SoL-DeepONet: Solver-In-The-Loop Deep Operator Networks for Parametric PDEs

Átila Ambrósio Luna
Department of Mathematics
Pontificia Universidade Catolica
Rio de Janeiro, Brazil
lunaatila@tamu.edu

Ulisses Braga-Neto
Department of Electrical and Computer Engineering
Texas A&M University
College Station, TX, USA
ulisses@tamu.edu

Maziar Raissi
Department of Applied Mathematics
University of California
Riverside, USA
maziar.raissi1@ucr.edu

Eduardo Gildin
Department of Petroleum Engineering
Texas A&M University
College Station, TX, USA
egildin@tamu.edu

Abstract

We propose solver-in-the-loop deep operator networks (SoL-DeepONet), a new family of physics-informed neural operator for PDEs, based on enforcing consistency with a conventional PDE solver. SoL-DeepONet combines a DeepONet with a solver-in-the-loop training strategy, in which a conventional solver update replaces PDE residuals computed by automatic differentiation. At training time, the solver advances the network prediction at time t_n by one time step, and the network learns to match this solution at t_{n+1}. We present experimental results where SoL-DeepONet learns the time-evolution operator of the one-dimensional viscous Burgers equation for diverse initial conditions and multiple viscosity values. SoL-DeepONet attains a mean relative L^2 error of 5.88% at 0.051 sec per training iteration, compared with 1.38% at at 5.263 sec/it achieved by the Physics-Informed DeepONet (PI-DeepONet) algorithm, demonstrating that the solver-in-the-loop paradigm remains competitive in accuracy but is two orders of magnitude faster to train.

1 Introduction

Physics-Informed Neural Networks (PINNs) enforce the governing equations by penalizing residuals computed via automatic differentiation at collocation points in space–time. While successful in many applications, PINNs exhibit several limitations: (i) the cost of higher-order derivatives through automatic differentiation can be substantial; (ii) training may stall or converge to poor minima due to the multi-part training loss; and (iii) standard PINN formulations typically train a separate network for each configuration or initial condition (Raissi et al., 2019).

To address these challenges, solver-in-the-loop approaches such as NewPINNs (Makki et al., 2026) replace PDE residuals with supervision generated by a numerical solver. In this paradigm, a classical scheme (e.g., finite differences) advances the network prediction by one time step and acts as a teacher, while the network learns to reproduce the solver output. This yields more stable gradients, can accelerate convergence, and embeds the numerical physics directly into the training loop without explicit residual computation.

In parallel, operator-learning architectures such as DeepONet (Lu et al., 2021) aim to learn mappings between infinite-dimensional function spaces. Rather than training one network per initial condition (IC), a single DeepONet maps any admissible IC to the corresponding solution field, providing a natural framework to generalize across families of inputs and parameters. Physics-Informed DeepONet (PI-DeepONet) combines the DeepONet architecture with physics-informed training, incorporating PDE residuals via automatic differentiation in the loss. This inherits the issues PINN has, such as

