# Physics-Informed Machine Learning: A survey on Problems, Methods and Applications

Zhongkai Hao, Songming Liu, Yichi Zhang, Chengyang Ying, Yao Feng
Hang Su, Jun Zhu

# Content

- Introduction—why and how PIML
- Problem Formulation
- Neural Simulation
  - Neural Solver
  - Neural Operator
  - Theory
- Inverse Problem (Inverse Design)
- Computer Vision & Reinforcement Learning

# Introduction

- Limitations of Statistical ML models
  - Not robust
  - Lack interpretability
  - Violate physical constraints
- Current statistic learning are not aware of the internal physical mechanism that generates the data

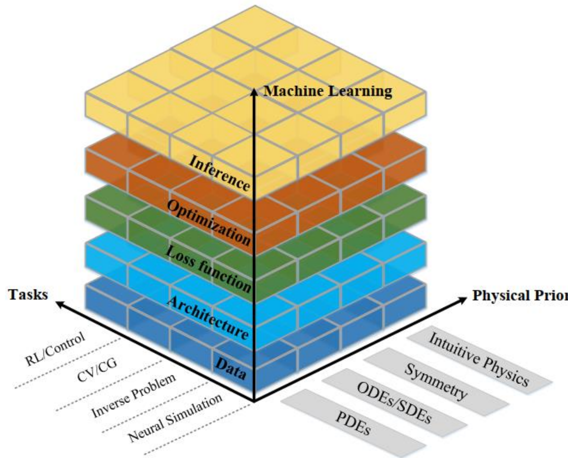
# Introduction

- Concept of Machine Learning
  - *build models that leverage empirical data to improve performance on some set of tasks*

- Concept of Physics-Informed Machine Learning
  - *build models that leverage empirical data* ***and available physical prior knowledge*** *to improve performance on a set of tasks that* ***involve a physical mechanism***

# Introduction

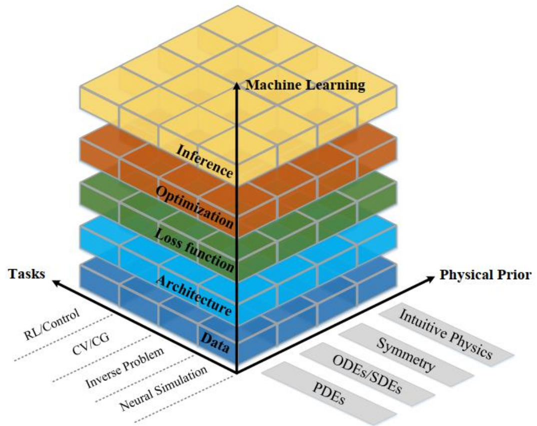
- Representation of physical prior
  - PDEs/ODEs/SDEs
  - Symmetry
  - Intuitive physics
- How to encode physical prior
  - Data
  - Architecture
  - Loss functions
  - Optimizer
  - Inference



(a)

# Introduction

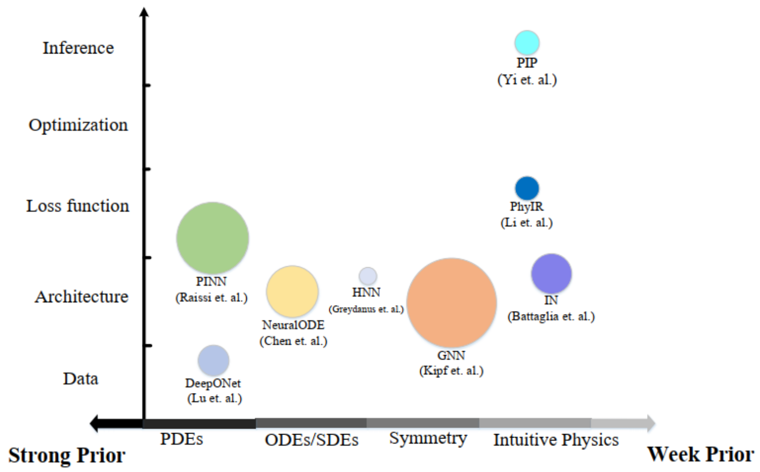
- Tasks of PIML
  - Neural Simulation
    - Neural Solver (PINN…)
    - Neural Operator (DeepONet, FNO…)
  - Inverse Problem
  - Computer Vision/Computer Graphics
  - Reinforcement Learning/Control

