Mini-Project Presentation

Neural Network Approach to PDEs

Physics Informed Neural Networks (PINN)

Utkarsh Raj - 2023UEE0158
Somil Pandita - 2023UEE1053
Sujal Kapoor - 2023UEE0154

Introduction to PINNs

- History:
  - Introduced by Raissi, Perdikaris, & Karniadakis (2017–2019).
  - Bridges AI and physics by embedding PDEs into neural network loss functions.
- Key Idea:
  - Leverage automatic differentiation to enforce physical laws (PDEs) during training.
  - Combines data-driven learning with domain knowledge.

PINNs Architecture

Workflow:

1. Construct $u_t + \mathcal{N}[u;\lambda] = 0, x \in \Omega, t \in [0,T]$
2. Define data points (boundary/collocation).
3. Loss = Data mismatch + PDE residual.
4. Train via optimization (e.g., L-BFGS).

Key Innovation:

- Automatic differentiation computes PDE residuals (e.g., u_t,u_{xx}).

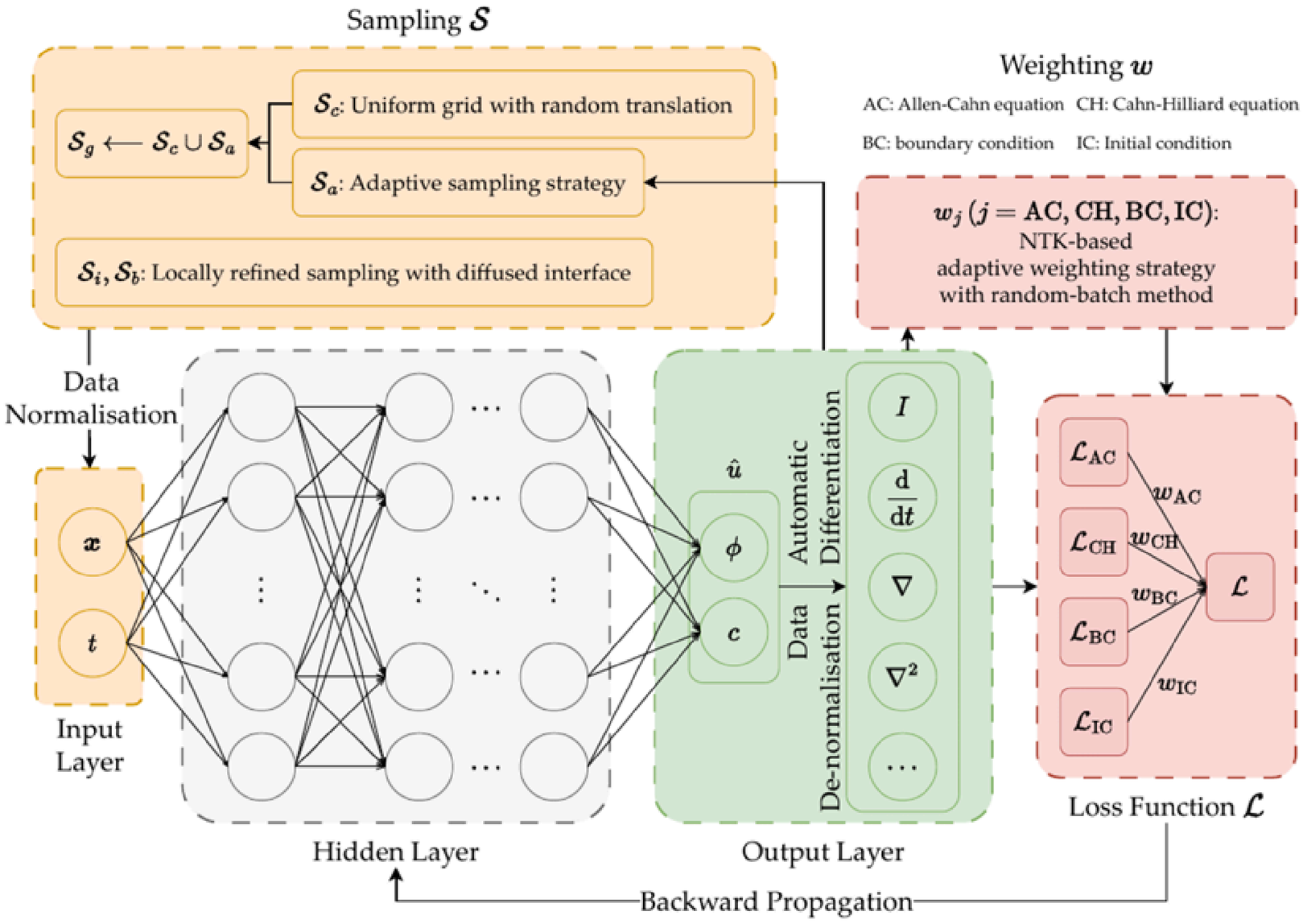
Visual: Simplified PINN architecture diagram (input: t,x; output: u; loss components).

