PIXEL: Physics-Informed Cell Representations for Fast and Accurate PDE Solvers

Namgyu Kang[1] **Byeonghyeon Lee**[1] **Youngjoon Hong**[2] **Seok-Bae Yun**[2] **Eunbyung Park**[1,3*]
[1]Department of Artificial Intelligence [2]Department of Mathematics
[3]Department of Electrical and Computer Engineering
Sungkyunkwan University
{kangnamgyu27, leebh0102, hongyj, sbyun01, epark}@skku.edu

Abstract

Physics-informed neural networks (PINNs) have recently emerged and succeeded in various PDEs problems with their mesh-free properties, and unsupervised training. However, their slower convergence speed and relatively inaccurate solutions often limit their broader applicability. This paper proposes a new kind of data-driven PDEs solver, physics-informed cell representations (PIXEL), elegantly combining classical numerical methods and learning-based approaches. We adopt a grid structure from the numerical methods to improve accuracy and convergence speed and overcome the spectral bias presented in PINNs. Moreover, the proposed method enjoys the same benefits in PINNs, e.g., using the same optimization frameworks to solve both forward and inverse PDE problems and readily enforcing PDE constraints with modern automatic differentiation techniques. Project URL: https://namgyukang.github.io/PIXEL/

1 Introduction

Physics-informed neural networks (PINNs) have recently received significant attention as new data-driven Partial Differential Equation (PDE) solvers [11]. However, PINNs suffer from slow convergence rates, and they often fall short of the desired accuracy [3, 16, 17, 19].

In addition, Multi-layer perceptron (MLP) architecture is known to have spectral bias, which prioritizes learning low-frequency components. Recent studies have shown that spectral bias [10] indeed exists in PINN models [8, 18] and this tendency towards smooth function approximation often leads to failure to accurately capture high-frequency components or singular behaviors in solution functions.

In this paper, we propose physics-informed cell representations (coined as *PIXEL*), which is jointly trained with shallow neural networks to improve convergence rates and accuracy. Inspired by classical numerical solvers that use grid points, we divide solution space into many subspaces and allocate trainable parameters for each cell. The key motivation of PIXEL is to disentangle the trainable parameters with respect to the input coordinates. In neural network-only approaches, such as PINNs, all network parameters are affected by the entire input domain space. Therefore, parameter updates for specific input coordinates influence the outputs of other input subspaces. On the other hand, each input coordinate has dedicated trainable parameters updated only for certain input coordinates in PIXEL. This parameter separation technique has been explored in visual computing domains [2, 5, 12, 13, 9, 1] and has shown remarkable success in terms of convergence speed of the training procedure.

Furthermore, PIXEL is immuned to spectral bias presented in PINNs. A root cause of the bias is the shared parameters of neural networks for the entire input space. However, PIXEL, each cell is

*Corresponding author.

DLDE Workshop in the 36th Conference on Neural Information Processing Systems (NeurIPS 2022).

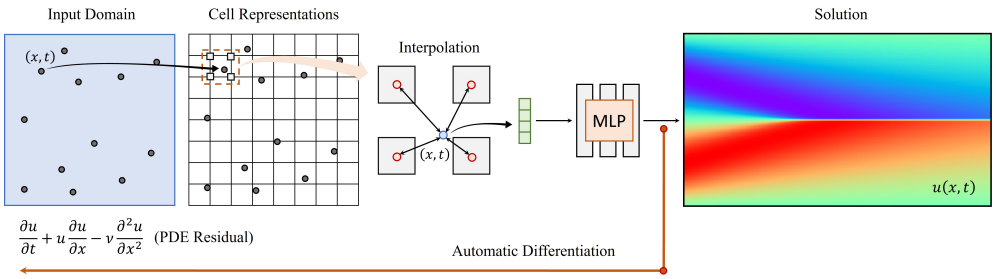

Figure 1: Overall PIXEL architecture for a PDE solver

only responsible for a small sub-region of the input domain. Therefore, a large difference between neighboring cell values can better approximate high-frequency components or singular behaviors.

Our proposed model PIXEL uses a differentiable interpolation scheme to implement virtually infinite resolution grids, and the resulting representations are differentiable with respect to input coordinates.

2 PIXEL

2.1 Physics-informed neural networks

We briefly review physics-informed neural networks (PINNs). Let us begin with the initial-boundary value problems with PDEs. A general formulation can be written as follows:

$$\mathcal{N}_{x,t}[u](x,t) = f(x,t), \qquad x \in \Omega, t \in [0,T], \tag{1}$$

$$u(x,0) = g(x), \qquad x \in \Omega, \tag{2}$$

$$\mathcal{B}_{x,t}[u](x,t) = h(x,t), \qquad x \in \partial\Omega, t \in [0,T], \tag{3}$$

where $\mathcal{N}_{x,t}[\cdot]$ is a linear or non-linear differential operator, $\mathcal{B}_{x,t}[\cdot]$ is also a differential operator for boundary conditions, and the initial conditions are denoted as $u(x,0) = g(x)$. $u(x,t)$ represents the unknown solution function and PINNs use neural networks, $u_\theta(x,t)$, parameterized by the trainable model parameters θ, to approximate the solution. Then, neural networks are trained by minimizing the following loss function.

$$\mathcal{L}(\theta) = \lambda_{\text{res}}\mathcal{L}_{\text{res}}(\theta) + \lambda_{\text{ic}}\mathcal{L}_{\text{ic}}(\theta) + \lambda_{\text{bc}}\mathcal{L}_{\text{bc}}(\theta) + \lambda_{\text{data}}\mathcal{L}_{\text{data}}(\theta), \tag{4}$$

where, $\mathcal{L}_{\text{res}}, \mathcal{L}_{\text{ic}}, \mathcal{L}_{\text{bc}}, \mathcal{L}_{\text{data}}$ are PDE residual, initial condition, boundary condition, and observational data loss functions, respectively. λ are weighting factors for each loss term. Each loss term is usually defined as mean square loss functions over sampled points. For example, a PDE residual loss $\mathcal{L}_{\text{res}}$ over N_{res} collocation points can be written as,

$$\mathcal{L}_{\text{res}}(\theta) = \frac{1}{N_{\text{res}}} \sum_{i=1}^{N_{\text{res}}} |\mathcal{N}_{x,t}[u_\theta](x_i,t_i) - f(x_i,t_i)|^2. \tag{5}$$

2.2 Neural networks and grid representations

The proposed architecture consists of a small neural network and a feature extractor of input coordinates. We approximate solution functions by a neural network f parameterized by θ,

$$u(x,t) \approx f(\phi(\hat{x},\hat{t},\mathcal{C});\theta), \tag{6}$$

where $\mathcal{C}$ is a grid representation and ϕ is a feature extractor given input coordinates and the grid representation using an interpolation scheme. Note that both $\mathcal{C}$ and θ are model parameters and updated during the training procedure. The dimension of $\mathcal{C}$ is determined by the dimension of the spatial input domain. For example, if $x \in \Omega \subset \mathbb{R}$ then $\mathcal{C} \in \mathbb{R}^{c \times H \times W}$ is a three dimensional tensor, where the channel size c, and H and W are spatial and temporal grid sizes, respectively. $\hat{x} \in [1,H]$ and $\hat{t} \in [1,W]$ are normalized input coordinates assuming input domain $\Omega \subset \mathbb{R}$ and $[0,T]$ are tightly bounded by a rectangular grid. If $x \in \Omega \subset \mathbb{R}^2$ then $\mathcal{C}$ is a four dimensional tensor, and if $x \in \Omega \subset \mathbb{R}^3$ then $\mathcal{C}$ is a five dimensional tensor [2].

