Journal of Nonlinear Waves (2026), **2**, e23, 1–47
doi:10.1017/jnw.2026.10050

CAMBRIDGE UNIVERSITY PRESS

RESEARCH ARTICLE

Soliton solutions to KdV and KP equations using physics-informed neural networks

Karnav Raval[1], Nadia Aiaseh[1], Nima Hosseini Dashtbayaz[2], Olga Trichtchenko[1] and Boyu Wang[2]

[1]Department of Physics and Astronomy, Western University, London, ON, Canada
[2]Department of Computer Science, Western University, London, ON, Canada
Corresponding author: Olga Trichtchenko; Email: otrichtc@uwo.ca

Keywords: nonlinear interactions; physics-informed neural networks; solitons

(Received 27 January 2026 UTC; revised 31 July 2026 UTC; accepted 2 September 2026 UTC)

Abstract
Partial differential equations (PDEs) model a wide range of physical phenomena but are often difficult to solve analytically or numerically, particularly when they are nonlinear and contain high-order derivative terms, as in water-wave problems. Although many numerical methods exist, they are typically tailored to specific equations. Physics-informed neural networks (PINNs) have recently emerged as a flexible alternative by incorporating the governing PDE, together with initial and boundary conditions, directly into the network's loss function. In this work, we systematically investigate the ability of PINNs to capture the nonlinear dynamics of the Korteweg–de Vries (KdV) equation and its two-dimensional generalization, the Kadomtsev–Petviashvili (KP) equation. Recovering nonlinear soliton interactions from the governing PDE alone therefore provides a stringent test of whether the network has learned the underlying dynamics rather than simply reproducing training data. We examine increasingly complex interacting-soliton configurations for KdV and analogous line-soliton and junction solutions for KP, evaluating how accuracy, training time and network complexity scale with problem difficulty. We also compare network architectures, activation functions and optimization strategies while benchmarking PINN performance against analytic solutions.

1. Introduction

The goal of this work is to examine how well physics-informed neural networks (PINNs) can solve nonlinear partial differential equations (PDEs). In particular, we are interested in whether a PINN can capture genuinely nonlinear behaviour, such as the soliton interactions studied here, when that behaviour is never directly supplied to the network, and in characterizing how reliably it does so as the complexity of the underlying solution increases. We focus on a class of equations that are well studied analytically and primarily used for modelling water waves in both two and three dimensions – the Korteweg–de Vries (KdV) and the Kadomtsev–Petviashvili (KP) equations. In particular, we study how well PINNs can find soliton solutions to these nonlinear, dispersive PDEs. Solitons propagate without changing shape, by balancing nonlinearity and dispersion. After their nonlinear interaction, they emerge from collisions unchanged except for a possible phase shift. Thus, a numerical solver can only be considered effective if it accurately captures this characteristic behaviour. Aside from KP and KdV's modelling applications, they are completely integrable and exactly solvable [42], which implies that they admit well-behaved solutions.

It is well known that PDEs are hard to solve both numerically and analytically in all generality, in particular if they are nonlinear. The benefit of focusing on KdV and KP and their soliton solutions is that these are well-studied equations with available analytical solutions. There are various methods for

© The Author(s), 2026. Published by Cambridge University Press. This is an Open Access article, distributed under the terms of the Creative Commons Attribution licence (http://creativecommons.org/licenses/by/4.0), which permits unrestricted re-use, distribution and reproduction, provided the original article is properly cited.

https://doi.org/10.1017/jnw.2026.10050 Published online by Cambridge University Press

finding soliton solutions for KdV and KP, such as the inverse scattering transform [2, 4, 15, 17, 18, 38], Riemann–Hilbert methods [45] and the Darboux transforms [40], just to name a few. In this work, we use the form of solutions given by Hirota [23–25, 49] for comparison to the numerical results we obtain from PINNs, as this method easily adapts to N-soliton solutions.

It is important to note that solutions to KdV and KP have also been found numerically. Historically, finite difference schemes were the first to be used, going back to Zabusky and Kruskal's original study of KdV solitons [57]. Such schemes remain a natural choice for lower-order or more general equations, but for KdV and KP specifically they are typically outperformed by pseudospectral (Fourier) methods, which exploit the fact that the linear dispersive terms in both equations are diagonal in Fourier space. The benchmark comparative study of Taha and Ablowitz [51] established this directly for KdV, showing that pseudospectral schemes reach a given accuracy substantially faster than finite differences for the one- and two-soliton problem, and pseudospectral or split-step Fourier integration has since become the standard approach for this class of equations.

The KP equation poses an additional complication for classical solvers, since its line-soliton solutions are not spatially localized and instead extend to the boundary of any finite computational domain [8]. Ablowitz and Baldwin [1], whose analytic line-soliton solutions we use to define our own initial and boundary data, report X- and Y-type line-soliton interactions as a naturally occurring phenomenon on shallow beaches and give closed-form Hirota solutions for them, but do not address this numerical boundary treatment. Kao and Kodama [30] instead address it with a windowing method, which smoothly damps the solution near the transverse domain boundaries so that a standard Fourier pseudospectral discretization can be used, with the true line-soliton solution recovered outside the windowed region. This approach achieves high (spectral) accuracy once the windowing scheme is in place, but requires this equation-specific boundary treatment, in contrast to the mesh-free approach used here, where the boundary is instead enforced directly as a labelled loss term. PINNs themselves have since been compared directly against classical solvers [6, 21, 46], generally finding that classical methods remain faster and more accurate for problems with fully known initial and boundary data, though often requiring exactly this kind of equation-specific discretization choice that a mesh-free method does not need. We return to a quantitative comparison between our own PINN results and these classical benchmarks in Section 4.

More recently, PINNs have emerged as a method to find solutions to PDEs [47], which relies on the property of feedforward neural networks being universal function approximators [13, 26]. One of the first prominent works to apply PINNs to the KdV equation was [47], who proposed using neural networks to solve forward and inverse problems involving nonlinear PDEs. PINNs are different from conventional neural networks as they incorporate physical laws, in the form of PDEs, in their loss function. This allows for a reduction in the number of labelled training examples needed, which traditional neural networks require. Furthermore, traditional neural networks learn by being given exact input and output pairs to the network (i.e. supervised learning), but using the PDE in the loss allows the network to learn the solution within the problem domain without knowing the exact solution for the training points. In practice, the governing equation constrains the solution space to those that are admitted by the PDE, while the initial conditions (ICs) and boundary conditions (BCs) appear as supervised loss terms, ensuring the solution is unique. In their work, they illustrated how a PINN could approximate solutions to the nonlinear Schrödinger equation with periodic BCs, as well as the Allen–Cahn and KdV equations. They also considered the inverse problem of finding the PDEs from data, illustrating how this would work for Navier–Stokes and KdV, by analysing how well the PINN can find the parameters in the PDE.

Since then, the optimization procedure for finding soliton solutions to KdV and modified KdV equations has been refined [35] by balancing the different terms in the loss function to avoid oscillations in the gradients. For consideration of solitary wave solutions to the mKdV equation and the improved Boussinesq equation, hyperparameters were introduced to speed up the optimization procedure, resulting in more accurate solutions in [20]. Higher-dimensional PDEs such as the KP equation have been explored using PINNs to find breather and lump solutions, among others [41]. For a full review

https://doi.org/10.1017/jnw.2026.10050 Published online by Cambridge University Press

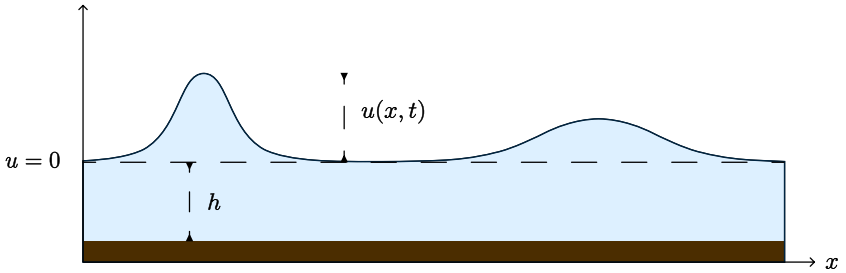


Figure 1. *$u(x,t)$ denotes the surface of the wave from its undisturbed level at $u=0$ with depth h.*

of PINNs as applied to fluid dynamics, see [58], and for a more comprehensive review of PINNs, see [52].

In this work, we carry out a systematic study of how PINN performance scales with the complexity of the KdV and KP soliton solutions being approximated, and of how this performance depends on the choices involved in training. We train and evaluate PINNs across an increasing number of interacting solitons for KdV, from one soliton up to three, and analogous line-soliton junction configurations for KP, including a three-line-soliton '2-in-3-out' junction, quantifying accuracy, training time and required network architecture in each case (Section 4). Throughout, the IC supplied to the network never encodes the nonlinear soliton interaction itself, as it consists only of the individual, single-soliton profiles (equation (2.5)) that make up the solution, placed far enough apart (within some numerical tolerance) that they do not yet interact. The PINN must therefore recover the full nonlinear interaction and phase shifts encoded in the coefficients A_{ij} of equation (2.4) from the governing PDE alone. This flexibility means the same PINN framework can be adapted across different ICs, recovering a full multi-soliton interaction from initial data describing only individual, well-separated solitons. However, we find that the network architecture and training strategy required to do so reliably are not obvious in advance, and become increasingly demanding as the number of interacting solitons grows, a point we return to in Section 3.

The paper is laid out as follows. First, we describe the equations and particular types of solutions that we are considering in Section 2. We then describe the neural network architecture we use for the optimization problem of minimizing the given loss, as well as the chosen hyperparameters in Section 3. We finish by presenting the types of one- and two-soliton solutions for both the KdV and KP equations in Section 4, and conclude our findings and comment on the performance of PINNs in Section 5.

2. Mathematical formulation

The KdV equation is a well-studied model for inviscid surface waves in shallow water in one dimension, first proposed by Boussinesq in 1872 [10], then Korteweg and de Vries in 1895 [33]. In non-dimensional form, it is given by

$$u_t + 6uu_x + u_{xxx} = 0, \tag{2.1}$$

where $u(x,t)$ is the vertical displacement from equilibrium, (x,t) are spatial and time variables, and subscripts denote partial derivatives. The scenario is depicted in Figure 1. This equation includes both a nonlinear and a dispersive term, and admits soliton solutions [2, 14, 48]. The solitary profiles interact nonlinearly and emerge unchanged, except for a phase shift, and it is due to this particle-like behaviour that they are often referred to as solitons [57]. It is known that KdV is integrable and exhibits N-soliton solutions [17, 34].

The KdV equation (2.1) can be extended to capture two-dimensional, small-amplitude waves in shallow water by considering the KP equation. It was first used by Kadomtsev and Petviashvili in 1970

https://doi.org/10.1017/jnw.2026.10050 Published online by Cambridge University Press

when they investigated the stability of the KdV soliton with transverse perturbations [29]. This equation was also derived in the context of water waves by considering the motion of irrotational, inviscid and incompressible fluid, assuming small-amplitude waves and shallow water where the motion is primarily one-dimensional [2, 3, 50]. The non-dimensional KP equation is given by

$$(u_t + 6uu_x + u_{xxx})_x + 3\sigma u_{yy} = 0, \text{ with } \sigma = \pm 1, \tag{2.2}$$

where the subscripts denote partial derivatives and the displacement from equilibrium is denoted by $u(x, y, t)$ with the spatial variables (x, y) and time t. The parameter σ can take on values of ± 1. When $\sigma = -1$, equation (2.2) is referred to as KPI, and when $\sigma = 1$, it is KPII. For this work, we focus on soliton solutions of KPII, as this is the regime of weak surface tension, which is valid for waves in the ocean and, in our opinion, a more interesting regime to examine using PINNs due to its practical applications. It is notable that every solution to the KdV is also a solution to KPII. It has also been shown that KPII has N-line-soliton solutions, wherein the N solitons travel in different directions and interact obliquely [49].

There are many ways to write the soliton solutions; we focus on the Hirota notation [1, 22, 23], where the solution can be represented as

$$u = 2\frac{\partial^2}{\partial x^2} \ln(F^{(N)}) \tag{2.3}$$

with

$$F^{(N)} = \sum_{\mu_i} \exp\left(\sum_{i=1}^{N} \mu_i \eta_i + \sum_{i<j}^{N} \mu_i \mu_j a_{ij}\right) \tag{2.4}$$

where N denotes the number of solitons and $\mu_i \in \{0, 1\}$ for $i, j \in \{1, \ldots, N\}$

$$\begin{aligned} \eta_i &= k_i x - k_i^3 t + \delta_i && \text{for KdV,} \\ \eta_i &= k_i \left(x + P_i y - (k_i^2 + 3\sigma P_i^2)t\right) + \delta_i && \text{for KP} \end{aligned}$$

and $a_{ij} = \ln(A_{ij})$, where $i < j$ with

$$\begin{aligned} A_{ij} &= \frac{(k_i - k_j)^2}{(k_i + k_j)^2} && \text{for KdV,} \\ A_{ij} &= \frac{(k_i - k_j)^2 - \sigma(P_i - P_j)^2}{(k_i + k_j)^2 - \sigma(P_i - P_j)^2}, && \text{for KP.} \end{aligned}$$

if we consider k, λ to be the wavenumbers in x and y, respectively, where for convenience we have introduced $P = \lambda/k$ [1]. Moreover, δ_i is a constant denoting the phase shift, with x, y and t representing the spatial and temporal variables, respectively.

In this work, we focus on one- and two-soliton solutions to KdV and KP, which take the same form but differ in the number of spatial dimensions. More explicitly, the one-soliton solutions are given by (2.3) with

https://doi.org/10.1017/jnw.2026.10050 Published online by Cambridge University Press

$$F^{(1)} = \sum_{\mu_1 \in \{0,1\}} e^{\mu_1 \eta_1} = 1 + e^{\eta_1}. \tag{2.5}$$

We can simplify this in the case of KdV to be

$$u^{(1)}_{\text{KdV}} = \frac{k^2}{2} \operatorname{sech}^2\left(\frac{\eta_1}{2}\right).$$

In this form, it is easy to see that

$$c = k^2 \quad \text{and} \quad A = \frac{c}{2}$$

where c is the phase speed, and A is the amplitude of the soliton. Observe that the speed of individual solitons is proportional to A, with larger amplitude waves travelling faster. For the two-soliton solution, after enumerating over all possible combinations of $\mu_1, \mu_2 \in \{0, 1\}$, we obtain

$$F^{(2)} = \sum_{\mu_1, \mu_2 \in \{0,1\}} e^{\mu_1 \eta_1 + \mu_2 \eta_2 + \mu_1 \mu_2 a_{12}} = 1 + e^{\eta_1} + e^{\eta_2} + A_{12} e^{\eta_1 + \eta_2}. \tag{2.6}$$

It is important to note that the analytical form of the solution is only used to define the IC and, in the case of PINNs for the KP equation, boundary data. Otherwise, the PINN is not aware of the analytical solution for any other points in the domain, as explained further in Section 3. The only other way the analytical solution is used is to compute the accuracy of the PINN-predicted solution, i.e. in the error analysis. We also note that we visualize the effects of nonlinearity for the KdV two-soliton solution by comparing the linear combination of single soliton solutions to the two-soliton PINN-predicted solution. When doing these comparisons, in order to make sure we overlay the analogous solitons prior to their interaction, we need to apply phase shifts. We shift each soliton of the linear combination by

$$\delta_1 = 0 \tag{2.7}$$

$$\delta_2 = a_{12} = 2 \ln\left(\frac{k_1 - k_2}{k_1 + k_2}\right). \tag{2.8}$$

Observe that only the smaller (slower) soliton needs a shift.

3. Physics-informed neural networks (PINNs)

The main goal of this work is to observe if PINNs can accurately find nonlinear and dispersive soliton solutions to well-understood KdV and KP equations ((2.1) and (2.2)) corresponding to analytical solutions given by equation (2.3). We use the implementation presented in [47] and employ a deep, fully connected feedforward neural network. Computing solutions involves initializing, training and then using a PINN.

The network is comprised of input, hidden and output layers, which in turn are made up of neurons (or nodes). In the case of the KdV equation, the input layer is composed of two nodes which correspond to its independent variables (x, t). In the KP case, we add an additional node such that there are three nodes, which correspond to (x, y, t). The output node represents the PINN-predicted solution $u_{\text{PINN}}(x, y, t)$. The layers are fully connected in a feedforward way, with connections between layers parameterized by weights and biases, which we collectively refer to as trainable parameters θ.

https://doi.org/10.1017/jnw.2026.10050 Published online by Cambridge University Press

The hyperbolic tangent activation function is applied to layer outputs prior to passing them to a following layer. We compare this choice against seven alternative activation functions in Appendix D, where tanh is shown to reach comparable accuracy to the best-performing alternatives at substantially lower training cost.

We can consider the PINN as a multitask learner with the objective of simultaneously learning the data, composed of particular ICs and BCs, while also satisfying the PDE. We define the set of points used for training as

$$\mathcal{X}_{\text{train}} = \mathcal{X}_{\text{IC}} \cup \mathcal{X}_{\text{BC}} \cup \mathcal{X}_{\text{PDE}} = \{\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^N\}$$

consisting of points $\mathbf{x}^i = \{x^i, y^i, t^i\}$, where x^i, y^i and t^i are drawn from a given distribution, and i denotes the training sample. Note that the training set $\mathcal{X}_{\text{train}}$ is composed of disjoint sets $\mathcal{X}_{\text{IC}}$, $\mathcal{X}_{\text{BC}}$, which have associated solution data (labels), and $\mathcal{X}_{\text{PDE}}$ denoting points which are used to compute the PDE residual, defined as the left-hand side of equation (2.2). Hereafter, we refer to these data as initial, boundary and collocation points, respectively.

We define the training objective as a minimization of the total loss L, which is given by the weighted sum

$$L(\theta, \mathcal{X}) = w_{\text{IC}} L_{\text{IC}}(\theta, \mathcal{X}_{\text{IC}}) + w_{\text{BC}} L_{\text{BC}}(\theta, \mathcal{X}_{\text{BC}}) + w_{\text{PDE}} L_{\text{PDE}}(\theta, \mathcal{X}_{\text{PDE}}), \tag{3.1}$$

where w_m are the loss weights (not to be confused with a trainable parameter), and L_m is the loss of a particular task, defined to be the mean squared error given as

$$L_{\text{BC}} = \frac{1}{N_{\text{BC}}} \sum_{i=1}^{N_{\text{BC}}} |u_{\text{exact}}(\mathbf{x}^i_{\text{BC}}) - \hat{u}^i_{\text{PINN}}|^2,$$

$$L_{\text{IC}} = \frac{1}{N_{\text{IC}}} \sum_{j=1}^{N_{\text{IC}}} |u_{\text{exact}}(\mathbf{x}^j_{\text{IC}}) - \hat{u}^j_{\text{PINN}}|^2,$$

$$L_{\text{PDE}} = \frac{1}{N_{\text{PDE}}} \sum_{k=1}^{N_{\text{PDE}}} |f_{\text{exact}}(\mathbf{x}^k_{\text{PDE}}) - \hat{f}_{\text{PINN}}(\mathbf{x}^k_{\text{PDE}})|^2.$$

Training the PINN involves finding the optimal parameters θ which minimize the total loss L. This is done using a set of optimizers, for a particular training set $\mathcal{X}_{\text{train}}$. The number of training samples is denoted as N_{BC}, N_{IC} and N_{PDE} for the BC, IC and collocation points. We denote u_{exact} as the exact solution evaluated on initial and boundary points, and $\hat{u}_{\text{PINN}}$ denotes the PINN's output value given the corresponding input points. Finally, f_{exact} is the right-hand side of the PDE, either equation (2.1) or (2.2), which in our case is trivial ($f_{\text{exact}} := 0$). Given that the BC and IC are prescribed, this corresponds to having labelled data, in contrast to the interior points which are not labelled.

Training data for the IC and BC are evenly spaced, and domain points are sampled from a uniform distribution. In this instance-based approach, treating $u(x, y, t)$ as a scalar dependent on a set of values within the domain implies that the PDE is satisfied pointwise. In addition, we can evaluate the model by comparing the PINN prediction to the analytical solution on a set of test points, which may differ from the training points. In this way, the PINN solution is mesh–free, which is greatly advantageous.

Computing these solutions involves initializing a PINN, and then training it with a particular training set $\mathcal{X}_{\text{train}}$ to obtain optimal parameters θ, followed by using it on a test set $\mathcal{X}_{\text{train}}$. The overall procedure is shown in Algorithm 1. The parameters θ of the network were initialized using the Xavier normal distribution [19]. For the beginning stages (1000 epochs) [37], the algorithm used to minimize the loss term was the Adam (Adaptive Moment Estimation) optimizer relying on stochastic gradient descent

https://doi.org/10.1017/jnw.2026.10050 Published online by Cambridge University Press

[32]. To refine the training, the L-BFGS (limited-memory Broyden–Fletcher–Goldfarb–Shanno) algorithm was used in later stages [39, 43], which is a quasi-Newton method relying on the Jacobian and a pseudo-inverse of the Hessian to optimize.