01. Physics-informed neural networks (PINNs) leverage recent developments in automatic differentiation to compute derivatives of neural networks with respect to their input co-ordinates and model parameters.

02. These networks are constrained to respect symmetries, invariances, or conservation principles originating from the physical laws governing the observed data, as modeled by general time-dependent and nonlinear partial differential equations (PDEs).

03. Physics-informed neural networks integrate information from both measurements and PDEs by embedding the PDEs into the loss function using automatic differentiation.

Architecture



Continuous Time Models

General Form:

$$f := u_t + \mathcal{N}[u]$$

Loss Function:

$$MSE = MSE_u + MSE_f$$

Case Studies:

1. Burgers' Equation
2. KdV Equation
3. Cahn-Hilliard Equation

$$MSE_f = \frac{1}{N_f} \sum_{i=1}^{N_f} |f(t_f^i, x_f^i)|^2.$$

$$MSE_u = \frac{1}{N_u} \sum_{i=1}^{N_u} |u(t_u^i, x_u^i) - u^i|^2,$$

Burgers' Equation

$$u_t + uu_x - (0.01/\pi)u_{xx} = 0, x \in [-1, 1], t \in [0, 1]$$

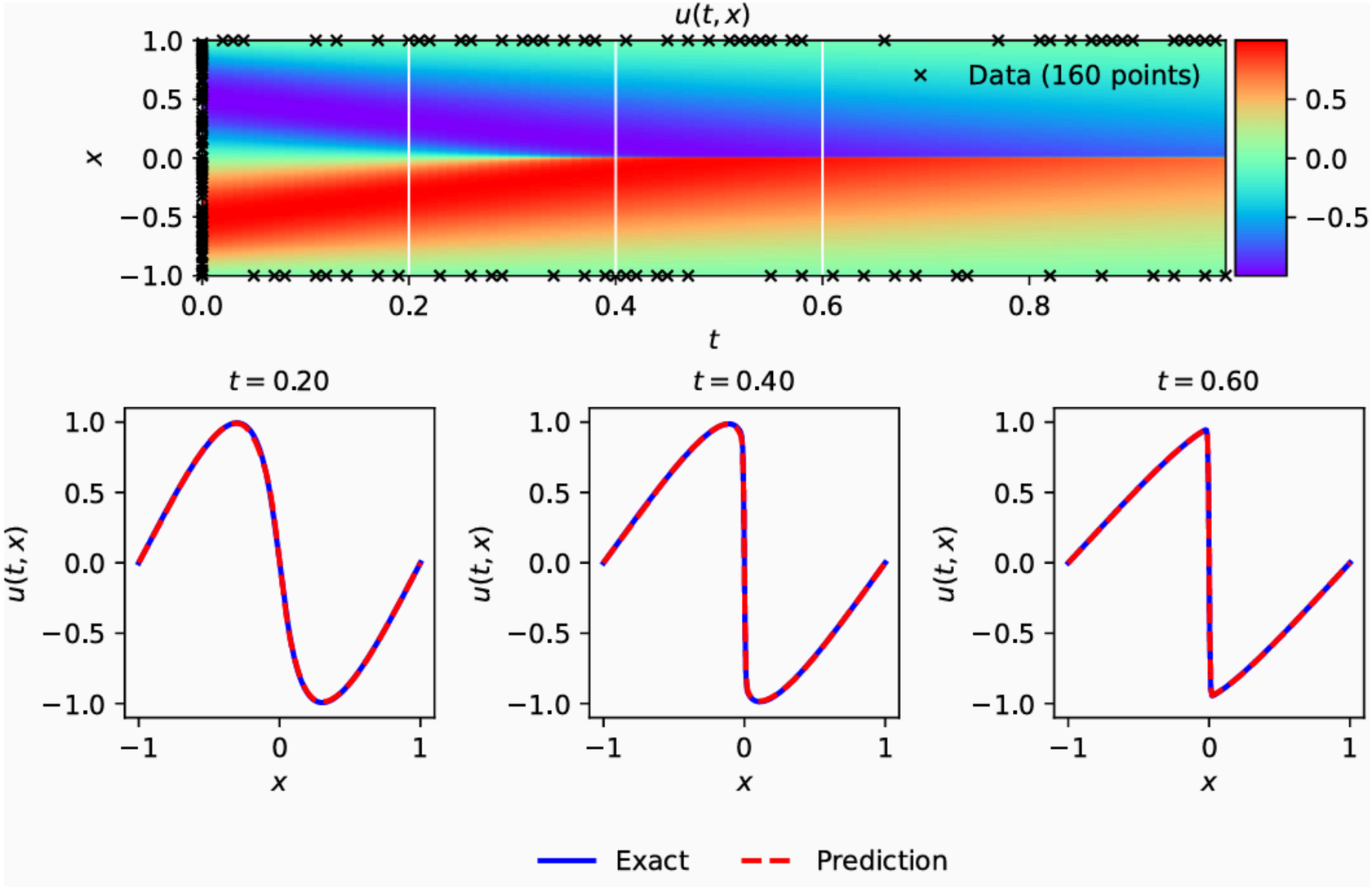