[2] Without temporal coordinates, e.g., Helmholtz equation, $\mathcal{C}$ is three or four dimensional tensors, respectively.

2.3 Mesh-agnostic representations through interpolation

In two dimensional grid cases, $x \in \Omega \subset \mathbb{R}$ and $\mathcal{C} \in \mathbb{R}^{c \times H \times W}$, the following is a feature extractor.

$$\phi(\hat{x}, \hat{t}, \mathcal{C}) = \sum_{i=1}^{H} \sum_{j=1}^{W} \mathcal{C}_{ij} k(\max(0, 1 - |\hat{x} - i|)) k(\max(0, 1 - |\hat{y} - j|)), \tag{7}$$

where $\mathcal{C}_{ij} \in \mathbb{R}^c$ denotes cell representations at (i, j), and $k : [0, 1] \to [0, 1]$ represents a monotonically increasing smooth function. Given normalized coordinates $(\hat{x}, \hat{t})$, it looks up neighboring points (2^{d+1} points) and computes the weighted sum of the representations according to a predefined kernel function. It is differentiable w.r.t input coordinates so that we can easily compute partial derivatives for PDE differential operator $\mathcal{N}[\cdot]$ by using automatic differentiation.

To support higher-order gradients, we use a cosine interpolation kernel which is multiple differentiable. We use, $k(x) := \frac{1}{2}(1 - \cos(\pi x))$ because it is everywhere continuous and infinitely differentiable.

2.4 Multigrid representations

The more fine-grained grids, the less chance a grid cell would see the points. It would result in highly overfitted solutions.

Inspired by recent hierarchical grid representations [14, 9], we suggest to use multigrid representations. We stack up multiple coarse-grained grid representations and the representations at each collocation point are computed by summing up the representations from all grids. With a slight abuse of notation, a multigrid representation is defined as four dimensional tensors in two dimensional grids $\mathcal{C} \in \mathbb{R}^{M \times c \times H \times W}$. Then we can reformulate an interpolation function as,

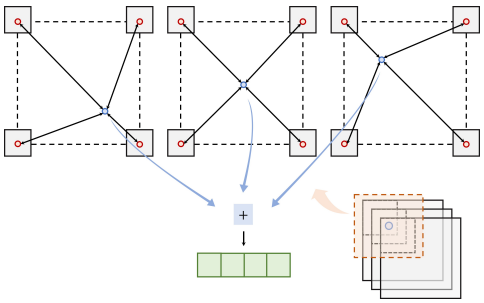

Figure 2: Multigrid representations

$$\phi_{\text{multi}}(\hat{x}, \hat{t}, \mathcal{C}) = \sum_{i=1}^{M} \phi(\hat{x} + \frac{(i-1)}{M}, \hat{t} + \frac{(i-1)}{M}, \mathcal{C}^i), \tag{8}$$

where $C^i \in \mathbb{R}^{c \times H \times W}$ denotes a grid representation. Multigrid approach was very critical to overall PIXEL performance. Without this, we observed that PIXEL suffers from serious overfitting issues.

3 Experiments

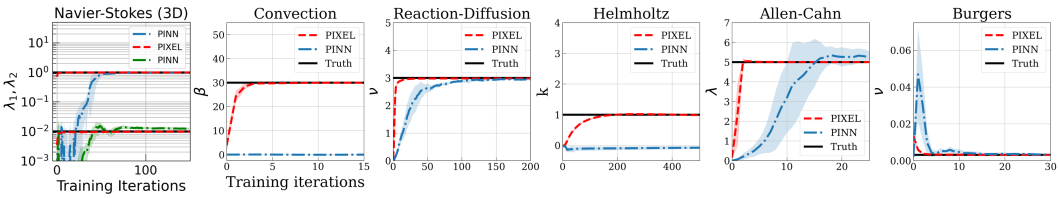

Figure 3: In the inverse problems, PIXEL shows a faster convergence and more accurate predictions. The shaded areas show 95% confidence intervals of 5 different runs with different *random* seeds.

We experimented on 6 various 2D and 3D PDEs by using L-BFGS optimizer. We implemented a customized CUDA kernel of the triple backward grid sampler that supports high-order gradients [15], the runtime and memory capacity were efficiently reduced. *Navier-Stokes Eq*, [11] shows the result of the inverse problem, which is the multivariate coefficient simultaneous prediction. *Convection Eq*, [3] shows that without sequential training, the original PINNs has failed to find accurate solutions. *Reaction-diffusion Eq*, which is also a PDE that the original PINNs have worked poorly [3]. About *Helmholtz Eq*, [17] has reported that the original PINN has struggled to find an accurate solution. We tested both low and high frequency parameters setting. *Allen-Cahn Eq* is a non-linear second-order PDE that is known to be challenging to solve using conventional PINNs [17], and a few techniques, including adaptive re-sampling [20] and weighting algorithms [7, 4, 19], have been proposed. Finally *Burgers Eq*, which is non-linear and known to have a singular behavior.

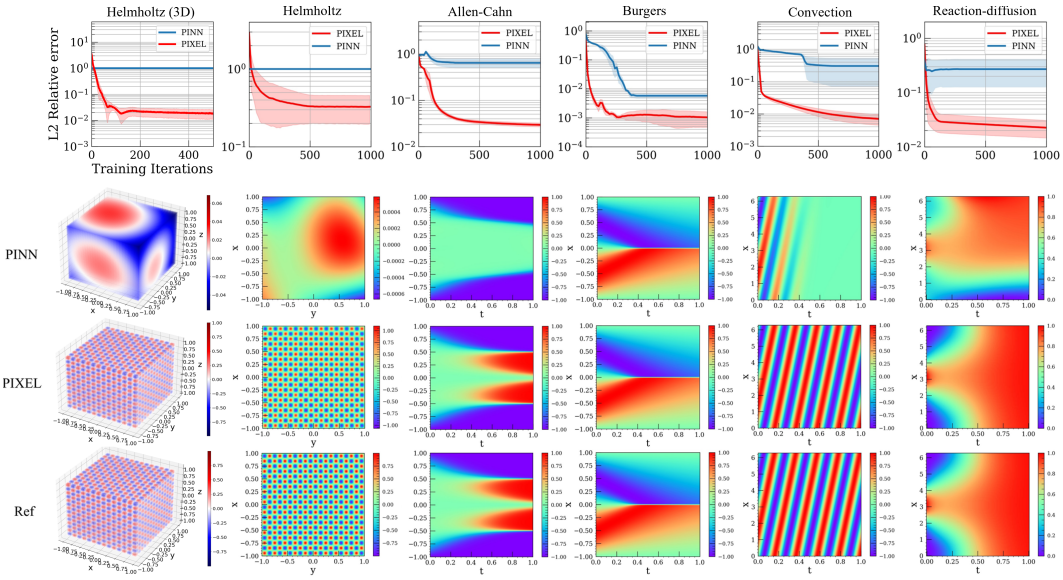

Figure 4: For the forward problem, training loss curves and solutions of PDEs: the shaded areas show 80% confidence intervals of 5 different runs with different *random* seeds (100, 200, 300, 400, 500).

