# Multi-fidelity Physics-constrained Neural Network and Its Application in Materials Modeling

Dehao Liu, Yan Wang*

Georgia Institute of Technology

yan.wang@me.gatech.edu

http://msse.gatech.edu

IDETC/CIE 2019

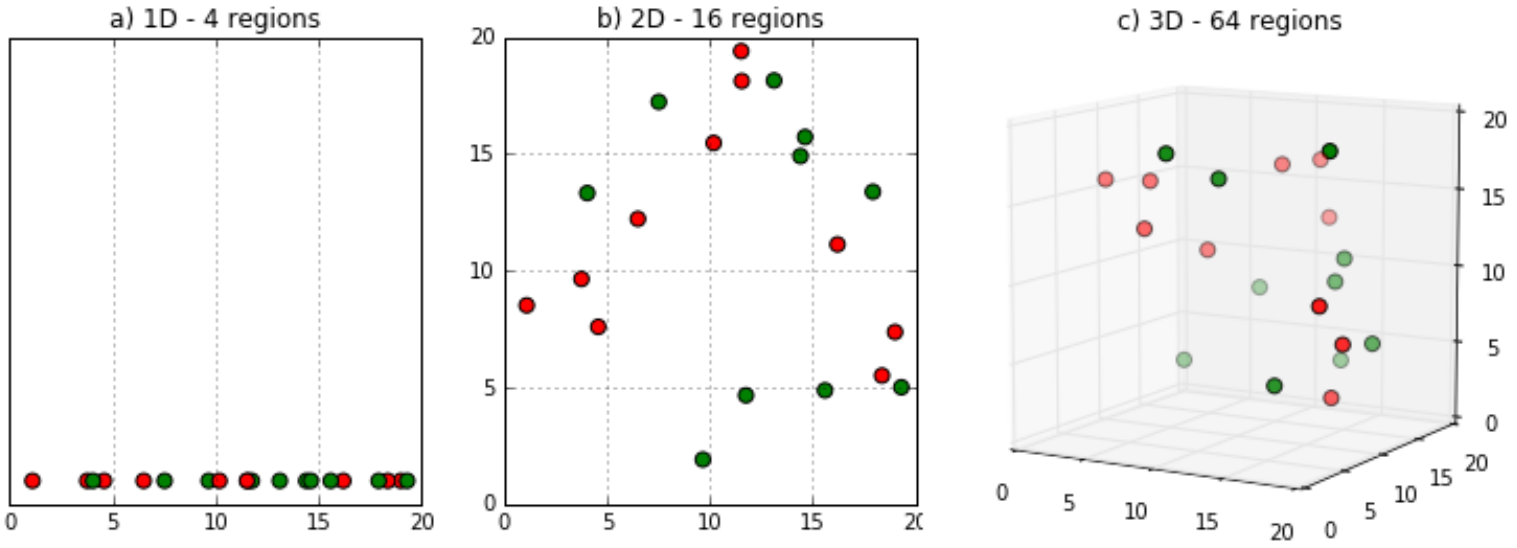
August 19, 2019, Anaheim, CA, USA

# Outline

- **Background**
- **Methodology**
  - Training of Physics-Constrained Neural Network (PCNN)
  - Construction of Multi-fidelity Physics-Constrained Neural Network (MFPCNN)
  - The Setup of Computational Scheme
- **Computational Results**
  - Heat Transfer
  - Phase Transition
  - Dendritic Growth
- **Summary**

# Background

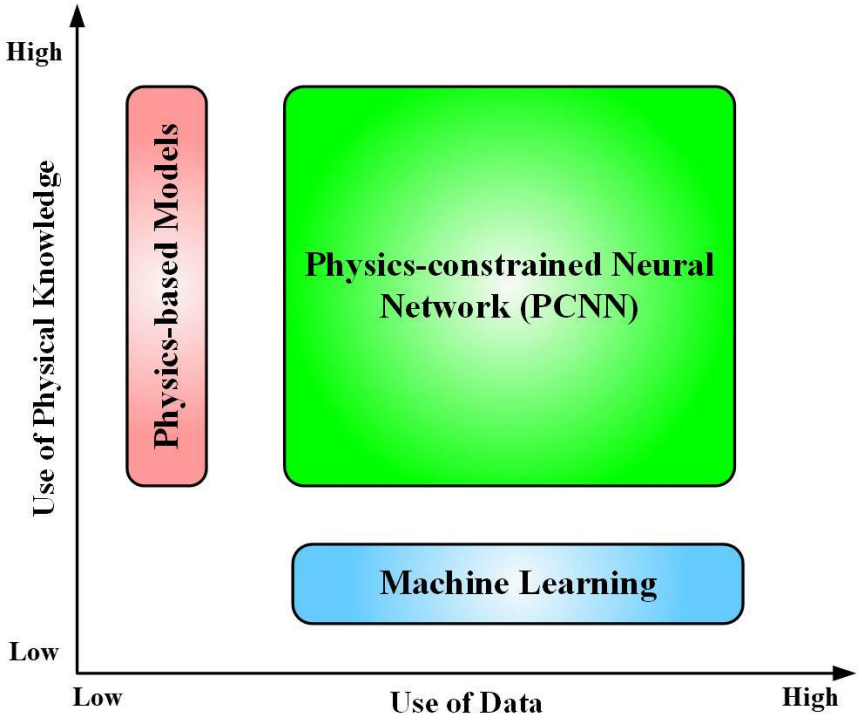
- Curse of dimensionality in training the Machine Learning tools
- Available data is scarce in some domains (e.g., materials design)
- Small data set results in a spurious relationship




(Prasad Pore, https://www.kdnuggets.com/2017/04/must-know-curse-dimensionality.html)

# Background

- Physics-based models ← scientific knowledge ← ancient data
- Machine learning models ← modern data
- How can we combine the strength of both physics-based and machine learning models?

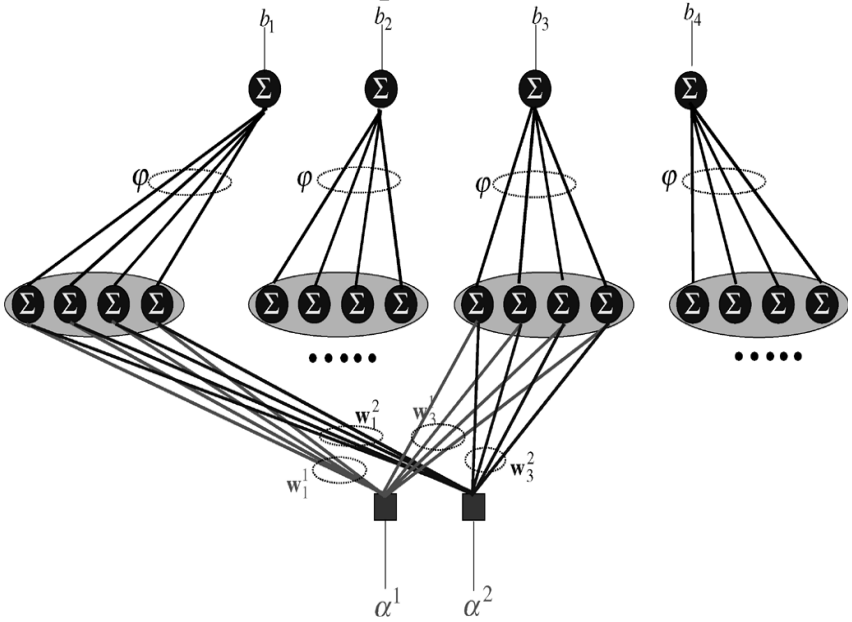


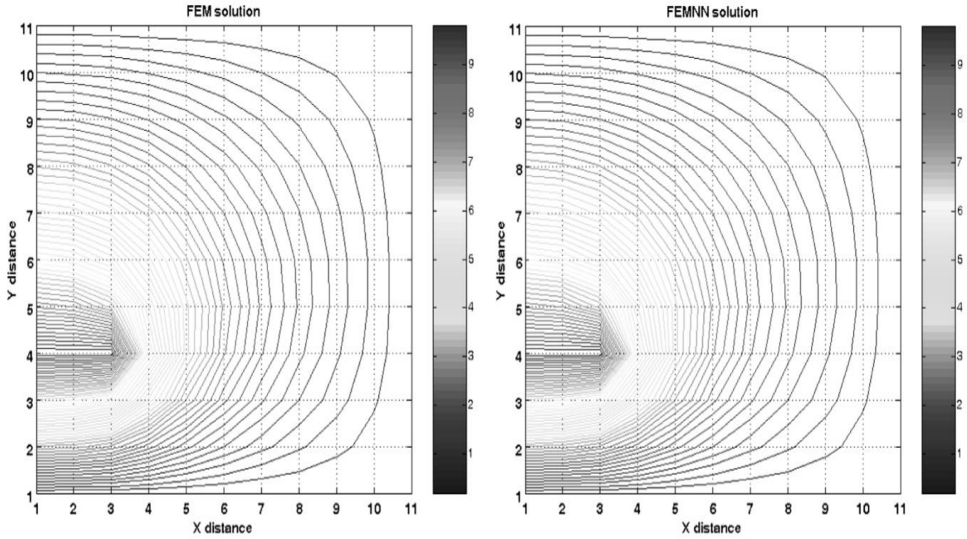
# “Physical Meanings” of Neural Networks