$$\text{initial condition } u(0, x) = -\sin(\pi x) \quad \text{boundary conditions } u(t, -1) = u(t, 1) = 0.$$

- NN: 9 layers, 3021 parameters.

Results:

- Relative L_2-error: 1.2×10^{-3}.
- Trend: Increasing N_u (boundary points) and N_f (collocation points) improves accuracy.

Numerical Results



Error Analysis

	2000	4000	6000	8000	10000
20	0.535667241	0.049695846	0.014588796	0.013886805	0.010786311
40	0.249364823	0.030792063	0.0091363	0.006794268	0.003439679
60	0.185548812	0.022392275	0.008312913	0.00668326	0.005492738
80	0.174256042	0.019572591	0.00463555	0.003494029	0.002582258
100	0.151201725	0.004660192	0.004047269	0.002652773	0.001231156

Table 1: N_f vs N_u graph

	2000–4000	4000–6000	6000–8000	8000–10000
20	5.2225	5.8639	4.2605	0.2210
40	7.3256	5.1587	4.2234	1.3273
60	8.3370	5.2153	3.4444	0.9779
80	8.8426	5.3923	5.0068	1.2669
100	9.3693	8.5816	0.4902	1.8931

Table 2: Rate table for N_f vs N_u

KdV Equation

$$u_t + \lambda_1 u u_x + \lambda_2 u_{xxx} = 0 \qquad x \in [-1, 1], t \in [0, 1]$$

$$\text{initial condition } u(0, x) = \cos(\pi x) \qquad \text{periodic boundary conditions.}$$