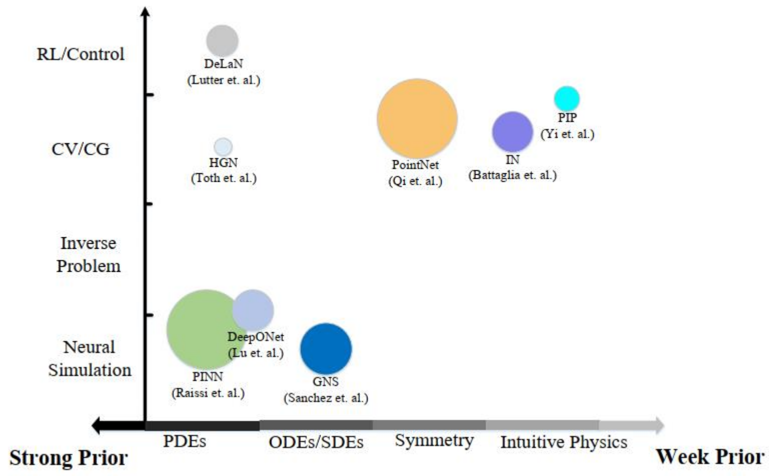


(a)

# Introduction

- Representative works
  - Methods for incorporating physical prior (left)
  - Works for solving different tasks (right)



# Problem Formulation

- Problem formulation
  - View ML as an optimization problem

$$\min_{f \in \mathcal{H}} \mathcal{L}(f; \mathcal{D}) + \Omega(f; \mathcal{D}).$$

- The root of physical prior is data is physical

$$\mathcal{F}(\mathcal{D}) = 0,$$

  - Data $\mathcal{D}_p = P(\mathcal{D})$
  - Architecture $f \in \mathcal{H}_p \subseteq \mathcal{H}.$
  - Loss/Reg $\mathcal{L}_p(f; \mathcal{D})$ or $\Omega_p(f; \mathcal{D})$
  - Optimization $OPT_p$
  - Inference $g_p(x, f(\boldsymbol{x}))$

# Neural Simulation

- Notations and Problem Formulation
  - PDEs

$$\mathcal{F}\left(u, \frac{\partial u}{\partial x_1}, \cdots \frac{\partial u}{\partial x_d}, \frac{\partial^2 u}{\partial x_1^2}, \frac{\partial^2 u}{\partial x_1 \partial x_2}, \cdots \frac{\partial^2 u}{\partial x_d^2}, \ldots; \theta\right)(x_i, t) = 0,$$

$$\mathcal{I}\left(u, \frac{\partial u}{\partial x_1}, \cdots \frac{\partial u}{\partial x_d}, \frac{\partial^2 u}{\partial x_1^2}, \frac{\partial^2 u}{\partial x_1 \partial x_2}, \cdots \frac{\partial^2 u}{\partial x_d^2}, \ldots; \theta\right)(x_i, t_0) = 0,$$

$$\mathcal{B}\left(u, \frac{\partial u}{\partial x_1}, \cdots \frac{\partial u}{\partial x_d}, \frac{\partial^2 u}{\partial x_1^2}, \frac{\partial^2 u}{\partial x_1 \partial x_2}, \cdots \frac{\partial^2 u}{\partial x_d^2}, \ldots; \theta\right)(x_i, t) = 0.$$

