want to emphasize that no examples of the solution are known at these points, and therefore no expensive and slow simulators are required to build the training set. To estimate the parameters of the PIANN we minimize eq. (9) by running ADAM optimizer for 200 epochs with a learning rate of 0.001.

Figure 2 shows the residual value for the testing dataset for the different epoch for $M = 4.5$. We can observe a fast convergence of the method and a cumulative value of the residual smaller than 10^{-4} after a few epochs. This demonstrates that we are minimizing the residuals in eq. (1) and subsequently solving the the equation that governs BL.

Figure 3 shows the comparison between the analytical solution (red) and the solution obtained by our PIANN (blue) for different M used during training. Top, middle and bottom rows correspond to $M = 2, M = 48$ and $M = 98$, respectively, and the columns from left to right, correspond to different time steps $t = 0.04, t = 0.20$, and $t = 0.40$, respectively. We can distinguish three regions of interest in the solution. Following increasing x coordinates, the first region on the left is one where the water saturation varies smoothly following the rarefaction part of the solution. The second region is a sharp saturation change that corresponds to the shock in the solution and the third region is ahead of the shock, with undisturbed water saturation values that are still at zero. For all cases, we observe that the PIANN properly learns the correct rarefaction behavior of the first region and approximates the analytical solution extremely well. In the third region, the PIANN also fits to the analytical solution perfectly and displays an undisturbed water saturation at zero. As for any classical numerical methods, the shock region is the most challenging to resolve.

Around the shock, the PIANN seems unable to perfectly resolve a sharp front, and the represented behavior is a shock that is smoothed over and displays non-monotonic artifacts upstream and downstream of the front. The location of the shock is, however, well captured. Such a behavior is reminiscent of the behavior observed in higher-order finite difference methods, where slope-limiters are often used to correct for the non-physical oscillations.

Importantly, it means the PIANN needs to learn where the shock is located in order to fit differently to both sides of it. This is the role of the attention mechanism of our architecture. On top of each figure we have visualized the attention map introduced by for every timestep. These maps visualize the attention weight $\alpha_{i,j}$ to predict the variable ui. We observe that in all cases the attention mechanism identifies the discontinuity, water front, and subsequently modifies the behavior of the network in the three different regions described above. This shows that attention mechanism provides all the necessary information to capture discontinuities

automatically without the necessity of training data or a prior knowledge. Finally, it is important to note that attention weights of the attention mechanism are constant when the shock/discontinuity disappears.

In addition, we test the behavior of our methodology to provide solutions for BL at points within the training range of M : $M = 4.5$ and $M = 71$. In other words, we want to check the capability of our PIANN model to interpolate solutions. Figure 5 shows that our PIANNs provide solutions that correctly detect the shock.

We also test the PIANN to extrapolate solutions out of the range of the training set: $M = 140$, $M = 250$ and $M = 500$. Figure 6 shows a degradation of the results when M is far from the original training set. We observe that our method predicts the behavior of the shock for $M = 140$. However, the shock is totally missed for $M = 500$ and as such, retraining the model is recommended with higher values of M. It is important to note that the results show that our method is stable and converges for the different cases.

We test how the neural residual error progresses based on different Δt and Δx resolutions. Results are shown in table 1, and demonstrate that our PIANN obtains smaller residuals when the resolution of the training set increases. However, we observe that changes in the residual are not highly significant. This is an advantage with respect to traditional numerical methods such as CFD, where smaller values of Δt are necessary to capture the shock and guarantee convergence and stability.

Finally, we have compared the results with central and upwind finite difference schemes for the term of the vector of fluxes. The first-order upwind difference introduces a dissipation error when applied to the residual of the Buckley- Leverett equation, which is equivalent to regularizing the problem via artificial diffusion. Figure 4 shows that both approaches present similar results respect to the analytical solution. The fact that both central and upwind differences yield similar predictions is important, because it suggests that the proposed PIANN approach does not rely on artificial dissipation for shock capturing.

(1-1-5) Discussion

In this work, we have introduced a new method to solve hyperbolic PDEs. We propose a new perspective by focusing on network architectures rather than on residual regularization. We call our new architecture a physics informed attention neural network (PIANN).

PIANN's novel architecture is based on two assumptions. First, correlations between values of the solution at all the spatial locations must be exploited, and second, the architecture has to be flexible enough to identify the shock and capture different behaviors of the solution at different regions of the domain. We have proposed an encoder-decoder GRU-based network to use the most relevant information of the fully encoded information, combined with the use of an attention mechanism. The attention mechanism is responsible for identifying the shock location and adapting the behavior of the PIANN model.

Unlike previous methods in the literature, the loss function of PIANNs is based solely on the residuals of the PDE, and the initial and boundary conditions are introduced in the architecture. These are stronger constraints than the ones enforced by previous methods, since we do not allow room for learning error on the initial or boundary conditions. As a result, PIANN's training aims only at minimizing the residual of the PDE; no hyperparameters are needed to control the effect of initial and boundary conditions on the solution.

We have applied the proposed methodology to the non-concave flux Buckley-Leverett problem, which has hitherto been an open problem for PINNs. The experimental results support the validity of the proposed methodology and conclude that: i) during training, the residuals of the equation decrease quickly to values smaller than 10^{-4} , which means that our methodology is indeed solving the differential equation, ii) the attention mechanism automatically detects shock waves

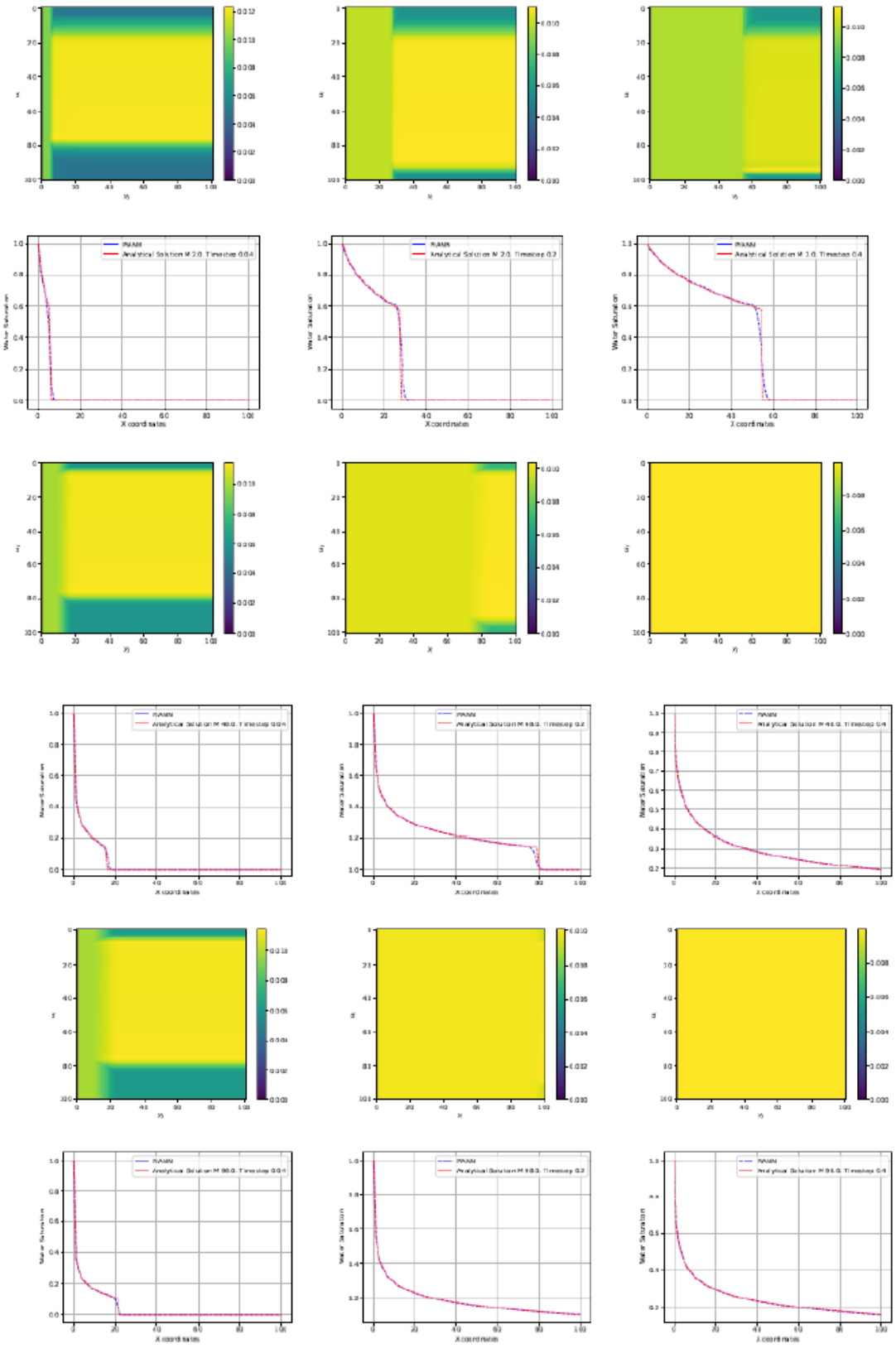


Figure 3: Top and bottom rows correspond to $M = 2$ and $M = 48$ and $M = 98$ for attention weights map and comparison of the predicted by the neural network and the exact solutions of the PDE, respectively. The columns from left to right, correspond to different time steps $t = 0.04, t = 0.20$ and $t = 0.40$

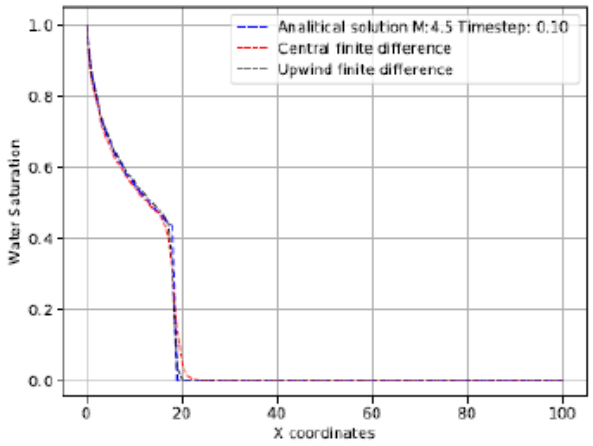


Figure 4: Comparison between solutions obtained with residuals evaluated using central and upwind finite differences, for $M = 4.5$.