- Artificial Neural Networks have been used to solve:
  - Finite element simulation problems
  - Optimization problems

# Finite-Element Neural Network

- Finite-element neural network (FENN) to solve differential equations for both forward and inverse problems (Ramuhalli et al. 2005; Xu et al. 2012)
  - The weights in a FENN are indexed elements of stiffness matrix and can be computed in advance without training
  - The number of weights is related to the number of nodes, which could be large for high-dimensional problems




(Ramuhalli et al. 2005)

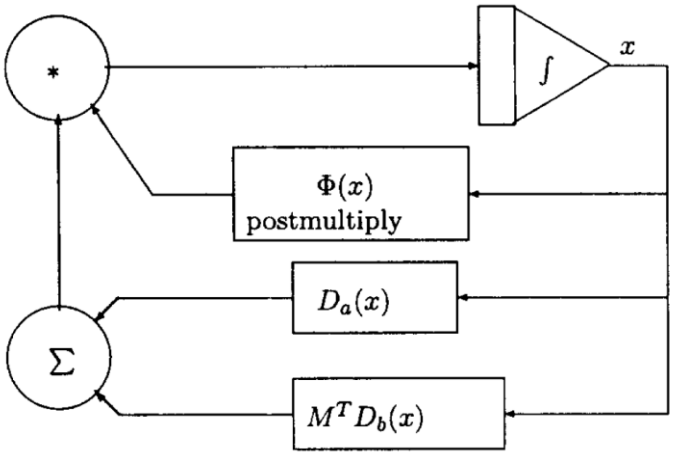
# Neural Network to Solve Optimization Problem

- Find root of $Mx + q = 0$, which also satisfies $x^T(Mx + q) = 0$, which is equivalent to solving

$$\min \frac{1}{2} x^T Q x + c^T x$$

$$\text{s.t. } Ax \geq b, x \geq 0$$

$$\text{where } M = \begin{bmatrix} Q & -A^T \\ A & 0 \end{bmatrix} \text{ and } q = \begin{bmatrix} c \\ -b \end{bmatrix}$$

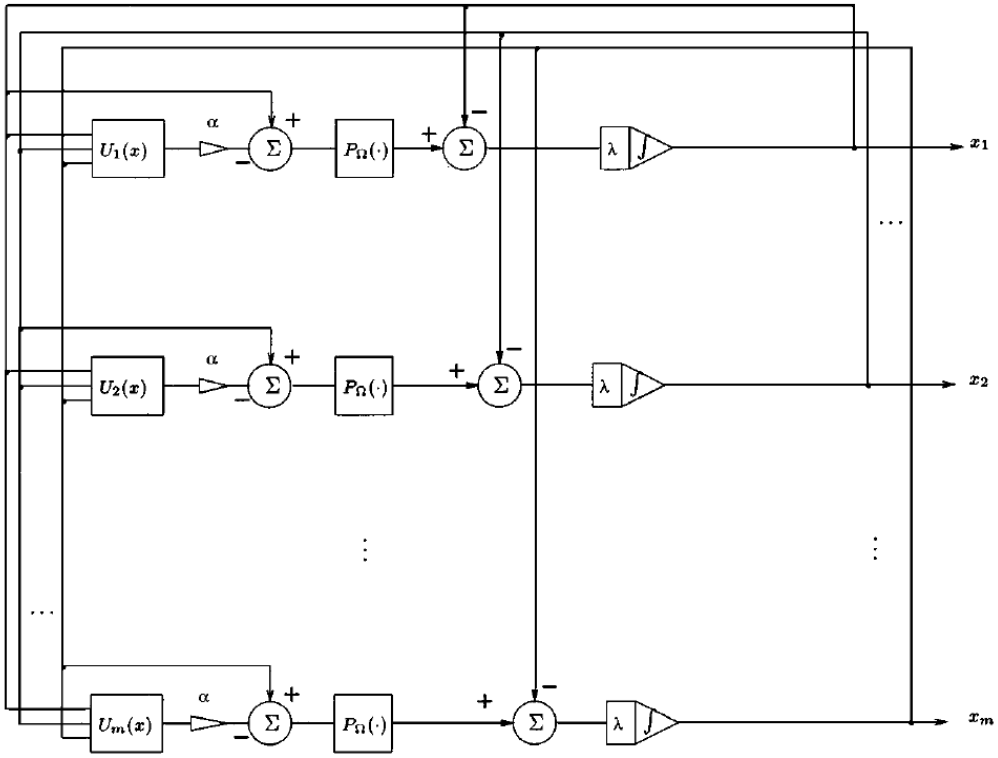



(Liang & Qi, 1999)

# Neural Network to Solve Optimization Problem

- Find root of $U(x) = Mx + q = 0$ by projection neural networks

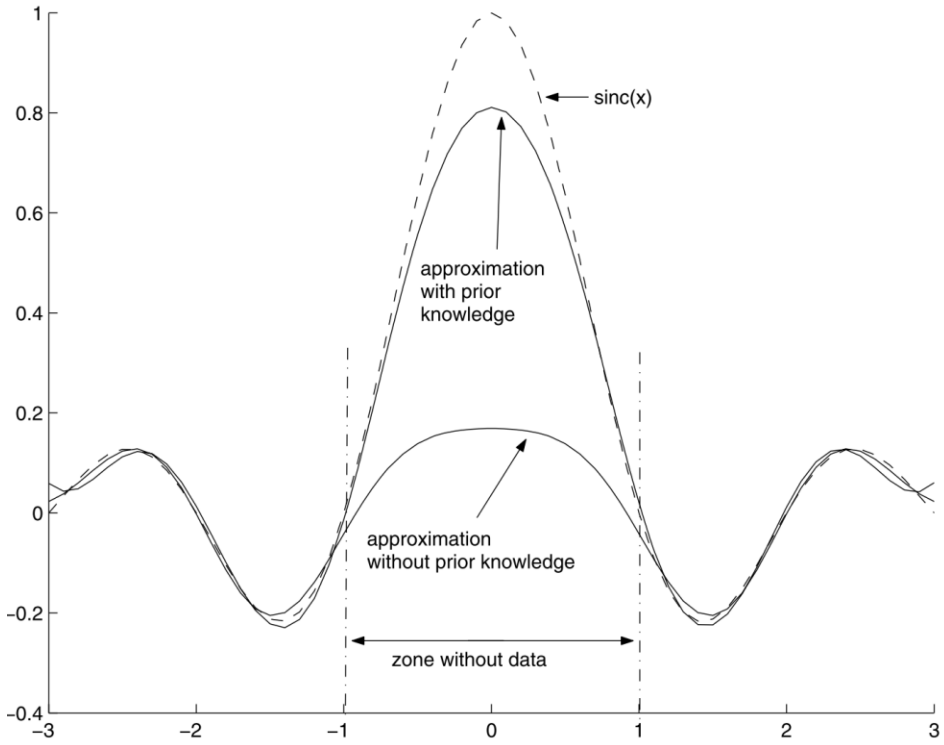
$$P_{\Omega}\big(x - (Mx + q)\big) = x$$




(Xia et al., 2002)