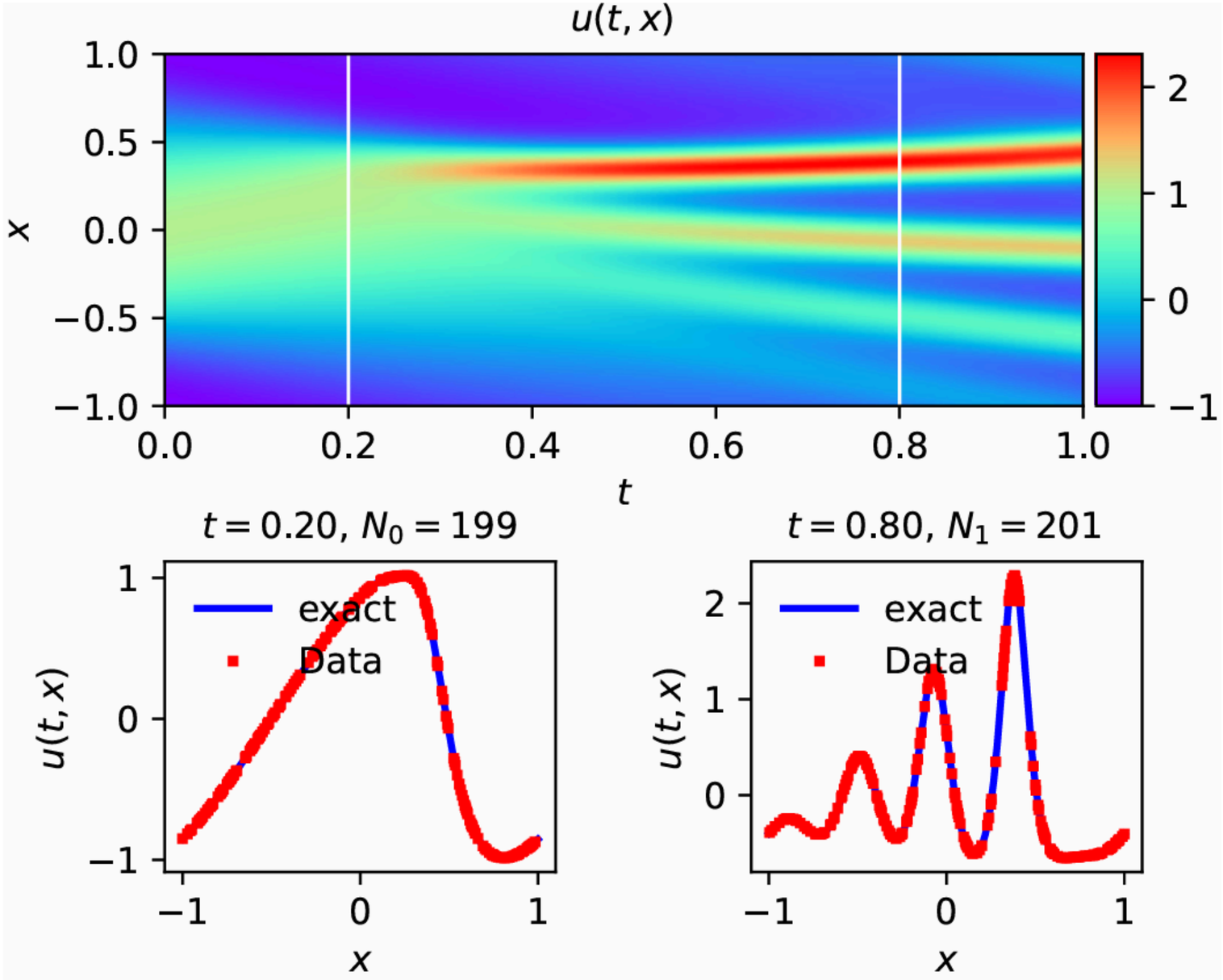
Key Feature:

- Physics-informed loss learns λ_1,λ_2.

Result:

- Example table snippet: N_f=10,000 reduces loss by ~80% vs. N_f=2,000.