Adam and L-BFGS have complementary strengths that motivate using them in sequence rather than either one alone. Adam [32] is a first-order, stochastic, adaptive-learning-rate method and comparatively cheaper per iteration, robust to the highly non-convex and initially noisy loss landscape typical of PINN training, and is generally not prone to becoming trapped in shallow local minima or saddle points early on. Its drawback is that it converges slowly once close to a minimum, and typically stalls at a relatively high loss value even after many epochs. L-BFGS [39, 43] is instead a quasi-Newton, second-order method that builds an approximation of the inverse Hessian from recent gradient history, and it converges much faster, but each iteration is more expensive, it is more sensitive to the quality of the starting point, and it can perform poorly or fail to make progress on the noisy, highly oscillatory loss surfaces that PDE residual terms typically produce early in training. Using L-BFGS alone from a random initialization is therefore prone to stalling in a poor local minimum before the optimizer ever reaches the basin in which its fast convergence properties are useful. The two-stage schedule used throughout this work with Adam for the first 1000 epochs, followed by L-BFGS until convergence, follows the original PINN formulation of [47] and is now standard practice in the PINN literature. It is intended to combine Adam's early robustness with L-BFGS's fast late-stage refinement.

Algorithm 1: Overall PINNs workflow

Step *1: Initialization*
- (a) Define the neural network architecture by setting number of hidden layers and neurons per hidden layer (see Table 3).
- (b) Initialize the network parameters θ by sampling from Xavier normal distribution.

Step *2: Training*
- (a) Set the training hyperparameters (see Table 2).
- (b) Define the loss function L using equation (9).
- (c) Generate training data: randomly sample collocation points $\mathcal{X}_{\text{PDE}}$ from a uniform distribution and evenly sample initial and boundary points $\mathcal{X}_{\text{IC}}$, $\mathcal{X}_{\text{BC}}$.
- (d) Train using Adam:
  for $epoch = 1$ **to** 1000 **do**
  1. Evaluate $u_{\text{PINN}}(\mathcal{X}_{\text{train}})$ on all training points
  2. Compute loss $L(\theta, \mathcal{X}_{\text{train}})$ and $\nabla_\theta L$
  3. Update network parameters θ using Adam

  end
- (e) Train using L-BFGS:
  while $|\Delta L| > tolerance$ **do**
  1. Evaluate $u_{\text{PINN}}(\mathcal{X}_{\text{train}})$ on all training points
  2. Compute loss $L(\theta, \mathcal{X}_{\text{train}})$ and $\nabla_\theta L$
  3. Update network parameters θ using L-BFGS

  end
- (f) Save the optimized network parameters θ.

Step *3: Testing*
- (a) Create the equally spaced test dataset $\mathcal{X}_{\text{test}}$.
- (b) Evaluate the trained PINN on test dataset $\hat{u}_{\text{PINN}}(\mathcal{X}_{\text{test}})$.
- (c) Compare $\hat{u}_{\text{PINN}}$ to u_{exact} using equation (10).

4. Results

In this section, we present the PINN-approximated solutions for the KdV and KP equations, (2.1) and (2.2), respectively. All computations were performed using Python with PyTorch 2.5 on an NVIDIA GeForce RTX 4090 GPU using CUDA 12.4. The collocation points were sampled randomly, and data for the IC and BC were sampled evenly. Finding particular N-soliton solutions involves training a PINN with the corresponding IC and BC for the case of interest. The parameters for the ICs are given in Table 1

https://doi.org/10.1017/jnw.2026.10050 Published online by Cambridge University Press

Table 1. *Parameters used for ICs*

	KdV		KP		
Parameter	**1-soliton**	**2-soliton**	**1-soliton**	**2-soliton (X)**	**2-soliton (Y)**
			x-direction		
k_1	0.9	1.0	1/2	1/2	1/2
k_2	—	$\sqrt{1.2/4}$	—	1/2	1.0
			y-direction		
P_1	—	—	2/3	$-2/3$	3/4
P_2	—	—	—	2/3	1/4

and were obtained from [5] for KdV or [1] for KP in order to match well-understood results. We note that the ICs for the two-soliton solution for KdV are given by a superposition of two separate one-soliton solutions (2.5) with the phase δ_i computed using (2.7), representing two solitons far enough apart such that the interaction term is negligible. In all other cases, the phases for ICs are set to zero. For KP, the IC is such that the interaction of the line solitons is outside of the computational domain. This is done to see if the PINN can capture the nonlinearity without it being explicitly given.

The set of parameters used in the optimization procedure is given in Table 2. Note that automatic differentiation [7] is used, as implemented in PyTorch, to compute loss gradients and the derivatives appearing in the KdV and KP equations (2.1) and (2.2). Once the network is trained, the solution is examined on a test set of uniformly generated $N_{\text{test}} = 1 \times 10^6$ points $\mathcal{X}_{\text{test}}$, which do not necessarily correspond to the training set. The point-wise normalized absolute error between the PINN predicted solution $\hat{u}_{\text{PINN}}$ and the exact solution u_{exact} given by (2.3) with (2.4) is computed by

$$e_l = \frac{|u_{\text{exact}}(\mathbf{x}^l) - \hat{u}_{\text{PINN}}(\mathbf{x}^l)|}{\max\limits_{\mathbf{x}\in\mathcal{X}_{\text{test}}} |u_{\text{exact}}(\mathbf{x}^l)|}, \tag{4.1}$$

where $l = 1, 2, \ldots, N_{\text{test}}$. We often report the maximum point-wise normalized absolute error, which we denote by $||e||_\infty = \max\limits_{1\le l\le N_{\text{test}}} (e_l)$. This was chosen because (4.1) gave the best measure of how accurately we captured the soliton solutions.

The architecture of the PINN, the number of points used in the training sets, as well as the weights for each term in the loss function (3.1) for the cases we considered, are summarized in Table 3. The architecture (number of layers and neurons per layer) for the KdV cases was chosen based on the architecture which gave the lowest final loss among those tested, from the results as given in Appendix A. From this table, we see that the number of neurons and layers increases as the number of solitons increases in the case of KdV where the BCs are trivial (zero). On the contrary, these hyperparameters do not change for the case of KP line solitons interacting, resulting in x or y junctions because the complexity of those solutions is relatively similar to each other, and the BCs are prescribed exactly, but the number of layers does increase between one- and two-soliton solutions. The loss weight (w_{BC}) for the zero BCs for KdV is less important and is set to be smaller relative to other weights than in the case of KPII, where the BCs are nontrivial (see Appendix B for loss histories separated according to BC, IC and PDE components). The maximum error (equation (4.1)), the final training loss (using equation (3.1)) and the time it takes to train one instance of the PINN for each scenario considered in this work are summarized in Table 4. From this table, we see the PINN optimizes the loss to a small tolerance (at least 1×10^{-6}) in minutes. However, the maximum error increases as the number of solitons increases for KdV, where the BCs are trivially zero, as this part of the loss term does not contribute as much as the equivalent in the line soliton case of KPII.

https://doi.org/10.1017/jnw.2026.10050 Published online by Cambridge University Press

Table 2. *Optimizer parameters used, common to all training cases and runs*

Parameter	**Value**
Adam	
Learning rate	1×10^{-3}
Number of epochs	1000
L-BFGS	
Learning rate	1.0
Max iterations	5×10^{4}
Max evaluations	1×10^{5}
Tolerance (grad)	1×10^{-8}
Tolerance (change)	1×10^{-16}
History size	300
Line search method	Strong Wolfe

Table 3. *A summary of architecture size, training data counts and loss weights used for PINNs trained to solve the KdV and KP equations for various solution cases*

		Architecture		**Training Data**			**Loss Weights**		
	Case	Neurons	Layers	N_{BC}	N_{IC}	N_{PDE}	w_{BC}	w_{IC}	w_{PDE}
KdV	1-soliton	16	4	1×10^{4}	3×10^{4}	5×10^{4}	1.0	5.0	15.0
	2-soliton	62	7	1×10^{4}	1×10^{4}	1×10^{5}	1.0	10.0	100.0
KP	1-soliton	32	3	1×10^{4}	1×10^{4}	5×10^{4}	1.0	1.0	10.0
	x-junction	32	8	1×10^{4}	1×10^{4}	5×10^{4}	1.0	1.0	10.0
	y-junction	32	8	1×10^{4}	1×10^{4}	5×10^{4}	1.0	1.0	10.0

Notes: We see that the number of neurons and layers increases when going from one to two solitons in the case of KdV, but does not change depending on the junction type for KP, as expected.

Table 4. *A summary of the performance (max normalized absolute error $||e||_{\infty}$, final training loss and training time) of PINNs, trained and tested for various solution cases of the KdV and KP equations*

		Performance		
	Case	$\|\|e\|\|_{\infty}$	Final loss	Time [mm:ss]
KdV	1-soliton	2.92×10^{-3}	1.08×10^{-7}	00:20
	2-soliton	1.02×10^{-1}	4.48×10^{-7}	10:39
KP	1-soliton	2.74×10^{-2}	6.61×10^{-8}	02:58
	x-junction	1.55×10^{-2}	2.31×10^{-7}	03:57
	y-junction	1.59×10^{-2}	1.14×10^{-6}	03:02

Notes: We see the PINN optimizes the loss to a small tolerance (at least 1×10^{-6}) in a relatively short amount of time (min). However, the maximum error increases for the case of KdV as we go from one to two solitons.

https://doi.org/10.1017/jnw.2026.10050 Published online by Cambridge University Press

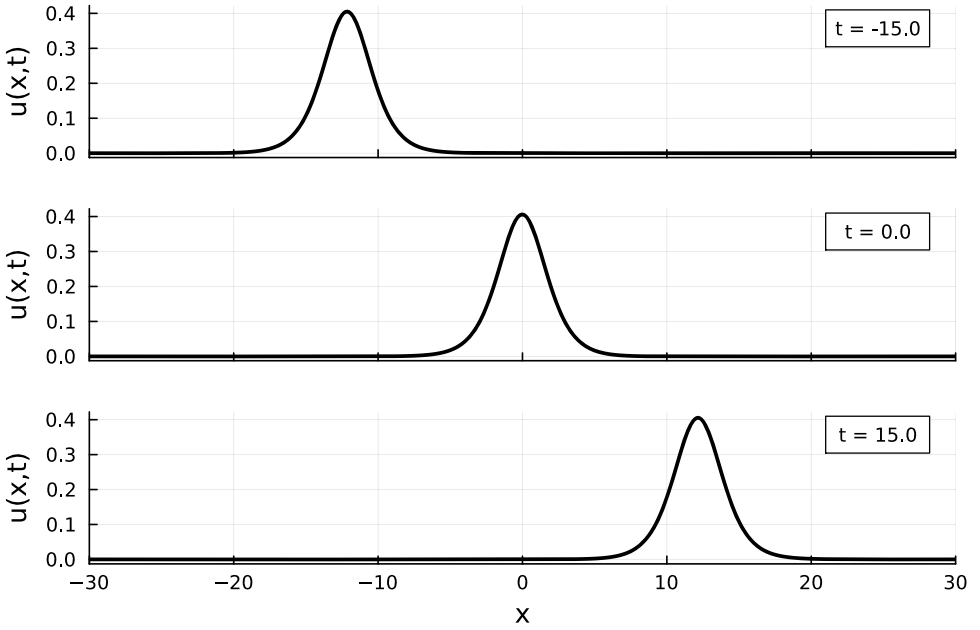


Figure 2. *Solutions to the KdV equation for the single-soliton IC computed by a PINN over a test set of points within the training domain. Plots (top to bottom) show profiles at the initial, middle and final training time. The soliton is seen travelling at a constant speed while maintaining its shape, as expected.*

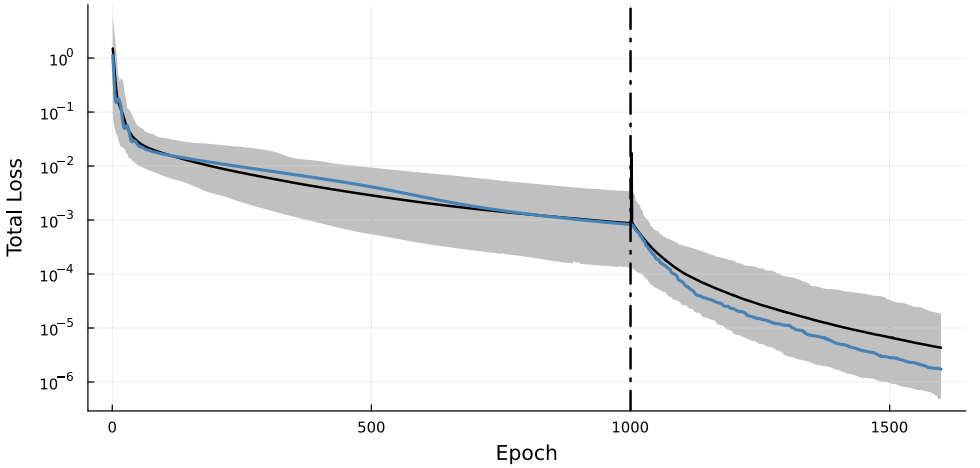


Figure 3. *Total loss during training of a PINN that solves the KdV equation for the single soliton IC. The blue solid line shows the training history of the PINN giving solutions seen in Figure 2; the black solid line represents the mean, with the grey region representing the 0.02–0.98 inter-quantile range. The dashed vertical line represents where the optimizer was switched from Adam to L-BFGS.*

The solutions that were found by the PINN were examined using $N_{\text{test}} = 1 \times 10^6$ test set of points evenly sampled from the domain. For the KdV, these solutions are shown in Figure 2 for the one-soliton case and Figure 6 for the two-soliton case. From both of these cases, we can see the solitons travel at a constant speed while maintaining their shape and amplitude, except during the nonlinear interaction depicted in Figure 6. When compared with a superposition of two single solitons, depicted by the dashed

https://doi.org/10.1017/jnw.2026.10050 Published online by Cambridge University Press

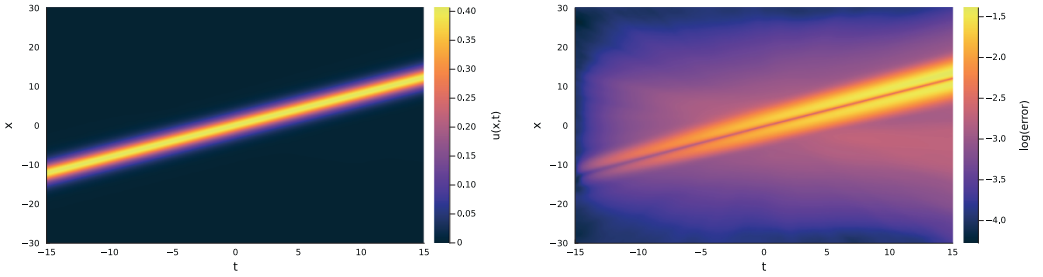


Figure 4. *Space-time portraits of the one-soliton solutions to the KdV equation (2.1) (left) and the log of normalized absolute error (right) given by (4.1) between the approximate and analytic solution evaluated on the test set of points, for the solutions seen in Figure 2. The largest error is for later times, near the location of the largest part of the soliton.*

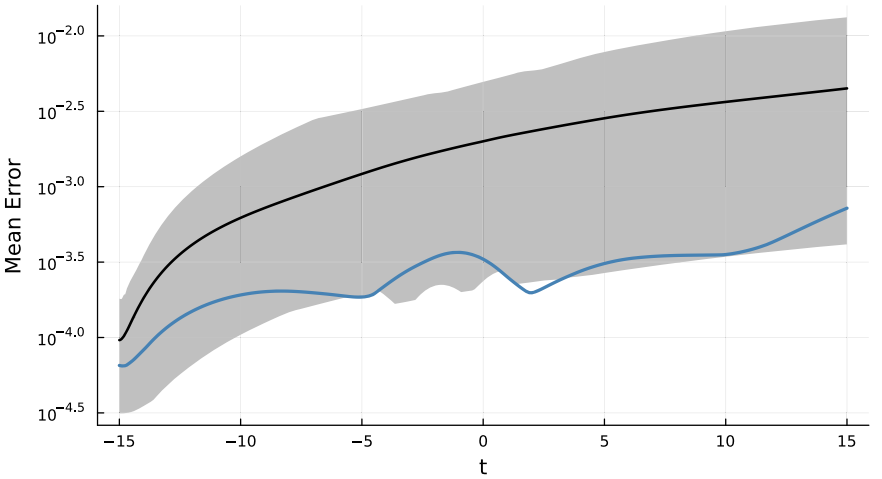
blue lines in the same figure, it can be seen that the interaction is truly nonlinear and that the solitons emerge from the interaction with only a change in phase.

The training loss given by equation 3.1 that was optimized for the case of one- and two-soliton ICs of the KdV equation is shown in Figures 3 and 7, where the blue line shows the training history of the PINN that computes the particular solutions we show in this work, with the black solid line representing the mean and grey region representing the 0.02–0.98 inter-quantile range for 100 training runs (replications). Note that the plots are truncated to that of the shortest training run, with the full component-wise loss histories shown in Appendix B, in Figure B1. The variance in training runs is explained due to having a random initialization of the network's trainable weights and biases, and the set of collocation points chosen randomly within the domain. The dashed vertical line represents where the optimizer was switched from Adam to L-BFGS. As can be seen, in particular from the training loss for the KdV two-soliton case, when using the Adam optimizer, the loss tends to oscillate, suggesting the optimizer keeps missing the direction of the smallest gradient. While it may be possible to adjust the learning rates or other parameters to prevent this from happening, when switching to the L-BFGS, this oscillatory behaviour is no longer seen, and the optimizer performs better at finding a path to a minimum of the total loss.

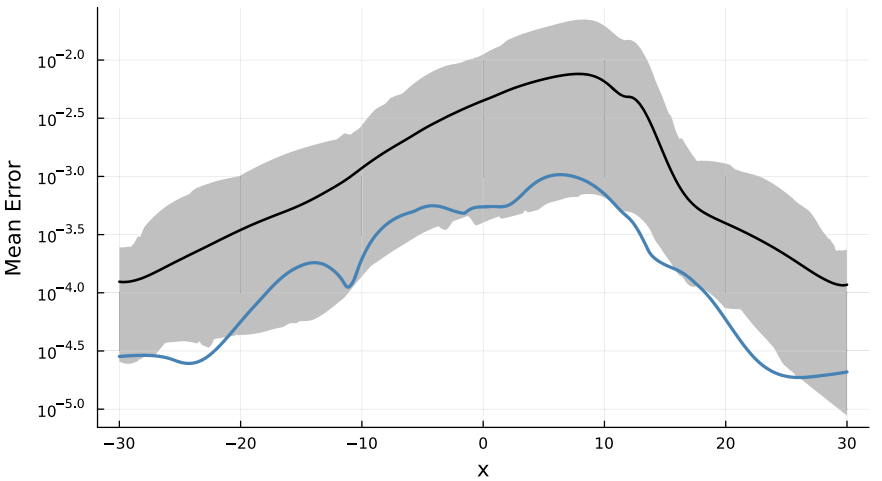
When comparing the PINN-predicted solutions to KdV to the analytic solutions using (4.1), we see from Table 4 that the maximum normalized error for one soliton is 2.92×10^{-3} and for two solitons is 1.02×10^{-1}, with Figure 5 and Figure 8 showing in more detail in which region these errors are largest. These figures depict the mean error averaged over t (top) or x (bottom), with the grey region again representing the 0.02–0.98 inter-quantile range for 100 training runs (replications), with the black line representing the mean and the blue line representing the particular solutions we examined. As expected, since the ICs and the BCs are in the loss function, the error in these regions is smallest. Figures 4 and 9 show the space-time portrait of the KdV one- and two-soliton solutions on the left, with the log of the normalized error for 100 training runs (replications) on the right. It can be seen that the PINN solution captures the amplitude of the solitons and the constant speed, but the largest error is at the furthest time examined, with the PINN not capturing the shape of the largest solitons as well.