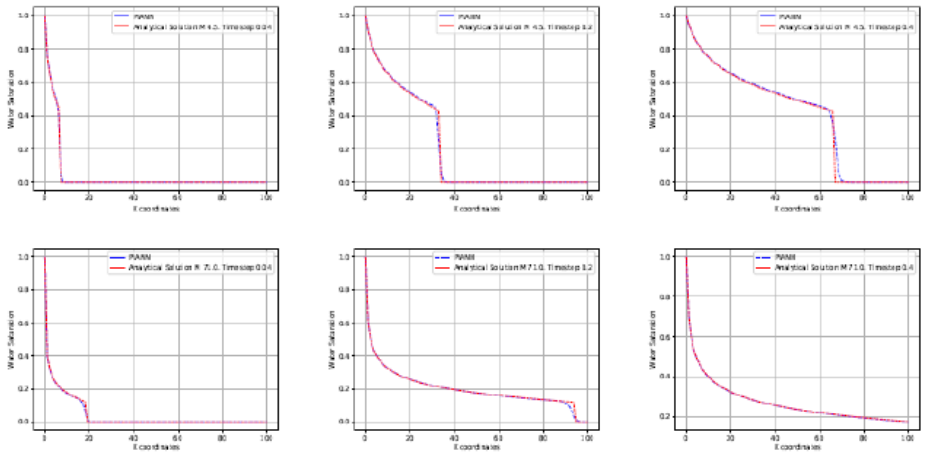


Figure 5: Top and bottom rows correspond to $M = 4.5$ and $M = 71$ for comparison of the predicted by the neural network and the exact solutions of the PDE, respectively. The columns from left to right, correspond to different time steps
$t = 0.04, t = 0.20$ and $t = 0.40$

of the solution and allows the PIANN to fit to the different behaviors of the analytical solution, and iii) the PIANN is able to interpolate solutions for values of the mobility ratio M inside the range of training set, as well as to extrapolate when the value of M is outside the range. However, we observe that if M is too far away from range of the training set, the quality of the solution decreases. In that case, a retraining of the network is recommended. iv) We observe that the residuals decrease when the resolution of the training set increases. However, the change in the residuals is not highly significant. This is advantageous with respect to traditional numerical methods where small values of Δt are needed to capture the shock and guarantee convergence and stability.

In conclusion, the proposed methodology is not confined by the current limitations of deep learning for solving hyperbolic PDEs with shock waves, and opens the door to applying these techniques to real-world problems, such as challenging reservoir simulations or carbon sequestration. It is plausible that this method could be applied to model many processes in other domains which are described by non-linear PDEs with shock waves.

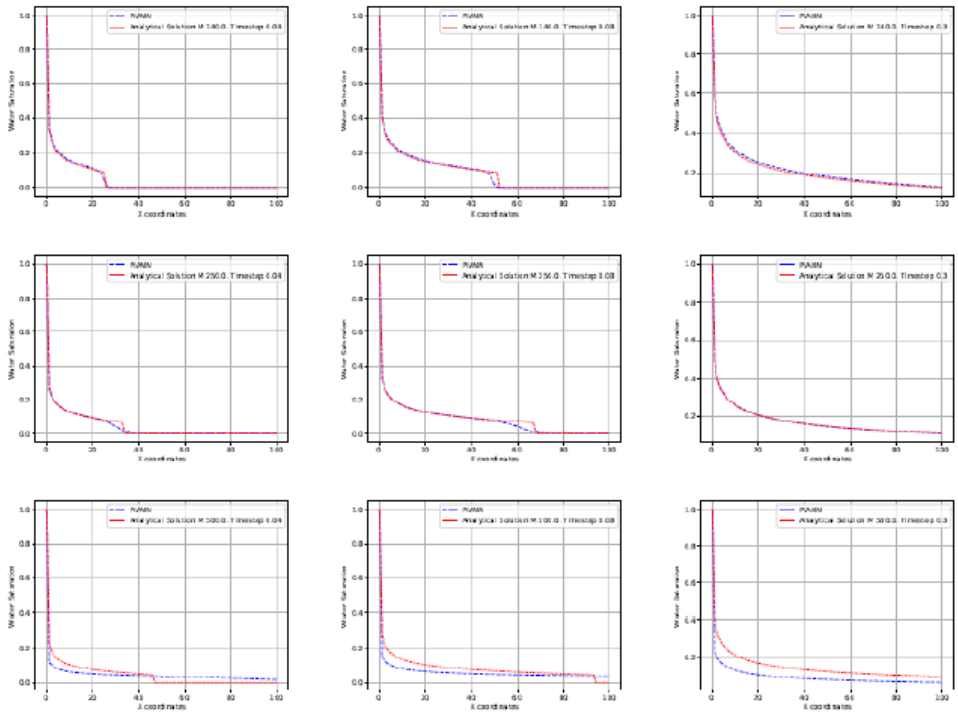


Figure 6: Top and bottom rows correspond to $M = 140$ and $M = 250$ and $M = 500$ comparison of the predicted by the neural network and the exact solutions of the PDE, respectively. The columns from left to right, correspond to different time steps $t = 0.04, t = 0.08$ *and* $t = 0.30$

Table 1: Residual calculation for different resolution of Δt and Δx for $M = 4.5$

Resolution	Residual Error
$\Delta x = 1 * 10^{-2}\ \Delta t = 1 * 10^{-2}$	$1 * 10^{-4}$
$\Delta x = 5 * 10^{-3}\ \Delta t = 5 * 10^{-3}$	$9 * 10^{-5}$
$\Delta x = 1 * 10^{-3}\ \Delta t = 1 * 10^{-3}$	$8.7 * 10^{-5}$

(1-2) GPT-PINN: Generative Pre-Trained Physics-Informed Neural Networks toward non-intrusive Meta-learning of parametric PDEs

(1-2-1) Reduced Basis Method

RBM is a linear reduction method that has been a popular option for rigorously and efficiently simulating parametric PDEs. Its hallmark feature is a greedy algorithm embedded in an offlineonline decomposition procedure. The offline (i.e. training) stage is devoted to a judicious and error estimate-driven exploration of the parameter-induced solution manifold. It selects a number of representative parameter values via a mathematically rigorous greedy algorithm . During the online stage, a reduced solution is sought in the terminal surrogate space for each unseen parameter value. Moreover, unlike other reduction techniques (e.g. proper orthogonal decomposition (POD)-based approaches), the number of full order inquiries RBM takes offline is minimum, i.e. equal to the dimension of the surrogate space. To demonstrate the main ideas, we consider a generic parameterized PDE as follows

$$\mathcal{F}(u; \mathbf{x}, \boldsymbol{\mu}) = f, \quad x \in \Omega \subseteq \mathbb{R}^{d}. \tag{1}$$

Here $\mathcal{F}$ encodes a differential operator parameterized via $\mu \in D \subset R^{d_s}$ together with necessary boundary and initial conditions. The parameter can be equation coefficients, initial values, source terms, or uncertainties in the PDE for the tasks of the uncertainty quantification, etc. $\mathcal{F}$ can depend on the solution and its (space- and time-) derivatives of various orders. We assume that we have available a numerical solution $u(x;\mu) \in X_h$ obtained by a high fidelity solver, called Full Order Model (FOM) and denoted as $FOM(\mu, X_h)$, and X_h is the discrete approximation space the numerical solution u belongs to.

A large number of queries of $u(\cdot,\mu)$ can be prohibitively expensive because the $FOM(\mu, X_h)$ has to be called many times. Model order reduction (MOR) aims to mitigate this cost by building efficient surrogates. One idea is to study the map

$$\mu \mapsto u(\cdot,\mu) \in X_h$$

and devise an algorithm to compute an approximation $u_N(\cdot,\mu)$ from an N-dimensional subspace X_N of X_h , such that

$$u_N(\cdot,\mu) \approx u(\cdot,\mu) for\ all\ \mu \in D$$

This reduced order model (ROM) formulation at a given μ is denoted by ROM (μ, X_N) , and is much cheaper to solve than FOM (μ, X_h) and can be conducted during the Online stage.

Algorithm 1 Classical RBM for parametric PDE (1): Offline stage

Input: A (random or given) $\boldsymbol{\mu}^1$, training set $\Xi \subset \mathcal{D}$.

Initialization: Solve FOM$(\boldsymbol{\mu}^1, X_h)$ and set $X_1 = \text{span}\{u(\cdot;\boldsymbol{\mu_1})\}$, $n = 2$.

1: **while** *stopping criteria not met,* **do**
2: Solve ROM$(\boldsymbol{\mu}, X_{n-1})$ for all $\boldsymbol{\mu} \in \Xi$ and compute error indicators $\Delta_{n-1}(\boldsymbol{\mu})$.
3: Choose $\boldsymbol{\mu}^n = \arg\max_{\mu \in \Xi} \Delta_{n-1}(\boldsymbol{\mu})$.
4: Solve FOM$(\boldsymbol{\mu}_n, X_h)$ and update $X_n = X_{n-1} \bigoplus \{u(\cdot;\boldsymbol{\mu}_n)\}$.
5: Set $n \leftarrow n + 1$.
6: **end while**

Output: Reduced basis set X_N, with N being the terminal index.

The success of RBM relies on the assumption that $u(\cdot\,; D)$ has small Kolmogorov N-width [34], defined as

$$d_N\left[u\left(\cdot;\mathcal{D}\right)\right] := \inf_{\substack{X_N \subset X_h \\ \dim X_N = N}} \sup_{\mu \in \mathcal{D}} \inf_{v \in X_N} \|u(\cdot,\boldsymbol{\mu}) - v\|_X .$$

A small d_N means that the solution to eq. (1) for any μ can be well-approximated from X_N that represents the outer infimum above. The identification of a near-infimizing subspace X_N is one of the central goals of RBM, and is obtained in the so-called Offline stage. RBM uses a greedy algorithm to find such X_N. The main ingredients are presented in Algorithm 1. The method explores the training parameter set $\Xi \subset D$ guided by an error estimate or an efficient and effective error indicator $\Delta_n(\mu)$ and intelligently choosing the parameter ensemble $\{\mu^n\}_{n=1}^N$ so that

$$X_N := \text{span}\{u(\cdot;\boldsymbol{\mu}^n)\}_{n=1}^N, \text{ and } u_N(\cdot,\boldsymbol{\mu}) = \sum_{n=1}^{N} c_n(\boldsymbol{\mu})u(\cdot,\boldsymbol{\mu}^n). \quad (2)$$

An offline-online decomposed framework is key to realize the speedup. Equipped with this robust and tested greedy algorithm, physics-informed reduced solver, rigorous error analysis, and certifiable convergence guarantees, RBM algorithms have become the go-to option for efficiently simulating parametric PDEs and established in the modern scientific computing toolbox and have benefited from voluminous research with theoretical and algorithmic refinement . One particular such development was the empirical error indicator of the L1-based RBM by Chen and his collaborators where $\Delta_{n-1}(\mu)$ was taken to be $\| c(\mu) \|_1$.
Here $c(\mu)$ is the coefficient vector of $u_N(\cdot,\mu)$ under the basis $\{u(\cdot;\mu n)\}_{n=1}^N$ *and* $\| \cdot \|_1$ represents the $\ell 1$-norm. As shown in , $c(\mu)$ represents a Lagrange interpolation basis in the parameter space implying that the indicator Δ_n represents the corresponding Lebesgue constant . The L1 strategy to select the parameter samples then controls the growth of the Lebesgue constants and hence is key toward accurate interpolation. This strategy, "free" to compute albeit not as traditionally rigorous, inspires the greedy algorithm of our GPT-PINN, to be detailed in Section 3.