# Constrained ANN Training

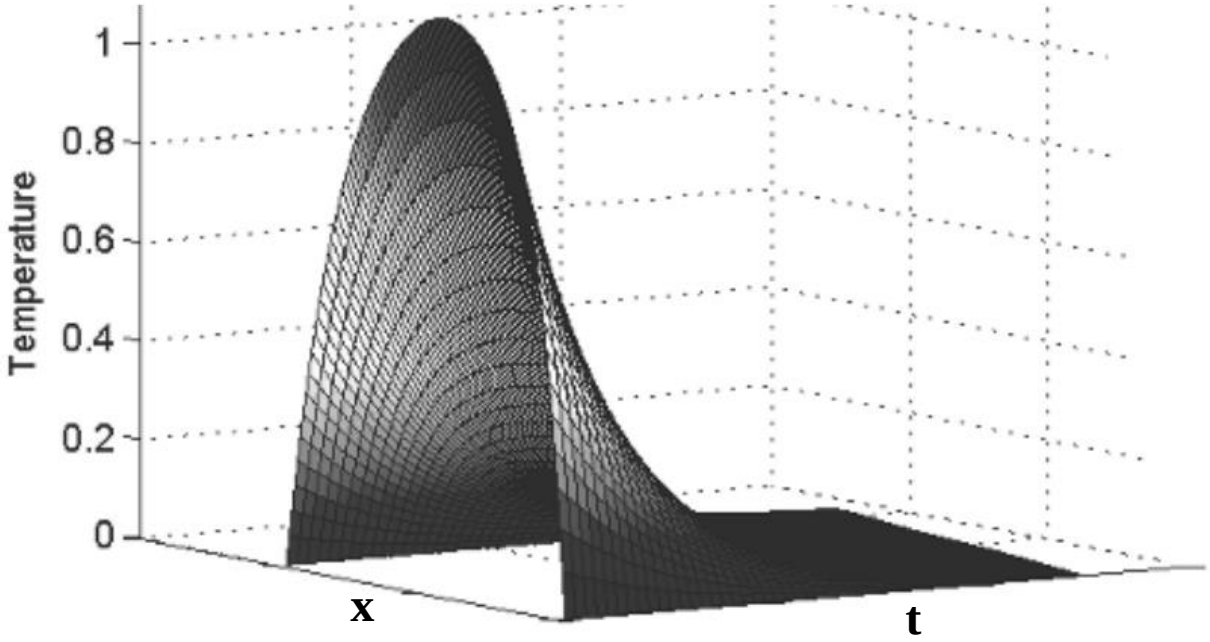
- Prior knowledge of derivatives ($f'$) is incorporated in training as constraints (Lauer & Bloch 2007; Han & Huang 2008)
  - Reduce the required amount of training data




(Lauer and Bloch, 2007)

# Physics-Constrained Neural Network

- Prior knowledge can be incorporated into neural networks in the form of partial differential equations (PDEs) (Cursi & Koscianski 2007)
  - Constraints as guidance for searching direction
  - The training efficiency has limitation in high-dimensional problems




Cursi and Koscianski, 2007

# Multi-Fidelity Physics-Constrained Neural Network

- We propose a multi-fidelity physics-constrained neural network (MFPCNN)
  - combining low-fidelity physics-constrained neural network (LFPCNN) and high-fidelity physics-constrained neural network (HFPCNN)

- The overall training cost (# of training data) is reduced

- **Methodology**
  - Demonstration of Physics-Constrained Neural Network (PCNN) with PDEs
  - Construction of Multi-fidelity Physics-Constrained Neural Network (MFPCNN)
  - The Setup of Computational Scheme

# Training of PCNN

- The prior knowledge of physics:
  - A general time-dependent PDE

$$P\left(u, \frac{\partial u}{\partial t}, \frac{\partial u}{\partial \mathbf{x}}, \frac{\partial^2 u}{\partial t^2}, \frac{\partial^2 u}{\partial \mathbf{x}^2}, \ldots\right) = f(t, \mathbf{x}),\ t \in [0, T],\ \mathbf{x} \in \Omega$$

  - Initial condition

$$I\left(u, \frac{\partial u}{\partial t}, \frac{\partial^2 u}{\partial t^2}, \ldots\right) = g(\mathbf{x}),\ t = 0,\ \mathbf{x} \in \Omega$$

  - Boundary condition

$$S\left(u, \frac{\partial u}{\partial t}, \frac{\partial u}{\partial \mathbf{x}}, \frac{\partial^2 u}{\partial t^2}, \frac{\partial^2 u}{\partial \mathbf{x}^2}, \ldots\right) = h(t, \mathbf{x}),\ t \in [0, T],\ \mathbf{x} \in \partial\Omega$$

# Training of PCNN

- The prior knowledge of physics:
  - A general time-dependent PDE

$$\mathbf{D}[u(t,\mathbf{x})] = f(t,\mathbf{x}),\ t \in [0,T],\ \mathbf{x} \in \Omega$$

  - Initial condition

$$\mathbf{\Lambda}[u(0,\mathbf{x})] = g(\mathbf{x}),\qquad t = 0,\qquad \mathbf{x} \in \Omega$$

  - Boundary condition

$$\mathbf{\Gamma}[u(t,\mathbf{x})] = h(t,\mathbf{x}),\qquad t \in [0,T],\qquad \mathbf{x} \in \partial\Omega$$

# Training of PCNN

- A multilayer perceptron (MLP) $U(t, \mathbf{x})$ is trained to approximate the true solution $u(t, \mathbf{x})$
- The weights of a PCNN can be learned by minimizing the mean square loss or cost function

$$E = \lambda_T E_T + \lambda_p E_p + \lambda_I E_I + \lambda_s E_s,$$

where

$$E_T = \frac{1}{N_T}\sum_{i=1}^{N_T}\left|U(t_i^T, \mathbf{x}_i^T) - T(t_i^T, \mathbf{x}_i^T)\right|^2, \; E_P = \frac{1}{N_P}\sum_{i=1}^{N_P}\left|\mathbf{D}\left[U(t_i^P, \mathbf{x}_i^P)\right] - f(t_i^P, \mathbf{x}_i^P)\right|^2,$$

$$E_I = \frac{1}{N_I}\sum_{i=1}^{N_I}\left|\mathbf{\Lambda}\left[U(t_i^I, \mathbf{x}_i^I)\right] - g(\mathbf{x}_i^I)\right|^2, \; E_S = \frac{1}{N_S}\sum_{i=1}^{N_S}\left|\mathbf{\Gamma}\left[U(t_i^S, \mathbf{x}_i^S)\right] - h(t_i^S, \mathbf{x}_i^S)\right|^2,$$

and $$\lambda_T + \lambda_P + \lambda_I + \lambda_s = 1.$$

- The weights are adaptive during the training

$$E = \frac{E_T^2 + E_P^2 + E_I^2 + E_S^2}{E_T + E_P + E_I + E_S}$$

# Construction of MFPCNN

- Construct MFPCNN based on
  - Low-Fidelity PCNN with the *complete* time period $t \in [0, T]$
  - High-Fidelity PCNN with a *shorter* time period $t \in [0, T_0]$ $(T_0 < T)$

- The discrepancy between the predictions of Low-Fidelity PCNN $U_L(t, \mathbf{x})$ and High-Fidelity PCNN $U_H(t, \mathbf{x})$ is

$$\delta(t, \mathbf{x}) = U_H(t, \mathbf{x}) - U_L(t, \mathbf{x}), \ t \in [0, T_0], \ \mathbf{x} \in \Omega$$

- Discrepancy artificial neural network (DANN) is constructed to predict the discrepancy $U_\delta(t, \mathbf{x})$ in the complete time period $t \in [0, T]$

# Construction of MFPCNN

- The weights of the DANN can be learned by using the discrepancy $\delta(t, \mathbf{x})$ as training data to minimize the mean squared error loss

$$E_\delta = \frac{1}{N_{total}^\delta} \sum_{i=1}^{N_{total}^\delta} |U_\delta(t_i, \mathbf{x}_i) - \delta(t_i, \mathbf{x}_i)|, \ t \in [0, T_0], \ \mathbf{x} \in \Omega$$

- It is assumed the evolution of the difference between the HFPCNN and LFPCNN in a long time period $t \in [0, T]$ can be captured by the DANN using the discrepancy $\delta(t, \mathbf{x})$ as training data in a short time period $t \in [0, T_0]$

- The prediction from MFPCNN comes from the prediction of LFPCNN and DANN

$$U_M(t, \mathbf{x}) = U_L(t, \mathbf{x}) + U_\delta(t, \mathbf{x}), \ t \in [0, T], \ \mathbf{x} \in \Omega$$

# Demonstration Setup

- Tensorflow is used
- The training data for the ANN, LFPCNN and MFPCNN comes from FEM solutions

The setup for different ML models in the heat transfer example

| ML model | Structure | Amount of training data (t×x×y) | Number of physical constraints (t×x×y) | Time period/s |
|---|---|---|---|---|
| **ANN** | 30-20-30-20 | 21×6×6 | 0 | [0, 1] |
| **PCNN1, PCNN2, PCNN3** | 30-20-30-20 | 21×6×6 | 41×11×11 | [0, 1] |

# Demonstration Setup

The setup for different ML models in the phase transition example

| ML model | Structure | Amount of training data (t×x×y) | Number of physical constraints (t×x×y) | Time period/s |
|---|---|---|---|---|
| **ANN** | 30-20-30-20 | 21×6×6 | 0 | [0, 1] |
| **LF-PCNN** | 30-20-30-20 | 21×6×6 | 21×11×11 | [0, 1] |
| **HF-PCNN1** | 30-20-30-20 | 9×21×21 | 5×11×11 | [0, 0.2] |
| **HF-PCNN2** | 30-20-30-20 | 18×21×21 | 10×11×11 | [0, 0.2], [0.8, 1] |
| **DANN1** | 5-5-5-5 | 9×26×26 | 0 | [0, 0.2] |
| **DANN2** | 10-10-10-10 | 9×26×26 | 0 | [0, 0.2] |
| **DANN3** | 5-5-5-5 | 18×26×26 | 0 | [0, 0.2], [0.8, 1] |
| **DANN4** | 10-10-10-10 | 18×26×26 | 0 | [0, 0.2], [0.8, 1] |
| **GP1** | RBF kernel | 9×26×26 | 0 | [0, 0.2] |
| **GP2** | RBF kernel | 18×26×26 | 0 | [0, 0.2], [0.8, 1] |