The solutions of the KPII equations as approximated by the PINN and examined on a test set of points are shown in Figure 10 for the one-line-soliton case, Figure 13 for the case where two-line solitons interact in an x-junction and Figure 16 for the case where the two-line solitons interact in a y-junction, with ICs shown in Table 1. We see that the one-line soliton travels at constant speed without changing shape. For the case of interacting solitons, the IC is set such that the interaction is outside the domain, but we observe that the PINN captures it well once it enters the computational domain.

https://doi.org/10.1017/jnw.2026.10050 Published online by Cambridge University Press



(a) Error averaged over t



(b) Error averaged over x

Figure 5. *The absolute point-wise normalized error (4.1) averaged over (top) t and (bottom) x, of the one-soliton solution for the KdV equation (2.1) computed by the PINN over the test domain. The blue line represents the PINN giving solutions seen in Figure 2; the black line represents the mean, with the grey region representing the 0.02–0.98 inter-quantile range. The error is smallest at the boundaries and for the initial time, as expected from the form of the loss given by (3.1).*

The training losses given by equation (3.1) that were optimized for the KPII equation are shown in Figures 12, 15 and 18, where the blue line shows the training history of the PINN that computes the particular solutions we show in this work, with the black solid line representing the mean and the grey region representing the 0.02–0.98 inter-quantile range for 50 training runs (replications). The full component-wise loss histories for the PINNs solving the KPII equation are shown in Figures B2 and B3. The dashed vertical line represents where the optimizer was changed from Adam to L-BFGS. These loss curves appear to have less spread between different runs than the ones for KdV, and we postulate that this is due to the nontrivial BCs that were sampled from exact solutions for the KPII equation.

https://doi.org/10.1017/jnw.2026.10050 Published online by Cambridge University Press

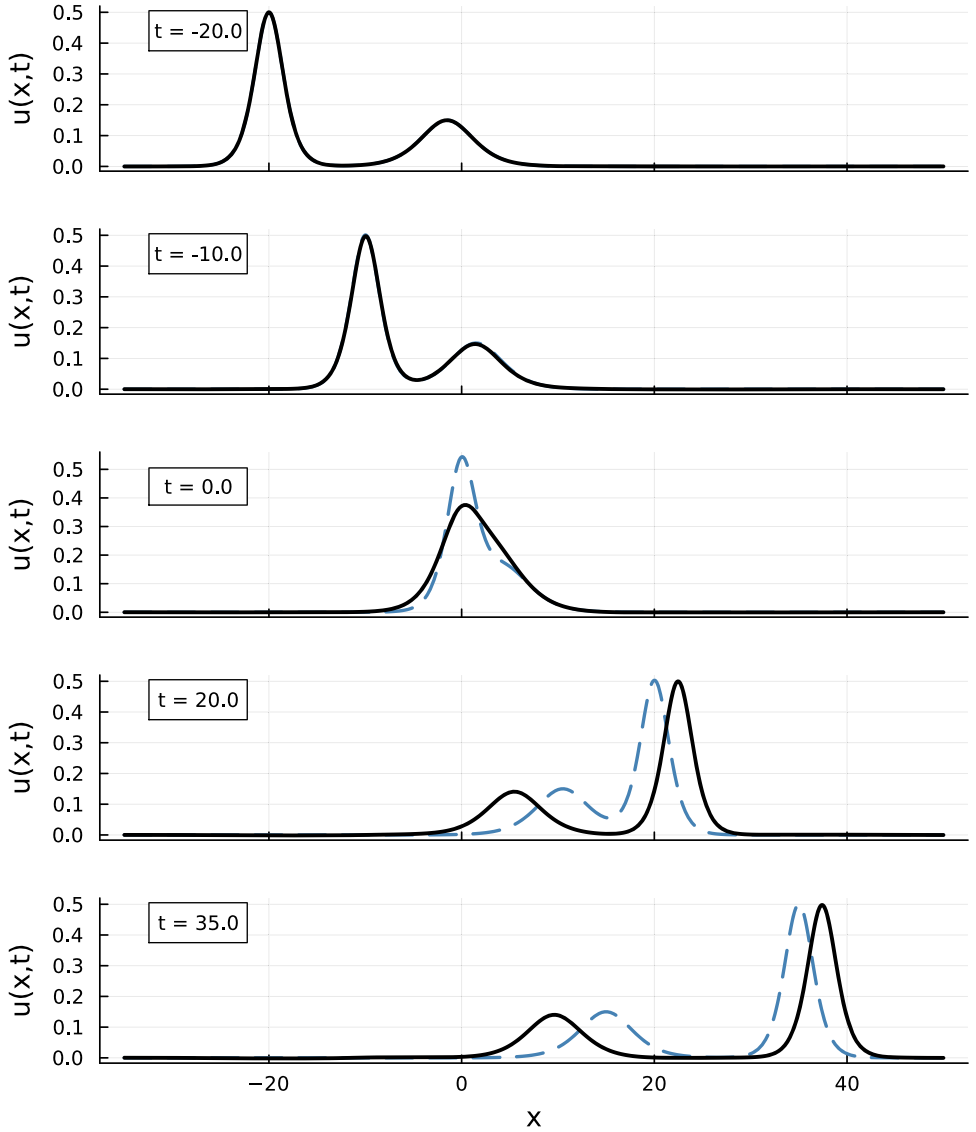


Figure 6. *Solutions to the KdV equation for the two-soliton IC computed by a PINN and shown at various times (as labelled), with topmost and bottom-most panels corresponding to initial and final training times, respectively. The solid line represents the nonlinear solution approximated by the PINN, and the dashed blue line is the linear combination of single soliton solutions. This highlights that the two solitons of the approximated solution emerge from the nonlinear interaction while maintaining their shape and amplitude but each with a phase shift.*

When comparing the KPII solutions from the PINN to the analytic solutions using (4.1), the maximum normalized error is always on the order of 10^{-2} with Figure 11, Figure 14 and Figure 17 showing the mean error averaged over y, t (top), x, t (middle) and x, y (bottom), with the grey region representing the 0.02–0.98 inter-quantile range for 50 training runs (replications), the black line representing the mean and the blue line representing the particular solutions we examined. As expected, because the ICs and the BCs are enforced via the loss function, the error in these regions is smallest. It can also be seen that the PINN solution has a larger error in the region where the line solitons interact

https://doi.org/10.1017/jnw.2026.10050 Published online by Cambridge University Press

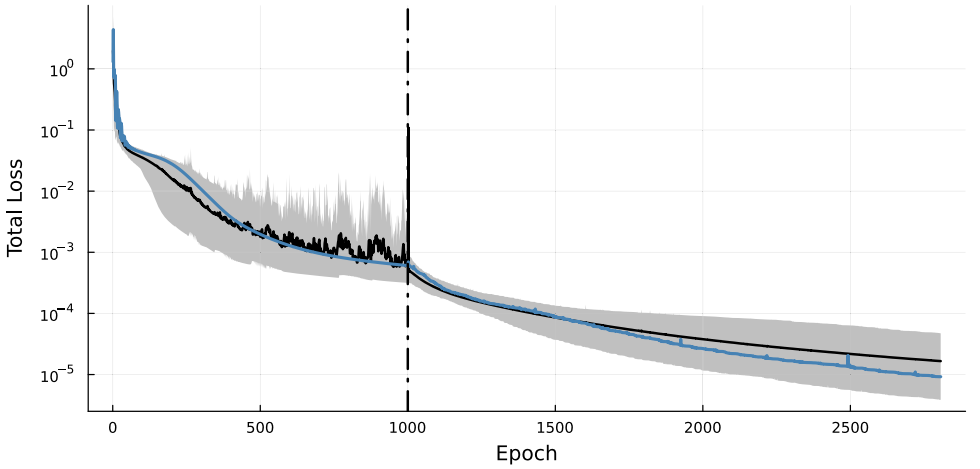


Figure 7. *Total loss during training of a PINN that solves the KdV equation (2.1) for the two-soliton IC. The blue solid line shows the training history of the PINN giving solutions seen in Figure 2; the black solid line represents the mean, with the grey region representing the 0.02–0.98 inter-quantile range. The dashed vertical line represents where the optimizer was switched from Adam to L-BFGS.*

than that in the region where there is no interaction; however, the overall mean error is still relatively small.

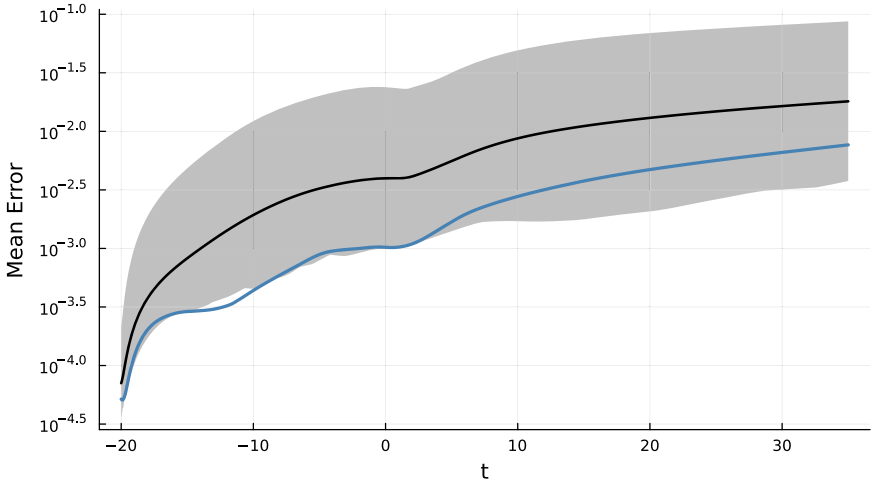
4.1. Generalization to more solitons

We now extend the PINN approach beyond the one- and two-soliton cases of Section 4 to three-soliton KdV solutions and a three-line-soliton KP junction ('2-in-3-out'). As shown below, the fixed architecture and static collocation sampling sufficient for one and two solitons no longer reliably converge at this level of complexity. This motivated the residual-based adaptive sampling scheme summarized in Algorithm 2, given in full in Appendix C, together with a direct comparison against non-adaptive sampling for the two-soliton case and further error and loss detail for both three-soliton cases below.

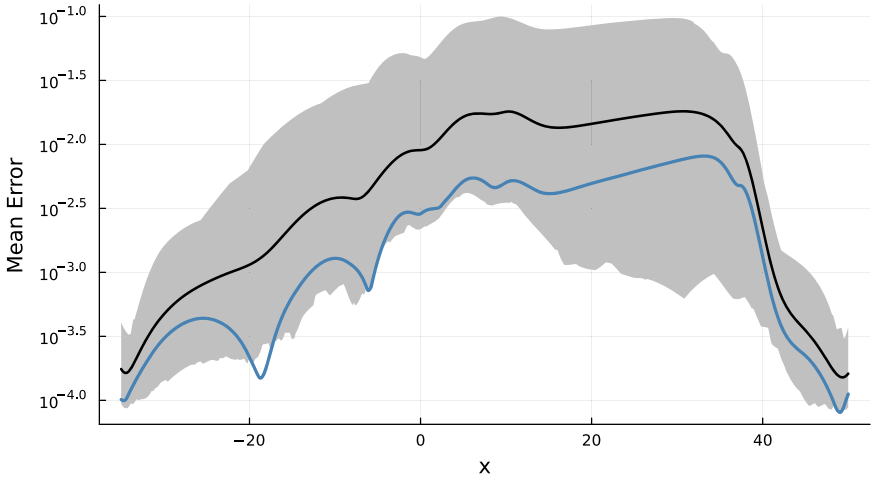
Figure 19 shows the resulting KdV solution at five times spanning training. At early and intermediate times ($t - 6$), the PINN-predicted solution (black) closely tracks the linear combination of individual solitons (blue dashed), since the true nonlinear interaction has not yet substantially diverged from a linear superposition at this stage; by $t = 12.5$, once the three solitons are interacting, the PINN solution visibly departs from this linear combination, indicating that the nonlinear interaction – never supplied in the IC – is being recovered from the PDE residual alone. At the latest times shown ($t = 31.2$, $t = 50.0$), it is the largest-amplitude, fastest-moving soliton (rightmost) that shows the largest growing positional and amplitude lag relative to the exact solution. Since the speed of an individual soliton is proportional to its amplitude (Section 2), this soliton travels furthest from the directly supervised initial data over the course of the domain, and is therefore the one the network must extrapolate furthest to represent accurately; this is consistent with the error concentrating along its trajectory in the space-time portrait (Figure 20). Since three well-separated solitons require a substantially larger spatial domain than the one- and two-soliton cases to avoid boundary effects, this case also required proportionally more collocation points and training time (Tables 5 and 6), even before accounting for the additional cost of the adaptive resampling stage itself.

Figure 21 shows the resulting KP solution for the three-line-soliton '2-in-3-out' junction at four times spanning training; as with the one- and two-line-soliton KP cases above, the IC is set so the interaction is outside the computational domain, and the junction is captured correctly once it enters the domain. Figure 22 shows the corresponding error and Figure 23 the training loss; as in the KdV three-soliton

https://doi.org/10.1017/jnw.2026.10050 Published online by Cambridge University Press



(a) Error averaged over t



(b) Error averaged over x

Figure 8. *The absolute point-wise normalized error (4.1) averaged over t (top) and x (bottom), of the two-soliton solution for the KdV equation (2.1) computed by the PINN over the test domain. The blue line represents the PINN giving solutions seen in Figure 6; the black line represents the mean, with the grey region representing the 0.02–0.98 inter-quantile range. The error is smallest at the boundaries and for the initial time, as expected from the form of the loss given by (3.1).*

case, error concentrates along the individual line-soliton trajectories and around the interaction region itself, rather than being spread uniformly across the domain.

4.2. Comparison with classical methods

When compared directly to classical numerical solvers for these same equations, our PINN results are slower and less accurate for a problem with known initial and boundary data, and this gap widens as solution complexity increases. Pseudospectral methods for the one- and two-soliton KdV problem typically reach relative errors of 10^{-6}–10^{-8} in well under a second of wall-clock time on a single core [51], compared to 2.92×10^{-3} in 20 s for one soliton and 1.02×10^{-1} in 10 min and 39 s for two solitons

https://doi.org/10.1017/jnw.2026.10050 Published online by Cambridge University Press

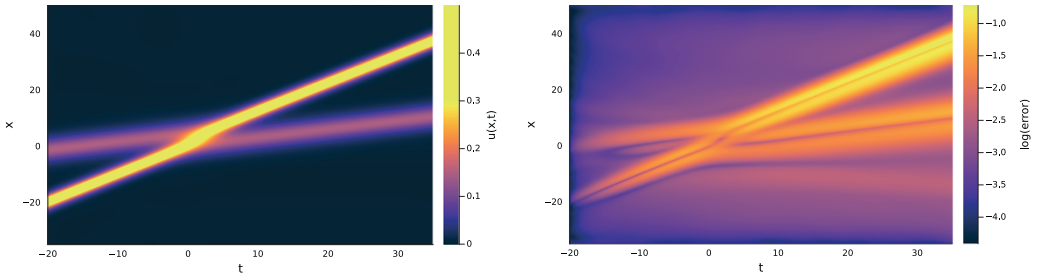


Figure 9. *Space-time portraits of the two-soliton solutions to the KdV equation (2.1) (left) and the log of normalized absolute error (right) given by (4.1) between the approximate and analytic solution evaluated on the test set of points, for the solutions seen in Figure 6. The largest error is for later times, at the location of the largest part of the soliton.*

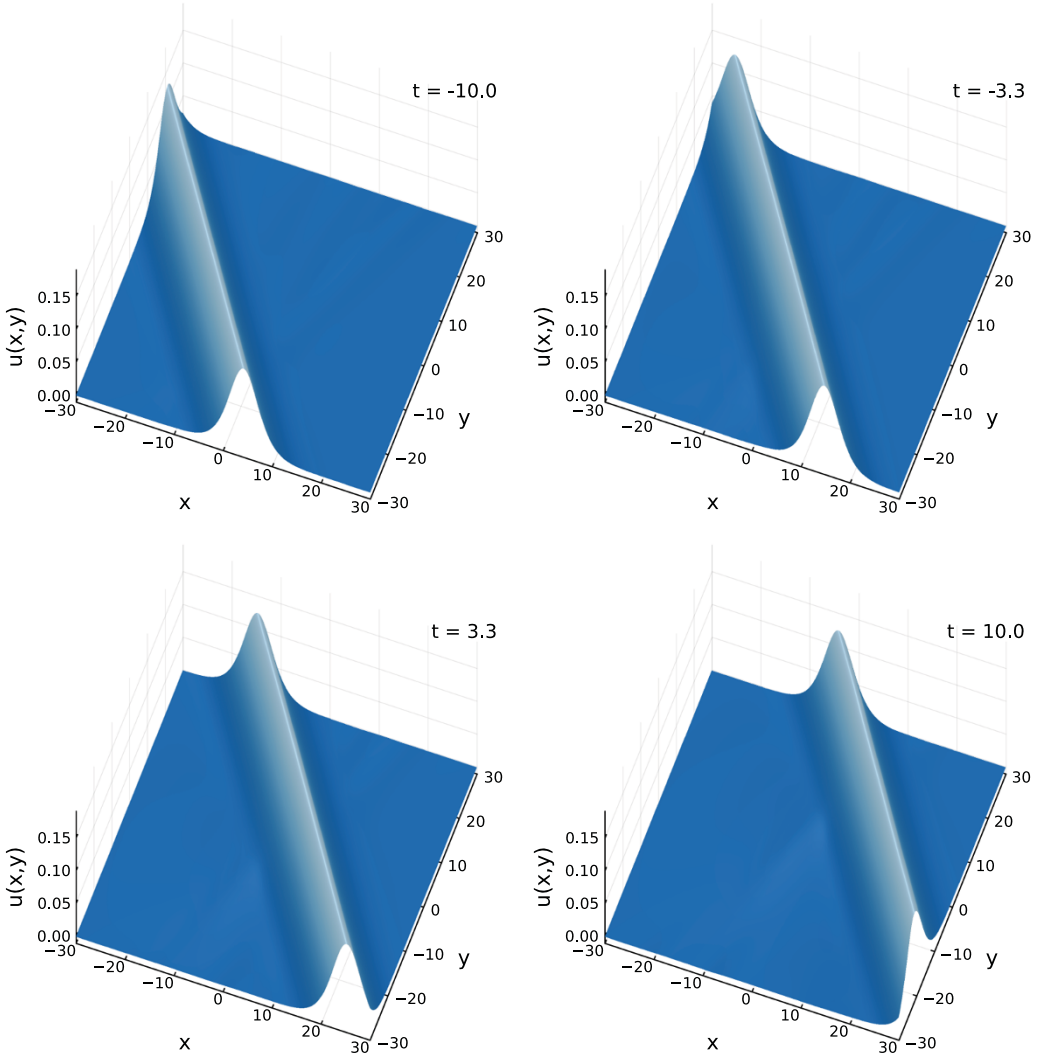
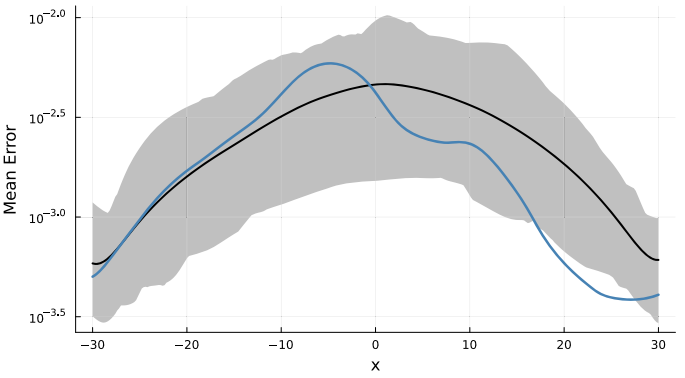
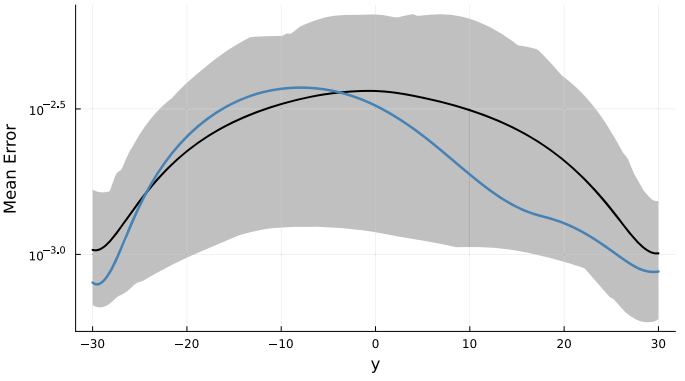


Figure 10. *Solutions to the KPII equation (2.2) for the single line soliton case computed by a PINN over a test set of points and shown at various times (as labelled) spanning the training domain. The set of parameters used for the soliton is given in Table 3, as in [1].*

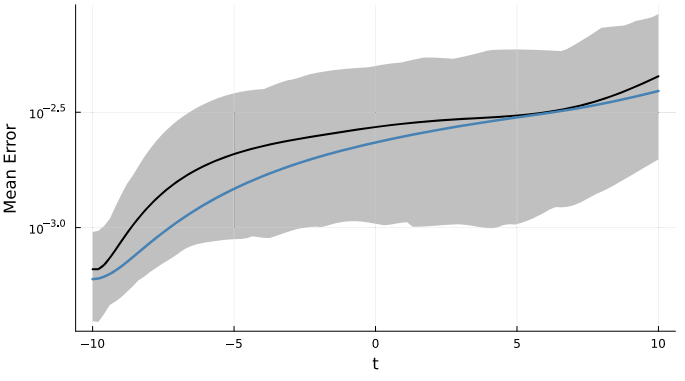
https://doi.org/10.1017/jnw.2026.10050 Published online by Cambridge University Press



(a) Error averaged over y and t



(b) Error averaged over x and t



(c) Error averaged over x and y

Figure 11. *The absolute point-wise normalized error (4.1) averaged over y and t (top), x and t (middle), and x and y (bottom), of the single line soliton solution for the KPII equation (2.2) computed by the PINN over the test domain. The blue solid line represents the PINN giving solutions seen in Figure 10; the black solid line represents the mean, with the grey region representing the 0.02–0.98 inter-quantile range. We note that the error is smallest at the boundaries and for the initial time, as expected from the form of the loss given by (3.1).*

https://doi.org/10.1017/jnw.2026.10050 Published online by Cambridge University Press

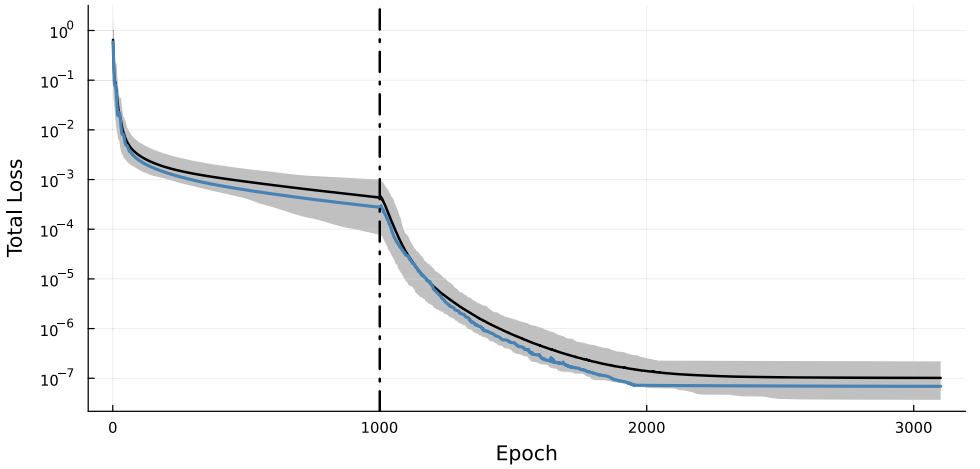


Figure 12. *Total loss during training of a PINN that solves the KPII equation (2.2) for the single line soliton IC. The blue solid line shows the training history of the PINN giving solutions seen in Figure 2; the black solid line represents the mean, with the grey region representing the 0.02–0.98 inter-quantile range. The dashed vertical line represents where the optimizer was switched from Adam to L-BFGS.*

with our non-adaptive PINN (Table 4). The residual-based adaptive sampling scheme introduced above and detailed in Appendix C narrows this gap somewhat for the two-soliton case, reducing the error to 6.13×10^{-2} in 6 min and 45 s (Table 6), but even with this improvement, it remains several orders of magnitude short of pseudospectral accuracy.