Methods	Convection	Reaction diffusion	Allen-Cahn	Burgers	Helmholtz (2D) ($a_1 = 1, a_2 = 4$)	Helmholtz (2D) ($a_1 = a_2 = 10$)	Helmholtz (3D) ($a_1 = a_2 = a_3 = 7$)
PINN (8-40)	N/A	N/A	N/A	5.60e-04	N/A	N/A	N/A
Sequential training	2.02-e02	1.56e-02	N/A	N/A	N/A	N/A	N/A
Learning rate annealing	N/A	N/A	N/A	N/A	3.69-e03	N/A	N/A
Self-attention	N/A	N/A	2.10e-02	N/A	N/A	N/A	N/A
Time marching	N/A	N/A	1.68e-02	N/A	N/A	N/A	N/A
Causal training	N/A	N/A	1.43e-03	N/A	N/A	N/A	N/A
Causal training (modified MLP)	N/A	N/A	**1.39e-04**	N/A	N/A	N/A	N/A
PINN (ours)	3.02e-01 (± 3.40e-01) (best : 2.45e-02)	2.46e-01 (± 2.25e-01) (best : 2.36e-02)	9.08e-01 (± 1.68e-02) (best : 5.23e-01)	5.77e-03 (± 1.74e-03) (best : 3.35e-03)	4.02e-01 (± 4.88e-01) (best : 2.30e-03)	1.00 (± 1.49e-06) (best: 1.00)	1.00 (± 7.19e-04) (best : 1.00)
PIXEL (16,4,16,16)	9.48e-03 (± 1.62e-03) (best : 6.39e-03)	1.63e-02 (± 2.11e-03) (best : 1.33e-02)	1.77e-02 (± 4.67e-03) (best : 9.64e-03)	9.98e-04 (± 3.70e-04) (best : 4.88e-04)	8.27e-03 (± 1.73e-03) (best : 2.71e-03)	3.05e-01 (± 2.38e-01) **(best : 7.47e-02)**	5.06e-03 (± 2.64e-03) **(best : 6.61e-04)**
PIXEL (64,4,16,16)	4.69e-03 (± 1.25e-03) **(best : 2.41e-03)**	8.11e-03 (± 8.74e-05) (best : 7.81e-03)	1.90e-02 (± 8.35e-03) (best : 4.56e-03)	6.20e-04 (± 2.09e-04) (best : 3.85e-04)	2.57e-03 (± 8.47e-04) (best : 1.14e-03)	4.26e-01 (± 3.10e-01) (best : 1.05e-01)	1.62e-01 (± 9.78e-02) (best : 5.54e-02)
PIXEL (96,4,16,16)	6.19e-03 (± 3.36e-03) (best : 3.12e-03)	8.26e-03 (± 1.18e-03) **(best : 7.16e-03)**	1.63e-02 (± 3.95e-03) (best : 8.86e-03)	7.01e-04 (± 3.60e-04) **(best : 3.20e-04)**	1.19e-03 (± 1.45e-04) **(best : 8.63e-04)**	3.11e-01 (± 1.43e-01) (best : 1.70e-01)	1.53e-01 (± 6.81e-02) (best : 1.34e-02)

Table 1: The comparisons to other methods (L_2 relative errors). The standard deviation for 5 experiments is shown with the mean in the table. We compared against PINN [11], Sequential training [3], Learning rate annealing [17], Self-attention [7], Time marching [6], and Causal training [16].

3.1 Results

For the forward problem, In *Convection equation*, the resulting solution image of PINN was not correctly updated for the later time domain, $t > 0.4$. However, PIXEL converged to a high accurate solution. For *Reaction-diffusion equation*, PINN showed a constant curve shape after 285 iterations in the averaged loss curve. Whereas PIXEL showed an exponential decay shape until 10,000 iterations. In *Helmholtz* which has the high-frequency components, as we expected, PINN has failed to converge to an accurate solution due to the spectral bias. In contrast, PIXEL obtained high-accuracy solutions quickly in 3-dimension as well as 2-dimension. For *Allen-Cahn*, which is known to be notoriously difficult, the previous studies have demonstrated that PINNs perform very poorly without additional training techniques, such as time marching techniques [6] or causal training [16]. However, our method can obtain accurate solutions without any additional methods.

For the inverse problem, PINN showed fluctuation in the prediction curve due to the *random* seed. In contrast, PIXEL showed robustness in predicting regardless of *random* seed in 3-dimension *Navier-Stokes equation* as well as 2-dimension equations. Except for Helmholtz equation which PINN failed to train, PIXEL showed convergence in a *few* iterations for PDEs. It was found that PIXEL achieves better performance than PINN at multivariate coefficient prediction of the inverse problem in 3-dimension *Navier-Stokes equation*.

Acknowledgments

We are thankful to Junwoo Cho for helpful discussion and contributions. This research was supported by the Ministry of Science and ICT (MSIT) of Korea, under the National Research Foundation (NRF) grant (2022R1F1A1064184, 2022R1A4A3033571), Institute of Information and Communication Technology Planning Evaluation (IITP) grants for the AI Graduate School program (IITP-2019-0-00421). The research of Seok-Bae Yun was supported by Samsung Science and Technology Foundation under Project Number SSTF-BA1801-02.

References

[1] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. Tensorf: Tensorial radiance fields. In *European Conference on Computer Vision (ECCV)*, 2022.

[2] Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5501–5510, June 2022.

[3] Aditi Krishnapriyan, Amir Gholami, Shandian Zhe, Robert Kirby, and Michael W Mahoney. Characterizing possible failure modes in physics-informed neural networks. *Advances in Neural Information Processing Systems*, 34, 2021.

[4] Dehao Liu and Yan Wang. A dual-dimer method for training physics-constrained neural networks with minimax architecture. *Neural Networks*, 136:112–125, 2021.

[5] Julien NP Martel, David B Lindell, Connor Z Lin, Eric R Chan, Marco Monteiro, and Gordon Wetzstein. Acorn: adaptive coordinate networks for neural scene representation. *ACM Transactions on Graphics (TOG)*, 40(4):1–13, 2021.

[6] Revanth Mattey and Susanta Ghosh. A novel sequential method to train physics informed neural networks for allen cahn and cahn hilliard equations. *Computer Methods in Applied Mechanics and Engineering*, 390:114474, 2022.

[7] Levi McClenny and Ulisses Braga-Neto. Self-adaptive physics-informed neural networks using a soft attention mechanism, 2020.

[8] Ben Moseley, Andrew Markham, and Tarje Nissen-Meyer. Finite basis physics-informed neural networks (fbpinns): a scalable domain decomposition approach for solving differential equations, 2021.

[9] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics*, 41(4):1–15, jul 2022.

[10] Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred Hamprecht, Yoshua Bengio, and Aaron Courville. On the spectral bias of neural networks. In *International Conference on Machine Learning*, pages 5301–5310. PMLR, 2019.

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

[12] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14335–14345, 2021.

[13] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5459–5469, June 2022.

[14] Towaki Takikawa, Joey Litalien, Kangxue Yin, Karsten Kreis, Charles Loop, Derek Nowrouzezahrai, Alec Jacobson, Morgan McGuire, and Sanja Fidler. Neural geometric level of detail: Real-time rendering with implicit 3D shapes. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 11358–11367, 2021.

[15] Jingwen Wang, Tymoteusz Bleja, and Lourdes Agapito. Go-surf: Neural feature grid optimization for fast, high-fidelity rgb-d surface reconstruction. In *2022 International Conference on 3D Vision (3DV)*. IEEE, 2022.

[16] Sifan Wang, Shyam Sankaran, and Paris Perdikaris. Respecting causality is all you need for training physics-informed neural networks, 2022.

[17] Sifan Wang, Yujun Teng, and Paris Perdikaris. Understanding and mitigating gradient flow pathologies in physics-informed neural networks. *SIAM Journal on Scientific Computing*, 43(5):A3055–A3081, 2021.