(1-2-2) Deep neural networks

Deep neural networks (DNN) have seen tremendous success recently when serving as universal approximators to the solution function (or certain quantity of interest (QoI) / observable) . First proposed in on an underlying collocation approach, it has been successfully used recently in different contexts. See and references therein. For a nonparametrized version (e.g. eq. (1) with a fixed parameter value), we search for a neural network $\Psi_{NN}(x)$ which maps the coordinate $x \in R^d$ to a surrogate of the solution, that is $\Psi_{NN}(x) \approx u(x)$.
Specifically, for an input vector x , a feedforward neural network maps it to an output, via layers of "neurons" with layer k corresponding to an affine-linear map C_k composed with scalar non-linear activation functions σ. That is,

$$\Psi_{\mathbf{NN}}^{\theta}(\mathbf{x}) = C_K \circ \sigma \circ C_{K-1} \ldots\ldots \circ \sigma \circ C_1(\mathbf{x}).$$

A justifiably popular choice is the $ReLU$ activation $\sigma(z) = max(z,0)$ that is understood as component-wise operation when z is a vector. For any $1 \le k \le K$, we define

$$C_k z_k = W_k z_k + b_k, \quad \text{for } W_k \in \mathbb{R}^{d_{k+1}\times d_k}, z_k \in \mathbb{R}^{d_k}, b_k \in \mathbb{R}^{d_{k+1}}.$$

To be consistent with the input-output dimension, we set $d_1 = d$ and $d_K = 1$. We concatenate the tunable weights and biases for our network and denote them as

$$\theta := \{W_k, b_k\}, \quad \forall\, 1 \le k \le K.$$

We have $\theta \in \Theta \subset \mathbb{R}^M$ with $M := \sum_{k=1}^{K-1}(d_k+1)d_{k+1}$. We denote this network by

$$\mathbf{NN}(d_1, d_2, \cdots, d_K). \quad (3)$$

Learning $\Psi_{NN}^{\theta}(x)$ then amounts to generating training data and determining the weights and biases θ by optimizing a loss function using this data.

(1-2-3) Physics-Informed Neural Network

We define our problem on the spatial domain $\Omega \subset R^d$ with boundary $\partial\Omega$, and consider timedependent PDEs with order of time-derivative $k = 1$ *or* 2 .

$$\begin{aligned} \frac{\partial^k}{\partial t^k}u(\mathbf{x},t) + \mathcal{F}[u(\mathbf{x},t)] &= 0 \qquad \mathbf{x}\in\Omega, \qquad t\in[0,T], \\ \mathcal{G}(u)(\mathbf{x},t) &= 0 \qquad \mathbf{x}\in\partial\Omega, \qquad t\in[0,T], \\ u(\mathbf{x},0) &= u_0(\mathbf{x}) \qquad \mathbf{x}\in\Omega. \end{aligned} \quad (4)$$

Here $\mathcal{F}$ is a differential operator as defined in Section 2.1 and $\mathcal{G}$ denotes a boundary operator. The goal of a PINN is to identify an approximate solution $u(x,t)$ via a neural network $\Psi_{NN}^{\theta}(x,t)$. Learning $\theta \in R^M$

requires defining a loss function whose minimum θ^* leads to $\Psi_{NN}^{\theta^*}$ approximating the solution to the PDE over the problem domain. PINN defines this loss as a sum of three parts, an integral of the local residual of the differential equation over the problem domain, that over the boundary, and the deviation from the given initial condition,

$$\mathcal{J}(u) = \int_{\Omega} \left\| \frac{\partial^k}{\partial t^k} u(\mathbf{x},t) + \mathcal{F}(u)(\mathbf{x},t) \right\|_2^2 + \|u(\mathbf{x},0) - u_0(\mathbf{x})\|_2^2 \, dx + \int_{\partial\Omega} \|\mathcal{G}(u)(\mathbf{x},t)\|_2^2 \, dx.$$

During training, we sample collocation points in certain fashion from the PDE space domain Ω, space-time domain $\Omega \times (0,T)$, and boundary $\partial\Omega \times [0,T], Co \subset \Omega \times [0,T]$ *and* $C_\partial \subset \partial\Omega \times [0,T]$ *and* $C_i \subset \Omega$, and use them to form an approximation of the true loss.

$$\begin{aligned} \mathcal{L}_{\text{PINN}}(\Psi_{\text{NN}}^{\theta}) = & \frac{1}{|\mathcal{C}_o|} \sum_{(\mathbf{x},t)\in\mathcal{C}_o} \left\| \frac{\partial^k}{\partial t^k}(\Psi_{\text{NN}}^{\theta})(\mathbf{x},t) + \mathcal{F}(\Psi_{\text{NN}}^{\theta})(\mathbf{x},t) \right\|_2^2 + \\ & \frac{1}{|\mathcal{C}_\partial|} \sum_{(\mathbf{x},t)\in\mathcal{C}_\partial} \left\| \mathcal{G}(\Psi_{\text{NN}}^{\theta})(\mathbf{x},t) \right\|_2^2 + \frac{1}{|\mathcal{C}_i|} \sum_{\mathbf{x}\in\mathcal{C}_i} \left\| \Psi_{\text{NN}}^{\theta}(\mathbf{x},0) - u_0(\mathbf{x}) \right\|_2^2. \end{aligned} \tag{5}$$

When the training converges, we expect that LPINN (Ψ_{NN}^{θ}) should be nearly zero.

(1-2-4) The GPT-PINN framework

Inspired by the RBM formulation eq. (2), we design the GPT-PINN. Its two components and design philosophy are depicted in Figure 1. As a hyper-reduced feedforward neural network $NN(2,n,1)$ with $1 \le n \le N$ (see eq. (3) for the notation), we denoted it by $NN^r(2,n,1)$. A key feature is that it has customized activation function in the neurons of its sole hidden layer. These activation functions are nothing but the pre-trained PINNs for the corresponding PDEs instantiated by the parameter values $\{\mu^1, \mu^2, \ldots\ldots, \mu^n\}$ chosen by a greedy algorithm that is specifically tailored for PINNs but inspired by the classical one adopted by RBM in Algorithm 1. The design of network architecture represents the first main novelty of the paper. To the best of our knowledge, this is the first time a whole (pre-trained) network is used as the activation function of one neuron.

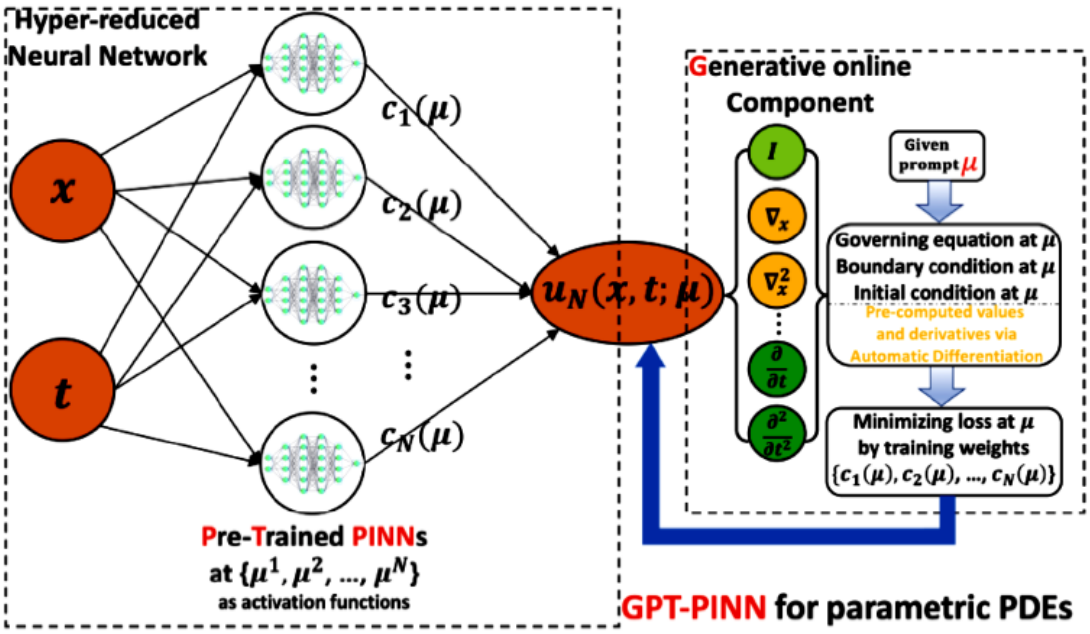


Figure 1: The GPT-PINN architecture. A hyper-reduced network adaptively embedding pretrained PINNs at the nodes of its sole hidden layer. It then allows a quick online generation of a surrogate solution at any given parameter value.

(1-2-5) The online solver of GPT-PINN

We first present the online solver, i.e. the training of the reduced network $NN^r(2,n,1)$, for any given μ . With the next subsection detailing how we "grow" the GPT-PINN offline from $NN^r(2,n,1)$ to $NN^r(2,n+1,1)$, we have a strategy of adaptively generating the terminal GPT-PINN, $NN^r(2,N,1)$. Indeed, given the simplicity of the reduced network, to train the weights $\{c_1(\mu), \dots\dots, c_n(\mu)\}$, no backpropagation is needed. The reason is that the loss function, similar to eq. (5), is a simple function containing directly and explicitly $\{c_1(\mu), \dots\dots, c_n(\mu)\}$ thanks to the reduced network structure of GPT-PINN. In fact, we denote by $\Psi_{NN}^{\theta^i}(\mathrm{x},\mathrm{t})$ the PINN approximation of the PDE solution when $\mu = \mu^i$. Given that $u_n(x,t;\mu) \approx \sum_{i=1}^n c_i(\mu)\,\Psi_{NN}^{\theta^i}(x,\mathrm{t})$[1], we can calculate the GPT-PINN loss as a function of the weights $c(\mu)$ as follows.