The three-soliton case, trained with the same adaptive scheme, reaches a relative error of 2.13×10^{-1} in 36 min and 33 s (Table 6), with the PINN-predicted solitons visibly lagging the analytic solution at the latest times examined, concentrated on the largest, fastest-moving soliton (Figures 19 and 20). This is consistent with the direct comparison of Ortiz, Marín and Ortiz [46], who report a Crank–Nicolson Fourier solver reaching target accuracy for the generalized KdV equation in ∼0.8 s, versus ∼65 s of PINN training for a comparable, though generally less accurate, solution. Bai, Koley, Mishra and Molinaro [6] similarly report degrading PINN efficiency with increasing soliton count for KdV-type equations, and Haight, Roudenko and Wang [21] find that classical solvers retain higher accuracy and efficiency than PINNs for single-instance one-dimensional dispersive-wave problems more broadly.

For the KP equation, our one-line-soliton and x-/y-junction results reach a maximum normalized error of order 10^{-2} (Table 4), and the three-line-soliton junction reaches 5.63×10^{-2} in 34 min and 59 s (Table 6). Because KP line solitons are not spatially localized and instead extend to the boundary of any finite computational domain, classical solvers require some form of domain truncation or absorbing/periodic boundary treatment tailored to the equation [1], in contrast to the approach used here, where the boundary is instead enforced directly as a labelled loss term; we are not aware of a directly comparable classical accuracy or timing benchmark for this specific line-soliton junction configuration.

5. Conclusion and discussion

In this work, we showed that PINNs can be used to find soliton solutions to nonlinear dispersive PDEs, focusing on the KdV and KP equations. Our results show that PINNs can successfully find one-, two- and three-soliton solutions for KdV, and single line-soliton, x-junction, y-junction and three-line-soliton ('2-in-3-out') solutions for KP. In all cases, the PINN was only given individual, non-interacting single

https://doi.org/10.1017/jnw.2026.10050 Published online by Cambridge University Press

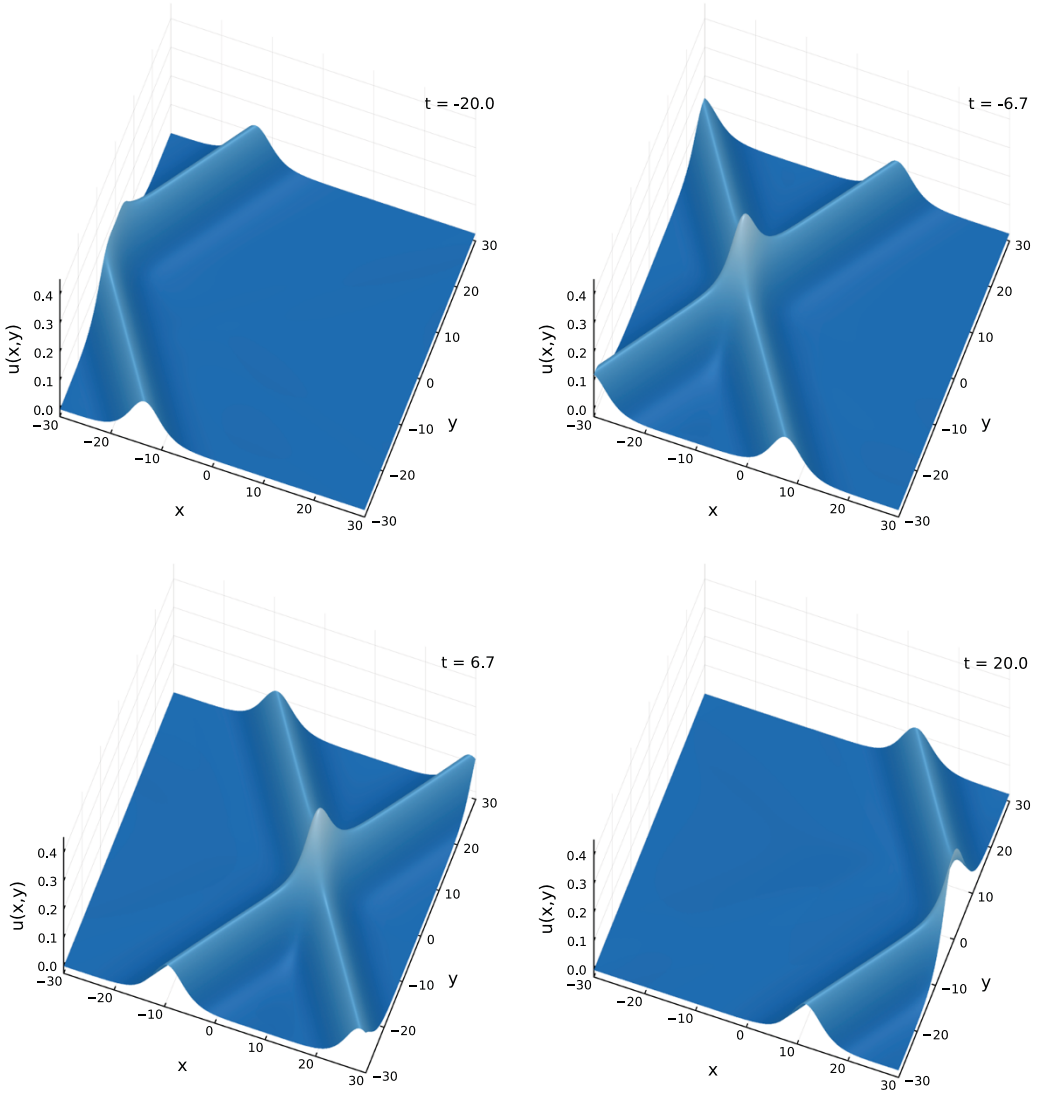
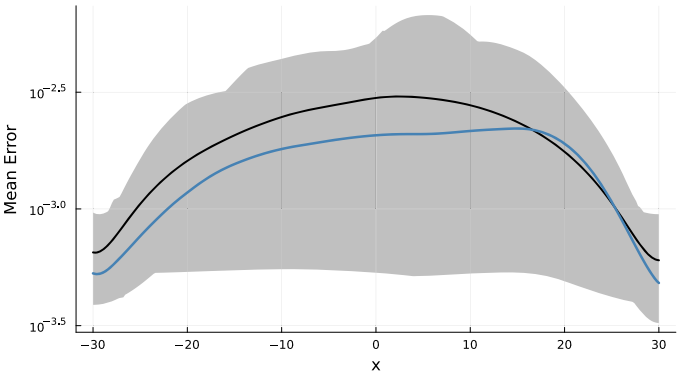


Figure 13. *Solutions to the KPII equation (2.2) for the case of two-line solitons interacting in an x-junction, computed by a PINN over a test set of points and shown at various times (as labelled) spanning the training domain. The set of parameters used for the soliton is given in Table 3, as in [1].*

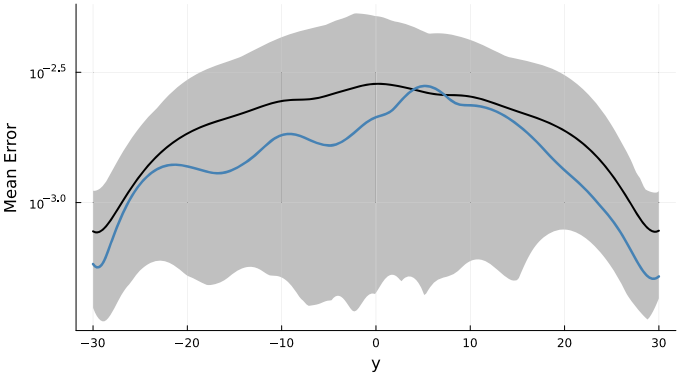
solitons in the IC, and the nonlinear interaction between them was captured without being supplied directly.

We illustrated the success of these PINN-approximated solutions by comparing them to the known analytic solutions and showed good agreement, with the maximum normalized error generally between 10^{-1} and 10^{-3} (see Table 4). The main features of solutions such as constant speed and wave amplitude of the single soliton (Figure 2 and Figure 10), as well as nonlinear interaction between two solitons and phase changes (Figure 6), were correctly captured. PINNs were also able to model the line soliton interactions in the KP equation, including x- and y-junctions (Figures 13 and 16). These cases were trained in under 11 min on a GPU. The three-soliton KdV case and the three-line-soliton KP junction took considerably longer, around 35 min each, and are discussed in more detail below.

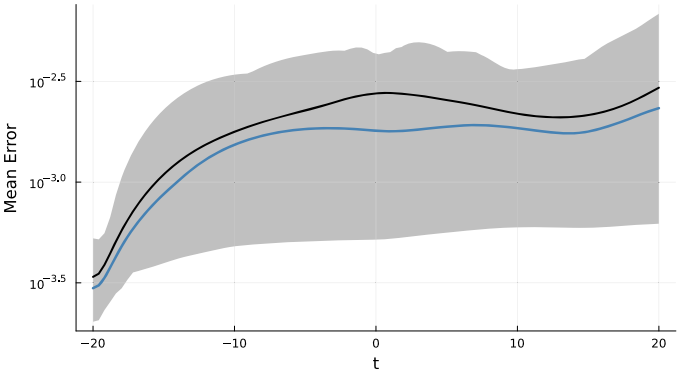
https://doi.org/10.1017/jnw.2026.10050 Published online by Cambridge University Press



(a) Error averaged over y and t



(b) Error averaged over x and t



(c) Error averaged over x and y

Figure 14. *The absolute point-wise normalized error (4.1) averaged over y and t (top), x and t (middle), and x and y (bottom), of the x-junction solution for the KPII equation (2.2) computed by the PINN over the test domain. The blue solid line represents the PINN giving solutions seen in Figure 13; the black solid line represents the mean, with the grey region representing the 0.02–0.98 inter-quantile range. We note that the error is smallest at the boundaries and for the initial time, as expected from the form of the loss given by (3.1).*

https://doi.org/10.1017/jnw.2026.10050 Published online by Cambridge University Press

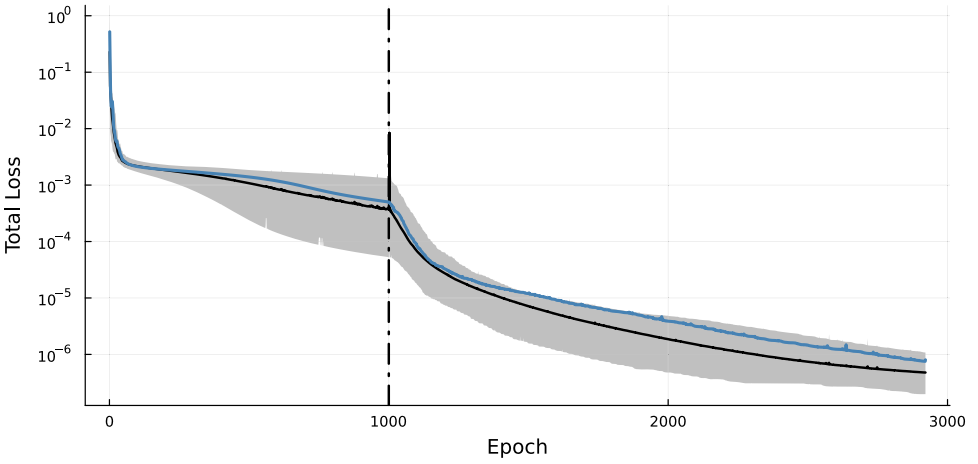


Figure 15. *Total loss during training of a PINN that solves the KPII equation (2.2) for the x-junction IC. The blue solid line shows the training history of the PINN giving solutions seen in Figure 2; the black solid line represents the mean, with the grey region representing the 0.02–0.98 inter-quantile range. The dashed vertical line represents where the optimizer was switched from Adam to L-BFGS.*

The feedforward, fully connected network was trained in two stages, first using the Adam optimizer, and then using the L-BFGS optimizer. This helped reduce the training error and improve accuracy (see Figures 3, 7, 12, 15 and 18). The performance of the PINNs depended on the choice of several types of hyperparameters, including architectural choices (number of hidden layers, number of neurons per layer), training parameters (learning rates, history size) and the weights used in the loss function. Some of these were adjusted depending on the equation and type of solution, and others were kept constant; see Tables 3 and 2. For example, more layers and training points were needed in the cases of the two-soliton KdV or KP solutions. We are not yet aware of a systematic way to choose these hyperparameters, particularly in the case of new problems or parameter regimes.

This sensitivity is most visible in the three-soliton KdV case. Despite training on an adaptively resampled set of collocation points and seeing only a moderate increase in training time relative to the two-soliton case (6:45 to 36:33, Table 6), the maximum normalized error more than triples (6.13×10^{-2} to 2.13×10^{-1}), and the PINN-predicted solitons visibly lag the analytic solution at the latest times examined (Figure 19). This suggests that scaling up the architecture and sampling strategy alone does not fully keep pace with the added solution complexity, and that reducing this lag likely requires changes to the training procedure itself, rather than simply a larger network or more collocation points.

Taken together with the comparison to classical numerical solvers in Section 4.2, this points to three current limitations of the PINN approach used here. First, our PINNs are slower and less accurate than classical pseudospectral solvers for these same equations, and this gap grows rather than shrinks as the number of solitons increases. Second, we do not have a systematic, reliable way to choose the network architecture or loss weights ahead of time for a new number of solitons or a new equation. Third, and related to the first point, accuracy does not simply improve by adding more collocation points or training time once the solution becomes complex enough, as seen in the three-soliton KdV case above. Addressing this last point is likely to require changes to the training procedure itself, and we discuss two promising directions for this below.

In our opinion, the main problem left to address is understanding how deep (number of layers) or wide (number of neurons) to make the network in order to accurately represent the solutions. With traditional methods, there are generally some guidelines for grid spacing or time steps to take when computing the solution that depend on the complexity of the equation or the solutions,

https://doi.org/10.1017/jnw.2026.10050 Published online by Cambridge University Press

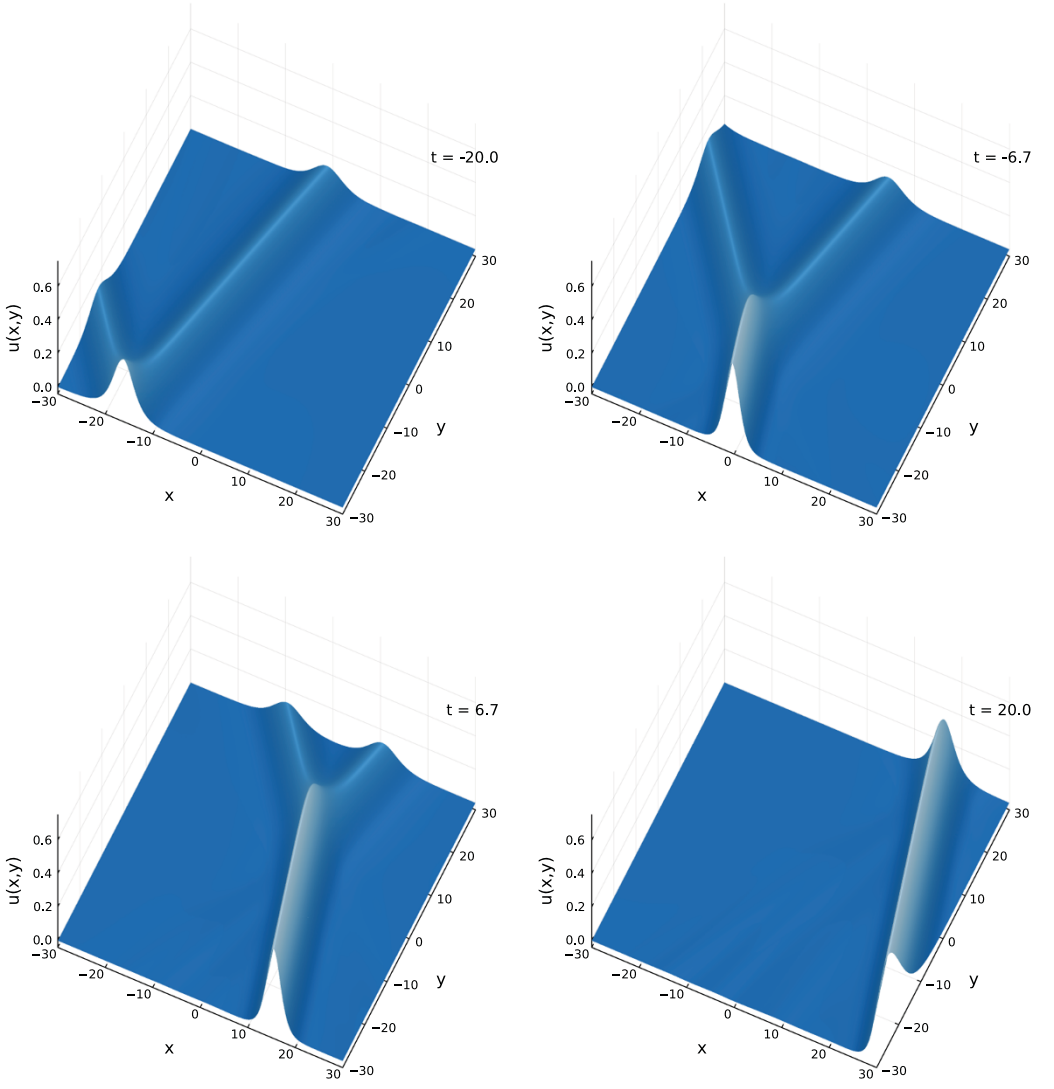
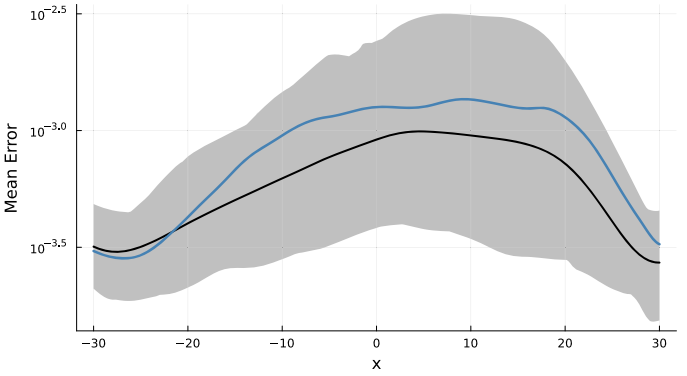


Figure 16. *Solutions to the KPII equation (2.2) for the case of two-line solitons interacting in a y-junction, computed by a PINN over a test set of points and shown at various times (as labelled) spanning the training domain. The set of parameters used for the soliton is given in Table 3, as in [1].*

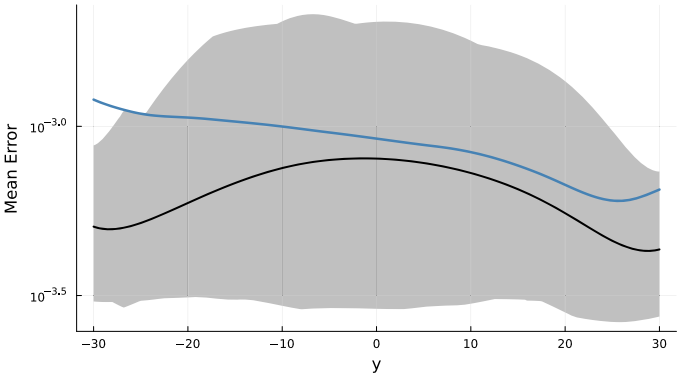
which lead to a more stable or accurate method, but this is not clear in the case of PINNs as of yet.

We also wish to highlight that in this work, we have used the most straightforward implementation and training procedure for PINNs, and there have been many additions to these methods over the past few years, since the implementation by [47]. To begin, other types of network architectures, such as those with residual connections [44], have been proposed. More general formulations which split the domain of interest into subdomains and train an ensemble of generally varying PINNs may also be useful, due to their ability to handle heterogeneity and be parallelized [27]. Rescaling the governing equations has also been shown to improve results [56]. Adding more physical knowledge such as conserved quantities (e.g. energy) could further constrain the solution space to those that are physically relevant [36].