[18] Sifan Wang, Hanwen Wang, and Paris Perdikaris. On the eigenvector bias of fourier feature networks: From regression to solving multi-scale pdes with physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering*, 384:113938, 2021.

[19] Sifan Wang, Xinling Yu, and Paris Perdikaris. When and why pinns fail to train: A neural tangent kernel perspective. *Journal of Computational Physics*, 449:110768, 2022.

[20] Colby L. Wight and Jia Zhao. Solving allen-cahn and cahn-hilliard equations using the adaptive physics informed neural networks, 2020.

A Experimental setup and details

	PDEs	Initial condition	Boundary condition	Inverse problem coefficient
Convection	$u_t + \beta u_x = 0$ $x \in [0, 2\pi],\ t \in [0, T]$	$u(x, 0) = \sin x$	$u(0, t) = u(2\pi, t)$	β
Reaction Diffusion	$u_t - \nu u_{xx} - \rho u(1-u) = 0$ $x \in [0, 2\pi], t \in [0, T]$	$u(x, 0) = h(x)$	$u(0, t) = u(2\pi, t)$	ν
Helmholtz (3D)	$\Delta u(x, y, z) + k^2 u(x, y) = q(x, y, z)$ $q(x, y, z) = k^2 \sin(a_1\pi x)\sin(a_2\pi y)\sin(a_2\pi z)$ $-(a_1\pi)^2 \sin(a_1\pi x)\sin(a_2\pi y)\sin(a_2\pi z)$ $-(a_2\pi)^2 \sin(a_1\pi x)\sin(a_2\pi y)\sin(a_2\pi z)$ $(x, y) \in [-1, 1]^2$	·	$u(x, y, z) = 0,$ $(x, y, z) \in \partial[-1, 1]^2$	·
Helmholtz (2D)	$\Delta u(x, y) + k^2 u(x, y) = q(x, y)$ $q(x, y) = k^2 \sin(a_1\pi x)\sin(a_2\pi y)$ $-(a_1\pi)^2 \sin(a_1\pi x)\sin(a_2\pi y)$ $-(a_2\pi)^2 \sin(a_1\pi x)\sin(a_2\pi y)$ $(x, y) \in [-1, 1]^2$	·	$u(x, y) = 0,$ $(x, y) \in \partial[-1, 1]^2$	k
Navier-Stokes (3D)	$u_t + \lambda_1(uu_x + vu_y) = -p_x + \lambda_2(u_x x + u_y y)$ $v_t + \lambda_1(uv_x + vv_y) = -p_y + \lambda_2(v_x x + v_y y)$ $u_x + v_y = 0$	·	·	λ_1, λ_2
Allen-Cahn	$u_t - 0.0001u_{xx} + \lambda u^3 - 5u = 0$ $x \in [-1, 1], t \in [0, 1], \lambda = 5$	$u(x, 0) = x^2 \cos(\pi x)$	$u(t, -1) = u(t, 1)$ $u_x(t, -1) = u_x(t, 1)$	λ
Burgers	$u_t + uu_x - \nu u_{xx} = 0$ $x \in [-1, 1],\ t \in [0, 1]$	$u(0, x) = -\sin(\pi x)$	$u(t, -1) = u(t, 1) = 0$	ν

Table 2: The formulations of various PDEs in our experiments of the forward and the inverse problem.

For all experiments, we used Limited-memory BFGS (L-BFGS) second-order optimization algorithms. In many cases of training PINNs, it outperforms other first-order optimization algorithms, such as ADAM or SGD. We set the learning rate to 1.0 and used the strong-wolfe line search algorithm. In every L-BFGS iteration, we randomly sample collocation points to make the model robust to the entire input and time domains for training PIXELs. We found that PINNs often have struggled to converge in this setting, so we initially sampled the collocation points and fixed them, which has been a common practice in PINNs literature. To compute the accuracy of the approximated solutions, we used the relative L_2 error, defined as $\frac{||u-\hat{u}||_2}{||u||_2}$, where $\hat{u}$ is a predicted solution and u is a reference solution. We used NVIDIA RTX3090 GPUs and A100 GPUs with 40 GB of memory. For all experiments, we used 2 hidden layers and 16 hidden dimensions for shallow MLP architecture, and a hyperbolic tangent activation function (tanh) was used. For coefficients of the loss function, we used $\lambda_{\text{ic}} = \lambda_{\text{bc}} = 1, \lambda_{\text{data}} = 0$.

Convection equation. A shallow MLP of 2 layers with 16 hidden units was used. The baseline PINN model was trained with the same number of data points from PIXEL. For the PINN model, we used 3 hidden layers and 50 hidden dimensions following the architecture in [1].

Reaction-diffusion equation. We used the same formulation in [1], and conducted experiments with the same PDE parameters ($\rho = 5$, $\nu = 3$). For training PINNs, we used 3 hidden layers and 50 hidden dimensions following the architecture in [1].

2D Helmholtz equation. We used the same formulation in [4]. The source term is given as $q(x, y) = -(a_1\pi)^2 u(x, y) - (a_2\pi)^2 u(x, y) + k^2 u(x, y)$. The analytic solution of this formulation is known as $u(x, y) = \sin(a_1\pi x)\sin(a_2\pi y)$. We tested the PDE parameters $k = 1$, $a_1 = 1$,and $a_2 = 4$. For a more complex setting, we also tested $k = 1$, $a_1 = 10$ and $a_2 = 10$. For the baseline PINN model, we used 7 hidden layers and 100 hidden dimensions following the architecture in [4].

3D Helmholtz equation. For 3D Helmholtz equation, the source term with $q(x, y, z) = -(a_1\pi)^2 u(x, y, z) - (a_2\pi)^2 u(x, y, z) - (a_3\pi)^2 u(x, y, z) + k^2 u(x, y, z)$ is used. We tested the complex PDE parameters setting $k = 1$, $a_1 = 7$, $a_2 = 7$, and $a_3 = 7$. MLP architecture is same as 2D problems.

Allen-Cahn equation. For the baseline PINN model, we used 6 hidden layers and 128 hidden dimensions following the architecture in [2] In the case of Allen-Cahn, there was a problem that NaN occurs in PINN when the seed is 400. Unlike PIXEL, PINN excludes seed 400 only in the case of Allen-Cahn.

Burgers equation. We used the same PDE parameter, $\nu = 0.01/\pi$ in [3]. For the baseline PINN model, we adopted the same architecture in [3], using 8 hidden layers and 40 hidden dimensions.

3D Navier-Stokes equation In this paper, the same Navier-Stokes equation with the original PINN paper [11] is used. We used 9 hidden layers and 20 hidden dimensions for the baseline PINN model to predict the multivariate coefficient in the inverse problem.

A.1 Hyperparameter of experiments

	Convection	Reaction diffusion	Helmholtz (3D)	Helmholtz (2D)	Allen-Cahn	Burgers
Grid sizes	(96, 4, 16, 16)	(96, 4, 16, 16)	(16, 4, 16, 16)	(96, 4, 16, 16)	(96, 4, 16, 16)	(96, 4, 16, 16)
# collocation pts	100,000	100,000	400,000	100,000	100,000	100,000
# ic pts	100,000	100,000	N/A	N/A	100,000	100,000
# bc pts	100,000	100,000	400,000	100,000	N/A	100,000
λ_{res}	0.005	0.01	0.01	0.0001	0.1	0.01

Table 3: Experimental details of the forward problems for training PIXELs: (96, 4, 16, 16) means, 96 multigrids, channel size of 4, the spatial grid size of 16, and temporal grid size of 16. # collocation pts, # ic pts, and # bc pts denote the number of collocation, initial condition, and boundary condition points, respectively.