  - Neural Solver

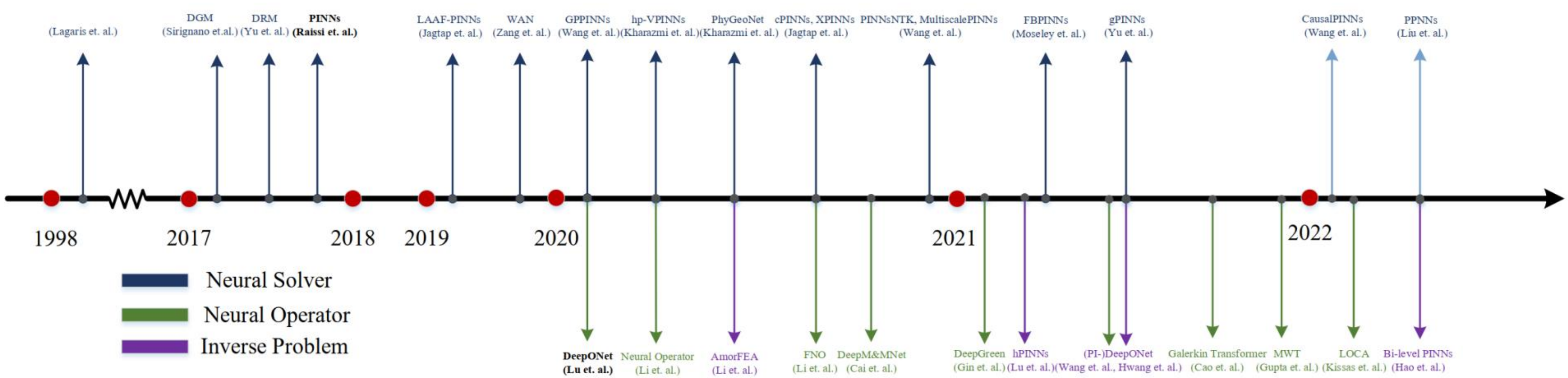
$$\min_{w \in W} \|u_w(\boldsymbol{x}) - \tilde{u}(\boldsymbol{x})\|,$$

  - Neural Operator

$$\min_{w \in W} \|G_w(\theta)(\boldsymbol{x}) - \tilde{G}(\theta)(\boldsymbol{x})\|,$$

# Neural Simulation

- Chronological overview
  - Neural Solver: DGM[1]/DRM[2]/PINN[3]...
  - Neural Operator: DeepONet[4]/FNO[5]…
  - Inverse Design: PINNs/DeepONets/AmorFEA[6]…

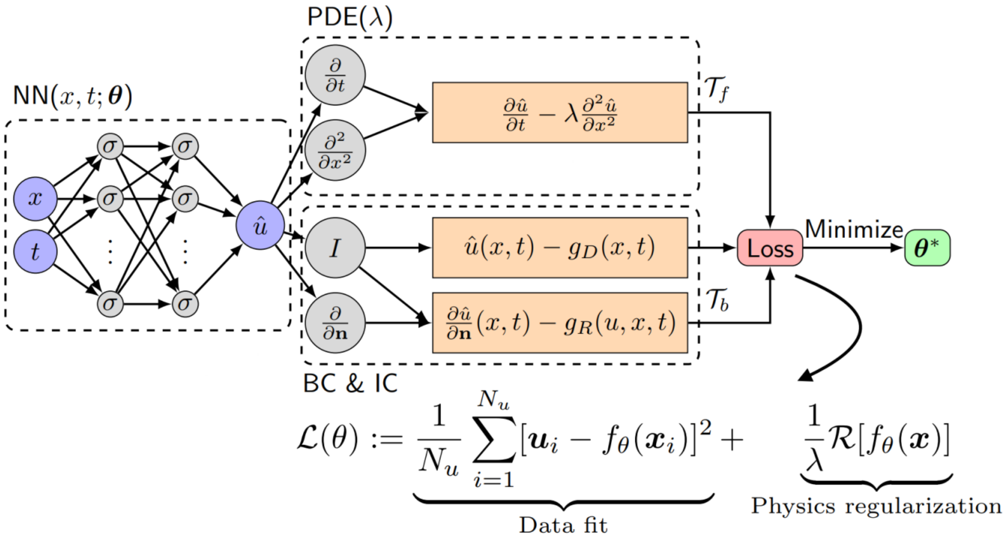


# Neural Solver

- Basic PINNs
  - Parametrize solution with NNs and optimize following loss

$$\mathcal{L} = \frac{\lambda_r}{|\Omega|}\int_{\Omega}\|\mathcal{F}(u_w;\theta)(\boldsymbol{x})\|^2 \mathrm{d}\boldsymbol{x} + \frac{\lambda_i}{|\Omega_0|}\int_{\Omega_0}\|\mathcal{I}(u_w;\theta)(\boldsymbol{x})\|^2 \mathrm{d}\boldsymbol{x}$$
$$+\frac{\lambda_b}{|\partial\Omega|}\int_{\partial\Omega}\|\mathcal{B}(u_w;\theta)(\boldsymbol{x})\|^2 \mathrm{d}\boldsymbol{x} + \frac{\lambda_d}{N}\sum_{i=1}^{N}\|u_w(\boldsymbol{x}_i) - u(\boldsymbol{x}_i)\|^2,$$