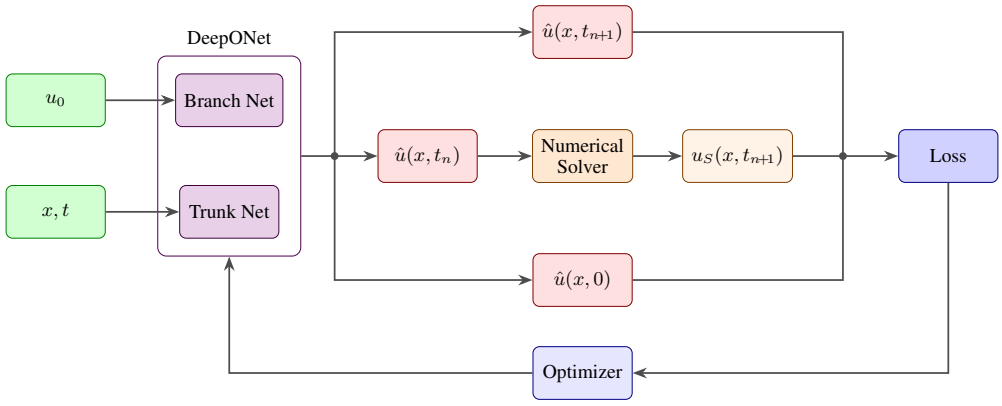

Figure 1: SoL-DeepONet architecture. The branch net encodes a discretized initial condition u_0 into a latent vector $\mathbf{b} \in \mathbb{R}^{64}$. The trunk net maps query points (x, t) to a latent vector $\boldsymbol{\tau} \in \mathbb{R}^{64}$. The output is computed as $\hat{u} = \boldsymbol{\tau} \cdot \mathbf{b} + \text{bias}$. During training, the numerical solver advances the network's prediction at t_n to produce a target at t_{n+1}, replacing traditional PDE residual computation. (Diagram adapted from Makki et al. (2026).)

expensive derivative computation and convergence issues due to a multi-part loss function (Wang et al., 2021).

In this work, we introduce the solver-in-the-loop deep operator network (SoL-DeepONet), by adapting the solver-consistent paradigm of NewPINNs to DeepONets. We keep the original unstacked DeepONet architecture unchanged, thereby enabling a direct comparison between residual-based supervision (PI-DeepONet) and solver-in-the-loop supervision (SoL-DeepONet). Our main empirical finding is that SoL-DeepONet is competitive in accuracy with PI-DeepONets while being two orders of magnitude faster to train.

2 METHODS

2.1 DEEPONET ARCHITECTURE

Our model is an unstacked DeepONet that maps a (discretized) initial condition u_0 at the branch input and queries (x, t) at the trunk input to the solution $u(x, t)$. The branch net codes the discretized IC into a latent vector, while the trunk net encodes the queries into another latent vector. The prediction is obtained by an inner product plus bias (Lu et al., 2021). Working formally, let $\boldsymbol{b} \in \mathbb{R}^p$ and $\boldsymbol{\tau} \in \mathbb{R}^p$ denote the outputs of the branch and trunk networks, respectively (in our experiments, the latent dimension is $p = 64$). The DeepONet prediction is

$$\hat{u}(x, t; u_0) = \sum_{k=1}^{p} b_k \tau_k + \beta, \tag{1}$$

where β is a learnable scalar bias. The architecture is unstacked: branch and trunk do not share weights but only the latent dimension.

2.2 TRAINING PROCEDURE: SOLVER-IN-THE-LOOP

Training operates on consecutive time pairs (t_n, t_{n+1}). For each pair (x, t) the trunk net is evaluated at both times (t_n, t_{n+1}). A single optimization step consists of (1) Sample initial conditions from the GRF; (2) For each IC u_0, compute $\hat{u}_{u_0}(x, t_n)$ and $\hat{u}_{u_0}(x, t_{n+1})$ with DeepONet; (3) Apply the numerical solver to $\hat{u}_{u_0}(x, t_n)$ (here, an FTCS scheme) to obtain $u_{\text{solver}}(x, t_{n+1})$; (4) Compute the solver loss $\mathcal{L}_{\text{solver}} = \text{MSE}\big(\hat{u}_{u_0}(x, t_{n+1}), u_{\text{solver}}(x, t_{n+1})\big)$; (5) Compute the initial-condition loss at $t = 0$, $\mathcal{L}_{\text{IC}} = \text{MSE}\big(\hat{u}_{u_0}(x, 0), u_0\big)$; (6) Combine both losses into the total loss $\mathcal{L} = \mathcal{L}_{\text{solver}} + \mathcal{L}_{\text{IC}}$; (7) Compute the loss gradient and update the trunk neural network weights.