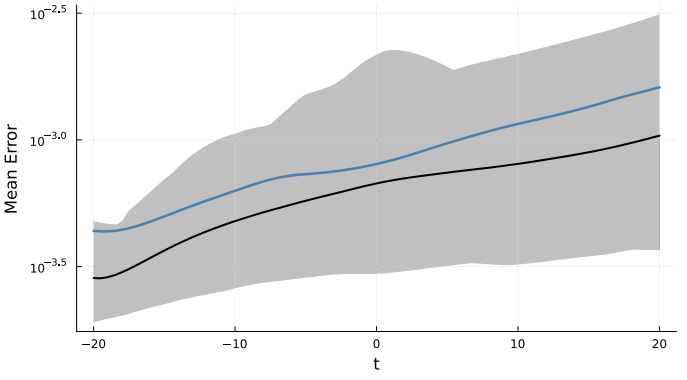
https://doi.org/10.1017/jnw.2026.10050 Published online by Cambridge University Press



(a) Error averaged over y and t



(b) Error averaged over x and t



(c) Error averaged over x and y

Figure 17. *The absolute point-wise normalized error (4.1) averaged over y and t (top), x and t (middle), and x and y (bottom), of the y-junction solution for the KPII equation (2.2) computed by the PINN over the test domain. The blue solid line represents the PINN giving solutions seen in Figure 16; the black solid line represents the mean, with the grey region representing the 0.02–0.98 inter-quantile range. We note that the error is smallest at the boundaries and for the initial time, as expected from the form of the loss given by (3.1).*

https://doi.org/10.1017/jnw.2026.10050 Published online by Cambridge University Press

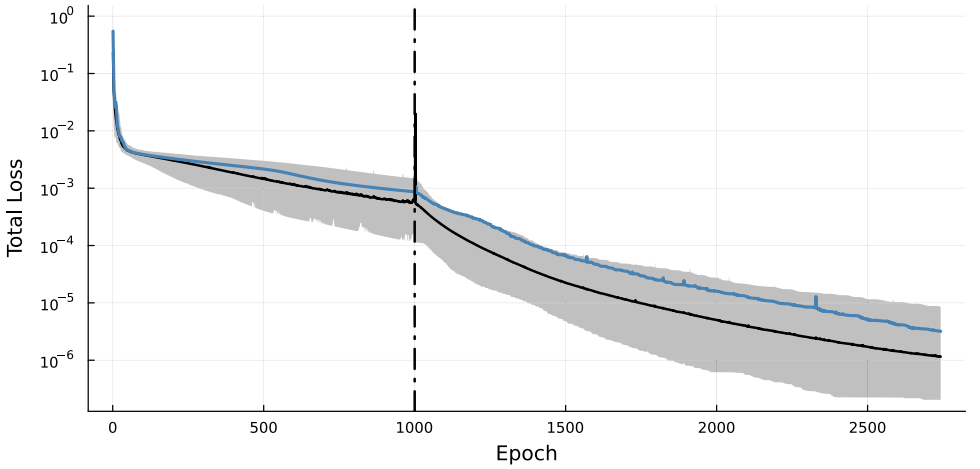


Figure 18. *Total loss during training of a PINN that solves the KPII equation (2.2) for the y-junction IC. The blue solid line shows the training history of the PINN giving solutions seen in Figure 2; the black solid line represents the mean, with the grey region representing the 0.02–0.98 inter-quantile range. The dashed vertical line represents where the optimizer was switched from Adam to L-BFGS.*

Table 5. *KP and KdV 3-soliton parameters and training data*

		Architecture		**Training data**				**Loss weights**		
	Case	Neurons	Layers	N_{BC}	N_{IC}	N_{PDE}	Adaptive	w_{BC}	w_{IC}	w_{PDE}
KdV	2-soliton	62	7	1×10^4	1×10^4	1×10^5	True	1.0	10.0	100.0
	3-soliton	72	6	1×10^4	1×10^4	1×10^5	True	1.0	10.0	100.0
KP	2-in-3 out	62	10	5×10^4	1×10^4	5×10^4	False	1.0	1.0	10.0

Table 6. *KP 2-in-3 out and KdV 2- and 3-soliton performance corresponding to PINNs from Table 5*

		Performance		
	Case	$\|\|e\|\|_\infty$	Final loss	Time [mm:ss]
KdV	2-soliton	6.13×10^{-2}	1.68×10^{-6}	6:45
	3-soliton	2.13×10^{-1}	6.26×10^{-7}	36:33
KP	2-in-3 out	5.63×10^{-2}	1.09×10^{-3}	34:59

The training process can also be augmented in a variety of ways. For example, the condition of a static training set can be relaxed, whereby the training data is updated in an adaptive manner during the training process, based on difficulty [11, 16, 55], an approach we used above for the three-soliton KdV and KP junction cases. The weighting between different loss terms can also be variable, set in an adaptive manner throughout training [9, 12, 31, 54, 59]. This could help directly with the error growth we observe away from the initial and boundary data, since it would let the network place more weight on the unlabelled, interior part of the loss exactly where and when it is struggling most, rather than relying on a fixed balance between loss terms set in advance. The parameters of the activation function can also be adaptive and included in the set trainable parameters, which has been shown to improve training speed [28]. Causality may also be respected, such that the network is encouraged to learn the solution at earlier times prior to later times, which has been shown to improve results [53]. This is a

https://doi.org/10.1017/jnw.2026.10050 Published online by Cambridge University Press

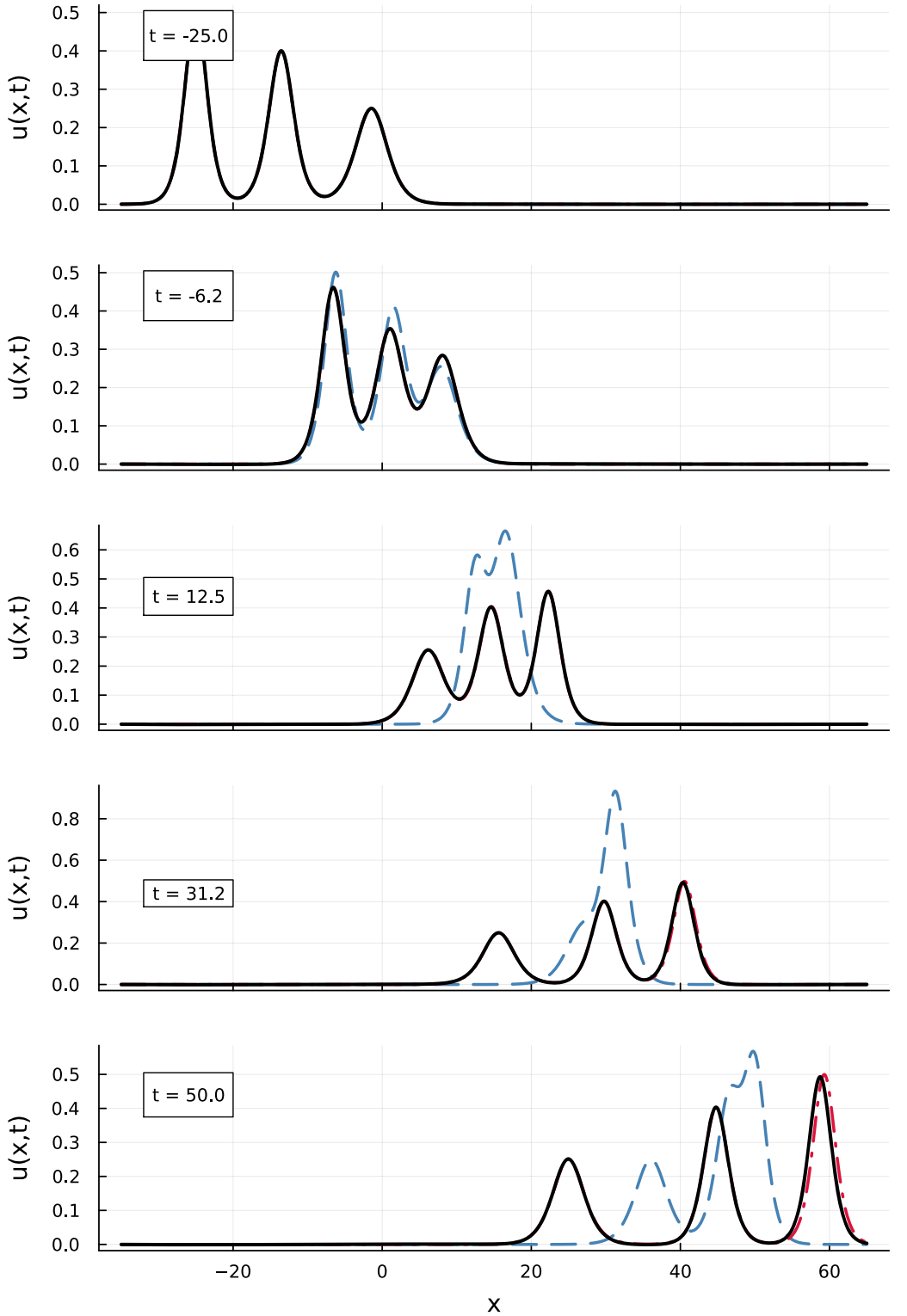


Figure 19. *KdV solutions for three-soliton IC. The black line shows the PINN-predicted solution, the orange dot-dash line shows the exact analytical solution and the blue dotted line shows the linear combination of individual solitons.*

https://doi.org/10.1017/jnw.2026.10050 Published online by Cambridge University Press

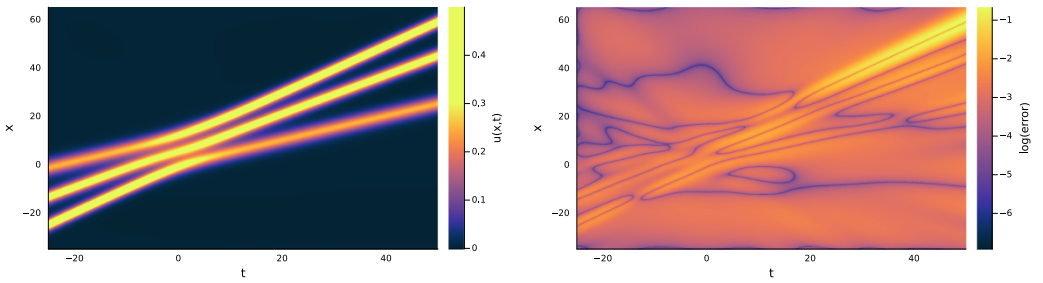


Figure 20. *KdV 3-soliton: space-time portrait (left) and normalized absolute error (right).*

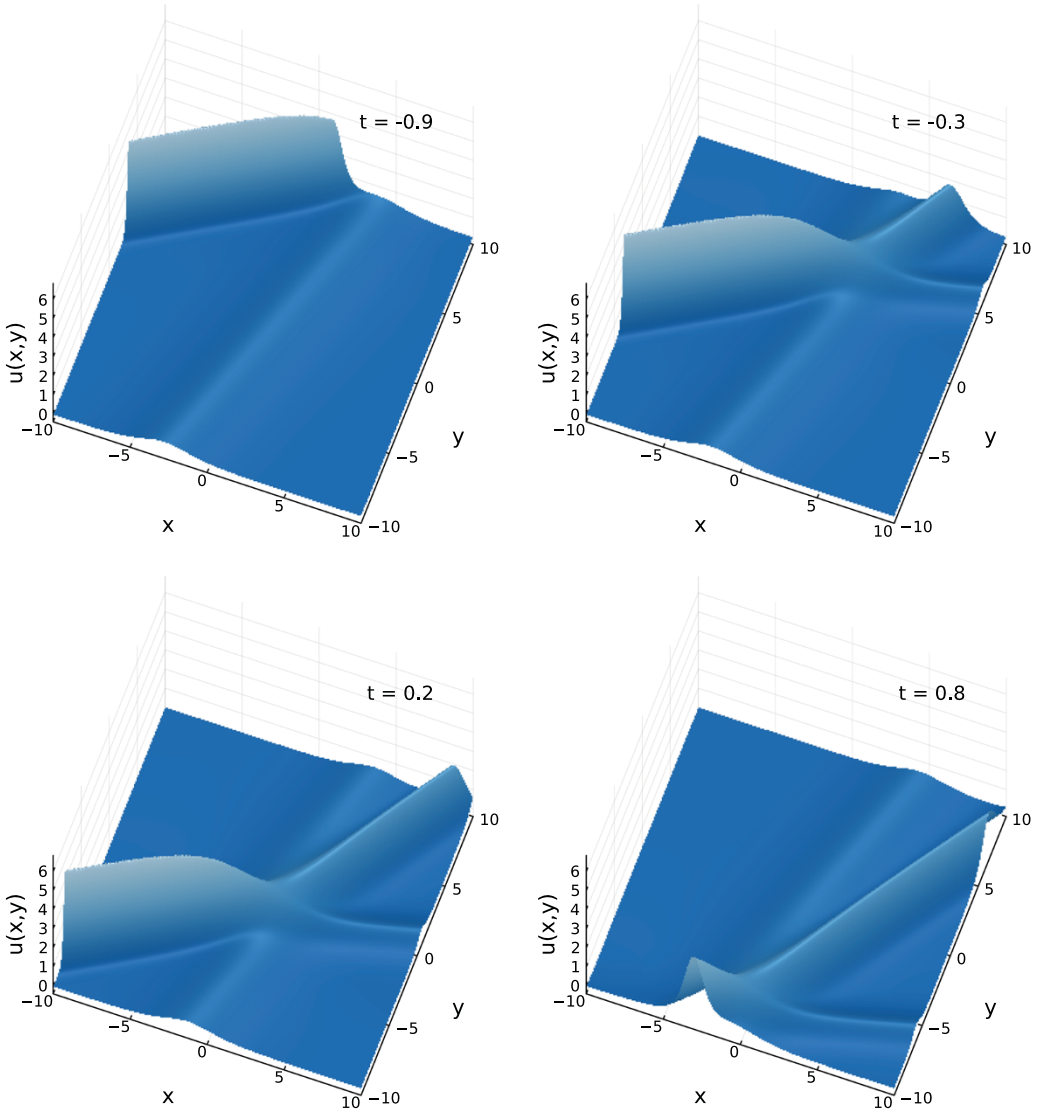
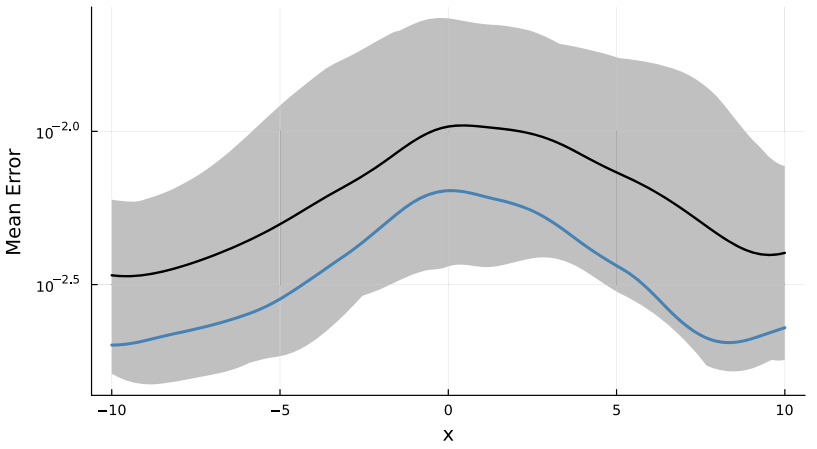
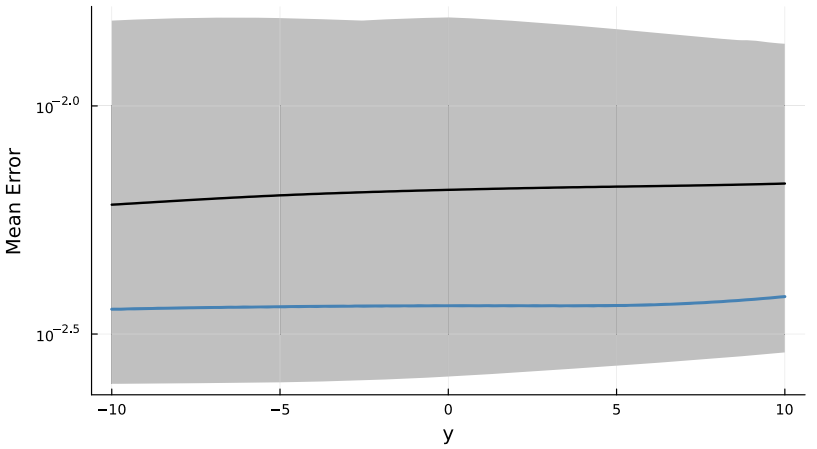


Figure 21. *KP 2-in-3 out interaction.*

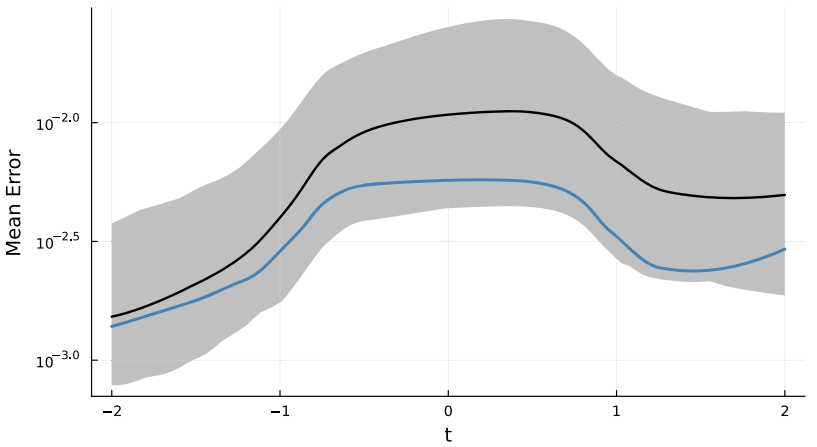
https://doi.org/10.1017/jnw.2026.10050 Published online by Cambridge University Press



(a) Error averaged over y and t



(b) Error averaged over x and t



(c) Error averaged over x and y

Figure 22. *KP three-soliton error.*

https://doi.org/10.1017/jnw.2026.10050 Published online by Cambridge University Press

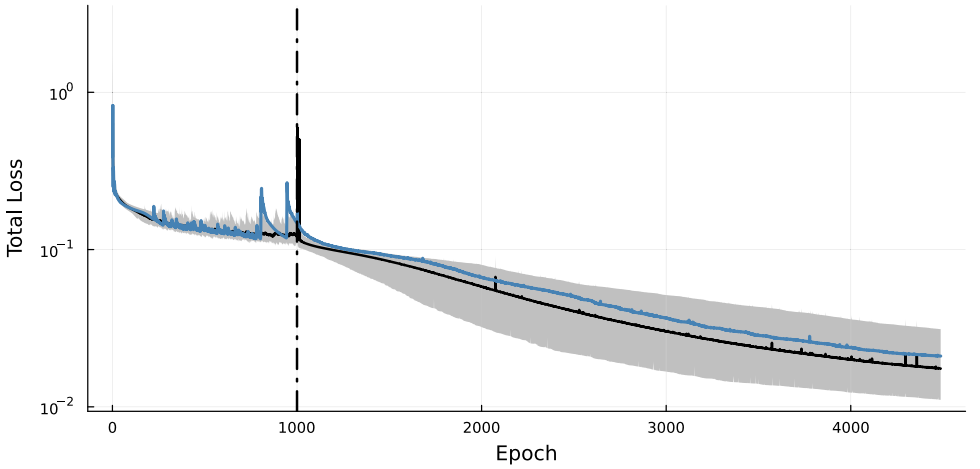


Figure 23. *KP three-soliton loss.*

particularly natural fit for the pattern of error growth we see in our own results, since it directly targets the later times where error tends to be largest.

Moreover, the work we presented here was done using our own code written in PyTorch to better understand how the network learns the solution, but existing packages such as DeepXDE [37] or NeuralPDE [60] could be used for their robust implementations and ease of use.

Data availability statement. The data that support the findings of this study are available from the corresponding author upon reasonable request.

Acknowledgements. We thank the reviewers and editors for their comments, which substantially expanded the scope and depth of this work.

Author contributions. Conceptualization: All authors. Methodology: All authors. Software development: K.R., N.A. and N.H.D. Analysis and validation: All authors. Writing – original draft: K.R., N.A. and O.T. and editing by all authors. All authors approved the final submitted draft.

Funding statement. O.T. acknowledges the support of the Natural Sciences and Engineering Research Council of Canada (NSERC) through Discovery Grant RGPIN-2020-06417.

Competing interests. The authors have no conflicts to disclose.

References

[1] **Ablowitz MJ and Baldwin DE** (2012) Nonlinear shallow ocean-wave soliton interactions on flat beaches. *Physical Review E* **86**(3), 036305.
[2] **Ablowitz MJ and Clarkson PA** (1991) *Solitons, Nonlinear Evolution Equations and Inverse Scattering*. **149**. Cambridge: Cambridge University Press.
[3] **Ablowitz MJ and Segur H** (1979) On the evolution of packets of water waves. *Journal of Fluid Mechanics* **92**(4), 691–715.
[4] **Ablowitz MJ, Yaacov DB and Fokas AS** (1983) On the inverse scattering transform for the kadomtsev-petviashvili equation. *Studies in Applied Mathematics* **69**(2), 135–143.
[5] **Anco S and Willoughby MR** Kdv solitons.
[6] **Bai G, Koley U, Mishra S and Molinaro R** (2021) Physics informed neural networks (PINNs) for approximating nonlinear dispersive PDEs. *arXiv preprint arXiv:2104.05584*.
[7] **Baydin AG, Pearlmutter BA, Radul AA and Siskind JM** (2018) Automatic differentiation in machine learning: A survey. *Journal of Machine Learning Research* **18**(153), 1–43.
[8] **Biondini G, Maruno KI, Oikawa M and Tsuji H** (2009) Soliton interactions of the kadomtsev–petviashvili equation and generation of large-amplitude water waves. *Studies in Applied Mathematics* **122**(4), 377–394.
[9] **Bischof R and Kraus MA** (2025) Multi-objective loss balancing for physics-informed deep learning. *Computer Methods in Applied Mechanics and Engineering* **439**, 117914.