Numerical Results



Error Analysis

	2000	4000	6000	8000	10000
20	0.0415	0.0384	0.0323	0.0315	0.0260
40	0.0381	0.0342	0.0241	0.0227	0.0202
60	0.0310	0.0270	0.0232	0.0212	0.0206
80	0.0297	0.0240	0.0230	0.0198	0.0155
100	0.0197	0.0176	0.0150	0.01189	0.01181

Table 3: N_f vs N_u graph

N_u	2000–4000	4000–6000	6000–8000	8000–10000
20	0.1120	0.4266	0.0872	0.8599
40	0.1558	0.8632	0.2080	0.5229
60	0.1993	0.3741	0.3134	0.1287
80	0.3074	0.1050	0.5208	1.0972
100	0.1626	0.3942	0.8077	0.0303

Table 4: Rate table for N_f across N_u

Cahn–Hilliard Equation

$$\frac{\partial u}{\partial t} = \nabla \cdot \left(M_c \nabla \frac{\delta G}{\delta u} \right) = \nabla \cdot (M_c \nabla \mu)$$

$$\mu = \frac{\delta G}{\delta u} = RT \left[\ln u - \ln(1-u) \right] + L(1-2u) - a_c \nabla^2 u$$

$$M_c = \left[\frac{D_A}{RT} u + \frac{D_B}{RT} (1-u) \right] u(1-u) = \frac{D_A}{RT} \left[u + \frac{D_B}{D_A} (1-u) \right] u(1-u)$$

$$x \in [-1, 1], t \in [0, 1]$$

Success cases

Application: Phase separation in materials.

Result:

- Loss decreases by ~50% when N_u increases from 20 to 100.

Error Analysis

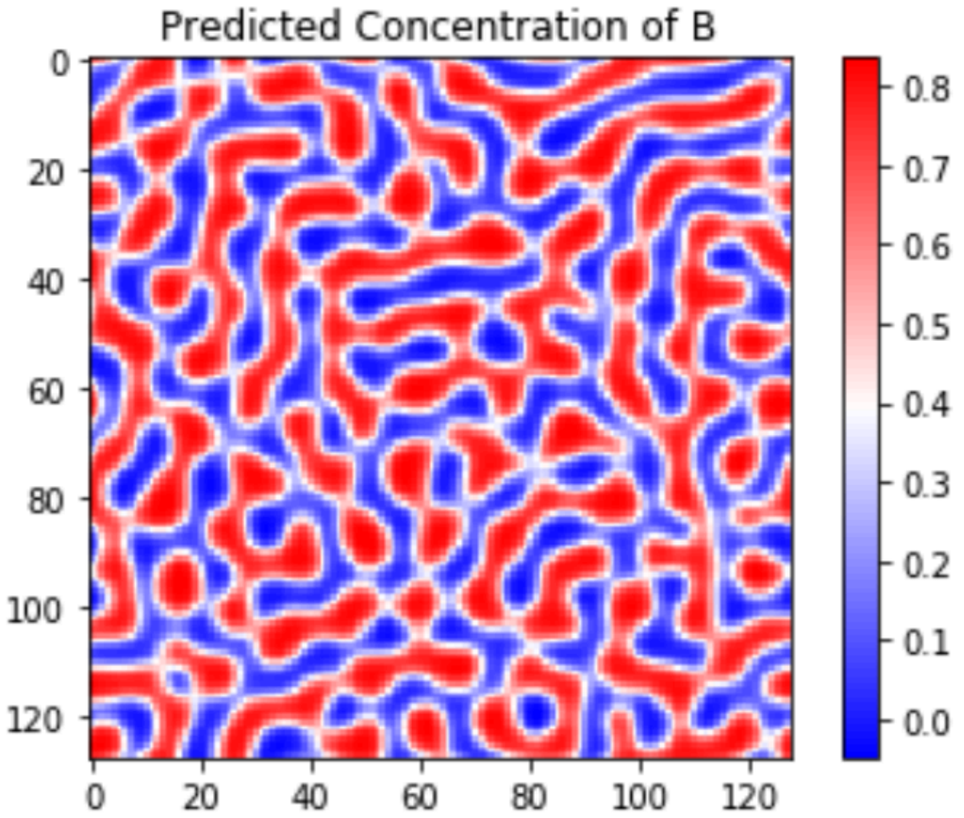
	2000	4000	6000	8000	10000
20	0.13893	0.13718	0.01532	0.01444	0.01433
40	0.13580	0.09106	0.01387	0.01375	0.01306
60	0.11929	0.08882	0.01351	0.01297	0.01276
80	0.08474	0.07869	0.01333	0.01278	0.01227
100	0.07005	0.06778	0.01286	0.01273	0.01214