$$\mathcal{L}_{\mathrm{PINN}}^{\mathrm{GPT}}(\mathbf{c}(\boldsymbol{\mu})) = \frac{1}{|\mathcal{C}_o^r|}\sum_{(\mathbf{x},t)\in\mathcal{C}_o}\left\|\frac{\partial^k}{\partial t^k}\left(\sum_{i=1}^n c_i(\boldsymbol{\mu})\Psi_{\mathbf{NN}}^{\theta^i}\right)(\mathbf{x},t)+\mathcal{F}\left(\sum_{i=1}^n c_i(\boldsymbol{\mu})\Psi_{\mathbf{NN}}^{\theta^i}\right)(\mathbf{x},t)\right\|_2^2 + \frac{1}{|\mathcal{C}_\partial^r|}\sum_{(\mathbf{x},t)\in\mathcal{C}_\partial}\left\|\mathcal{G}\left(\sum_{i=1}^n c_i(\boldsymbol{\mu})\Psi_{\mathbf{NN}}^{\theta^i}\right)(\mathbf{x},t)\right\|_2^2 + \frac{1}{|\mathcal{C}_i^r|}\sum_{\mathbf{x}\in\mathcal{C}_i}\left\|\sum_{i=1}^n c_i(\boldsymbol{\mu})\Psi_{\mathbf{NN}}^{\theta^i}(\mathbf{x},0)-u_0(\mathbf{x})\right\|_2^2. \tag{6}$$

The online collocation sets $C_0^r \subset \Omega \times [0,\mathrm{T}]$, $C_\partial^r \subset \partial\Omega \times [0,\mathrm{T}]$ and $C_i^r \subset \Omega$ are used, similar to eq. (5), to generate an approximation of the true loss. They are taken to be the same as their full PINN counterparts C_0 , C_∂ , C_i in this paper but we note that they can be fully independent. The training of $NN^r(2,n,1)$ is then simply

$$\mathbf{c} \leftarrow \mathbf{c} - \delta_r \nabla_{\mathbf{c}} \mathcal{L}_{\mathrm{PINN}}^{\mathrm{GPT}}(\mathbf{c}) \tag{7}$$

Here $c = (c_1(\mu), \dots\dots, c_n(\mu))^T$ and δ_r is the online learning rate. The detailed calculations of eq. (6) and eq. (7) are given in A for the first numerical example. Those for the other examples are very similar and thus omitted. We make the following three remarks to conclude the online solver.

1. Precomputation for fast training of $NN^r(2,n,1)$: Due to the linearity of the derivative operations and the collocation nature of loss function, a significant amount of calculations of eq. (6) can be precomputed and stored. These include the function values and all (spatial and time) derivatives involved in the operators $\mathcal{F}$ and $\mathcal{G}$ of the PDE eq. (4):

$$\Psi_{\mathbf{NN}}^{\theta^i}(\mathcal{C}),\ \frac{\partial^k}{\partial t^k}\left(\Psi_{\mathbf{NN}}^{\theta^i}\right)(\mathcal{C})\ (k=1 \text{ or } 2),\ \nabla_{\mathbf{x}}^{\ell}\Psi_{\mathbf{NN}}^{\theta^i}(\mathcal{C})\ (\ell = 1,2,\cdots) \text{ for } \mathcal{C} = \mathcal{C}_o^r, \mathcal{C}_\partial^r, \mathcal{C}_i^r. \tag{8}$$

Once these are precomputed, updating c according to eq. (7) is very efficient. It can even be made independent of $|C|$.

2. Non-intrusiveness of GPT-PINN: It is clear that, once the quantities of eq. (8) are extracted from the full PINN, the online training of $NN^r(2,n,1)$ is independent of the full PINN. GPT-PINN is therefore non-intrusive of the Full Order Model. One manifestation of this property is that, as shown in our third numerical example, the full PINN can be adaptive while the reduced PINN may not be.

3. The error indication of $NN^r(2,n,1)$: One prominent feature of RBM is its a posteriori error estimators/indicators which guides the generation of the reduced solution space and certifies the accuracy of the surrogate solution. Inspired by this classical design, we introduce the following quantity that measures how accurate $NN^r(2,n,1)$ is in generating a surrogate network at a new parameter μ.

$$\Delta_{\mathbf{NN}}^r(\mathbf{c}(\boldsymbol{\mu})) \triangleq \mathcal{L}_{\mathrm{PINN}}^{\mathrm{GPT}}(\mathbf{c}(\boldsymbol{\mu})). \tag{9}$$

We remark that this quantity is essentially free since it is readily available when we train $NN^r(2,n,1)$ according to eq. (7). The adoption of the training loss of the meta-network as an error indicator, inspired by the residual-based error estimation for traditional numerical solvers such as FEM, represents the second main novelty of this paper.

(1-2-6) Training the reduced network GPT-PINN: the greedy algorithm

Algorithm 2 GPT_PINN for parametric PDE: Offline stage

Input: A (random or given) μ^1, training set $\Xi_{\text{train}} \subset \mathcal{D}$, full PINN.

1: Train a full PINN at μ^1 to obtain $\Psi_{\text{NN}}^{\theta^1}$. Precompute quantities necessary for $\nabla_{\mathbf{c}}\mathcal{L}_{\text{PINN}}^{\text{GPT}}$ at collocation nodes $\mathcal{C}_o^r$, $\mathcal{C}_\partial^r$, and $\mathcal{C}_i^r$, see eq. (8). Set $n = 2$.
2: **while** *stopping criteria not met,* **do**
3: Train $\mathsf{NN}^{\mathsf{r}}(2, n-1, 1)$ at μ for all $\mu \in \Xi_{\text{train}}$ and record the indicator $\Delta_{\mathsf{NN}}^r(\mathbf{c}(\mu))$.
4: Choose $\mu^n = \arg\max_{\mu \in \Xi_{\text{train}}} \Delta_{\mathsf{NN}}^r(\mu)$.
5: Train a full PINN at μ^n to obtain $\Psi_{\text{NN}}^{\theta^n}$. Precompute quantities necessary for $\nabla_{\mathbf{c}}\mathcal{L}_{\text{PINN}}^{\text{GPT}}$ at collocation nodes $\mathcal{C}_o^r$, $\mathcal{C}_\partial^r$, and $\mathcal{C}_i^r$, see eq. (8).
6: Update the GPT_PINN by adding a neuron to the hidden layer to construct $\mathsf{NN}^{\mathsf{r}}(2, n, 1)$.
7: Set $n \leftarrow n + 1$.
8: **end while**

Output: GPT_PINN $\mathsf{NN}^{\mathsf{r}}(2, N, 1)$, with N being the terminal index.

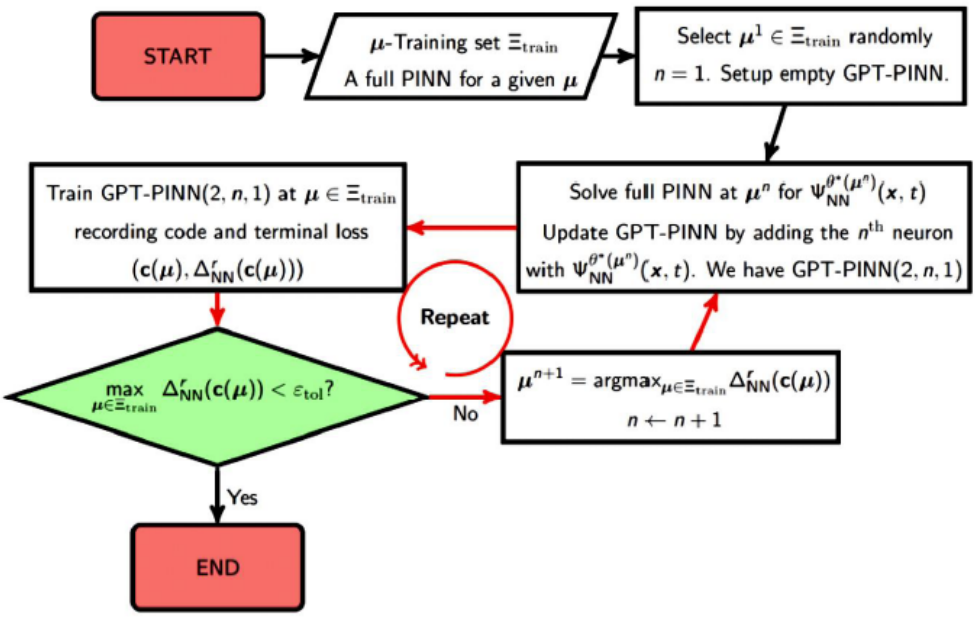


Figure 2: Flowchart of the GPT-PINN Offline training stage. With the online solver described in Section 3.1, we are ready to present our greedy algorithm. Its main steps are outlined in Algorithm 2 with its flowchart provided in Figure 2. The meta-network

adaptively "learns" the parametric dependence of the system and "grows" its sole hidden layer one neuron/network at a time in the following fashion. We first randomly select, in the discretized parameter domain Ξ_{train} , one parameter value μ^1 and train the associated (highly accurate) PINN $\Psi_{NN}^{\theta^1}$.. The algorithm then decides how to "grow" its meta-network by scanning the entire discrete parameter space Ξtrain and, for each parameter value, training this reduced network (of 1 hidden layer with 1 neuron $\Psi_{NN}^{\theta^1}$. As it scans, it records an error indicator $\Delta_{NN}^r c(\mu)$. The next parameter value μ^2 is the one generating the largest error indicator. The algorithm then proceeds by training a full PINN at μ^2 and therefore grows its hidden layer into two neurons with customized (but pre-trained) activation functions $\Psi_{NN}^{\theta^1}$ and $\Psi_{NN}^{\theta^2}$. This process is repeated until the stopping criteria is met which can be either that the error indicator is sufficiently small or a

pre-selected size of the reduced network is met. At every step, we select the parameter value that is approximated most badly by the current meta-network. We end by presenting how we initialize the weights $c(\mu)$ when we train $NN^r(2, n-1, 1)$ on Line 3 of Algorithm 2. They are initialized by a linear interpolation of up to 2^{d_s} closest neighbors of μ within the chosen parameter values $\{\mu^1, \ldots, \mu^N\}$. Recall that ds is the dimension of the parameter domain.

(1-2-7) Related work

The last two to three years have witnessed an increasing level of interest toward metalearning of (parameterized or unparameterized) PDEs due to the need of repeated simulations and the remarkable success of PINNs in its original form or adaptive ones. Here we mention a few representative ones and point out how our method differentiates from theirs.

Metalearning via PINN parameters. the authors adopt statistical (e.g. regression) and numerical (e.g. RBF/spline interpolation) methods to build a surrogate for the map from the PDE parameter μ to the PINN parameter (weights and biases, θ). They are shown to be superior than MAML for parameterized PDEs which was shown to outperform LEAP. Both are general-purpose meta-learning methods. However, the online solver (i.e. regression or interpolation) of ignores the physics (i.e. PDE). The method assumes that the μ-variation of the PINN weights and biases is analogous to that of the PDE solution.

DeepONet. Aiming to learn nonlinear operators, a DeepONet consists of two sub-networks, a branch net for encoding the input function (e.g source/control term, as opposed to PDE coefficients) at a fixed number of sensors, and a trunk net for encoding the locations for the output functions. It does not build in the physics represented by the dynamical system or PDE for a new input. Moreover, it is relatively data-intense by having to scan the entire input function space such as Gaussian random field or orthogonal polynomial space.