	Navier-Stokes (3D)	Convection	Reaction diffusion	Helmholtz ($a_1 = 1, a_2 = 4$)	Allen-Cahn	Burgers
Grid sizes	(150, 4, 16, 16)	(192, 4, 16, 16)	(192, 4, 16, 16)	(16, 4, 16, 16)	(192, 4, 16, 16)	(192, 4, 16, 16)
# collocation pts	100,000	100,000	100,000	100,000	100,000	100,000
# ic pts	100,000	100,000	100,000	N/A	100,000	100,000
# bc pts	100,000	100,000	100,000	100,000	N/A	100,000
λ_{res}	1.25	0.005	0.005	0.00001	0.1	0.0005

Table 4: Experimental details of the inverse problems (PIXEL)

Table 3 and Table 4 shows the experimental details for the forward and inverse problems respectively. Note that all other hyperparameters of the forward problems and the inverse problems are same explained in main text, including the architecture of PINN, confirming that the proposed method is not sensitive to hyperparameters. For the inverse problems, The number of data points used for ground truth data points is 1,000,000 for Navier-Stokes equation, 25,600 for convection equation, Burger equation, and Reaction-diffusion equation. The Helmholtz equation uses 490,000 and the Allen-Cahn equation uses 102,912.

B Grid size and the number of data points

This section studies the relationship between the amounts of training data points and the grid sizes. We introduced multigrid representations that inject a smooth prior into the whole framework, which would reduce the required data points for each training iteration. We demonstrate this with the convection equation example by varying the number of training data points (collocation and initial condition) and the number of multigrid representations. We fixed the grid size 16 and the channel size 4, and varied the number of multigrids from 4 to 64. We reported the L_2 relative errors after 500 L-BFGS iterations. As shown in Table 5, our method is robust to the amounts of training data points. Although we can achieve higher accuracy with more training points, it performs comparably with a few data regimes.

Multigrid sizes	5,000 (# pts)	10,000	20,000	50,000	100,000
(4, 4, 16, 16)	2.35e-01	2.32e-01	2.25e-01	2.23e-01	2.21e-01
(8, 4, 16, 16)	5.59e-02	4.11e-02	3.10e-02	2.95e-02	3.73e-02
(16, 4, 16, 16)	4.47e-02	2.99e-02	2.51e-02	8.01e-03	1.91e-02
(32, 4, 16, 16)	4.40e-02	2.69e-02	1.13e-02	1.13e-02	8.22e-03

Table 5: Varying the amounts of training points and the number of multigrids (L_2 relative errors)

C The visualization of multigrid representations

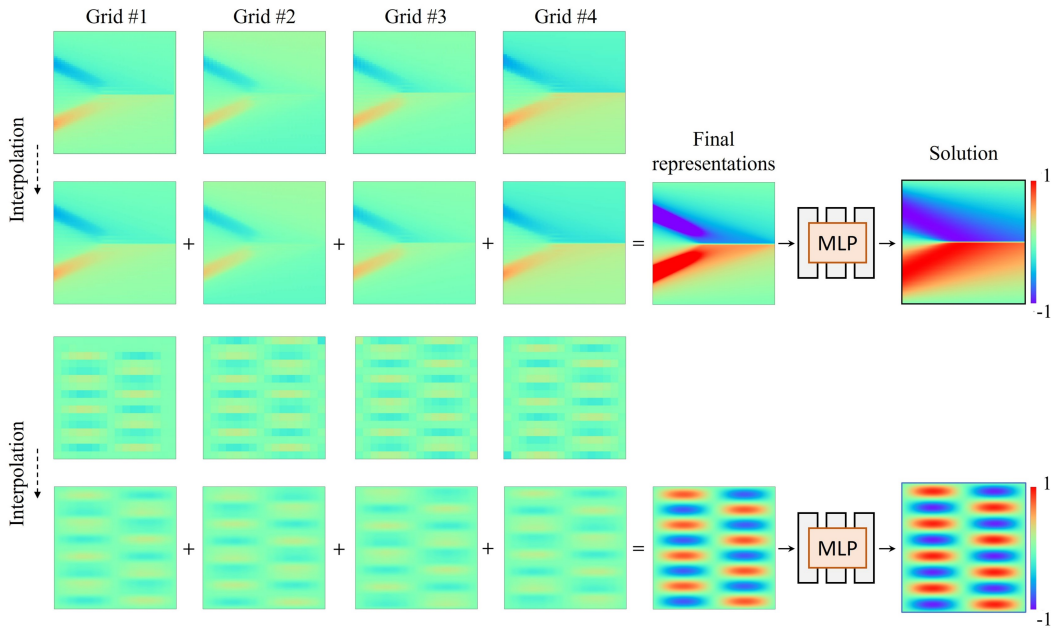

Figure 5: Visualization of multigrid representations for Burgers and Helmholtz equations (best viewed zoomed in): The first row shows image plots of each grid representation, and the second row shows the representations after the interpolation. The final representations are obtained through the sum of each interpolated cell, followed by an MLP to generate the solution. We used (4,4,64,64) and (4,4,16,16) multigrid representations for Burgers and Helmholtz, respectively.

We demonstrate the intermediate results in Figure 5. In Burgers example (the first and the second rows), we used (4,1,64,64) configuration and two layers of MLP with 16 hidden units. As we expect, each grid show foggy images since the final solution will be the sum of all multigrid representations. Also, we shifted each grid, which resulted in slightly different images from each other. The final solution is completed in the last stage by filling up the remaining contents using an MLP. Importantly, we note that the singular behavior (shock, a thin line in the middle of the solution image) is already well captured by the grid representations. The role of MLP was to represent the smooth global

component in solution function. Therefore, the proposed grid representations and an MLP combine each other's strengths to obtain better final solutions.

In Helmholtz example, we used small size grids (4,4,16,16). Thus, we can observe notable differences after cosine interpolation (grid-like pattern before the interpolation). We also note that the grid representations already represent complex patterns, and the last MLP stage also refined the solution by darkening the colors in the solution image.