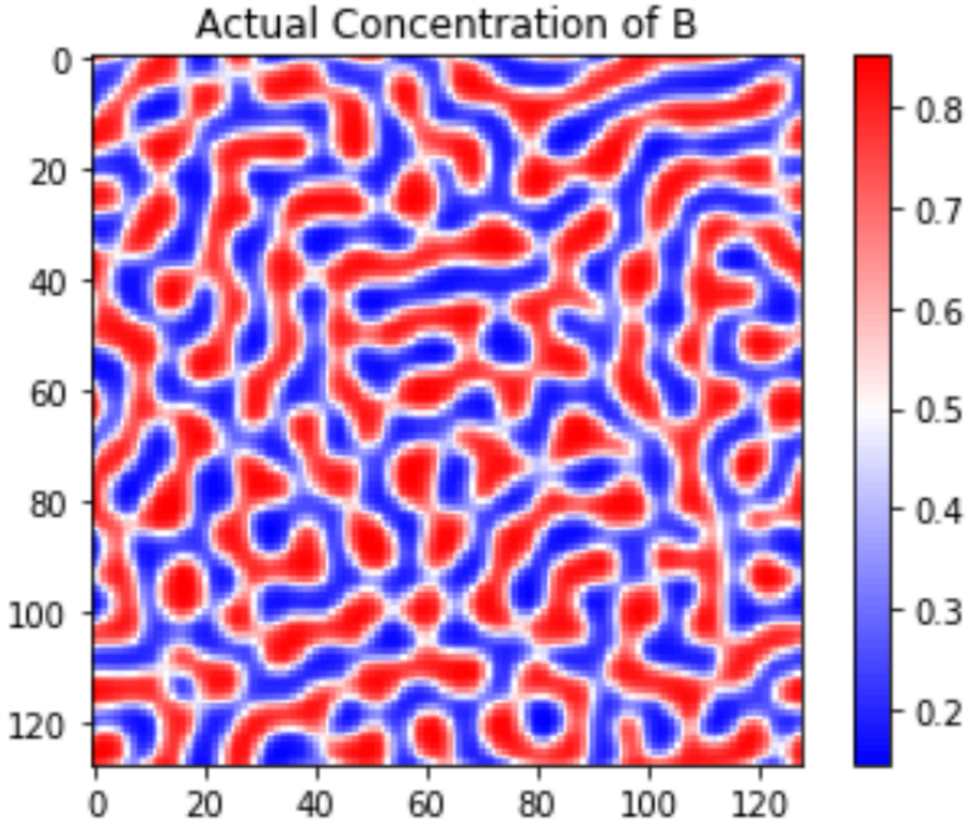
Table 5: N_u vs N_f Table

	2000–4000	4000–6000	6000–8000	8000–10000
20	0.01837	5.40603	0.20720	0.03252
40	0.57668	4.64011	0.03065	0.23107
60	0.42562	4.64447	0.14070	0.07284
80	0.10693	4.37789	0.14887	0.18278
100	0.04740	4.09853	0.03770	0.20941

Table 6: Computed values across intervals of 2000 units.