3 RESULTS

3.1 PROBLEM SETUP

We consider the one-dimensional viscous Burgers equation

$$u_t + u\,u_x = \nu\,u_{xx}, \quad (x,t) \in [0,1]^2, \tag{2}$$

with periodic boundary conditions $u(0,t) = u(1,t)$, and $u_x(0,t) = u_x(1,t)$, where ν is the kinematic viscosity. This is a classical benchmark for machine-learning methods for PDEs, due to its nonlinear convective term, shock formation at low viscosity ν, and dissipative behavior (Burgers, 1948). In operator-learning settings, Burgers offers a controlled environment to test generalization across initial conditions (Wang et al., 2022; Lu et al., 2021). We discretize space with $N_x = 101$ uniformly spaced points, so that $\Delta x = 0.01$. For training the operator, we use a coarse time step $\Delta t_{\text{train}} = 0.1$, yielding time levels $t_n = 0, 0.1, \ldots, 1.0$ (11 snapshots). The finite-difference solver employed inside the training loop uses a much smaller time step $\Delta t_{\text{solver}} = 5 \times 10^{-4}$ and performs $N_i = 200$ sub-steps to advance from t_n to t_{n+1}, thus matching $\Delta t_{\text{train}} = N_i \Delta t_{\text{solver}}$ (LeVeque, 2002).

3.2 INITIAL CONDITION GENERATION

Training initial conditions are sampled as Gaussian random fields following the protocol of PI-DeepONet (Wang et al., 2022). We use the same covariance operator and parameters, with variance $\sigma^2 = 625$, correlation length parameter $\tau = 5$, and smoothness exponent $\gamma = 4$, and truncate the expansion at $K_{\text{GRF}} = 2048$ Fourier modes to match the periodic domain. We do not normalize the sampled ICs; their typical amplitude satisfies $|u_0|_{\max} \approx 0.3$, which remains well within the Courant–Friedrichs–Lewy stability limit of the FTCS solver. A pool of $N_{\text{pool}} = 1000$ ICs is pre-generated at the beginning of training, and at each iteration we randomly sample $N_{\text{ICS}} = 16$ ICs from this pool.

3.3 OPTIMIZATION AND HYPERPARAMETERS

We train the network using the Adam optimizer with initial learning rate $\eta = 3 \times 10^{-4}$ (Raissi et al., 2019). A StepLR scheduler decays the learning rate by a factor of 0.9 every 1500 iterations. Training runs for a total of $80\,000$ iterations. All experiments were conducted on a single NVIDIA GTX 1650 GPU with 16 GB of system RAM. On this hardware, our solver-in-the-loop DeepONet implementation takes approximately 0.051 seconds per training iteration, whereas a PI-DeepONet implementation with the same architecture and data pipeline takes 5.263 seconds per iteration, confirming a speedup of two orders of magnitude in training throughput. Table 1 summarizes all hyperparameters used in our experiments. These choices follow the NewPINNs configuration (Makki et al., 2026) as closely as possible, with modifications only where required by the GRF setup.

3.4 QUANTITATIVE EVALUATION

We evaluate the learned operator on 1000 test ICs drawn from the same GRF distribution but with a different random seed. Reference solutions are computed with a spectral solver on a finer grid. Table 2 summarizes the comparison between PI-DeepONet and SoL-DeepONet. While our method exhibits a slightly higher mean error, it makes up for that with a training time that is two orders of magnitude shorter, making it a more practical choice when computational efficiency is a priority. Importantly, SoL-DeepONet does not require computing PDE residuals via automatic differentiation; supervision is provided only by the solver at intervals of $\Delta t_{\text{train}} = 0.1$.

3.5 QUALITATIVE EVALUATION

Figure 2 presents qualitative comparisons between SoL-DeepONet predictions and reference spectral solutions for six representative GRF initial conditions at viscosity $\nu = 0.01$. Each row corresponds to a different initial condition and each column to a different time snapshot. Relative L^2 errors are reported over each plot (best seen by magnifying the figure). Steep gradients and shock-like structures emerge rapidly, and the error concentrates near these high-gradient regions. The

Table 1: Hyperparameters used in SoL-DeepONet training.

Spatial points	$N_x = 101$
Spatial step	$\Delta x = 0.01$
Train time step	$\Delta t_{\text{train}} = 0.1$
Solver time step	$\Delta t_{\text{solver}} = 5 \times 10^{-4}$
Solver sub-steps	$N_i = 200$
GRF modes	$K_{\text{GRF}} = 2048$
ICs per step	$N_{\text{ICS}} = 16$
IC pool size	$N_{\text{pool}} = 1000$
Latent dimension	$p = 64$
Learning rate	3×10^{-4}
LR scheduler	StepLR$(1500, 0.9)$
GRF σ	625
GRF τ	5
GRF γ	4

Table 2: Training comparison between PI-DeepONet Wang et al. (2022) and SoL-DeepONet.