[10] **Boussinesq J** (1872) Théorie des ondes et des remous qui se propagent le long d'un canal rectangulaire horizontal, en communiquant au liquide contenu dans ce canal des vitesses sensiblement pareilles de la surface au fond. *Journal de mathéMatiques Pures et appliquées* **17**, 55–108.

[11] **Celaya A, Fuentes D and Riviere B** (2025) An adaptive collocation point strategy for physics informed neural networks via the qr discrete empirical interpolation method. *arXiv preprint arXiv:2501.07700*.

[12] **Chen Z, Badrinarayanan V, Lee CY and Rabinovich A** (2018) Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. International conference on machine learning, 2018, Stockholm, Sweden. PMLR, pp. 794–803.

[13] **Dissanayake M and Phan-Thien N** (1994) Neural-network-based approximations for solving partial differential equations. *Communications in Numerical Methods in Engineering* **10**(3), 195–201.

[14] **Drazin PG and Johnson RS** (1989) *Solitons: An introduction*. Cambridge: Cambridge University Press.

[15] **Fokas AS and Ablowitz MJ** (1983) On the inverse scattering of the time-dependent schrödinger equation and the associated kadomtsev-petviashvili (i) equation. *Studies in Applied Mathematics* **69**(3), 211–228.

[16] **Gao Z, Yan L and Zhou T** (2023) Failure-informed adaptive sampling for pinns. *SIAM Journal on Scientific Computing* **45**(4), A1971–A1994.

[17] **Gardner CS, Greene JM, Kruskal MD and Miura RM** (1967) Method for solving the korteweg-devries equation. *Physical Review Letters* **19**(19), 1095.

[18] **Gardner CS, Greene JM, Kruskal MD and Miura RM** (1974) Korteweg-devries equation and generalizations. vi. methods for exact solution. *Communications on Pure and Applied Mathematics* **27**(1), 97–133.

[19] **Glorot X and Bengio Y** (2010) Understanding the difficulty of training deep feedforward neural networks. Thirteenth International Conference on Artificial Intelligence and Statistics, Sardinia, Italy. PMLR, volume 9, pp. 249–256.

[20] **Guo Y, Cao X and Peng K** (2023) Solving nonlinear soliton equations using improved physics-informed neural networks with adaptive mechanisms. *Communications in Theoretical Physics* **75**(9), 095003.

[21] **Haight C, Roudenko S Wang Z** (2025) Soliton profiles: Classical numerical schemes vs. neural network-based solvers. *arXiv preprint arXiv:2512.24634*.

[22] **Hietarinta J** (1997) Introduction to the hirota bilinear method. In Y Kosmann-Schwarzbach, B Grammaticos and KM Tamizhmani (eds.), *Integrability of Nonlinear Systems*. Berlin Heidelberg, Berlin, Heidelberg: Springer, pp. 95–103.

[23] **Hirota R** (1971) Exact solution of the korteweg—de vries equation for multiple collisions of solitons. *Physical Review Letters* **27**(18), 1192.

[24] **Hirota R** (2004) *The Direct Method in Soliton Theory*. Cambridge: Cambridge University Press. p. 155.

[25] **Hirota R** (1976) Direct methods of finding solutions of nonlinear evolution equations. In Miura RM (ed.) *Bäcklund Transformations Lecture Notes in Mathematics*, 515, New York: Springer Verlag.

[26] **Hornik K, Stinchcombe M and White H** (1989) Multilayer feedforward networks are universal approximators. *Neural Networks* **2**(5), 359–366.

[27] **Jagtap AD and Karniadakis GE** (2020) Extended physics-informed neural networks (xpinns): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations. *Communications in Computational Physics* **28**(5), 2002–2041.

[28] **Jagtap AD, Kawaguchi K and Karniadakis GE** (2020) Adaptive activation functions accelerate convergence in deep and physics-informed neural networks. *Journal of Computational Physics* **404**, 109136.

[29] **Kadomtsev BB and Petviashvili VI** (1970) On the stability of solitary waves in weakly dispersing media. In *Doklady Akademii Nauk*, Vol. 192, Russian Academy of Sciences, pp. 753–756.

[30] **Kao CY and Kodama Y** (2012) Numerical study of the KP equation for non-periodic waves. *Mathematics and Computers in Simulation* **82**(7), 1185–1218.

[31] **Kendall A, Gal Y and Cipolla R** (2018) Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. IEEE Conference on computer vision and pattern recognition, 2018, Salt Lake City, Utah, USA, pp. 7482–7491.

[32] **Kingma DP and Ba JL** (2014) Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[33] **Korteweg DJ and de Vries G** (1895) Xli. on the change of form of long waves advancing in a rectangular canal, and on a new type of long stationary waves. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* **39**(240), 422–443.

[34] **Lax D** (1968) Integrals of nonlinear equations of evolution and solitary waves. *Communications on Pure and Applied Mathematics* **21**, 467–490.

[35] **Li J, Chen J and Li B** (2022) Gradient-optimized physics-informed neural networks (gopinns): A deep learning method for solving the complex modified kdv equation. *Nonlinear Dynamics* **107**, 781–792.

[36] **Lin S and Chen Y** (2022) A two-stage physics-informed neural network method based on conserved quantities and applications in localized wave solutions. *Journal of Computational Physics* **457**, 111053.

[37] **Lu L, Meng X, Mao Z and Karniadakis GE** (2021) Deepxde: A deep learning library for solving differential equations. *SIAM review* **63**(1), 208–228.

[38] **Manakov S and Zakharov V** (1981) Three-dimensional model of relativistic-invariant field theory, integrable by the inverse scattering transform. *Letters in Mathematical Physics* **5**, 247–253.

[39] **Matthies H and Strang G** (1979) The solution of nonlinear finite element equations. *International Journal for Numerical Methods in Engineering* **14**(11), 1613–1626.

https://doi.org/10.1017/jnw.2026.10050 Published online by Cambridge University Press

[40] **Matveev VB and Salle MA** (1992) *Darboux Transformations and Solitons*. Springer Series in Nonlinear Dynamics. Heidelberg: Springer Berlin.
[41] **Miao ZW and Chen Y** (2022) Physics-informed neural networks method in high-dimensional integrable systems. *Modern Physics Letters B* **36**(01), 2150531.
[42] **Mulase M** (1984) Complete integrability of the kadomtsev-petviashvili equation. *Advances in Mathematics* **54**(1), 57–66.
[43] **Nocedal J** (1980) Updating quasi-newton matrices with limited storage. *Mathematics of Computation* **35**(151), 773–782.
[44] **Noorizadegan A, Cavoretto R, Young DL and Chen CS** (2024) Stable weight updating: A key to reliable pde solutions using deep learning. *Engineering Analysis With Boundary Elements* **168**, 105933.
[45] **Novikov S, Manakov SV, Pitaevskii LP and Zakharov VE** (1984) *Theory of Solitons: The Inverse Scattering Method*. New York: Springer Science & Business Media.
[46] **Ortiz R, Marín A and Ortiz MA** (2025) Physics-informed neural networks and Fourier methods for the generalized Korteweg–de Vries equation. *Mathematics* **13**(9), 1521.
[47] **Raissi M, Perdikaris P and Karniadakis GE** (2019) 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.
[48] **Strutt JW**, 3rd Baron Rayleigh (1876) On waves. *Philosophical Magazine* **1**, 257–279.
[49] **Satsuma J** (1976) N-soliton solution of the two-dimensional korteweg-devries equation. *Journal of the Physical Society of Japan* **40**(1), 286–290.
[50] **Stokes GG** (1847) On the theory of oscillatory waves. *Transactions of the Cambridge Philosophical Society* **8**, 441–455.
[51] **Taha TR and Ablowitz MJ** (1984) Analytical and numerical aspects of certain nonlinear evolution equations. III. Numerical, Korteweg–de Vries equation. *Journal of Computational Physics* **55**(2), 231–253.
[52] **Toscano JD, Oommen V, Varghese AJ, Zou Z, Daryakenari NA, Wu C and Karniadakis GE** (2025) From pinns to pikans: Recent advances in physics-informed machine learning. *Machine Learning for Computational Science and Engineering* **1**(1), 1–43.
[53] **Wang S, Sankaran S and Perdikaris P** (2022) Respecting causality is all you need for training physics-informed neural networks. *arXiv preprint arXiv:2203.07404*.
[54] **Wang S, Teng Y and Perdikaris P** (2020) Understanding and mitigating gradient pathologies in physics-informed neural networks. *arXiv preprint arXiv:2001.04536*.
[55] **Wu C, Zhu M, Tan Q, Kartha Y and Lu L** (2023) A comprehensive study of non-adaptive and residual-based adaptive sampling for physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering* **403**, 115671.
[56] **Xu S, Dai Y, Yan C, Sun Z, Huang R, Guo D and Yang G** (2025) On the preprocessing of physics-informed neural networks: How to better utilize data in fluid mechanics. *Journal of Computational Physics* **528**, 113837.
[57] **Zabusky NJ and Kruskal MD** (1965) Interaction of "soliton"' in a collisionless plasma and the recurrence of initial states. *Physical Review Letters* **15**(6), 240.
[58] **Zhao C, Zhang F, Lou W, Wang X and Yang J** (2024) A comprehensive review of advances in physics-informed neural networks and their applications in complex fluid dynamics. *Physics of Fluids* **36**(10), 101301.
[59] **Zhou C, Chen J, Yang Z and Png CE** (2025) Dual-balancing for physics-informed neural networks. *arXiv preprint arXiv:2505.11117*.
[60] **Zubov K, McCarthy Z, Ma Y, Calisto F, Pagliarino V, Azeglio S, Bottero L, Luján E, Sulzer V, Bharambe A, Vinchhi N, Balakrishnan K, Upadhyay D and Rackauckas C** (2021) Neuralpde: Automating physics-informed neural networks (pinns) with error approximations.

https://doi.org/10.1017/jnw.2026.10050 Published online by Cambridge University Press

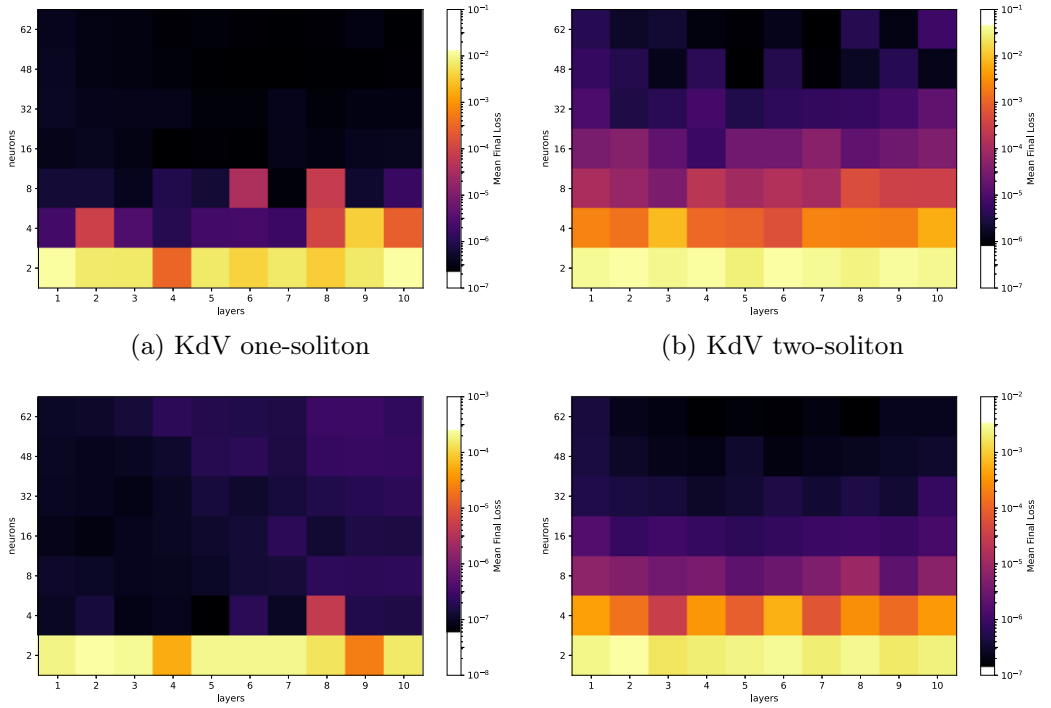
Appendix A. Architecture analysis

In this section, we provide the performance of PINNs trained to solve the KdV and KP equations for a variety of architectures (number of hidden layers and neurons per layer).(Hereafter, we refer to hidden layers and neurons per layer as 'layers' and 'neurons', for brevity. We also refer to each combination (layers, neurons) as an 'architecture'.) For each architecture case, we train and test $n = 10$ networks, and the mean result for a particular metric (final loss or max normalized absolute point-wise error, $||e||_\infty$) over the n replications is presented. All other hyperparameters are kept constant and are set as per Tables 2 and 3.

We first characterize the differences in training by examining the final loss achieved for each architecture. Then we test each architecture and observe the dependence of the maximum of the normalized absolute error on the architecture.

A.1. Final loss

Figure A1 shows the average value of the final loss achieved for each architecture that was trained, for one- and two-soliton solution cases of both KdV and KP.

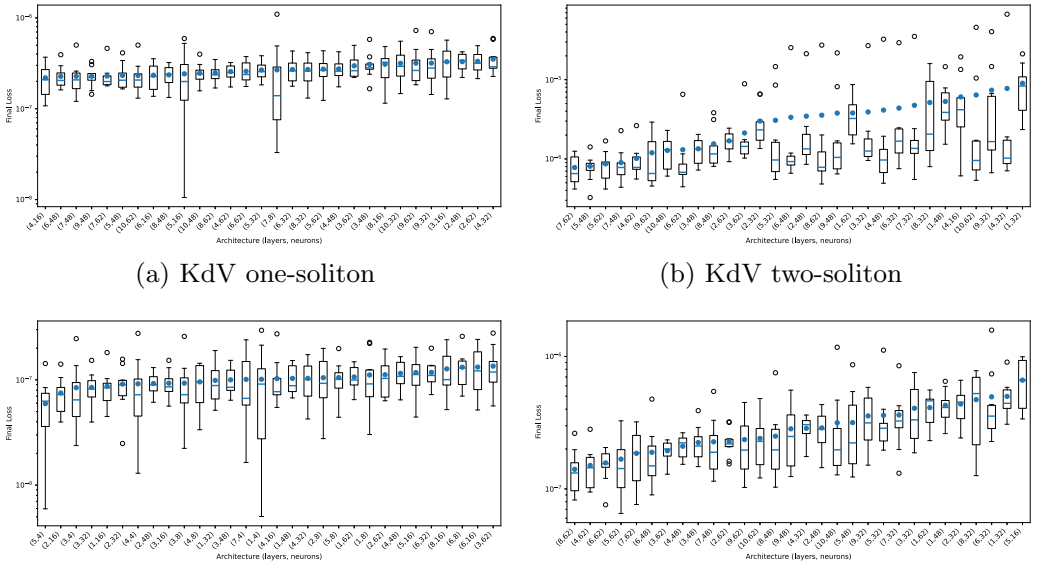


(a) KdV one-soliton (b) KdV two-soliton

(c) KP one-soliton (d) KP Y-Junction

Figure A1. *Average final loss for a variety of PINNs trained to solve the KdV and KP equations for various ICs. For each architecture, n = 10 replications were trained and tested.*

https://doi.org/10.1017/jnw.2026.10050 Published online by Cambridge University Press



(a) KdV one-soliton (b) KdV two-soliton

(c) KP one-soliton (d) KP y-junction

Figure A2. *Distribution of final loss for a variety of PINN architectures trained to solve the KdV and KP equations with various ICs. Boxes show the inter-quartile range (IQR, Q1–Q3), with the blue line and dots indicating the median and mean, respectively. Whiskers extend to (Q1 − 1.5 × IQR) and (Q3 + 1.5 × IQR), with hollow circles indicating outliers beyond this range. For each architecture, n = 10 replications were trained and tested.*

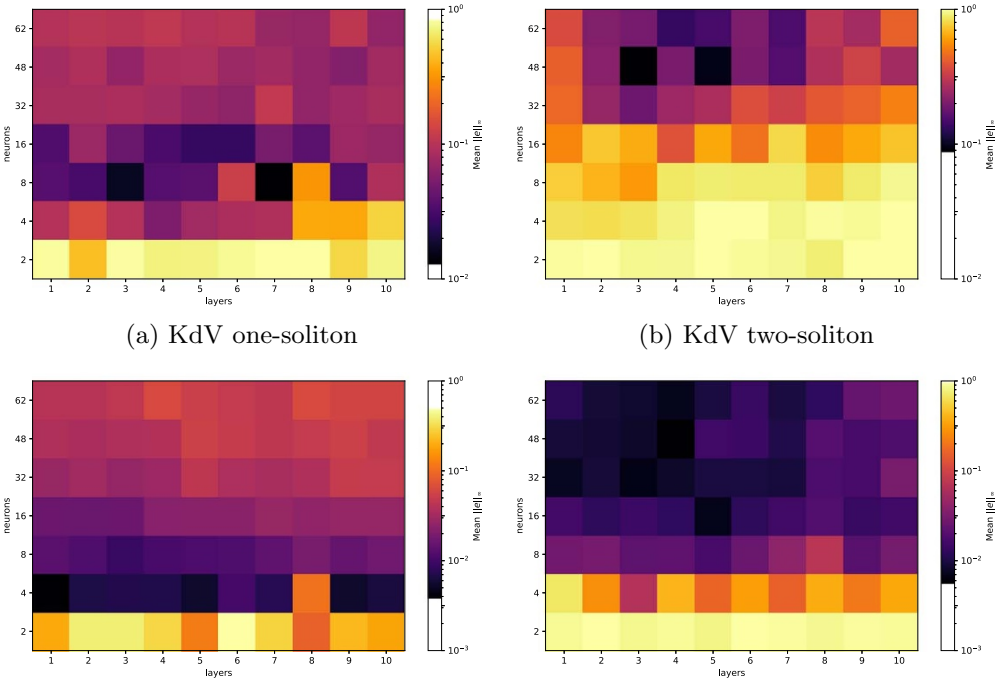
A.2. Max error

Figure A3 shows the average $||e||_\infty$ for each architecture considered, and the raw data are shown in Tables A1, A2, A3 and A4. Observe that while the architectures that perform well tend to be near one another, there is no consistent pattern between the cases that allows us to draw general conclusions about the types of architectures that may perform well for more solitons or different types of governing equations.

In Figure A4, we present box and whisker plots which show the distribution of the max normalized absolute error $||e||_\infty$ for the best 30 architectures, as determined and ordered by the lowest mean $||e||_\infty$. The blue dots indicate the mean $||e||_\infty$ and the blue lines indicate the median. The boxes show the inter-quartile range (IQR), the spread of the middle 50% of data between the 25th (Q1) and 75th (Q3) percentiles. The whiskers on either extrema extend to (Q1 − 1.5 × IQR) and (Q3 + 1.5 × IQR) with hollow circles indicating outliers beyond this range.

While the difference between the mean $||e||_\infty$ of the worst and best performing architecture of this set can be up to an order of magnitude, we also observe a large variance in these data, with whiskers sometimes spanning up to two orders of magnitude.