Metalearning loss functions. Authors concern the definition of the PINN loss functions. While it is in the parameterized PDE setting, the focus is a gradient-based approach to discover, during the offline stage, better PINN loss functions which are parameterized by e.g. the weights of each term in the composite objective function. The end goal is therefore improved PINN performance e.g. at unseen PDE parameters, due to the learned loss function configuration.

Metalearning initialization. the authors study the use of a meta network, across the parameter domain of a 1-D arc model of plasma simulations, to better initialize the PINN at a new task (i.e. parameter value).

MetaNO. The recent meta-learning approach for transferring knowledge between neural operators aims to transfer the learned network parameters $\theta(\mu)$ between different μ with only the first layer being retrained. Its resulting surrogate is fully data-driven, i.e. with no physics built in for a new value μ.

PRNN. The physics-reinforced neural network approach builds the map $\mu \mapsto c(\mu)$ via regression (i.e. no physics during the online evaluation for a new μ) although PDE residuals were considered during the supervised learning of the map via labelled data.

Our proposed GPT-PINN exploits the μ-variation of the PDE solution directly which may feature a Kolmogorov N-width friendlier to MOR approaches, see Figure 3, than the weights and biases. This is, in part, because that the weights and biases lie in a (much) higher dimensional space. Moreover, the meta-network of our approach, being a PINN itself, has physics automatically built in in the same fashion as the underlying PINNs. Lastly, our approach provides a surrogate solution to the unseen parameter values in addition to a better initialization transferred from the sampled PINNs. Most importantly, our proposed GPT-PINN embodies prior knowledge that is mathematically rigorous and PDE-pertinent into the network architecture. This produces strong inductive bias that usually leads to good generalization.

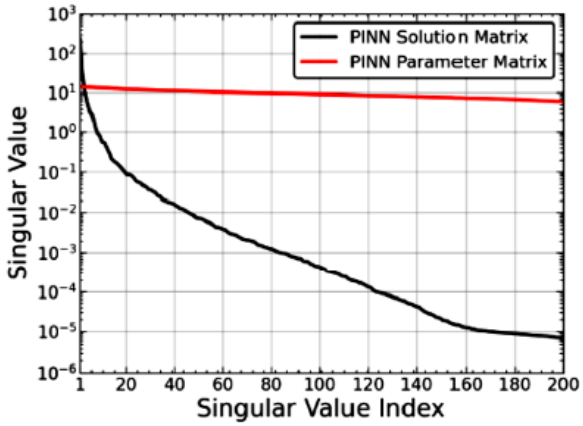


Figure 3: A motivating example showing that the solution matrix of a parametric PDE $\{u(\cdot,\mu^n)\}_{n=1}^{200}$ exhibits fast decay in its singular values (indicating fast decay of the Kolmogorov N-width of the solution manifold) while the network weights and biases manifold $\{\{\theta(\mu^n)\}_{n=1}^{200}\}200$ does not.

(1-3) Newton Informed Neural Operator for Solving Nonlinear Partial Differential Equations

(1-3-1) Nonlinear PDEs with multiple solutions

Significant mathematical models depicting natural phenomena in biology, physics, and materials science are rooted in nonlinear partial differential equations (PDEs) . These models, characterized by their inherent nonlinearity, present complex multi-solution challenges. Illustrative examples include string theory in physics, reaction-diffusion systems in chemistry, and pattern formation in biology . However, experimental techniques like synchrotronic and laser methods can only observe a subset of these multiple solutions. Thus, there is an urgent need to develop computational methods to unravel these nonlinear models, offering deeper insights into the underlying physics and biology . Consequently, efficient numerical techniques for identifying these solutions are pivotal in understanding these intricate systems. Despite recent advancements in numerical methods for solving nonlinear PDEs, significant computational challenges persist for large-scale systems. Specifically, the computational costs of employing Newton and Newton-like approaches are often prohibitive for the large-scale systems encountered in real-world applications. In response to these challenges , we propose an operator learning approach based on Newton's method to efficiently solve nonlinear PDEs.

(1-3-2) Related works

Indeed, there are numerous approaches to solving partial differential equations (PDEs) using neural networks. Broadly speaking, these methods can be categorized into two main types: function learning and operator learning.

In function learning, neural networks are used to directly approximate the solutions to PDEs. Function learning approaches aim to directly learn the solution function itself. On the other hand, in operator learning, the focus is on learning the operator that maps input parameters to the solution of the PDE. Instead of directly approximating the solution function, the neural network learns the underlying operator that governs the behavior of the system.

Function learning methods In function learning, a commonly employed method for addressing this problem involves the use of Physics-Informed Neural Network (PINN)-based learning approaches, as introduced by Raissi et al. and Deep Ritz Methods . However, in these methods, the task becomes particularly challenging due to the ill-posed nature of the problem arising from multiple solutions. Despite employing various initial data and training methods, attaining high accuracy in solution learning remains a complex endeavor. Even when a high-accuracy solution is achieved, each learning process typically results in the discovery of only one solution. The specific solution learned by the neural network is heavily influenced by the initial conditions and training methods employed. However, discerning the relationships between these factors and the learned solution remains a daunting task. the authors introduce HomPINNs for learning multiple solutions to PDEs, where the number of solutions that can be learned depends on the choice of "start functions." However, if the "start functions" are not appropriately selected, HomPINNs may fail to capture all solutions. In this paper, we present an operator learning approach combined with Newton's method to train the nonlinear solver. While this approach is not specifically designed for computing multiple solutions, it can be employed to compute them if suitable initial guesses are provided.

Operator learning methods Various approaches have been developed for operator learning to solve PDEs, including DeepONet , which integrates physical information , as well as techniques like FNO inspired by spectral methods, and MgNO , HANO , and WNO based on multilevel methods, and transformer-based neural operators . These methods focus on approximating the operator between the parameters and the solutions. Firstly, they require the solutions of PDEs to be unique; otherwise, the operator is not well-defined. Secondly, they focus on the relationship between the parameter functions and the solution, rather than the initial data and multiple solutions.

(1-3-3) Review of Newton Methods to Solve Nonlinear Partial Differential Equations

To tackle this problem Eq. (1), we employ Newton's method by linearizing the equation. For the Newton method applied to an operator, if we aim to find the solution of $\mathcal{F}(u) = 0$, the iteration can be written as:

$$\mathcal{F}'(u_n)u_{n+1} = \mathcal{F}'(u_n)u_n - \mathcal{F}(u_n) \Leftrightarrow \mathcal{F}'(u_n)\delta u = -\mathcal{F}(u_n)$$

where $\delta u = u_{n+1} - u_n$.

In this context, $\mathcal{F}'(u)v$ is the (Frechet) derivative of the operator, which is a linear operator to v , defined as follows :

To find $\mathcal{F}'(u)$ in $\mathcal{X}$, for any $v \in \mathcal{X}$,

$$\lim_{|v|\to 0} \frac{|\mathcal{F}(u+v) - \mathcal{F}(u) - \mathcal{F}'(u)v|}{|v|} = 0$$

where $|\cdot|$ denotes the norm in $\mathcal{X}$.

For solving Eq. (1), given any initial guess $u_0(x)$, for $i = 1, 2, \ldots, M$, in the $i-th$ iteration of Newton's method, we have $\tilde{u}(x) = u + \delta u(x)$ by solving

$$\begin{cases} \big(\mathcal{L} - f'(u)\big)\delta u = -\mathcal{L}u + f(u), x \in \Omega \\ \delta u = 0\,, \qquad\qquad\qquad\qquad x \in \delta\Omega \end{cases} \tag{2}$$

which is based on the fact that the (Frechet) derivative of $\mathcal{L} - f(\cdot)$ at u is $\mathcal{L} - f'(u)$. If Eq. (2) has a unique solution, then by solving Eq. (2) and repeating the process M times, we will obtain one of the solutions of the nonlinear equation (1). Denoting the mapping for u and δu, the solution of Eq. (2) with parameter u, as $\mathcal{G}(u) := \delta u$, we know that

$$\lim_{n\to\infty} (\mathcal{G} + \mathrm{Id})^n (u_0) = u^*\,,$$

where u^* is one of the solutions of Eq. (1). For different initial conditions, this process will converge to different solutions of Eq. (1), making this method suitable for finding multiple solutions. Furthermore , the Newton method is well-posed, meaning that each initial condition u_0 will converge to a single solution of Eq. (1) under appropriate assumptions (see Assumption 1). This approach helps to address the ill-posedness encountered when using PINNs directly to solve Eq. (1). However, repeatedly solving Eq. (1) can be computationally expensive, especially in high-dimensional cases or when a large number of discrete points are involved. In this paper, we tackle these challenges by employing neural networks.

(1-3-4) Neural Operator Structures

In this section, we introduce the structure of the neural operator to approximate the operator locally in the Newton methods from Eq.(2), i.e., $\delta u := \mathcal{G}(u)$, where δu is the solution of Eq.(2), which depends on u. If we can learn the operator G(u) well using the neural operator $\mathcal{O}(u;\theta)$, then for an initial function u0, assume the n-th iteration will approximate one solution, i.e., $(\mathcal{G} + \mathrm{Id})^n(u_0) \approx u^*$.Thus,

$$(\mathcal{O} + \mathrm{Id})^n(u_0) \approx (\mathcal{G} + \mathrm{Id})^n(u_0) \approx u^*$$

For another initial condition, we can evaluate our neural operator and find the solution directly.

Then we discuss how to train such an operator. To begin, we define the following shallow neural operators with p neurons for operators from $\mathcal{X}$ to $\mathcal{Y}$ as

$$\mathcal{O}(a;\theta) = \sum_{i=1}^{p} \mathcal{A}_i\, \sigma(\mathcal{W}_i a + \mathcal{B}_i) \qquad \forall a \in \mathcal{X} \tag{3}$$

where $\mathcal{W}_i \in \mathcal{L}(\mathcal{X},\mathcal{Y}), \mathcal{B}_i \in \mathcal{Y}, \mathcal{A}_i \in \mathcal{L}(\mathcal{Y},\mathcal{Y})$, and θ denote all the parameters in $\{\mathcal{W}_i, \mathcal{A}_i, \mathcal{B}_i\}_{i=1}^{p}$.

Here , $\mathcal{L}(\mathcal{X},\mathcal{Y})$ denotes the set of all bounded (continuous) linear operators between $\mathcal{X}$ and $\mathcal{Y}$, and $\sigma: \mapsto R$ defines the nonlinear point-wise activation.

In this paper, we will use shallow DeepONet to approximate the Newton operator. To provide a more precise description, in the shallow neural network, $\mathcal{W}_i$ represents an interpolation of operators. With proper and reasonable assumptions, we can present the following theorem to ensure that DeepONet can effectively approximate the Newton method operator. The proof will be provided in the appendix. Furthermore, MgNO is replaced by $\mathcal{W}$ as a multigrid operator , and FNO is some kind of kernel operator; our analysis can be generalized to such cases.