# Demonstration Setup

The setup for different ML models in the dendritic growth example

| ML model | Structure | Amount of training data (t×x×y) | Number of physical constraints (t×x×y) | Time period/s |
|---|---|---|---|---|
| **LF-PCNN1** | 30-20-30-20 | 2861 (random, $\Delta t = 0.1$) | 21×21×21 | [0, 1] |
| **HF-PCNN1** | 30-20-30-20 | 3901 (random, $\Delta t = 0.05$) | 5×21×21 | [0, 0.2] |
| **DANN1** | 5-5-5-5 | 9×51×51 | 0 | [0, 0.2] |
| **LF-PCNN2** | 30-20-30-20 | 2861 (random, $\Delta t = 0.1$) | 11×41×41 | [0, 1] |
| **HF-PCNN2** | 30-20-30-20 | 3901 (random, $\Delta t = 0.05$) | 3×41×41 | [0, 0.2] |
| **DANN2** | 5-5-5-5 | 9×51×51 | 0 | [0, 0.2] |

- **Computational Results**
  - Heat Transfer (Heat Equation)
  - Phase Transition (Allen-Cahn Equation)
  - Dendritic Growth (Heat and Allen-Cahn Equation)

# Heat Transfer

- Heat transfer in 2D domain with zero Neumann boundary condition is given by

$$\begin{cases} u_t - 0.01 * \left(u_{xx} + u_{yy}\right) = 0, \quad t, x, y \in [0,1], \\ u(0, x, y) = 0.5 * [sin(4\pi x) + sin(4\pi y)], \\ u_x(t, 0, y) = 0, \\ u_x(t, 1, y) = 0, \\ u_y(t, x, 0) = 0, \\ u_y(t, x, 1) = 0. \end{cases}$$

# Heat Transfer

- The total mean squared loss

$$E = \lambda_T E_T + \lambda_p E_p + \lambda_I E_I + \lambda_s E_s$$

- The training loss

$$E_T = \frac{1}{N_T}\sum_{i=1}^{N_T}\left|U\left(t_i^T, x_i^T, y_i^T\right) - T\left(t_i^T, x_i^T, y_i^T\right)\right|^2$$

- The loss of physical form

$$E_P = \frac{1}{N_P}\sum_{i=1}^{N_P}\left|U_t\left(t_i^P, x_i^P, y_i^P\right) - 0.01 * \left[U_{xx}\left(t_i^P, x_i^P, y_i^P\right) + U_{yy}\left(t_i^P, x_i^P, y_i^P\right)\right]\right|^2$$

- The loss of initial condition

$$E_I = \frac{1}{N_I}\sum_{i=1}^{N_I}\left|U\left(0, x_i^I, y_i^I\right) - 0.5 * \left[sin\left(4\pi x_i^I\right) + sin\left(4\pi y_i^I\right)\right]\right|^2$$

- The loss of boundary condition

$$E_S = \frac{1}{N_S}\sum_{i=1}^{N_S}\left[\left|U_x\left(t_i^S, 0, y_i^S\right)\right|^2 + \left|U_x\left(t_i^S, 1, y_i^S\right)\right|^2 + \left|U_y\left(t_i^S, x_i^S, 0\right)\right|^2 + \left|U_y\left(t_i^S, x_i^S, 1\right)\right|^2\right]$$

# Heat Transfer

- For PCNN1, the weights are equal and fixed

$$E = 0.25(E_T + E_p + E_I + E_s)$$

- For PCNN2, the weights are unequal and fixed

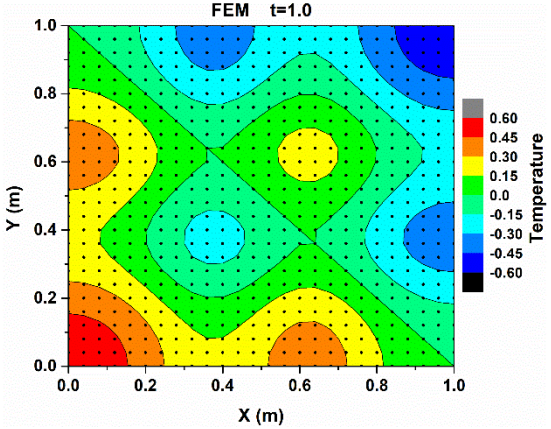
$$E = 0.125(E_T + 2E_p + 4E_I + E_s)$$

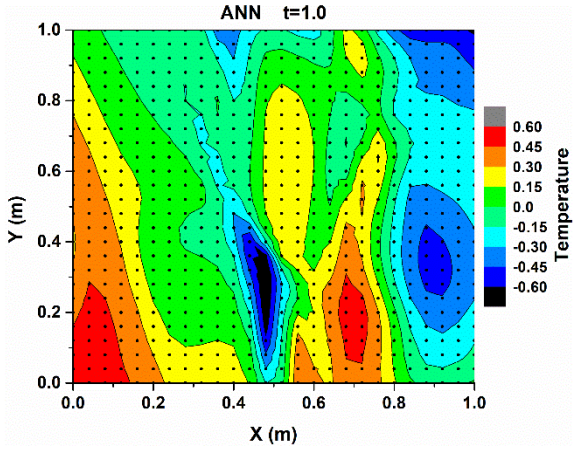
- For PCNN3, the weights are adaptive during the training

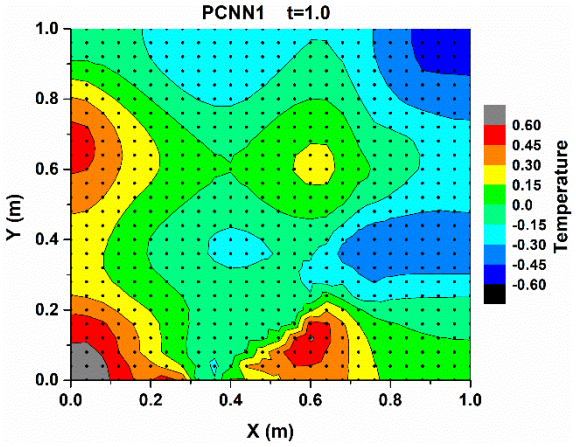
$$E = \frac{E_T^2 + E_P^2 + E_I^2 + E_S^2}{E_T + E_P + E_I + E_s}$$

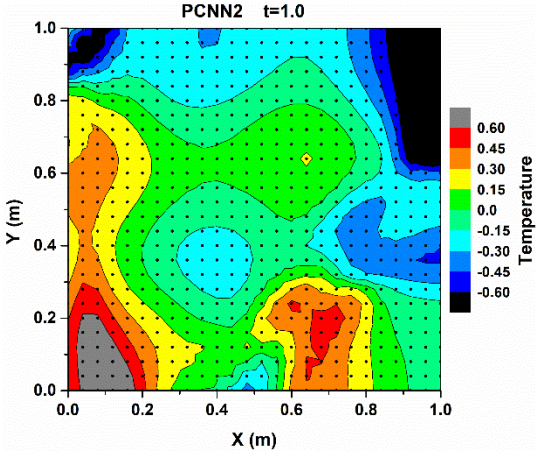
# Heat Transfer

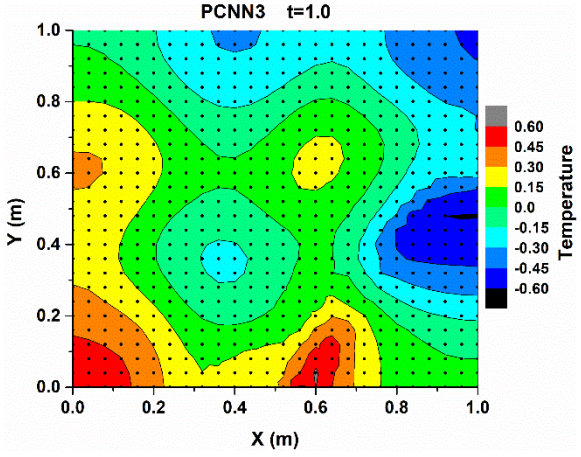
- The predictions of temperature field from different models when t=1