D Visualizations

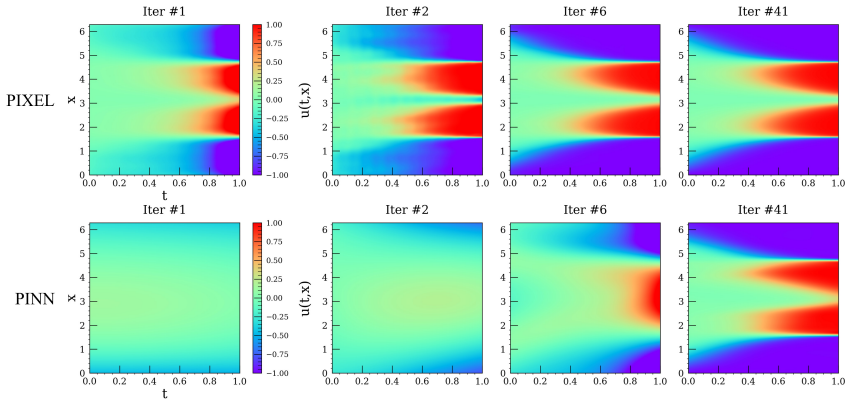

Figure 6: Inverse problem of allen-cahn equation

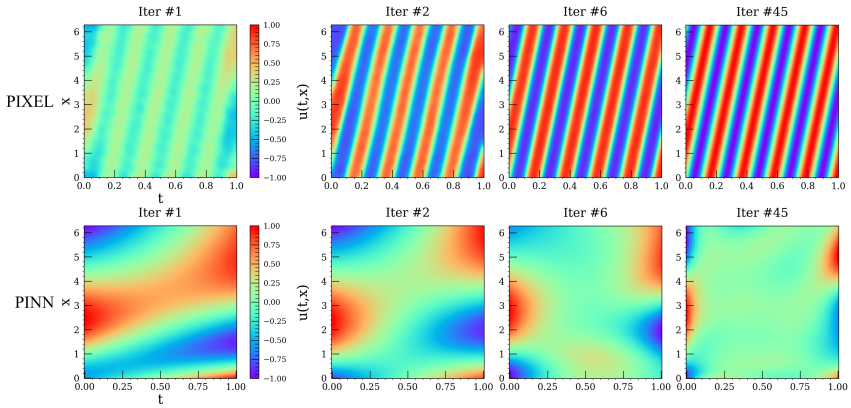

Figure 7: Inverse problem of convection equation

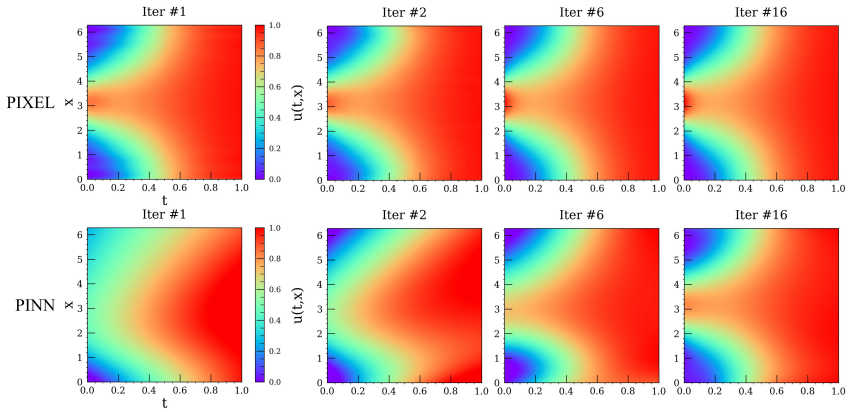

Figure 8: Inverse problem of reaction-diffusion equation

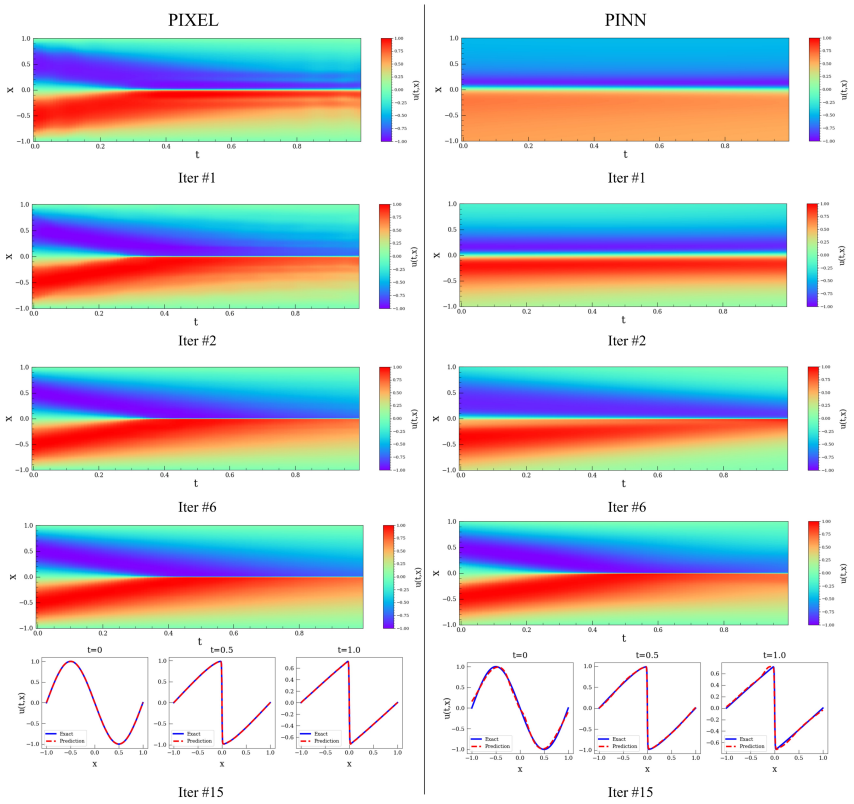

Figure 9: Inverse problem of burgers equation

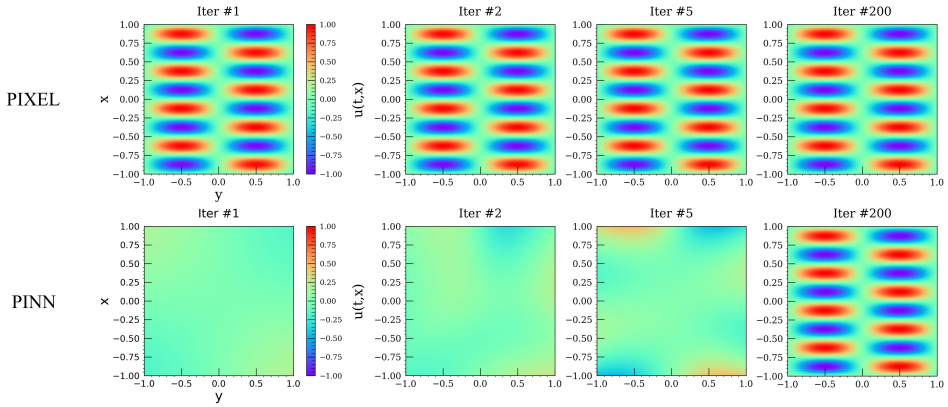

Figure 10: Inverse problem of helmholtz equation.