Before the proof, we need to establish some assumptions regarding the input space $\mathcal{X} \subset H^2(\Omega)$ of the operator and $f(u)$ in Eq. (1).

Assumption 1.

(i): For any $u \in \mathcal{X}$, we have that the linear equation $(\mathcal{L} - f'(u))\delta u = -\mathcal{L}u + f(u)$ is well-posed for solving δu.

(ii): There exists a constant F such that $\| f(x) \| W^{2,\infty}(R) \leq F$.

(iii): All coefficients in $\mathcal{L}$ are C^1 and $\partial\Omega \in C^2$.

(iv): $\mathcal{X}$ has a Schauder basis $\{b_k\}_{k=1}^{\infty}$, we define the canonical projection operator $\mathcal{P}_n$ based on this basis. The operator $\mathcal{P}_n$ projects any element $u \in \mathcal{X}$ onto the finite-dimensional subspace spanned by the first n basis elements $\{b_1, b_2, \ldots, b_n\}$. Specifically, for $u \in \mathcal{X}$, $u = \sum_{k=0}^{\infty} \alpha_k b_k$, let $\mathcal{P}_n(u) = \sum_{k=0}^{n} \alpha_k b_k$, where α_k are the coefficients in the expansion of u with respect to the basis $\{b_n\}$. The canonical projection operator $\mathcal{P}_n$ is a linear bounded operator on $\mathcal{X}$. According to the properties of Schauder bases, these projections $\mathcal{P}_n$ are uniformly bounded by some constant C.

Furthermore, we assume, for any $u \in \mathcal{X}, \epsilon > 0$ there exists a n such that

$$\|u - \mathcal{P}_n u\|_{H^2(\Omega)} \leq \epsilon \ , \qquad for\ all\ u \in \mathcal{X}$$

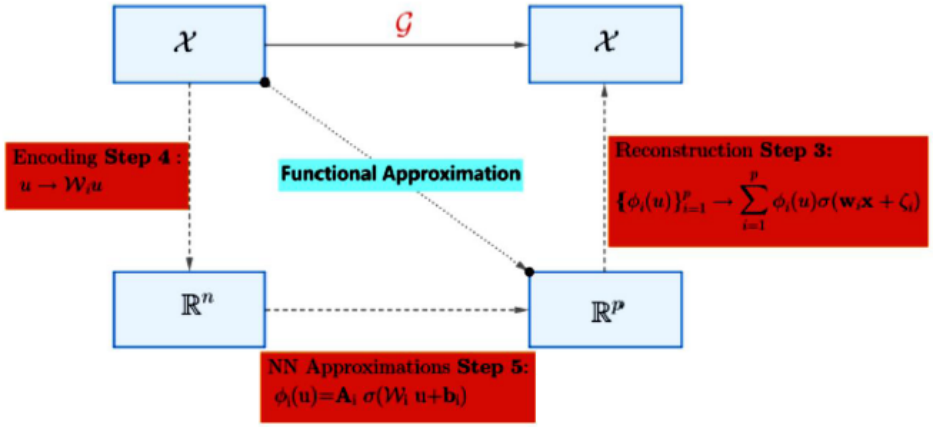


Figure 1: The sketch of proof for Theorem 1.

Theorem 1. Suppose $\mathcal{X} = \mathcal{Y} \subset H^2(\Omega)$ and Assumption 1 holds. Then, there exists a neural network $\mathcal{O}(a;\theta) \in \Xi p$ defined as

$$\Xi_p := \left\{ \sum_{i=1}^{p} \boldsymbol{A_i}\sigma\left(\mathcal{W}_i\boldsymbol{u} + \boldsymbol{b_i}\right)\sigma\left(\boldsymbol{w_i}\cdot\boldsymbol{x} + \zeta_i\right) | \mathcal{W}_i \in \mathcal{L}(\mathcal{X}, \mathbb{R}^m), \boldsymbol{b_i} \in \mathbb{R}^m, \boldsymbol{A_i} \in \mathbb{R}^{1\times m} \right\} \quad (4)$$

such that

$$\sup_{u\in\mathcal{X}} \left\| \sum_{i=1}^{p} \boldsymbol{A_i}\sigma\left(\mathcal{W}_i\boldsymbol{u} + \boldsymbol{b_i}\right)\sigma\left(\boldsymbol{w_i}\cdot\boldsymbol{x} + \zeta_i\right) - \mathcal{G}(u) \right\|_{L^2(\Omega)} \leq C_1 m^{-\frac{1}{n}} + C_2(\epsilon + p^{-\frac{2}{d}}), \quad (5)$$

where σ is a smooth non-polynomial activation function, n is shown in Assumption 1 and contained in $\mathcal{W}_i, C_1$ is a constant independent of m, ϵ, and p, C_2 is a constant depended on p, n and F (see in Assumption 1) is the scale of the P in Assumption 1. And ϵ depends on n . n and ϵ are defined in Assumption 1.

(1-3-5) Mean Square Loss

The Mean Square Error loss function is defined as:

$$\mathcal{E}_S(\theta) := \frac{1}{M_u \cdot M_x} \sum_{j=1}^{M_u} \sum_{k=1}^{M_x} \left| \mathcal{G}\left(u_j\right)\left(x_k\right) - \mathcal{O}\left(u_j;\theta\right)\left(x_k\right) \right|^2 \quad (6)$$

where $u_1, u_2, \ldots, u_{M_u} \sim \mu$ are independently and identically distributed (i.i.d) samples in $\mathcal{X}$, and $x_1, x_2, \ldots, x_{M_x}$ are uniformly i.i.d samples in Ω.

However, using only the Mean Squared Error loss function is not sufficient for training to learn the Newton

method, especially since in most cases, we do not have enough data $\{u_j, \mathcal{G}(u_j)\}_{u_j}^{M_u}$.

Furthermore, there are situations where we do not know how many solutions exist for the nonlinear equation (1). If the data is sparse around one of the solutions, it becomes impossible to effectively learn the Newton method around that solution.

Given that $\varepsilon_S(\theta)$ can be viewed as the finite data formula of $\varepsilon_{S_c}(\theta)$, where

$$\varepsilon_{S_c}(\theta) = \lim_{M_u, M_x \to \infty} \varepsilon_S(\theta)$$

The smallness of ε_{S_c} can be inferred from Theorem 1. To understand the gap between $\varepsilon_{S_c}(\theta)$ and $\varepsilon_S(\theta)$, we can rely on the following theorem. Before the proof, we need some assumptions about the data in $\varepsilon_S(\theta)$:

Assumption 2.

(i) Boundedness : For any neural network with bounded parameters, characterized by a bound B and dimension d_θ, there exists a function $\Psi: L^2(\Omega) \to [0, \infty)$ such that

$$|\mathcal{G}(u)(x)| \leqslant \Psi(u), \quad \sup_{\boldsymbol{\theta} \in [-B,B]^{d_\theta}} |\mathcal{O}(u;\boldsymbol{\theta})(x)| \leqslant \Psi(u), \quad \sup_{\boldsymbol{\theta} \in [-B,B]^{d_\theta}} |\mathcal{L}\mathcal{O}(u;\boldsymbol{\theta})(x)| \leqslant \Psi(u)$$

for all $u \in \mathcal{X}, x \in \Omega$, and there exist constants $C, \kappa > 0$, such that

$$\Psi(u) \leqslant C\,(1 + \|u\|_{H^2})^{\kappa}. \tag{7}$$

(ii) Lipschitz continuity : There exists a function $\Phi: L^2(\Omega) \to [0, \infty)$, such that

$$|\mathcal{O}(u;\boldsymbol{\theta})(x) - \mathcal{O}(u;\boldsymbol{\theta}')(x)| \leqslant \Phi(u)\,\|\boldsymbol{\theta} - \boldsymbol{\theta}'\|_{\ell^\infty} \tag{8}$$

for all $u \in \mathcal{X}, x \in \Omega$, and $\Phi(u) \leqslant C(1 + \|u\|_{H^2(\Omega)})^k$, for the same constants $C, \kappa > 0$ as in Eq. (7).

(iii) Finite measure: There exists $\alpha > 0$, such that

$$\int_{H^2(\Omega)} e^{\alpha \|u\|^2_{H^2(\Omega)}} \mathrm{d}\mu(u) < \infty.$$

Theorem 2. If Assumption 2 holds, then the generalization error is bounded by

$$\sup_{\boldsymbol{\theta} \in [-B,B]^{d_\theta}} |\mathbb{E}(\mathcal{E}_S(\boldsymbol{\theta}) - \mathcal{E}_{Sc}(\boldsymbol{\theta}))| \leqslant C\left[\frac{1}{\sqrt{M_u}}\left(1 + C d_{\boldsymbol{\theta}} \log(CB\sqrt{M_u})^{2\kappa+1/2}\right) + \frac{d_{\boldsymbol{\theta}}\sqrt{\log M_x}}{\sqrt{M_x}}\right],$$

where C is a constant independent of B, d_θ, M_x, and M_u. The parameter κ is specified in (7). Here , B represents the bound of parameters, and d_θ is the number of parameters.

Remark 1. Assumption 2 is easily satisfied if we consider $\mathcal{X}$ as the local function set around the solution, which is typically the case in Newton's methods. This aligns with our approach and the working region in the approximation part (see Remark 3). The error $sup_{\theta \in [-B,B]^{d_\theta}} |E(\varepsilon_S(\theta) - \varepsilon_{S_c}(\theta))|$ suggests that the network can perform well based on the loss function $\varepsilon_S(\theta)$. The reasoning is as follows: let $\theta_s = \arg\min_{\theta \in [-B,B]^{d_\theta}} \varepsilon_S(\theta)$ and $\theta_{S_c} = \arg\min_{\theta \in [-B,B]^{d_\theta}} \varepsilon_{S_c}(\theta)$. We aim for $E\varepsilon_{S_c}(\theta_s)$ to be small, which can be written as:

$$\mathbb{E}\mathcal{E}_{Sc}(\boldsymbol{\theta}_S) \leq \mathcal{E}_{Sc}(\boldsymbol{\theta}_{S_c}) + \mathbb{E}(\mathcal{E}_S(\boldsymbol{\theta}_S) - \mathcal{E}_{Sc}(\boldsymbol{\theta}_S)) \leq \mathcal{E}_{Sc}(\boldsymbol{\theta}_{S_c}) + \sup_{\boldsymbol{\theta} \in [-B,B]^{d_\theta}} |\mathbb{E}(\mathcal{E}_S(\boldsymbol{\theta}) - \mathcal{E}_{Sc}(\boldsymbol{\theta}))|,$$

where $\varepsilon_{S_c}(\theta_{S_c})$ is small, as demonstrated by Theorem 1 when B is sufficiently large.