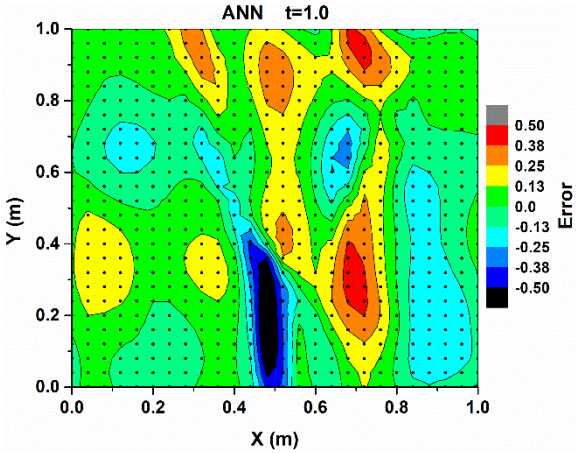



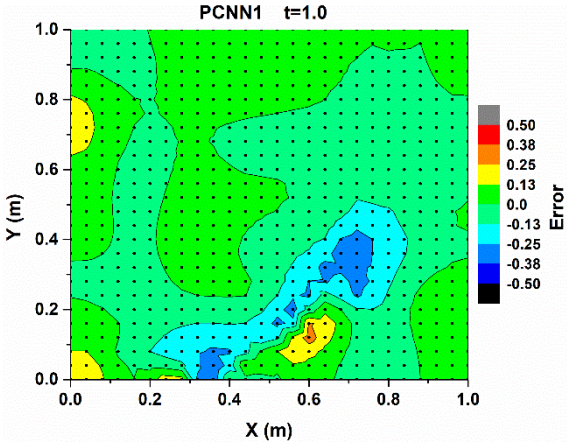
# Heat Transfer

- The errors of prediction compared to the FEM solution when t=1

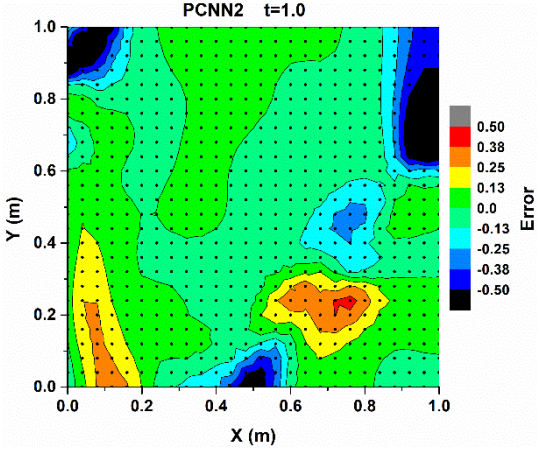
ANN



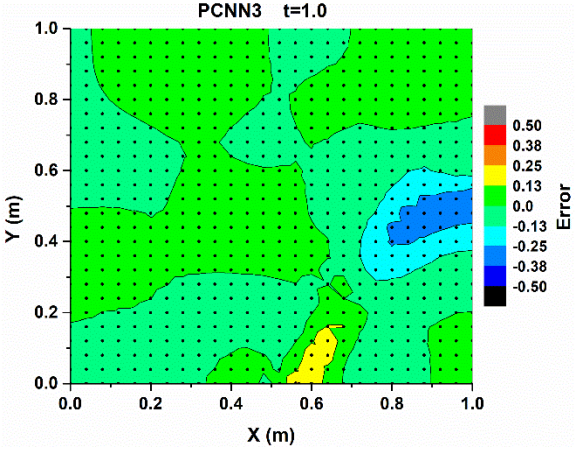

PCNN1




PCNN2

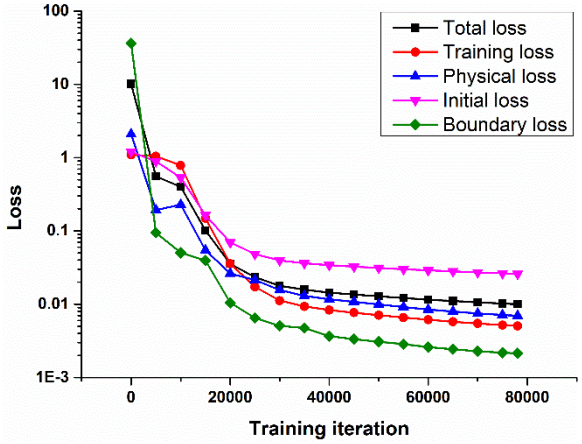



PCNN3

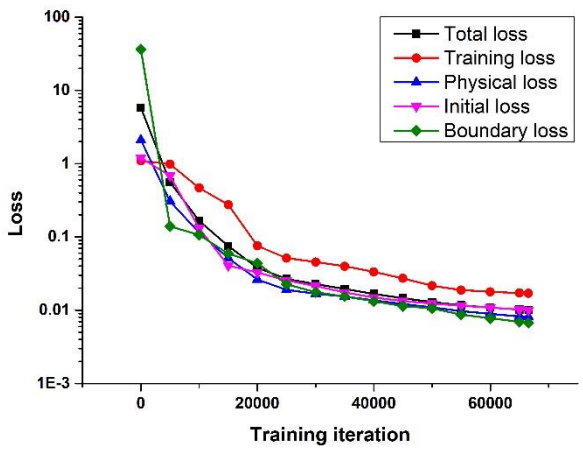


# Heat Transfer

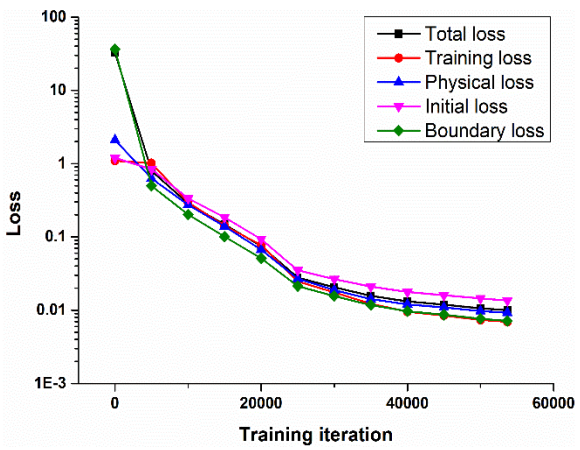
- Learning curves for different PCNNs




PCNN1




PCNN2



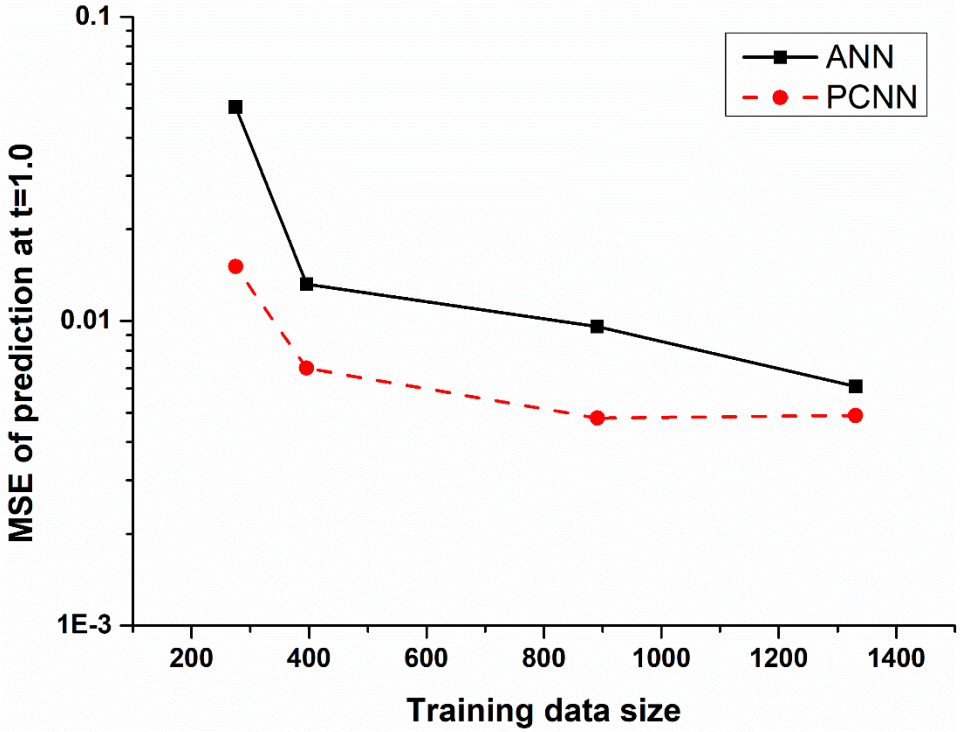

PCNN3

Quantitative comparison for different neural networks to solve the heat equation

| Neural network | Training time (second) | MSE of prediction at t = 0 | MSE of prediction at t = 1 |
|---|---|---|---|
| **ANN** | 8.66 | 0.1998 | 0.0293 |
| **PCNN1** | 1475.40 | 0.0225 | 0.0079 |
| **PCNN2** | 1259.91 | 0.0125 | 0.0350 |
| **PCNN3** | 1019.07 | 0.0139 | 0.0055 |