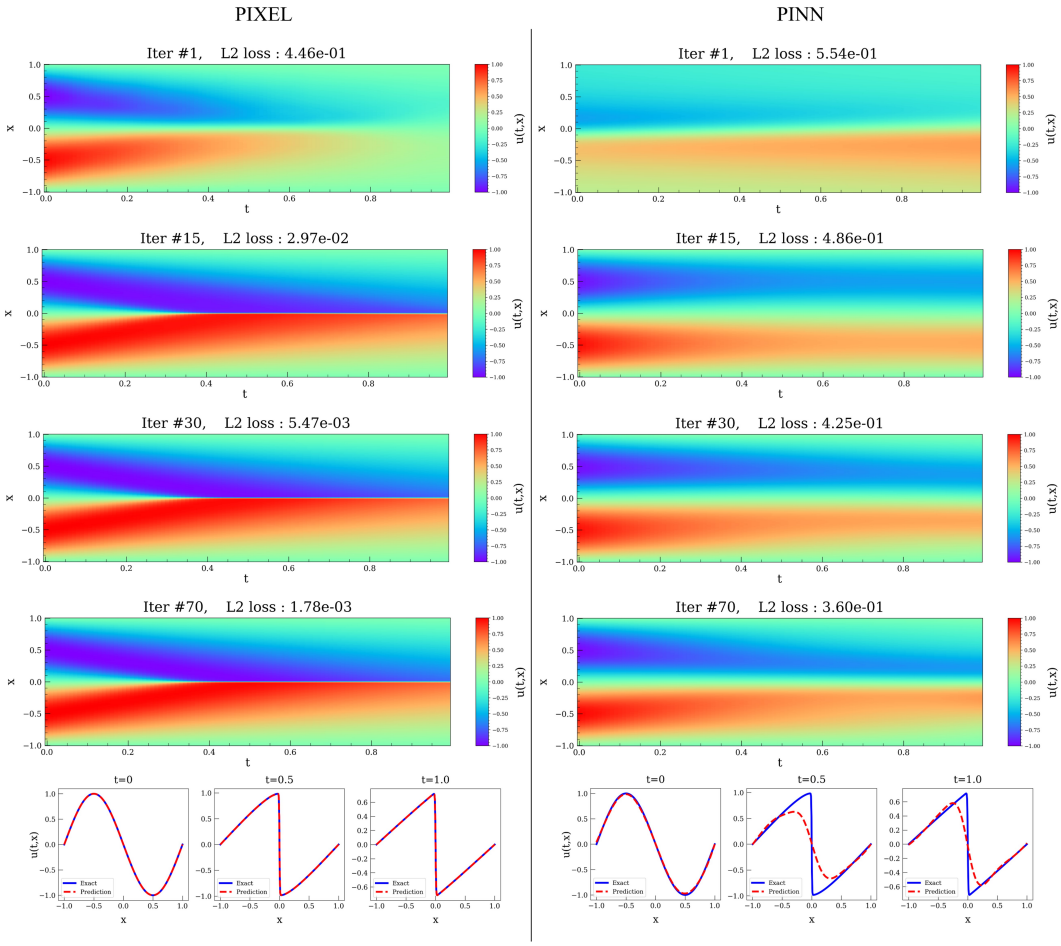

Figure 11: Forward problem of burgers equation

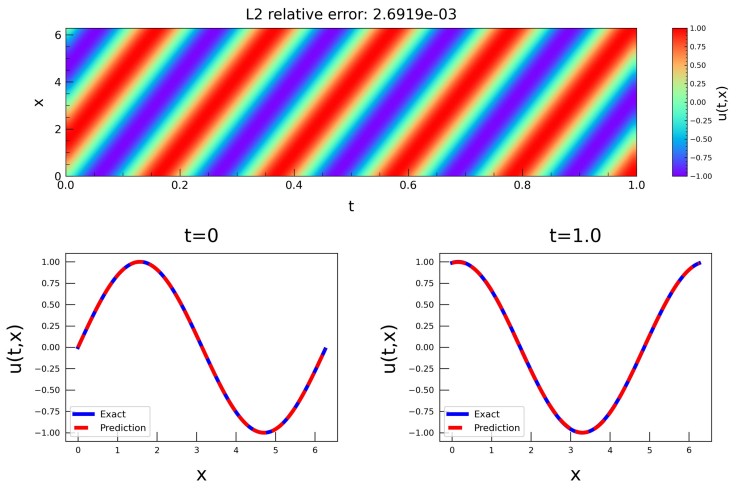

Figure 12: Convection equation results of PIXEL, at the 1500 iteration, the final relative L_2 error is 2.69e-03