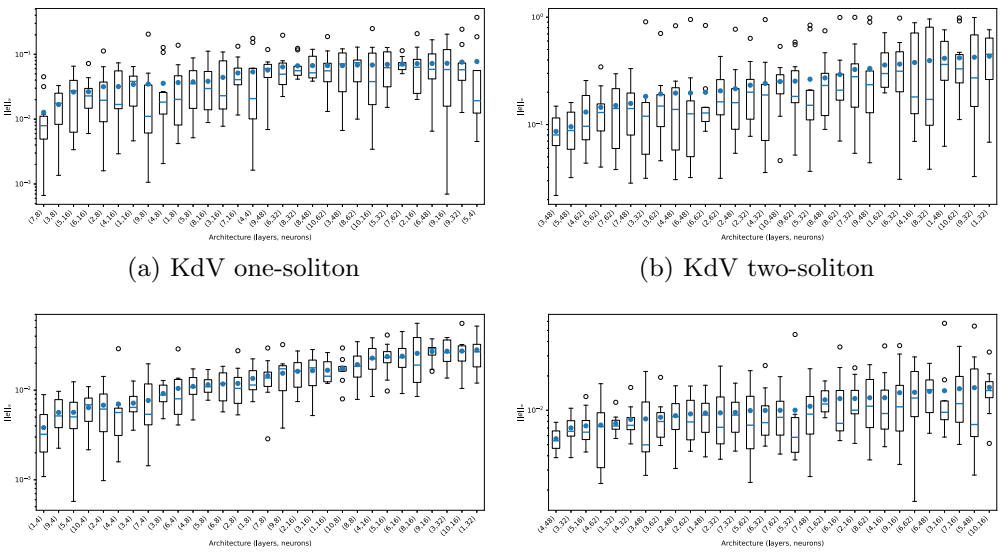
https://doi.org/10.1017/jnw.2026.10050 Published online by Cambridge University Press



(a) KdV one-soliton (b) KdV two-soliton

(c) KP one-soliton (d) KP Y-Junction

Figure A3. *Average max normalized absolute error $||e||_\infty$ for a variety of PINNs trained to solve the KdV and KP equations for various ICs. For each architecture, $n = 10$ replications were trained and tested.*



(a) KdV one-soliton (b) KdV two-soliton

(c) KP one-soliton (d) KP y-junction

Figure A4. *Distribution of max normalized absolute error $||e||_\infty$ for a variety of PINN architectures trained to solve the KdV and KP equations with various ICs. Boxes show the inter-quartile range (IQR, Q1–Q3), with the blue line and dots indicating the median and mean, respectively. Whiskers extend to $(Q1 - 1.5 \times IQR)$ and $(Q3 + 1.5 \times IQR)$, with hollow circles indicating outliers beyond this range. For each architecture, $n = 10$ replications were trained and tested.*

https://doi.org/10.1017/jnw.2026.10050 Published online by Cambridge University Press

https://doi.org/10.1017/jnw.2026.10050 Published online by Cambridge University Press

Table A1. *Average max normalized absolute error $||e||_\infty$ for a variety of PINNs trained to solve the KdV equation with a one-soliton IC*

	Layers									
Neurons	1	2	3	4	5	6	7	8	9	10
2	7.9 ± 4.3	4.4 ± 4.9	8.1 ± 4.1	6.8 ± 4.7	7.1 ± 4.7	7.6 ± 4.1	8.1 ± 4.3	8.2 ± 3.9	5.4 ± 4.9	7.2 ± 4.5
4	0.94 ± 1.4	1.4 ± 3.3	0.93 ± 1.4	0.53 ± 0.66	0.77 ± 1.2	0.87 ± 0.91	0.9 ± 0.82	3.7 ± 4.5	3.6 ± 4.8	5.2 ± 4.9
8	0.37 ± 0.42	0.32 ± 0.34	0.17 ± 0.11	0.36 ± 0.44	0.38 ± 0.27	1.2 ± 3.1	0.13 ± 0.14	3.1 ± 4.7	0.35 ± 0.62	0.9 ± 1.9
16	0.34 ± 0.21	0.72 ± 0.59	0.44 ± 0.38	0.32 ± 0.28	0.26 ± 0.21	0.26 ± 0.19	0.51 ± 0.37	0.38 ± 0.33	0.73 ± 0.7	0.68 ± 0.77
32	0.84 ± 0.42	0.83 ± 0.37	0.86 ± 0.82	0.8 ± 0.58	0.7 ± 0.43	0.64 ± 0.51	1.1 ± 0.61	0.66 ± 0.29	0.76 ± 0.69	0.83 ± 0.67
48	0.81 ± 0.54	0.88 ± 0.5	0.67 ± 0.43	0.86 ± 0.43	0.87 ± 0.44	0.72 ± 0.52	0.8 ± 0.57	0.67 ± 0.3	0.57 ± 0.31	0.78 ± 0.38
62	0.94 ± 0.47	1 ± 0.55	0.98 ± 0.53	1.1 ± 0.44	0.94 ± 0.52	0.99 ± 0.58	0.71 ± 0.2	0.68 ± 0.4	1 ± 0.68	0.67 ± 0.51

Notes: For each architecture, 10 replications were trained and tested. Results shown with multiplicative factor 10^{-1}.

https://doi.org/10.1017/jnw.2026.10050 Published online by Cambridge University Press

Table A2. *Average max normalized absolute error $||e||_\infty$ for a variety of PINNs trained to solve the KdV equation with a two-soliton IC*

	Layers									
Neurons	1	2	3	4	5	6	7	8	9	10
2	9.7 ± 0.81	9.9 ± 0.11	9.3 ± 1.7	9.5 ± 1.1	9.9 ± 0.1	9.6 ± 0.93	9.4 ± 1.3	8.9 ± 2.3	9.9 ± 0.12	9.9 ± 0.3
4	8.2 ± 1.3	8.1 ± 1.5	8.4 ± 2.9	9.1 ± 1.2	9.9 ± 0.15	9.9 ± 0.17	9.2 ± 1.7	9.7 ± 0.92	9.5 ± 0.95	9.9 ± 0.12
8	7.4 ± 2.3	6.6 ± 3.9	5.8 ± 4.3	8.5 ± 2.8	8.7 ± 2.4	8.7 ± 3	8.6 ± 3	7.5 ± 3.4	8.6 ± 2.8	9.4 ± 1.5
16	5.2 ± 3.2	7.2 ± 1.5	6.3 ± 3.5	3.8 ± 3.4	6.2 ± 3.9	4.6 ± 4.2	7.9 ± 3.1	5.4 ± 3.6	6.2 ± 3.8	7.1 ± 4.1
32	4.3 ± 2.4	2.3 ± 1.7	1.8 ± 2.6	2.4 ± 2.7	2.6 ± 2.9	3.7 ± 2.7	3.3 ± 2.8	3.9 ± 4	4.2 ± 3.6	5 ± 3.6
48	4.1 ± 2.3	2.2 ± 2.1	0.86 ± 0.4	2 ± 2.4	0.96 ± 0.44	2 ± 2.8	1.6 ± 0.99	2.7 ± 2	3.3 ± 3.3	2.5 ± 1.3
62	3.6 ± 1.7	2.1 ± 1.3	1.9 ± 1.9	1.3 ± 0.82	1.4 ± 0.88	2 ± 2.3	1.5 ± 0.99	2.9 ± 2.6	2.5 ± 1.9	4.2 ± 3

https://doi.org/10.1017/jnw.2026.10050 Published online by Cambridge University Press

Table A3. *Average max normalized absolute error $||e||_\infty$ for a variety of PINNs trained to solve the KP equation with a single line-soliton IC*

	Layers									
Neurons	1	2	3	4	5	6	7	8	9	10
2	2 ± 4	3.9 ± 5	3.9 ± 5	3 ± 4.8	1.2 ± 3.3	4.9 ± 5.1	2.9 ± 4.7	0.87 ± 2.5	2.3 ± 4.4	1.8 ± 3.8
4	0.038 ± 0.024	0.068 ± 0.044	0.072 ± 0.027	0.07 ± 0.079	0.057 ± 0.038	0.1 ± 0.074	0.077 ± 0.062	1.1 ± 3.1	0.056 ± 0.027	0.064 ± 0.029
8	0.14 ± 0.054	0.12 ± 0.067	0.092 ± 0.026	0.11 ± 0.039	0.11 ± 0.029	0.12 ± 0.046	0.15 ± 0.072	0.19 ± 0.082	0.15 ± 0.084	0.18 ± 0.056
16	0.17 ± 0.054	0.16 ± 0.069	0.17 ± 0.073	0.23 ± 0.091	0.24 ± 0.091	0.24 ± 0.11	0.28 ± 0.11	0.26 ± 0.17	0.27 ± 0.065	0.27 ± 0.12
32	0.28 ± 0.13	0.32 ± 0.13	0.27 ± 0.094	0.29 ± 0.11	0.44 ± 0.091	0.36 ± 0.13	0.34 ± 0.11	0.37 ± 0.13	0.49 ± 0.14	0.48 ± 0.22
48	0.37 ± 0.18	0.35 ± 0.11	0.37 ± 0.12	0.38 ± 0.056	0.55 ± 0.23	0.49 ± 0.15	0.45 ± 0.11	0.48 ± 0.15	0.55 ± 0.17	0.46 ± 0.12
62	0.41 ± 0.087	0.41 ± 0.15	0.45 ± 0.11	0.63 ± 0.19	0.52 ± 0.19	0.48 ± 0.12	0.45 ± 0.14	0.65 ± 0.16	0.58 ± 0.23	0.58 ± 0.18

Notes: For each architecture, 10 replications were trained and tested. Results shown with multiplicative factor 10^{-1}.

https://doi.org/10.1017/jnw.2026.10050 Published online by Cambridge University Press

Table A4. *Average max normalized absolute error $||e||_\infty$ for a variety of PINNs trained to solve the KP equation with a two-line-soliton y-junction IC*

	Layers									
Neurons	1	2	3	4	5	6	7	8	9	10
2	8.8 ± 2.6	9.5 ± 0.99	8.4 ± 2.8	8.9 ± 2.6	8.1 ± 2.7	9.6 ± 0.37	9.2 ± 1.6	8.7 ± 2.1	9.1 ± 1.9	8.9 ± 2
4	6.9 ± 3.5	2.7 ± 3.3	0.65 ± 0.62	4.1 ± 4.6	1.6 ± 2.3	3.2 ± 3.7	1.6 ± 2.3	3.7 ± 3.5	2.1 ± 2.5	3.5 ± 3.9
8	0.29 ± 0.23	0.31 ± 0.31	0.22 ± 0.13	0.23 ± 0.12	0.17 ± 0.17	0.26 ± 0.21	0.41 ± 0.94	0.73 ± 1.7	0.21 ± 0.21	0.3 ± 0.23
16	0.16 ± 0.091	0.13 ± 0.058	0.15 ± 0.16	0.13 ± 0.098	0.073 ± 0.029	0.13 ± 0.1	0.15 ± 0.11	0.19 ± 0.21	0.14 ± 0.11	0.16 ± 0.073
32	0.077 ± 0.017	0.095 ± 0.066	0.07 ± 0.022	0.083 ± 0.03	0.1 ± 0.13	0.1 ± 0.061	0.096 ± 0.042	0.18 ± 0.15	0.17 ± 0.11	0.31 ± 0.47
48	0.095 ± 0.04	0.09 ± 0.042	0.084 ± 0.067	0.056 ± 0.015	0.16 ± 0.16	0.15 ± 0.062	0.11 ± 0.068	0.2 ± 0.27	0.17 ± 0.096	0.19 ± 0.25
62	0.12 ± 0.034	0.093 ± 0.045	0.087 ± 0.042	0.074 ± 0.048	0.099 ± 0.07	0.14 ± 0.091	0.1 ± 0.052	0.13 ± 0.073	0.25 ± 0.094	0.27 ± 0.18

Notes: For each architecture, 10 replications were trained and tested. Results shown with multiplicative factor 10^{-1}.

Appendix B. Component-wise loss histories

In this appendix, we present the training loss histories of the PINNs used to compute the solutions shown in the main text. In Figure B1, we show the component-wise loss histories of the training process for PINNs that solve the KdV equation for the one- and two-soliton solutions shown in Figures 2 and 6, respectively.

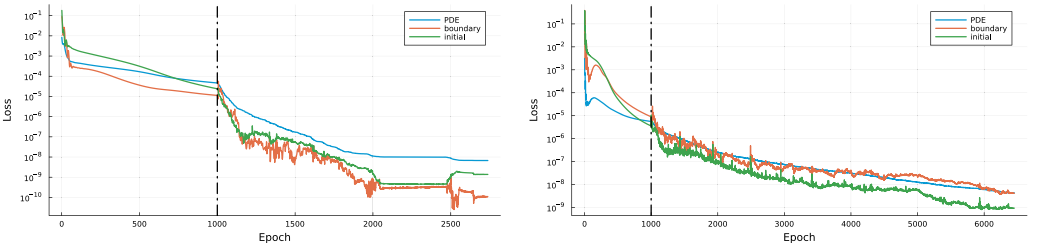


Figure B1. *Component-wise loss during training of PINNs that solve the KdV equation for (left) single-soliton and (right) two-soliton ICs.*

In Figure B2, we show the loss histories of the PINN that solves the KP equation for the single line-soliton solution shown in Figure 10.

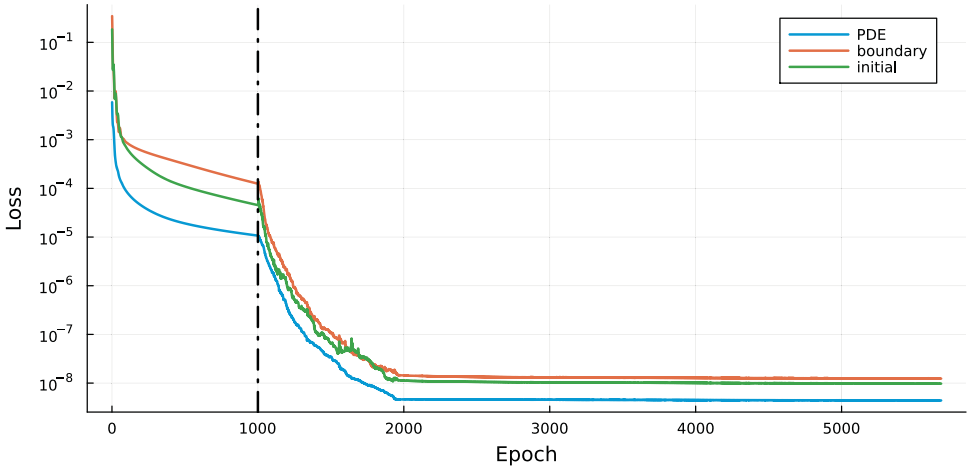


Figure B2. *Component-wise loss during training of a PINN that solves the KPII equation for a single line-soliton IC.*

https://doi.org/10.1017/jnw.2026.10050 Published online by Cambridge University Press

In Figure B3, we show the loss histories during training of PINNs that solve the KP equation for two-line-soliton solutions, corresponding to the x-junction and y-junction solutions shown in Figures 13 and 16, respectively. Observe that in the case of PINNs solving the KP equation, L_{BC} tends to be the largest throughout training, which we postulate is due to the nontrivial BCs, in contrast with the KdV case.

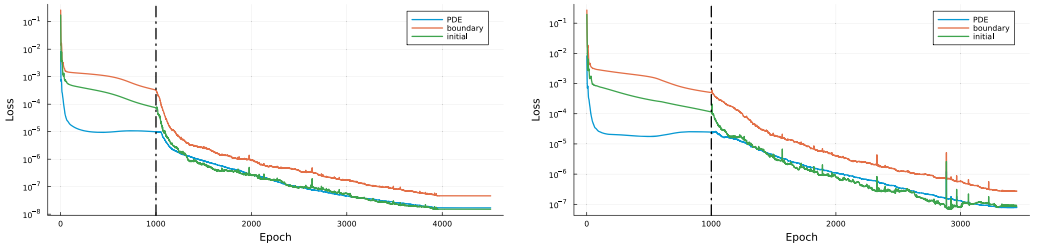


Figure B3. *Component-wise loss during training of PINNs that solve the KPII equation for two-line-soliton solutions, for the (left) x-junction and (right) y-junction ICs.*

Appendix C. Adaptive sampling

For the three-soliton KdV and KP '2-in-3-out' junction cases of Section 4 (§ 4.1), the fixed architecture and static, uniformly sampled collocation set used for the one- and two-soliton cases no longer reliably converge. This appendix introduces the residual-based adaptive sampling scheme used to address this, and quantifies its effect on the KdV two-soliton case, where a well-characterized non-adaptive baseline is already available (Figures 6–9) and can be compared directly against an otherwise identical run using adaptive sampling.

Algorithm 2: Adaptive Training Procedure

Stage A: Initial Training

1. Generate the initial training data $\mathcal{X}_{\text{train}}$, consisting of initial condition points, boundary condition points, and uniformly sampled PDE collocation points.
2. Train using Adam for n_A epochs.

Stage B: Residual-Based Adaptive Sampling

for $r = 1, 2, \ldots, R$ **do**

1. Sample a pool $\mathcal{P} = \{(x_i, t_i)\}_{i=1}^{M}$ of candidate PDE points from the interior.
2. Compute residuals $e_i = |f(x_i, t_i; \theta)|$ for all $(x_i, t_i) \in \mathcal{P}$.
3. Select the K points with largest e_i.
4. Append these K points to the PDE point set.
5. Train using Adam for n_B epochs.

Stage C: Fine-tuning

1. Train using L-BFGS until convergence.

Algorithm 2 outlines the residual-based adaptive sampling scheme used for the three-soliton KdV and KP junction cases of Section 4. Training proceeds in three stages. In Stage A, the network is trained exactly as in the one- and two-soliton cases (Algorithm 1), using Adam on a fixed, uniformly sampled set of collocation points for n_A epochs. In Stage B, rather than continuing to draw collocation points uniformly at random, a pool $\mathcal{P}$ of M candidate points is sampled from the interior of the domain at each of R rounds; the PDE residual is evaluated at every candidate point, and only the K points with the largest residual are retained and appended to the training set, after which the network is trained for a further n_B epochs on the enlarged collocation set. Because the PDE residual is largest precisely where the current network approximation is least accurate, this concentrates additional training signal

https://doi.org/10.1017/jnw.2026.10050 Published online by Cambridge University Press

exactly where uniform sampling under-resolves the solution, rather than spending the same additional points uniformly across regions the network has already learned well. Crucially, these residual points are drawn from the *unlabelled* interior of the domain, unlike the IC and BC points, which already carry direct supervision via $\mathcal{L}_{\text{IC}}$ and $\mathcal{L}_{\text{BC}}$. This scheme specifically targets the error that accumulates away from the supervised data, discussed further below. In Stage C, training concludes with L-BFGS, as in the non-adaptive cases, now applied to the enlarged, residual-informed collocation set.

To isolate the effect of adaptive sampling from other changes, we compare the two-soliton KdV case trained with the scheme above against the non-adaptive baseline of Section 4, using the same network architecture, training data counts, and loss weights (Table 3), with the only difference being the residual-based resampling stage (Algorithm 2, Stage B) inserted between the Adam and L-BFGS training stages.

Taken together, Figures C2–C5 show that residual-based adaptive sampling measurably reduces both the mean and worst-case point-wise error for the KdV two-soliton case relative to static sampling. In particular, since adaptive resampling only ever adds points to the interior, unlabelled portion of the collocation set and never to the IC or BC data, which are already directly supervised, the reduction in error is concentrated specifically in the middle of the domain, away from the initial and boundary data, rather than uniformly across it (Figure C3). This directly motivated using the same scheme for the three-soliton KdV and KP junction cases in Section 4, where static sampling alone was insufficient to reach convergence comparable to the one- and two-soliton cases. It does not eliminate the qualitative pattern of error growing away from the supervised initial and boundary data, only its magnitude and variance, and we do not view it as a complete solution to that phenomenon.

Figure 20 shows the space-time portrait and normalized absolute error (4.1) for the KdV three-soliton case, trained using the same adaptive scheme. Rather than growing uniformly with $|t|$, the error is concentrated in two distinct features: a band tracking the trajectory of the largest-amplitude soliton, which intensifies at later times, and a more complex, elevated-error region surrounding the interaction zone where the three characteristic lines cross ($t \approx -10$ to 10). This is consistent with the lag visible in Figure 19: even with adaptive sampling, error accumulates preferentially along the parts of the solution the network must extrapolate furthest from the supervised initial and boundary data, rather than simply as a function of elapsed time, which is the same pattern identified for the two-soliton case above, now at greater overall magnitude (Table 6).

https://doi.org/10.1017/jnw.2026.10050 Published online by Cambridge University Press

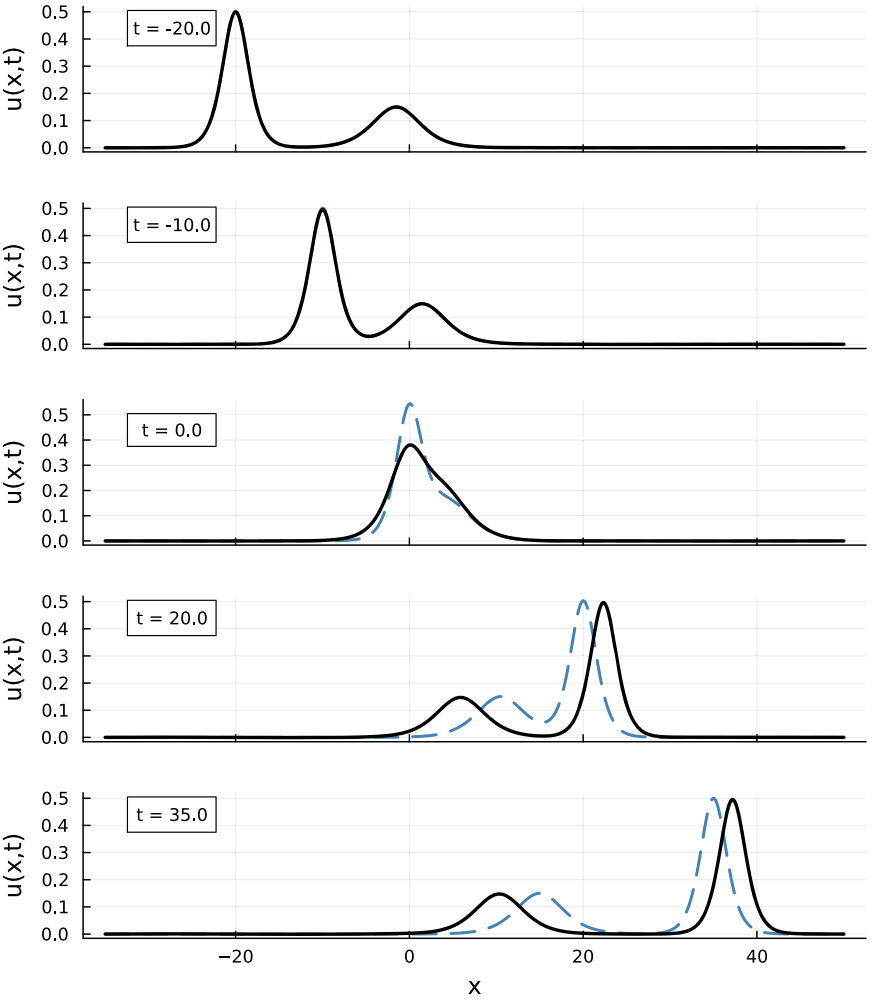


Figure C1. *Solutions to the KdV equation for the two-soliton IC computed by a PINN trained with residual-based adaptive sampling, shown at various times (as labelled), with topmost and bottom-most panels corresponding to initial and final training time. The solid line represents the nonlinear solution approximated by the PINN, and the dashed blue line is the linear combination of single soliton solutions. Compared to Figure 6, computed with static (non-adaptive) sampling.*

https://doi.org/10.1017/jnw.2026.10050 Published online by Cambridge University Press