(1-3-6) Newton Loss

As we have mentioned, relying solely on the MSE loss function can require a significant amount of data to achieve the task. However, obtaining enough data can be challenging, especially when the equation is complex and the dimension of the input space is large. Hence, we need to consider another loss function to aid learning, which is the physical information loss function , referred to here as the Network loss function. The Newton loss function is defined as:

$$\mathcal{E}_N(\theta) := \frac{1}{N_u \cdot N_x} \sum_{j=1}^{N_u} \sum_{k=1}^{N_x} |(\mathcal{L} - f'(u_j))\mathcal{O}(u_j;\theta)(x_k) + (\mathcal{L}u_j - f(u_j))(x_k)|^2 \quad (9)$$

where $u_1, u_2, \ldots, u_{N_u} \sim \nu$ are independently and identically distributed (i.i.d) samples in $\mathcal{X}$, and $x_1, x_2, \ldots, x_{N_x}$ are uniformly i.i.d samples in Ω.

Given that $\varepsilon_{\mathbf{N}}(\theta)$ can be viewed as the finite data formula of $\varepsilon_{\mathbf{N}_c}(\theta)$, where

$$\mathcal{E}_{Nc}(\boldsymbol{\theta}) = \lim_{N_u, N_x \to \infty} \mathcal{E}_S(\boldsymbol{\theta}).$$

To understand the gap between $\varepsilon_{\mathbf{N}_c}(\theta)$ and $\varepsilon_{\mathbf{N}}(\theta)$, we can rely on the following theorem:

Corollary 1. If Assumption 2 holds, then the generalization error is bounded by

$$\sup_{\boldsymbol{\theta} \in [-B,B]^{d_{\boldsymbol{\theta}}}} |\mathbb{E}(\mathcal{E}_N(\theta) - \mathcal{E}_{Nc}(\boldsymbol{\theta}))| \leqslant C \left[\frac{1}{\sqrt{N_u}} \left(1 + C d_{\boldsymbol{\theta}} \log(CB\sqrt{N_u})^{2\kappa+1/2}\right) + \frac{d_{\boldsymbol{\theta}} \sqrt{\log N_x}}{\sqrt{N_x}} \right],$$

where C is a constant independent of $B, d_{\boldsymbol{\theta}}, N_x, and\ N_u$. The parameter κ is specified in (7). Here , B represents the bound of parameters, and $d_{\boldsymbol{\theta}}$ is the number of parameters.

Remark 2. If we only utilize $\varepsilon_{\mathbf{s}}(\theta)$ as our loss function, as demonstrated in Theorem 2, we require both M_u and M_x to be large, posing a significant challenge when dealing with complex nonlinear equations. Obtaining sufficient data becomes a critical issue in such cases. In this paper, we integrate Newton's information into the loss function, defining it as follows:

$$\mathcal{E}(\boldsymbol{\theta}) := \lambda \mathcal{E}_S(\boldsymbol{\theta}) + \mathcal{E}_N(\boldsymbol{\theta}), \quad (10)$$

where $\varepsilon_{\mathbf{N}}(\theta)$ represents the cost associated with the unsupervised learning data. If we lack sufficient data for $\varepsilon_{\mathbf{s}}(\theta)$, we can adjust the parameters by selecting a small λ and increasing N_x and N_u. This strategy enables effective learning even when data for $\varepsilon_{\mathbf{s}}(\theta)$ is limited. We refer to this neural operator, which incorporates Newton information, as the Newton Informed Neural Operator.

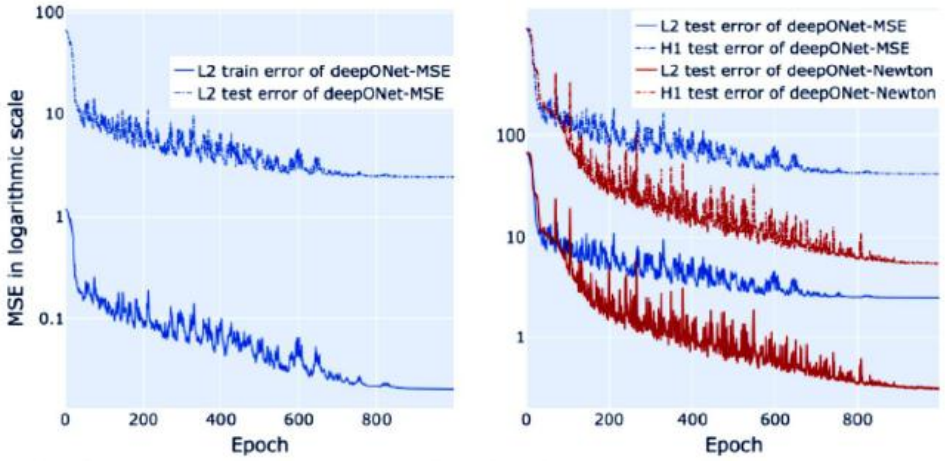
In the following experiment, we will use the neural operator established in Eq. (3) and the loss function in Eq. (10) to learn one step of the Newton method locally, i.e., the map between the input u and the solution δu in eq. (2). If we have a large dataset , we can choose a large λ in $\varepsilon(\theta)$ (10) ; if we have a small dataset, we will use a small λ to ensure the generalization of the operator is minimized. After learning one step of the Newton method using the operator neural networks, we can easily and quickly obtain the solution by the initial condition of the nonlinear PDEs (1) and find new solutions not present in the datasets.

(1-3-7) Experimental Settings

We introduce two distinct training methodologies. The first approach employs exclusively supervised data, leveraging the Mean Squared Error Loss (6) as the primary optimization criterion. The second method combines both supervised and unsupervised learning paradigms, utilizing a hybrid loss function 10 that integrates Mean Squared Error Loss (6) for small proportion of data with ground truth (supervised training dataset) and with Newton's loss (9) for large proportion of data without ground truth (unsupervised training dataset). We call the two methods method 1 and method 2. The approaches are designed to simulate a practical scenario with limited data availability, facilitating a comparison between these training strategies to evaluate their efficacy in small supervised data regimes. We chose the same configuration of the neural operator (DeepONet) which is aligned with our theoretical analysis.

Case 1: Convex problem

We consider $2D$ convex problem $\mathcal{L}(u) - f(u) = 0$, where $\mathcal{L}(u) := -\Delta u, f(u) : -u^2 + sin\, 5\pi(x+y)$ and $u = 0$ on $\partial\Omega$. We investigate the training dynamics and testing performance of neural operator (DeepONet) trained with two methods, focusing on Mean Squared Error (MSE) and Newton's loss functions. For method 1, we use 500 supervised data samples (with ground truth), while for method 2, we use 5000 unsupervised data samples (only with the initial state) along with supervised data samples, employing the regularized loss function as defined in Equation 10 with $\lambda = 0.01$.



(a) Training and testing errors using Method 1 (MSE loss for 500 supervised data)

(b) Comparison of models trained using Method 1 (deepONet-MSE) and Method 2 (deepONet-Newton)

Figure 2: Training and testing performance of DeepONet under different conditions.

MSE Loss Training (Fig. 2(a)): In method 1, Effective training is observed but exhibits poor generalization. The significantly larger testing error compared to the training error suggests that using only MSE loss is insufficient. Performance Comparison (Fig. 2(b)): DeepONet-Newton model (Method 2) exhibits superior performance in both L_2 and H_1 error metrics, highlighting its enhanced generalization accuracy. This study shows the advantages of using Newton's loss for training DeepONet models, illustrating that increasing the number of unsupervised samples via introducing Newton's loss leads to a substantial improvement in the test L_2 error and H_1 error.

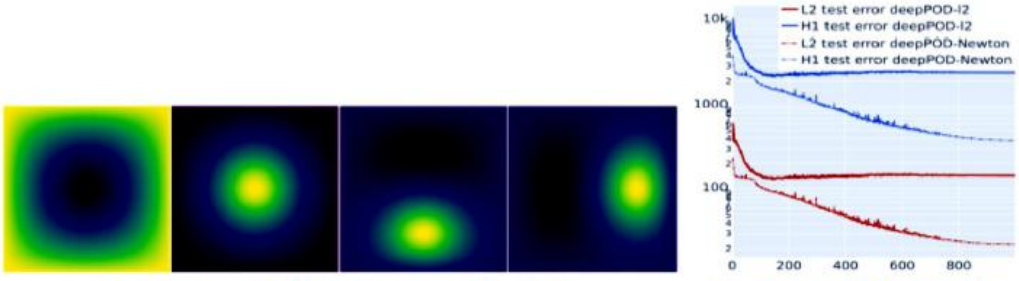
Case 2: Non-convex problem with multiple solutions

We consider a 2D Non-convex problem,

$$\begin{cases} -\Delta u(x,y) - u^2(x,y) = -s\sin(\pi x)\sin(\pi y) & in\ \Omega \\ u(x,y) = 0 & in\ \partial\Omega \end{cases} \tag{11}$$

where $\Omega = (0,1) \times (0,1)$. In this case, $\mathcal{L}(u) := -\Delta(u)$, $f(u) := u^2 - s\,sin(\pi x)\,sin(\pi y)$ and it has multiple solutions (see Figure 3 for its solutions).

In the experiment, we let one of the multiple ground truth solutions rarely touched in the supervised training dataset such that the neural operator trained via method 1 will saturate in terms of test error because it relies on the ground truth to recover all the patterns for multiple solution cases (as shown by the curves in Figure 3). On the other hand, the model trained via method 2 is less affected by the limited supervised data since the utilization of Newton's loss. One can refer to Appendix A for the detailed experiment setting.



(a) Solutions of 2D Non-convex problem (11)

(b) Method 1 VS Method 2

Figure 3: Solutions of 2D Non-convex problem (11)

Efficiency This case study highlights the superior efficiency of our neural operator-based method as a surrogate model for Newton's method. Both methods parallelize operations to solve 500/5000 Newton linear systems simultaneously, each with distinct initial states. The key advantage of the neural operator lies in its ability to batch the computation of these independent systems efficiently.

By efficiently batching and sampling a wide variety of initial states, the neural operator improves the likelihood of discovering to multiple solutions, particularly in nonlinear PDEs with complex solution landscapes. Consequently, while Newton's method alone does not inherently guarantee finding multiple solutions, the combination of rapid computation and extensive initial condition sampling enhances the chances of identifying multiple solutions.

For a fair comparison, the classical Newton solver was also parallelized using CUDA on a GPU. However, the neural operator naturally handles large batch sizes during inference, allowing it to process all systems in one go.

Parameter	Newton's Method	NINO
Number of Streams	10	-
Data Type	float32	float32
Execution Time for 500 linear Newton systems (s)	31.52	1.1E-4
Execution Time for 5000 linear Newton systems (s)	321.15	1.4E-4

Table 1: Benchmarking the efficiency of Newton Informed Neural Operator. Computational Time Comparison for Solving 500 and 5000 Initial Conditions.