Numerical Results





Discrete Time Models

$$u^{n+c_i} = u^n - \Delta t \sum_{j=1}^{3} a_{ij} \mathcal{N}[u^{n+c_j}], i = 1, ..., 3$$

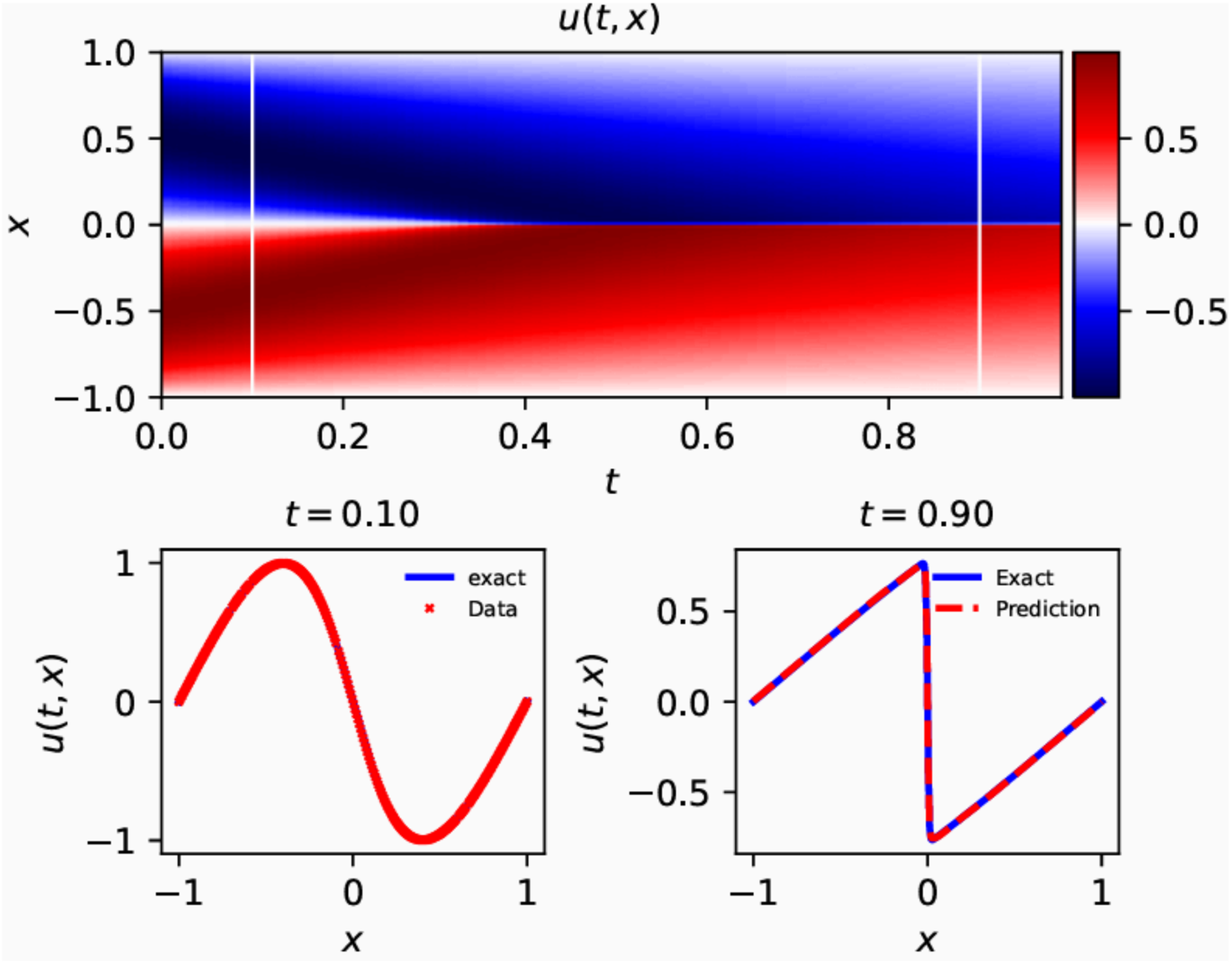
$$u^{n+1} = u^n - \Delta t \sum_{j=1}^{3} b_j \mathcal{N}[u^{n+c_j}]$$

Approach: Implicit Runge-Kutta for PDEs like Burgers' Equation.