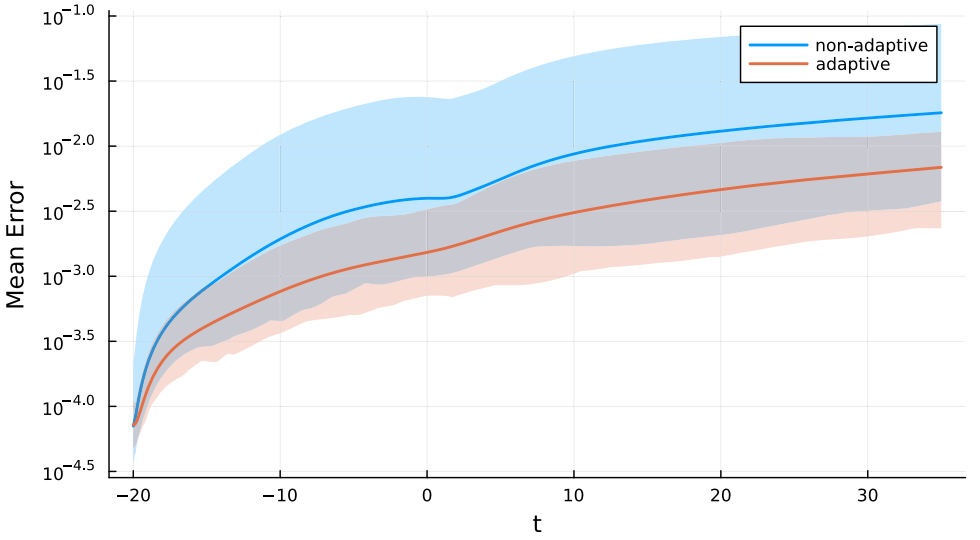


Figure C2. *Ensemble-averaged point-wise normalized error (4.1), averaged over x, as a function of t, comparing adaptive (residual-based) against non-adaptive (static) collocation sampling for the KdV two-soliton case. The adaptive scheme reduces the mean error across the full time range shown and substantially narrows the inter-quantile spread relative to non-adaptive sampling, indicating more consistent convergence across replications rather than only a lower typical error.*

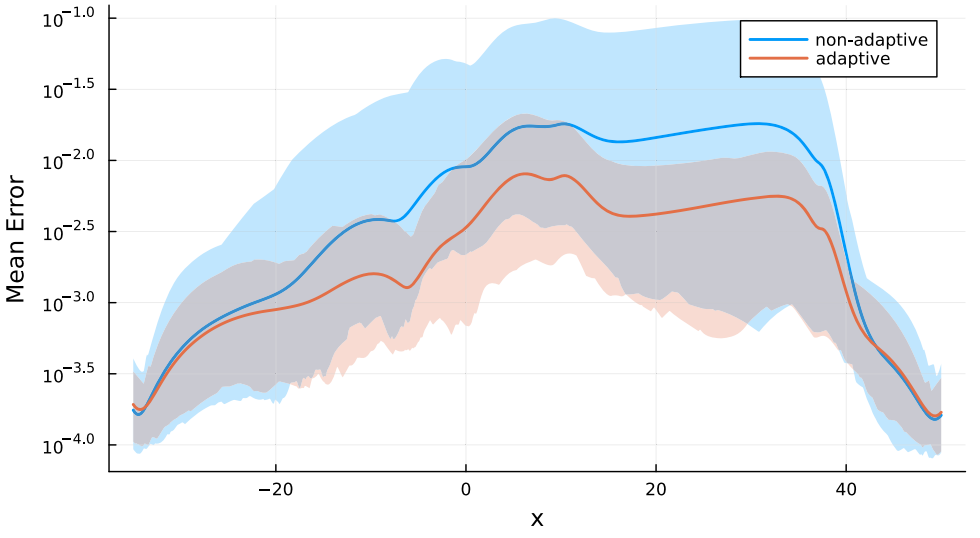


Figure C3. *Ensemble-averaged point-wise normalized error (4.1), averaged over t, as a function of x, comparing adaptive against non-adaptive collocation sampling for the KdV two-soliton case (logarithmic error axis). Both strategies show the same qualitative pattern – error smallest near the domain edges, where initial and boundary data directly supervise the loss, and largest in the interior – but adaptive sampling visibly compresses this interior error peak, directly reducing the mid-domain error concentrated in the unlabelled collocation points rather than at the IC/BC.*

https://doi.org/10.1017/jnw.2026.10050 Published online by Cambridge University Press

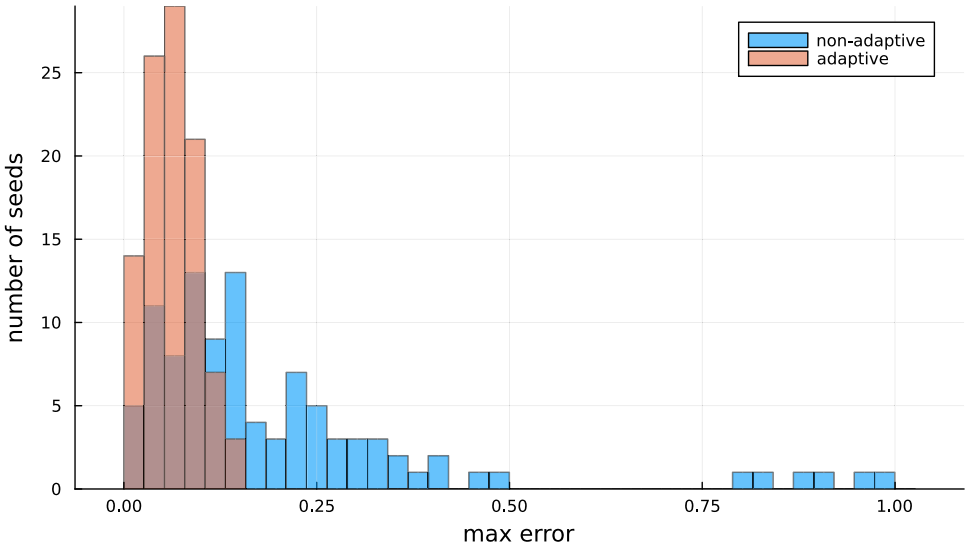


Figure C4. *Distribution of the maximum point-wise normalized error (4.1) $\|e\|_\infty$ across random seeds, for adaptive versus non-adaptive ('regular seed') sampling, for the KdV two-soliton case. Non-adaptive seeds span a wide range of maximum error, while adaptive seeds cluster towards substantially lower maximum error, indicating improved worst-case reliability across random initializations.*

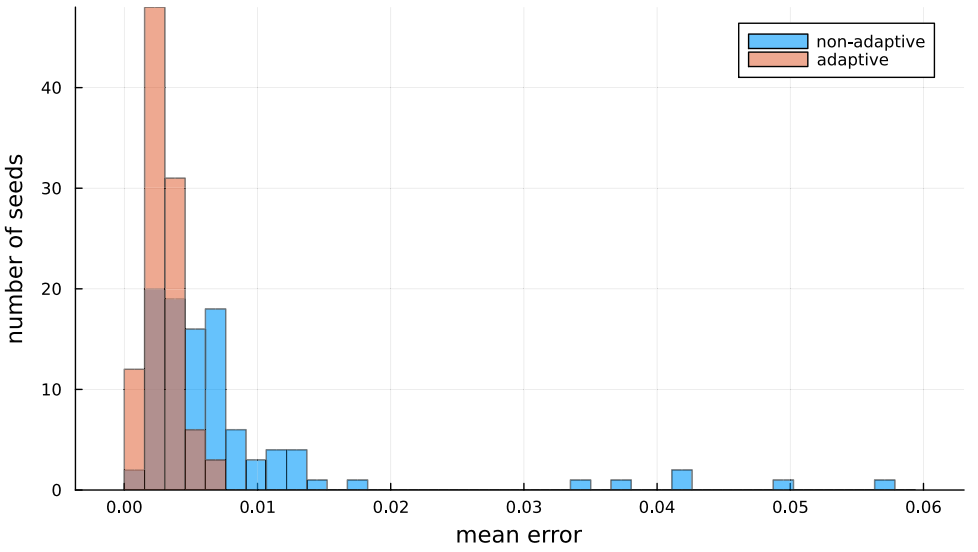


Figure C5. *Distribution of the mean point-wise normalized error (4.1) across random seeds, for adaptive versus non-adaptive ('regular seed') sampling, for the KdV two-soliton case. Adaptive sampling shifts this distribution sharply towards lower mean error and away from the broader, right-shifted distribution seen for non-adaptive sampling, indicating a reduction in both typical error and its variability across seeds.*

https://doi.org/10.1017/jnw.2026.10050 Published online by Cambridge University Press

Appendix D. Activation functions

This appendix compares eight activation functions – tanh, sine, softplus, mish, silu, gelu, elu and relu – for the KdV one-soliton case, by repeating the architecture sweep of Appendix A independently for each and comparing the resulting training time, final loss and maximum error.

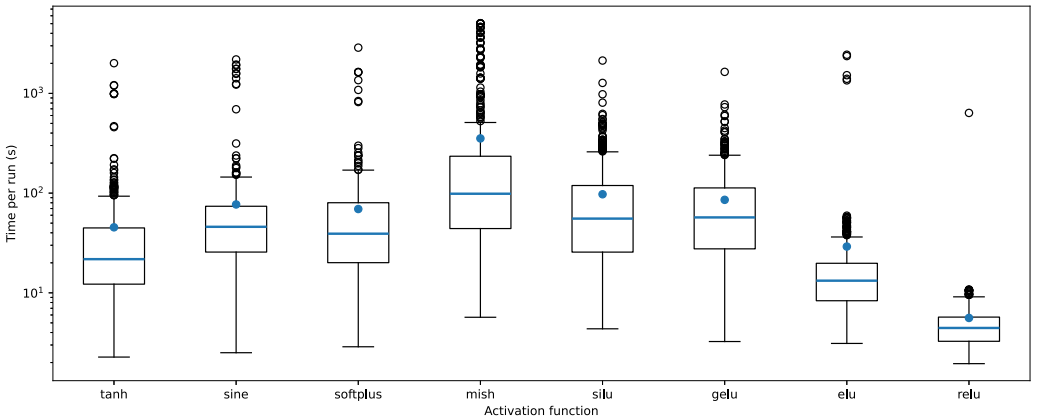


Figure D1. *Distribution of training times for a variety of activation functions.*

Table D1. *Summary of training times for the KdV one-soliton activation function study*

	Runs	Total (h)	Mean (s)	Median (s)	Min (s)	Max (s)
mish	700	68.74	353.5	98.6	5.7	5058.5
silu	700	18.94	97.4	55.5	4.4	2134.3
gelu	700	16.65	85.6	57.1	3.2	1643.4
sine	700	14.96	77.0	45.9	2.5	2189.3
softplus	700	13.47	69.3	39.2	2.9	2874.7
tanh	704	8.88	45.4	21.8	2.3	2008.3
elu	700	5.66	29.1	13.2	3.1	2441.2
relu	700	1.09	5.6	4.4	1.9	636.9

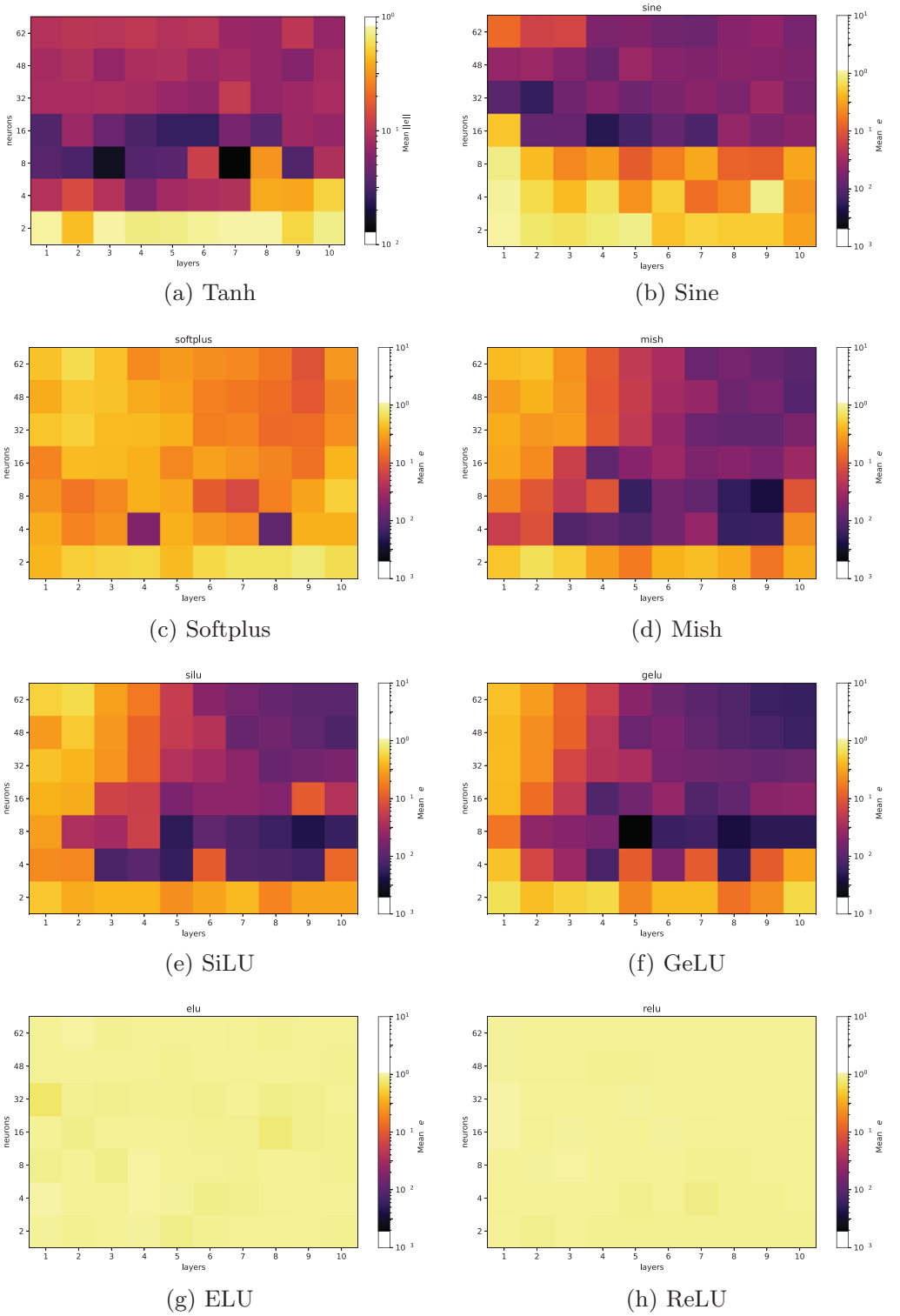
Notes: Each activation function was tested across the full architecture sweep described in Appendix A, with $n = 10$ independent training runs for each architecture. The table reports the total computational time, together with the mean, median, minimum and maximum training time per run.

D.1. Max error

D.2. Final loss

Table D1 and Figure D1 show that tanh has one of the lowest mean training times (45.4 s) among the eight functions tested. It is faster than sine, softplus, mish, silu and gelu, with only relu and elu training faster still. Comparing tanh's results in Appendix A (Figures A1(a) and A3(a)) against Figures D3 and D2, tanh reaches final loss and maximum error broadly comparable to the best-performing alternatives, while relu and elu, despite being fastest to train, show substantially higher error and loss across the architecture grid. This combination of competitive accuracy and comparatively low training cost,

https://doi.org/10.1017/jnw.2026.10050 Published online by Cambridge University Press



(a) Tanh (b) Sine

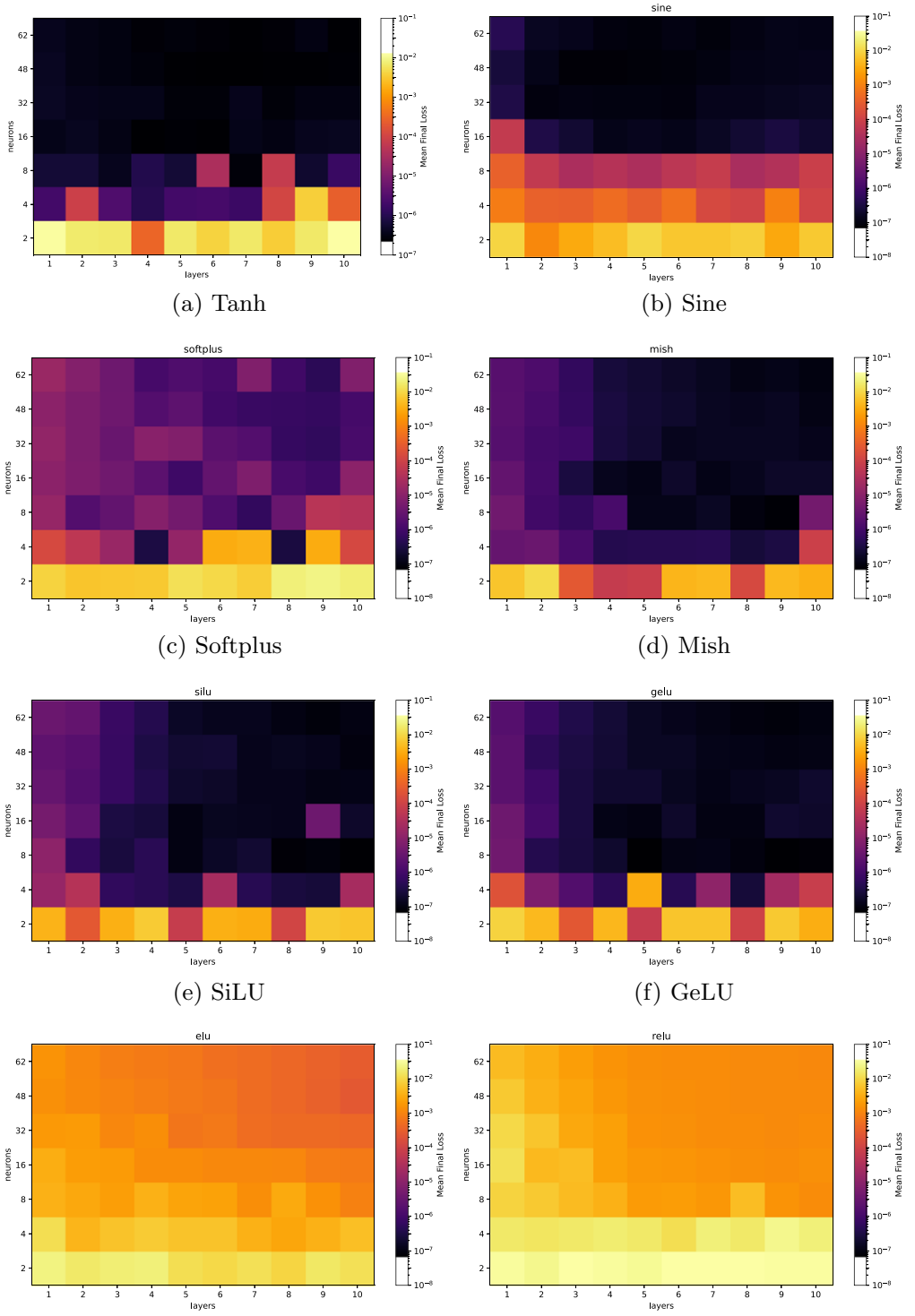
(c) Softplus (d) Mish

(e) SiLU (f) GeLU

(g) ELU (h) ReLU

Figure D2. *Average max normalized absolute error $\|e\|_\infty$ for KdV one-soliton PINNs across a grid of architectures, trained with each of eight activation functions. For each architecture, $n = 10$ replications were trained and tested.*

https://doi.org/10.1017/jnw.2026.10050 Published online by Cambridge University Press



(a) Tanh (b) Sine

(c) Softplus (d) Mish

(e) SiLU (f) GeLU

(g) ELU (h) ReLU

Figure D3. *Average final training loss for KdV one-soliton PINNs across a grid of architectures, trained with each of eight activation functions. For each architecture, $n = 10$ replications were trained and tested.*

https://doi.org/10.1017/jnw.2026.10050 Published online by Cambridge University Press

rather than being the single most accurate option, is why tanh, rather than a marginally more accurate but substantially slower alternative such as mish, is used for the results presented in the main body of this paper.

Cite this article: Raval K, Aiaseh N, Hosseini Dashtbayaz N, Trichtchenko O and Wang B (2026) Soliton solutions to KdV and KP equations using physics-informed neural networks. *Journal of Nonlinear Waves*, 1–47. https://doi.org/10.1017/jnw.2026.10050

https://doi.org/10.1017/jnw.2026.10050 Published online by Cambridge University Press