# Heat Transfer

- Convergence analysis for the ANN and the PCNN3



# Phase Transition

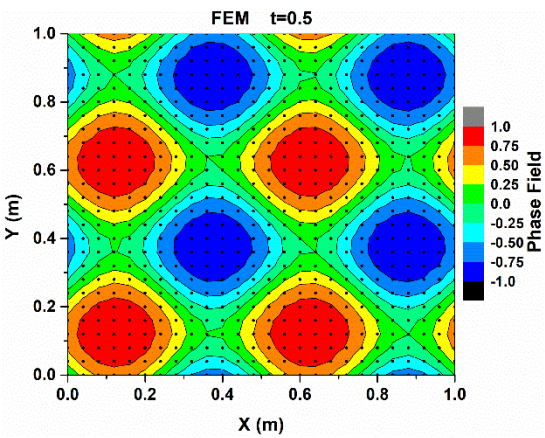
- The Allen-Chan equation is a basic model equation for the diffuse interface approach
- The Allen-Cahn equation in 2D with periodic boundary condition is given by

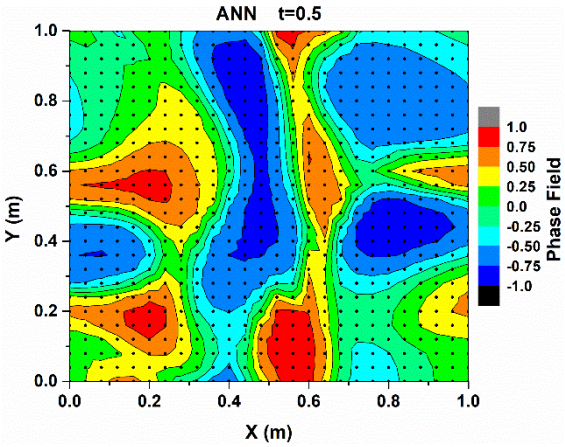
$$\begin{cases} u_t - 0.001 * \left(u_{xx} + u_{yy}\right) = u - u^3, \quad t, x, y \in [0,1], \\ u(0, x, y) = 0.5 * [sin(4\pi x) + sin(4\pi y)], \\ u(t, 0, y) = u(t, 1, y), \\ u_x(t, 0, y) = u_x(t, 1, y), \\ u(t, x, 0) = u(t, x, 1), \\ u_y(t, x, 0) = u_y(t, x, 1). \end{cases}$$

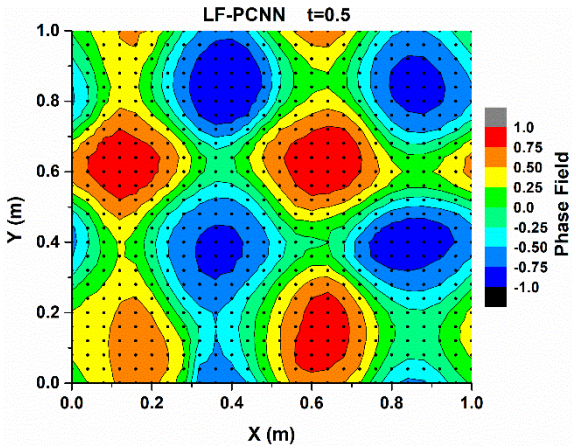
- In the total loss function, the weights are also adaptive

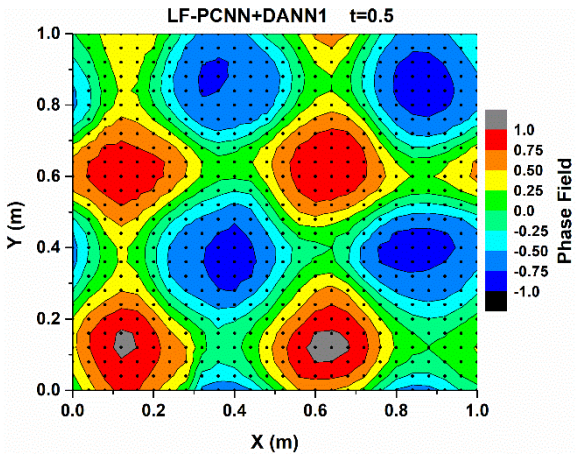
# Phase Transition

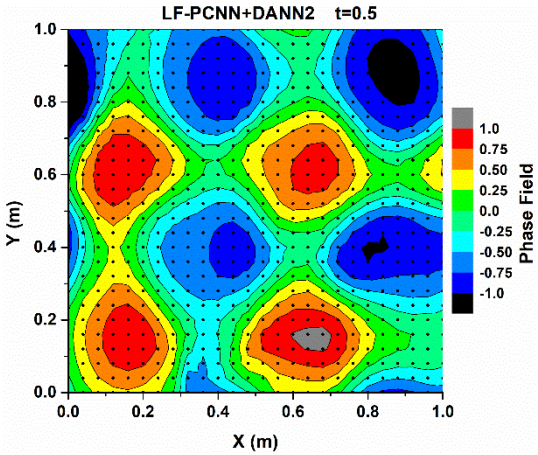
- The predictions of phase field from different models when t=0.5

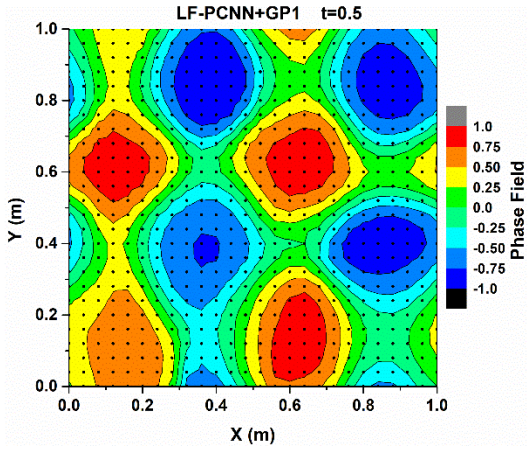


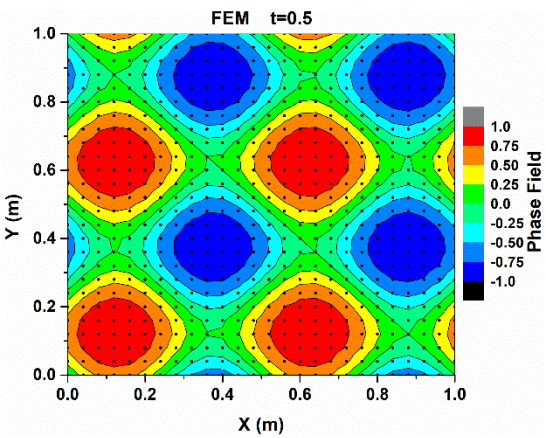



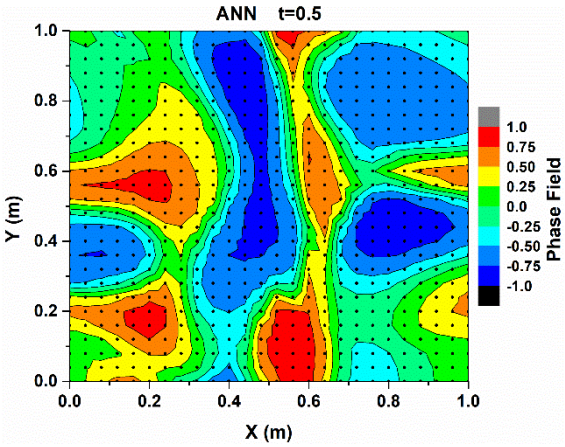



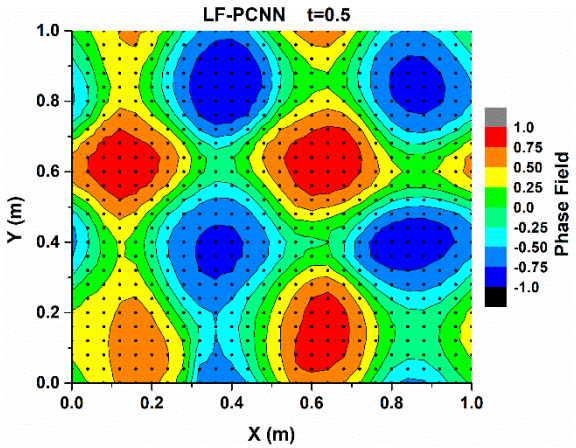



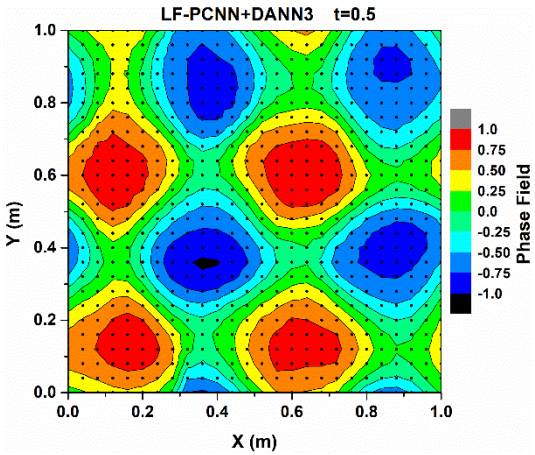
# Phase Transition

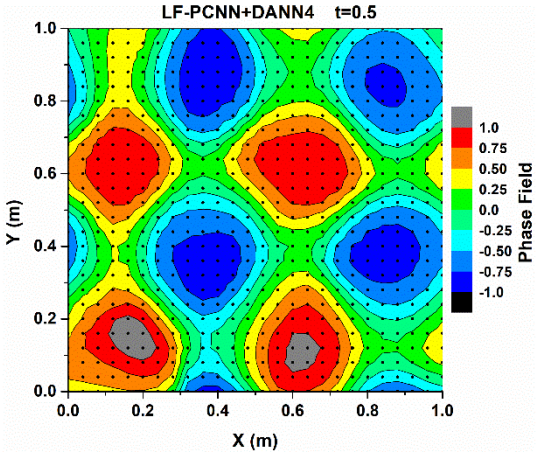
- The predictions of phase field from different models when t=0.5