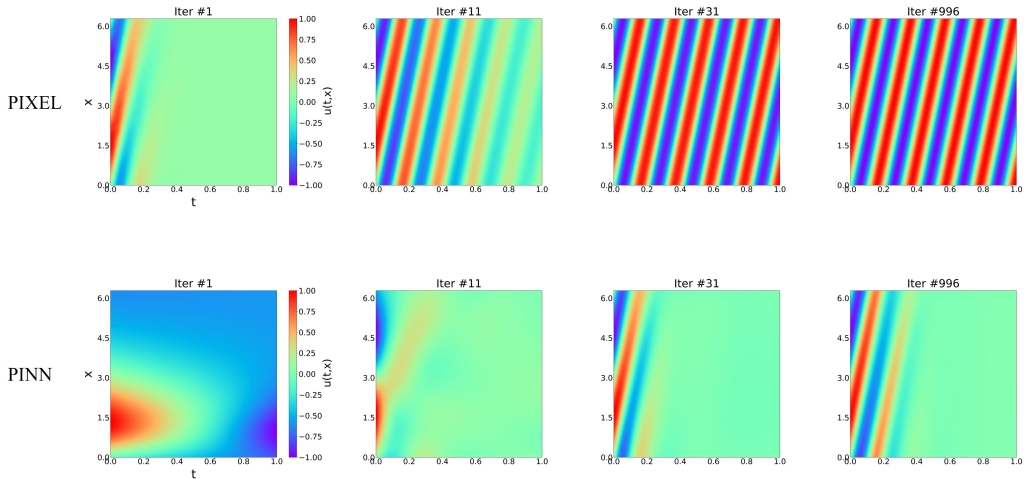

Figure 13: Forward problem of convection equation

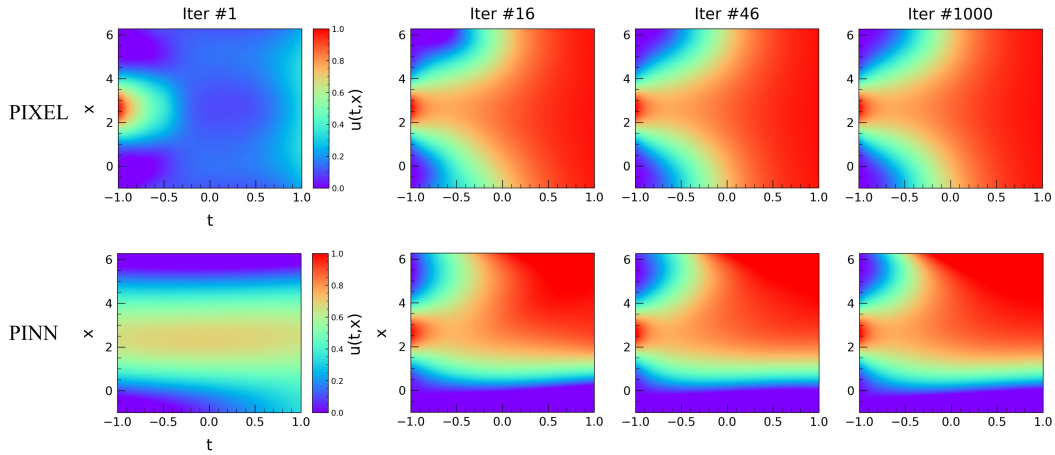

Figure 14: Forward problem of reaction-diffusion equation

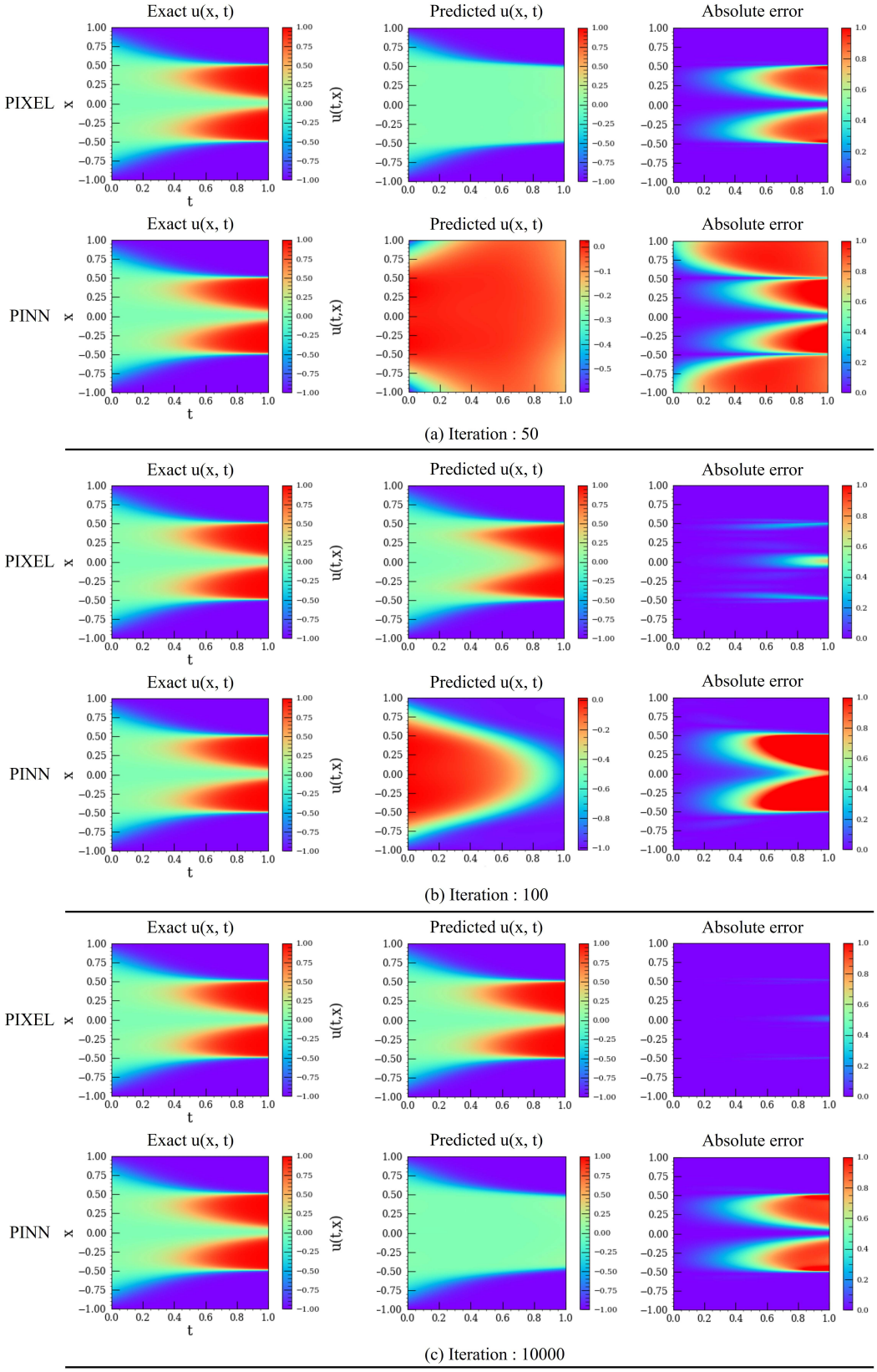

Figure 15: Forward problem of allen-cahn equation.

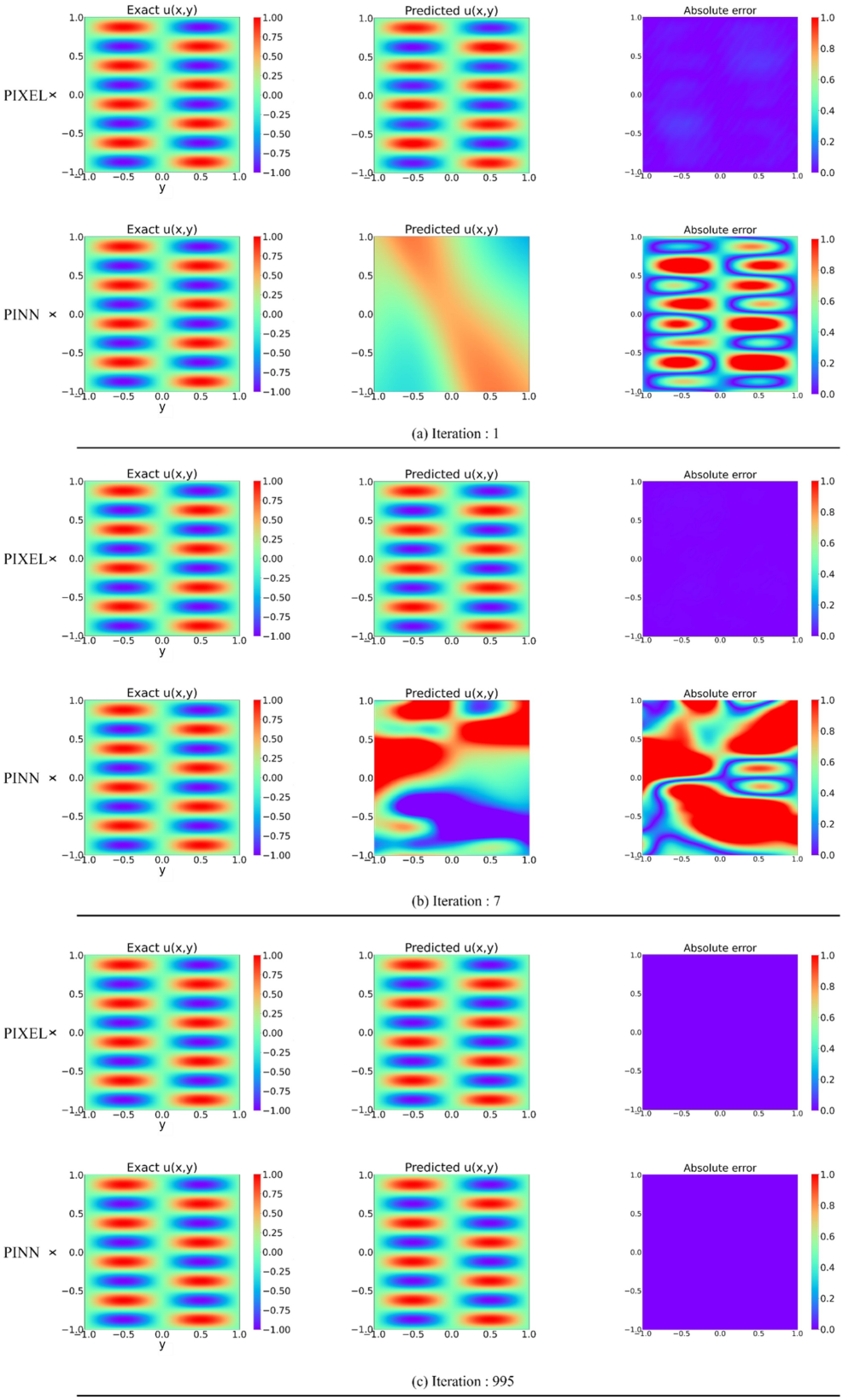

Figure 16: Forward problem of helmholtz equation

References

[1] Aditi Krishnapriyan, Amir Gholami, Shandian Zhe, Robert Kirby, and Michael W Mahoney. Characterizing possible failure modes in physics-informed neural networks. *Advances in Neural Information Processing Systems*, 34, 2021.

[2] Revanth Mattey and Susanta Ghosh. A novel sequential method to train physics informed neural networks for allen cahn and cahn hilliard equations. *Computer Methods in Applied Mechanics and Engineering*, 390:114474, 2022.

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

[4] Sifan Wang, Yujun Teng, and Paris Perdikaris. Understanding and mitigating gradient flow pathologies in physics-informed neural networks. *SIAM Journal on Scientific Computing*, 43(5):A3055–A3081, 2021.