	PI-DeepONet	**Sol DeepONet**
Per gradient update		
ICs per batch	50	16
IC loss points	50	8,080
BC loss points	100	periodic
PDE/Solver loss points	50	8,080
Total loss evaluations	200	16,160
Per iteration		
Gradient updates	1	10
Total loss evaluations	200	161,600
Training speed		
Throughput (it/s)	0.187	**19.505**
Time per iteration (s)	5.342	**0.051**
Speedup	$1\times$	**104.7**$\times$
Performance		
Mean L^2 error (%)	**1.38** ± 1.64	3.07 $\pm$**1.13**

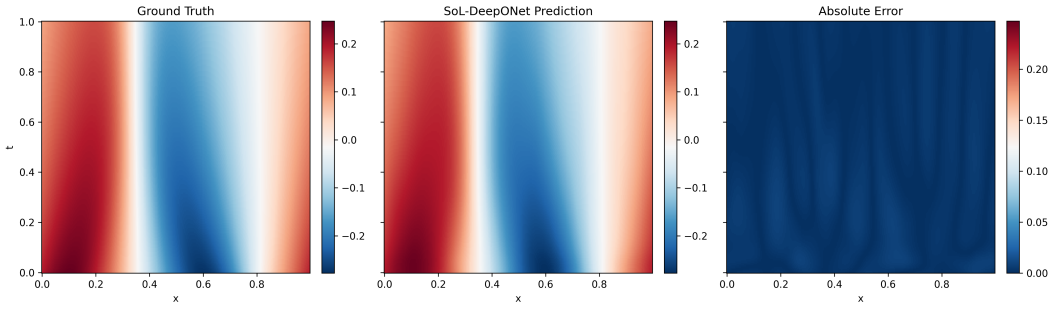

Figure 2: Qualitative comparison of SoL-DeepONet predictions (solid lines) against the reference spectral solver (dashed lines) for multiple GRF initial conditions at viscosity $\nu = 0.01$ across several time snapshots. Relative L^2 errors are reported over each plot (best seen by magnifying the figure).

relatively coarse training time step $\Delta t_{\text{train}} = 0.1$ means that solver supervision is sparse, which can limit accuracy around fast dynamics. Nonetheless, the network captures the overall propagation and dissipation patterns, even if sharp features are slightly smeared.

4 Conclusion

We have introduced solver-in-the-loop DeepONets, a new class of physics-informed neural operators that combine an unstacked DeepONet with a numerical solver embedded in the training loop. In experiments with the Burger's equation, SoL-DeepONet is competitive in accuracy with PI-DeepONet, but avoids the computational overhead of automatic differentiation required by PDE residuals, relying instead on discrete supervision from a classical solver, which leads to a training time that is two orders of magnitude shorter. Future work includes optimizing the DeepONet architecture for periodic inputs, exploring higher-order or more dissipative solver schemes (e.g., Lax–Wendroff or Crank–Nicolson) within the loop, refining the solver time discretization near shock regions, and extending the solver-in-the-loop operator-learning framework to higher-dimensional PDEs such as the Navier–Stokes equations.

REFERENCES

J. M. Burgers. A mathematical model illustrating the theory of turbulence. *Advances in Applied Mechanics*, 1:171–199, 1948.

Randall J. LeVeque. *Finite Volume Methods for Hyperbolic Problems*. Cambridge University Press, 2002.

Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, 2021.

Maedeh Makki, Satish Chandran, Maziar Raissi, Adrien Grenier, and Behzad Mohebbi. Newpinns: Physics-informing neural networks using conventional solvers for partial differential equations. *arXiv preprint*, 2026.

Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

Sifan Wang, Hanwen Wang, and Paris Perdikaris. Learning the solution operator of parametric partial differential equations with physics-informed DeepONets. *Science Advances*, 7(40):eabi8605, 2021.

Sifan Wang, Yujun Yu, and Paris Perdikaris. When and why PINNs fail to train: A neural tangent kernel perspective. *Journal of Computational Physics*, 449:110768, 2022.