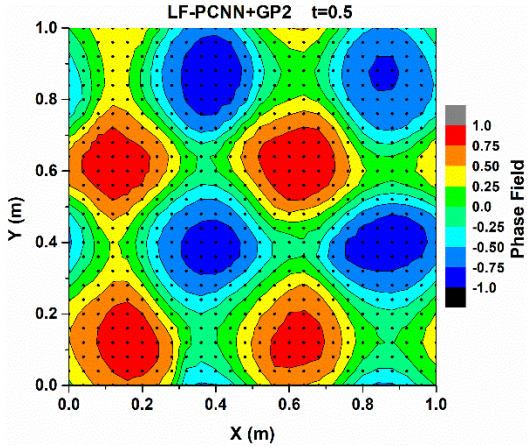


# Phase Transition

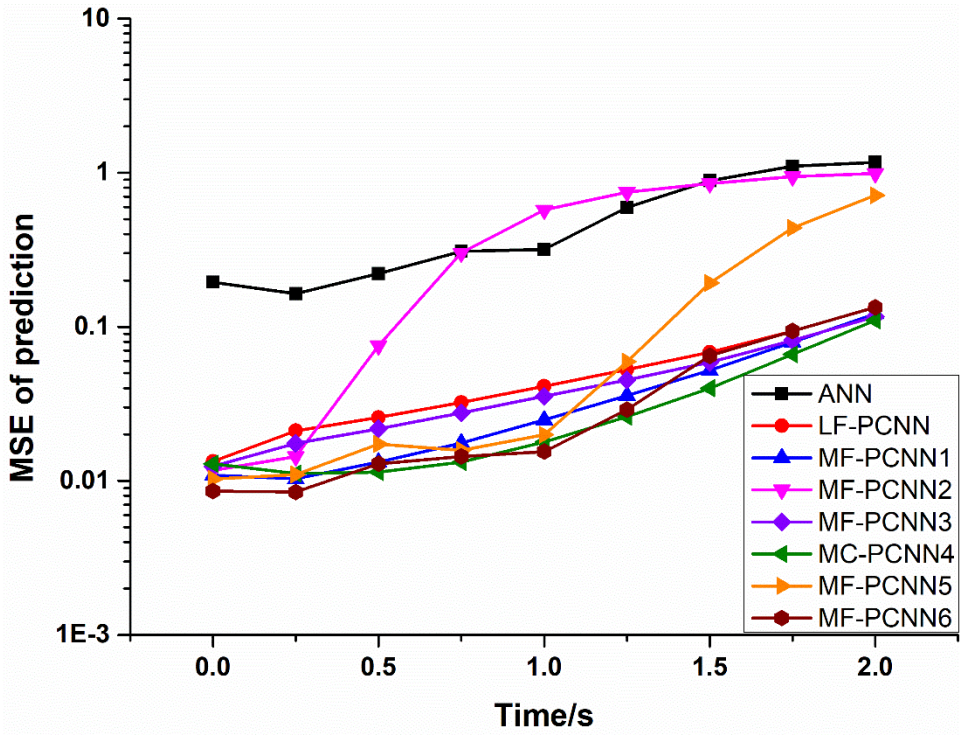
- The MSEs of prediction for most PCNNs are one order of magnitude lower than those for ANN by adding physical constraints
- Simpler structure of neural networks can reduce overfitting
- A simple ANN is more suitable than a GP to do the extrapolation of the discrepancy

Quantitative comparison between different ML models to solve the Allen-Cahn equation

| ML model | Training time (second) | MSE of prediction at t = 0.5 | MSE of prediction at t = 1.5 |
|---|---|---|---|
| **ANN** | 7.93 | 0.2215 | 0.8866 |
| **LF-PCNN** | 774.32 | 0.0258 | 0.0684 |
| **MF-PCNN1=LF-PCNN+DANN1** | 774.32+324.37+79.52=1178.21 | 0.0133 | 0.0521 |
| **MF-PCNN2=LF-PCNN+DANN2** | 774.32+324.37+25.19=1123.88 | 0.0753 | 0.8508 |
| **MF-PCNN3=LF-PCNN+GP1** | 774.32+324.37+62.58=1161.27 | 0.0218 | 0.0587 |
| **MF-PCNN4=LF-PCNN+DANN3** | 774.32+3095.68+100.38=3970.38 | 0.0114 | 0.0399 |
| **MF-PCNN5=LF-PCNN+DANN4** | 774.32+3095.68+58.01=3928.01 | 0.0173 | 0.1926 |
| **MF-PCNN6=LF-PCNN+GP2** | 774.32+3095.68+1498.41=5368.41 | 0.0129 | 0.0648 |

# Phase Transition

- The time period $t \in [1, 2]$ is outside the time range $t \in [0, 1]$ of LF training data for the LF-PCNN
- Among all ML models in this example, the MF-PCNN1 has the best performance since it has a relatively low training time and very good accuracy.



# Dendritic Growth

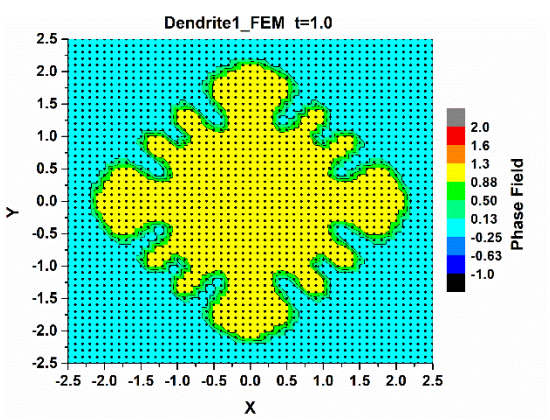
- The third example is dendritic growth during solidification, where heat transfer and phase transition are coupled with each other.
- Zero Neumann boundary conditions are applied

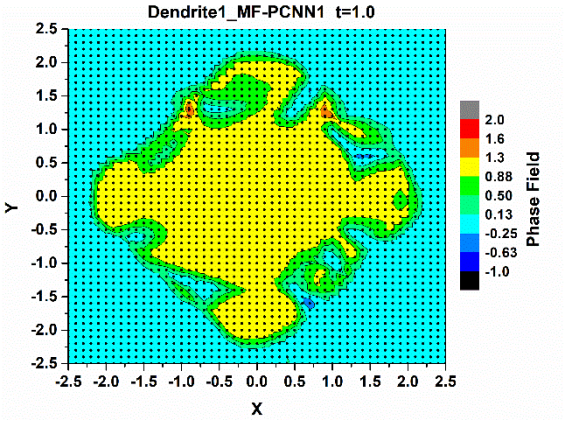
$$\begin{cases} 0.001p_t - 0.0001\left(p_{xx} + p_{yy}\right) = p(1-p)\left[p - 0.5 + \frac{0.9}{\pi}\tan^{-1}(10q_e - 10q)\right] \\ p(0,x,y) = exp\left(-\frac{x^2+y^2}{0.04}\right) \\ p_x(t,-2.5,y) = p_x(t,2.5,y) = p_y(t,x,-2.5) = p_y(t,x,2.5) = 0 \\ 0.001\left(q_t - q_{xx} + q_{yy}\right) = 0.001Kp_t \\ q(0,x,y) = 0 \\ q_x(t,-2.5,y) = q_x(t,2.5,y) = q_y(t,x,-2.5) = q_y(t,x,2.5) = 0 \\ t \in [0,1],\ x,y \in [-2.5,2.5] \end{cases}$$

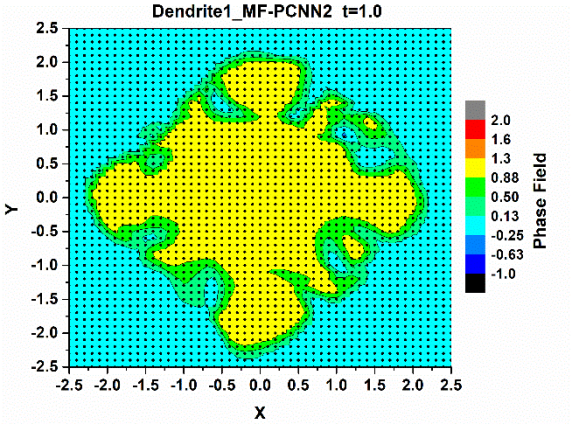
- $p$ is the phase field and $q$ is the temperature field. The liquidus temperature $q_e$ and latent heat $K$ are materials dependent
- In the total loss function, the weights are also adaptive