Case 3: The Gray-Scott model

The Gray-Scott model describes the reaction and diffusion of two chemical species, A and S , governed by the following equations:

$$\frac{\partial A}{\partial t} = D_A \Delta A - SA^2 + (\mu + \rho)A \,,$$
$$\frac{\partial S}{\partial t} = D_S \Delta S + SA^2 - \rho(1 - S) \,,$$

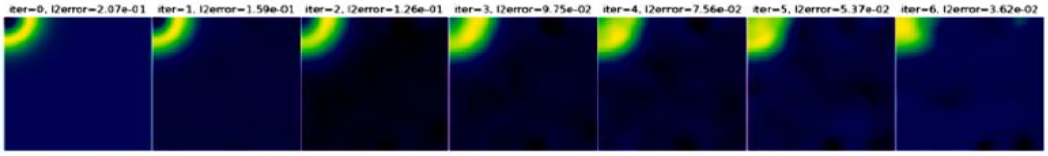
where D_A and D_S are the diffusion coefficients, and μ and ρ are rate constants.

Newton's Method for Steady-State Solutions :

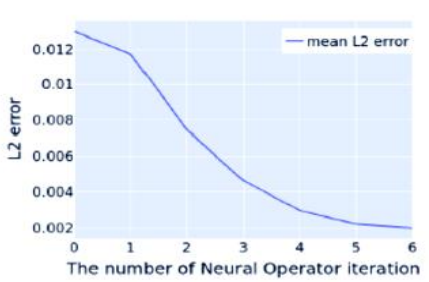
Newton's method is employed to find steady-state solutions ($\frac{\partial A}{\partial t} = 0$, $\frac{\partial S}{\partial t} = 0$) by solving the nonlinear system:

$$\begin{cases} 0 = D_A \Delta A - SA^2 + (\mu + \rho)A \,, \\ 0 = D_S \Delta S + SA^2 - \rho(1 - S) \,, \end{cases} \tag{12}$$

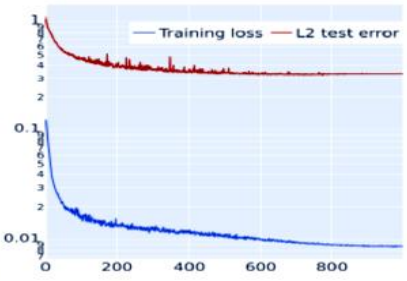
The Gray-Scott model is highly sensitive to initial conditions, where even minor perturbations can lead to vastly different emergent patterns. Please refer to Figure 4 for some examples of the patterns. This sensitivity reflects the model's complex, non-linear dynamics that can evolve into a multitude of possible steady states based on the initial setup. Consequently, training a neural operator to map initial conditions directly to their respective steady states presents significant challenges. Such a model must learn from a vast functional space, capturing the underlying dynamics that dictate the transition from any given initial state to its final pattern. This complexity and diversity of potential outcomes is the inherent difficulty in training neural operators effectively for systems as complex as the Gray-Scott model.



(a) An example demonstrating how the neural operator maps the initial state to the steady state in a iterative manner



(b) Average convergence rate of NINO.



(c) Training via method 2

Figure 4: The convergence behavior of the Neural Operator-based solver.

In subfigure (a), we use a ring-like pattern as the initial state to test our learned neural operator. This pattern does not appear in the supervised training dataset and lacks corresponding ground truth data. Instead, it is present only in the unsupervised data (Newton's loss), i.e., some data in Newton's loss will converge to this specific pattern. Despite this, our neural operator, trained using Newton's loss, can effectively approximate the mapping of the initial solution to its correct steady state. we further test our neural operator, utilizing it as a surrogate for Newton's method to address nonlinear problems with an initial state drawn from the test dataset. The curve shows the average convergence rate of $\| u - u_i \|$ across the test dataset, where u_i represents the prediction at the i-th step by the neural operator. In subfigure (c), we compare the Training Loss (Rescaled Newton's Loss) and Absolute L2 Test Error. The magnitudes are not directly comparable as they represent different metrics; however, the trends are consistent, indicating that the inclusion of unsupervised data and training with Newton's loss contributes to improved model performance.

References

[1] Terrence J. Sejnowski. The unreasonable effectiveness of deep learning in artificial intelligence. PNAS, 117:30033-30038, 2020.
[2] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078, 2014.
[3] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. arXiv preprint arXiv:1409.3215, 2014.
[4] D Jude Hemanth and V Vieira Estrela. Deep learning for image processing applications, volume 31. IOS Press, 2017.
[5] Sorin Grigorescu, Bogdan Trasnea, Tiberiu Cocias, and Gigel Macesanu. A survey of deep learning techniques for autonomous driving. Journal of Field Robotics, 37(3):362–386, 2020.
[6] Paul A Johnson, Bertrand Rouet-Leduc, Laura J Pyrak-Nolte, Gregory C Beroza, Chris J Marone, Claudia Hulbert, Addison Howard, Philipp Singer, Dmitry Gordeev, Dimosthenis Karaflos, et al. Laboratory earthquake forecasting: A machine learning competition. PNAS, 118:e2011362118, 2021.
[7] Ruben Rodriguez Torrado, Ahmed Khalifa, Michael Cerny Green, Niels Justesen, Sebastian Risi, and Julian Togelius. Bootstrapping conditional gans for video game level generation. In 2020 IEEE Conference on Games (CoG), pages 41–48. IEEE, 2020.
[8] Ruben Rodriguez Torrado, Philip Bontrager, Julian Togelius, Jialin Liu, and Diego Perez-Liebana. Deep reinforcement learning for general video game ai. In 2018 IEEE Conference on Computational Intelligence and Games (CIG), pages 1–8. IEEE, 2018.
[9] Yohai Bar-Sinai, Stephan Hoyer, Jason Hickey, and Michael P. Brenner. Learning data-driven discretizations for partial differential equations. PNAS, 116:15344–15349, 2019.
[10] F. Regazzoni, L. Dedé, and A. Quarteroni. Machine learning for fast and reliable solution of time-dependent differential equations. Journal of Computational Physics, 397:108852, 2019.
[11] E. Samaniego, C. Anitescu, S. Goswami, V. M. Nguyen-Thanh, H. Guo, K. Hamdia, X. Zhuang, and T. Rabczuk. An energy approach to the solution of partial differential equations in computational mechanics via machine learning: Concepts, implementation and applications. Computer Methods in Applied Mechanics and Engineering, 362:112790, 2020.
[12] Jiequn Han, Arnulf Jentzen, and Weinan E. Solving high-dimensional partial differential equations using deep learning. PNAS, 115:8505—-8510, 2018.

[13] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind. Automatic differentiation in machine learning: a survey. The Journal of Machine Learning Research, 18(1):5595–5637, 2017.
[14] P. Benner, S. Gugercin, and K. Willcox. A {Survey} of {Projection}-{Based} {Model} {Reduction} {Methods} for {Parametric} {Dynamical} {Systems}. SIAM Review, 57(4):483–531, jan 2015.
[15] P. Binev, A. Cohen, W. Dahmen, R. Devore, G. Petrova, and P. Wojtaszczyk. Convergence Rates for Greedy Algorithms in Reduced Basis Methods. SIAM J. MATH. ANAL, pages 1457–1472, 2011.
[16] W. Chen, Q. Wang, J. S. Hesthaven, and C. Zhang. Physics-informed machine learning for reduced-order modeling of nonlinear problems. Journal of Computational Physics, 446:110666, 2021.
[17] Y. Chen, S. Gottlieb, L. Ji, and Y. Maday. An eim-degradation free reduced basis method via over collocation and residual hyper reduction-based error estimation. Journal of Computational Physics, 444:110545, 2021.
[18] Y. Chen, J. Jiang, and A. Narayan. A robust error estimator and a residual-free error indicator for reduced basis methods. Computers & Mathematics with Applications, 77:1963–1979, 2019.
[19] G. Cybenko. Approximation by superpositions of a sigmoidal function. 2:303–314, 1989. [10] W. E, J. Han, and A. Jentzen. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. 5:349–380, 2017.
[20] W. E and B. Yu. The deep ritz method: A deep learning-based numerical algorithm for solving variational problems. 6:1–12, 2018.
[21] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In International conference on machine learning, pages 1126–1135. PMLR, 2017.
[22] S. Flennerhag, P. G. Moreno, N. D. Lawrence, and A. Damianou. Transferring knowledge across learning processes. arXiv preprint arXiv:1812.01054, 2018.
[23] I. Goodfellow, Y. Bengio, and A. Courville. Deep learning. MIT Press, 2016.
[24] B. Haasdonk. Chapter 2: Reduced Basis Methods for Parametrized PDEs ¯NA Tutorial Introduction for Stationary and Instationary Problems, pages 65–136.
[25] H. Amann and P. Hess. A multiplicity result for a class of elliptic boundary value problems. Proceedings of the Royal Society of Edinburgh Section A: Mathematics, 84(1-2):145–151, 1979.
[26] S. Brenner. The mathematical theory of finite element methods. Springer, 2008.
[27] B. Breuer, P. McKenna, and M. Plum. Multiple solutions for a semilinear boundary value problem: a computational multiplicity proof. Journal of Differential Equations, 195(1):243–269,2003.
[28] Shuhao Cao. Choose a transformer: Fourier or galerkin. Advances in Neural Information Processing Systems, 34, 2021.
[29] Mark C Cross and Pierre C Hohenberg. Pattern formation outside of equilibrium. Reviews of modern physics, 65(3):851, 1993.
[30] L. Evans. Partial differential equations, volume 19. American Mathematical Society, 2022.

[31] Somdatta Goswami, Aniruddha Bora, Yue Yu, and George Em Karniadakis. Physics-informed neural operators. 2022 arXiv preprint arXiv:2207.05748, 2022.
[32] R. Guo, S. Cao, and L. Chen. Transformer meets boundary value inverse problems. In The Eleventh International Conference on Learning Representations, 2022.
[33] J. Han, A. Jentzen, and W. E. Solving high-dimensional partial differential equations using deep learning. Proceedings of the National Academy of Sciences, 115(34):8505–8510, 2018. [10] W. Hao, S. Lee, X. Xu, and Z. Xu. Stability and robustness of time-discretization schemes for the allen-cahn equation via bifurcation and perturbation analysis. arXiv preprint arXiv:2406.18393, 2024.
[34] W. Hao, C. Liu, Y. Wang, and Y. Yang. On pattern formation in the thermodynamicallyconsistent variational gray-scott model. arXiv preprint arXiv:2409.04663, 2024.
[35] W. Hao and C. Xue. Spatial pattern formation in reaction–diffusion models: a computational approach. Journal of mathematical biology, 80:521–543, 2020.