Key Findings:

- Higher RK stages (q) and larger Δt improve stability.
- Example: For q=500, loss drops to 0.00230 at Δt=0.8.

Numerical Results



Error Analysis

Data Table

q	$\delta t = 0.2$	$\delta t = 0.4$	$\delta t = 0.6$	$\delta t = 0.8$
1	0.035028525	0.11004553	0.228297338	0.375338256
2	0.007118945	0.051910978	0.090987705	0.09543623
4	0.003480386	0.010396939	0.03266187	0.040469777
8	0.004592469	0.006252498	0.008332861	0.024373775
16	0.003610709	0.00476564	0.011588301	0.028217085
32	0.003573923	0.004282821	0.005074797	0.025276707
64	0.003372173	0.003741181	0.005913021	0.008323381
100	0.003298198	0.00281849	0.002983324	0.007659137
500	0.003292774	0.003592792	0.003480864	0.002296612

Rate Table

$q \setminus \delta$ (delta t)	0.2–0.4	0.4–0.6	0.6–0.8
1	1.6515	1.7998	1.7282
2	2.8663	1.3841	0.1659
4	1.5788	2.8232	0.7451
8	0.4452	0.7084	3.7309
16	0.4004	2.1915	3.0935
32	0.2611	0.4185	5.5811
64	0.1498	1.1290	1.1885
100	0.2268	0.1402	3.2774
500	0.1258	1.1037	1.4455

In our study, we were able to determine the relationship between Nu (number of boundary
points) and Nf (number of collocation points), showing that higher values of Nu and Nf
resulted in significantly more accurate predictions in the continuous case.

Furthermore,
in the discrete case, where Runge-Kutta (RK) method was employed, we
observed that larger values of q (number of time steps) Δt (time step size) led to
improved model performance and better approximation accuracy.

The methods showcase promising results across diverse problems in computational
science, opening new possibilities for data-driven modeling and scientific computing.

Results & Advantages

Trends from Tables:

- ↑ N_u,N_f→ ↓ MSE
- ↑ q,Δt→ ↓ Loss.

Advantages of PINNs:

- Handles nonlinearity without linearisation
- Data-efficient (works with sparse/noisy data).
- Flexible across PDE types (hyperbolic, dispersive, phase-field).

Conclusion

- PINNs unify physics and machine learning.
- Demonstrated success on Burgers, KdV, and Cahn-Hilliard equations.
- Future work: Adaptive sampling, domain decomposition.

Visual: Summary graphic of PINN applications.

References

- M. Raissi, P. Perdikaris, and G.E. Karniadakis, ”Physics-informed neural networks: Adeep learning framework for solving forward and inverse problems involving non linear partial differential equations,” Journal of Computational Physics, vol. 378, pp. 686–707, 2019. https://doi.org/10.1016/j.jcp.2018.10.045.
- L. Lu, X. Meng, Z. Mao, and G.E. Karniadakis, ”DeepXDE: A deep learning library for solving differential equations,” SIAM Review, vol. 63, no. 1, pp. 208–228, 2021. https://arxiv.org/abs/2102.04626.
- . A.D. Jagtap, K. Kawaguchi, and G.E. Karniadakis, ”Locally adaptive activation functions with slope recovery for deep and physics-informed neural networks,” Pro ceedings of the Royal Society A, vol. 476, no. 2239, 2020. https://arxiv.org/ abs/2004.04638
- **M. Raissi, P. Perdikaris, and G.E. Karniadakis, ”Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations.”**
- GitHub: maziarraissi/PINNs.

Thank you very much!