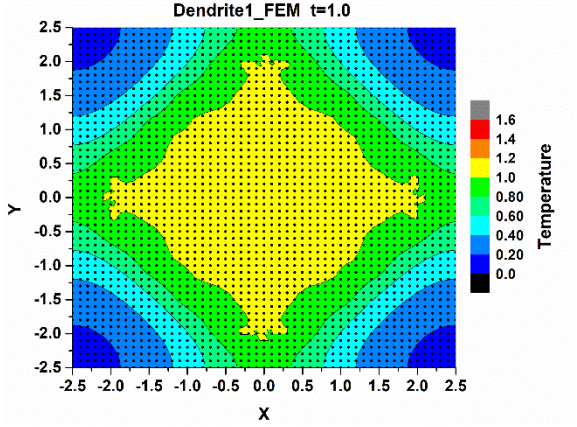
# Dendritic Growth

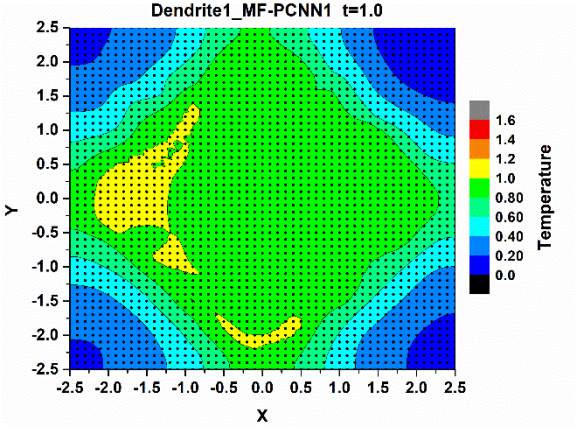
- The predicted phase fields and temperature fields from different models for the first material option ($q_e = 1$; $K = 2$) at $t$ = 1.0.

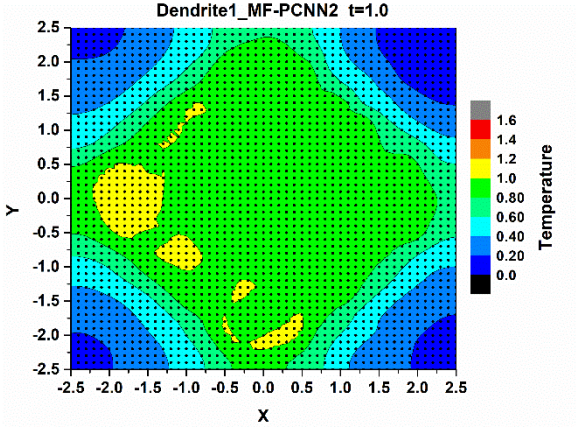


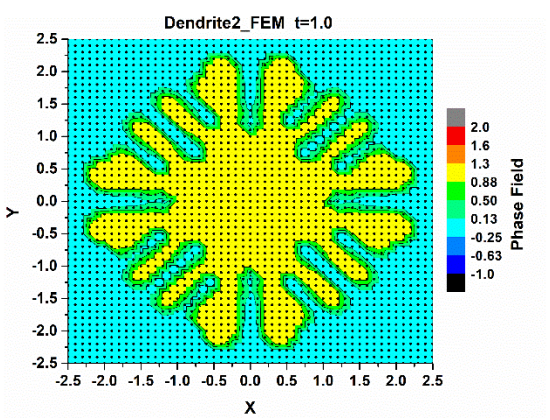



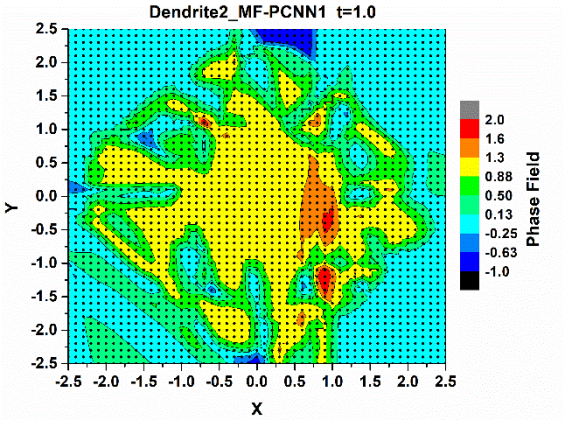



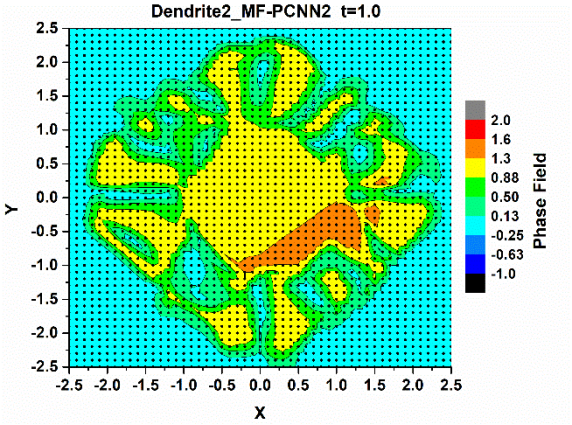



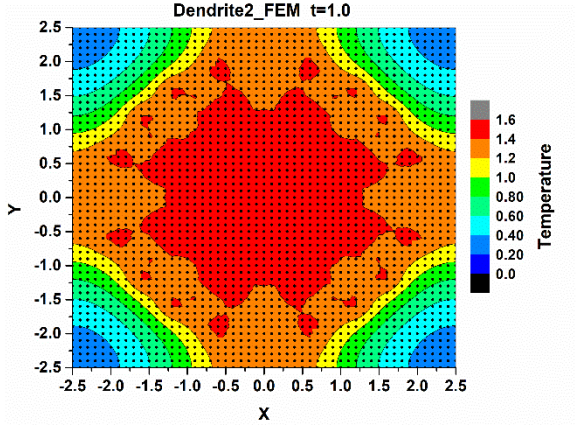
# Dendritic Growth

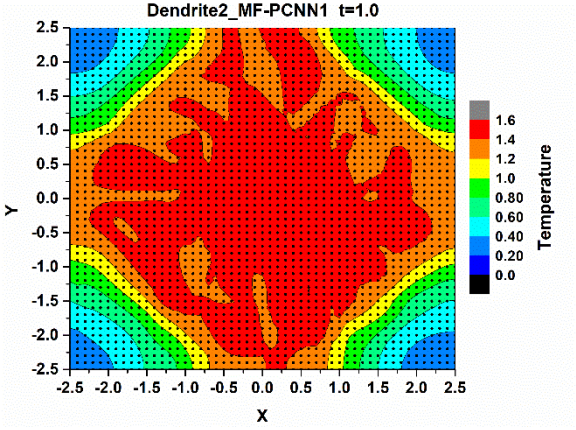
- The predicted phase fields and temperature fields from different models for the second material option ($q_e = 1.4;\ K = 2.8$) at $t$ = 1.0.

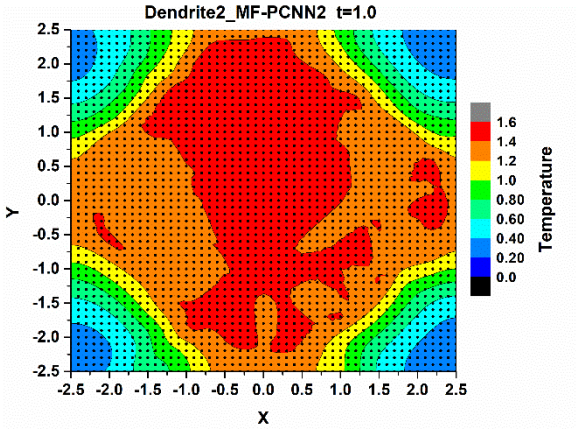


# Summary

- A novel MFPCNN is proposed to reduce the required amount of training data, where physical knowledge is applied to constrain neural networks
- The PCNN is effective for these two different types of PDEs with different boundary conditions, which demonstrate its generalization
- By using the MFPCNN, the desired accuracy can be attained without turning to the expensive HFPCNN for the whole time period
- The proposed method should not be regarded as the complete replacement of classical numerical simulation methods for solving partial differential equations (e.g., finite elements, spectral methods, etc.)

köszönöm תודה dĕkuji

mahalo 고맙습니다

# Thanks!

merci 谢谢 danke

Ευχαριστώ شكرا

どうもありがとう gracias