  - Graphical illustration of PINNs



# Neural Solver

- Variants of PINNs
  - Loss Reweighting and Data Resampling
  - Novel optimization targets
    - Numerical Differentiation
    - Variational Formulation
    - Regularization terms
  - Novel Neural Architectures
    - Activation functions
    - Feature preprocessing (embedding)
    - Boundary Encoding
    - Sequential Architecture/Convolutional Architecture
    - Domain Decomposition

# Neural Solver

- Loss Reweighting
  - Balance learning rates by gradient norms [7]

$$\hat{\lambda}_i = \frac{\max\{\nabla_w \mathcal{L}_r(w_n)\}}{|\overline{\nabla_w \mathcal{L}_i(w_n)}|}.$$

$$\lambda_i \leftarrow (1-\alpha)\lambda_i + \alpha\hat{\lambda}_i,$$

  - NTK reweighting
  - Variance reweighting
  - …
- Data Sampling
  - Sample points with higher error with IS [8]

$$\mathcal{L}_r = \mathbb{E}_{\boldsymbol{x}\sim q}\left[\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}\|\mathcal{F}(u)(\boldsymbol{x})\|^2\right]$$

$$q(\boldsymbol{x}_i) = \frac{\|\nabla_w l_r(w, \boldsymbol{x}_i)\|}{\sum_j \|\nabla_w l_r(w, \boldsymbol{x}_j)\|} \approx \frac{l_r(w, \boldsymbol{x}_i)}{\sum_j l_r(w, \boldsymbol{x}_j)}$$

# Neural Solver

- Novel Optimization Targets--Variational Formulation
  - For the following problem

$$\begin{aligned}\Delta u &= f(\boldsymbol{x}), x \in \Omega, \\ \frac{\partial u}{\partial n} &= 0, x \in \partial\Omega.\end{aligned}$$

  - PINN optimizes

$$\mathcal{L}(w) = \frac{\lambda_r}{|\Omega|}\int_\Omega \|\Delta u_w - f(\boldsymbol{x})\|^2 \mathrm{d}\boldsymbol{x} + \frac{\lambda_b}{|\partial\Omega|}\int_{\partial\Omega}\left\|\frac{\partial u_w}{\partial n}\right\|^2 \mathrm{d}\boldsymbol{x}$$

  - Deep Ritz Method [2] optimizes

$$\mathcal{J}(w) = \int_\Omega \left(\frac{1}{2}|\nabla u_w(\boldsymbol{x})|^2 - f(\boldsymbol{x})u_w(\boldsymbol{x})\right)\mathrm{d}\boldsymbol{x}.$$

  - VPINNs [9] choose a set of test functions

$$\mathcal{J}(w) = \frac{1}{K}\sum_{k=1}^{K}|\langle \mathcal{F}(u_w), v\rangle_\Omega|^2 + \lambda_b \frac{1}{N_b}\sum_{i=1}^{N_b}|u_w(\boldsymbol{x}_i) - g(\boldsymbol{x}_i)|^2.$$

# Neural Solver

- Novel Optimization Targets—Regularization terms
  - Gradient-enhanced PINNs [10]
  - For PDEs, we penalize itself as well as its derivatives

$$f\left(\mathbf{x}; \frac{\partial u}{\partial x_1}, \ldots, \frac{\partial u}{\partial x_d}; \frac{\partial^2 u}{\partial x_1 \partial x_1}, \ldots, \frac{\partial^2 u}{\partial x_1 \partial x_d}; \ldots; \boldsymbol{\lambda}\right) = 0, \quad \mathbf{x} = (x_1, \cdots, x_d) \in \Omega,$$

  - Loss function of gPINNs

$$\mathcal{L} = w_f \mathcal{L}_f + w_b \mathcal{L}_b + w_i \mathcal{L}_i + \sum_{i=1}^{d} w_{g_i} \mathcal{L}_{g_i}\left(\boldsymbol{\theta}; \mathcal{T}_{g_i}\right),$$

$$\mathcal{L}_{g_i}\left(\boldsymbol{\theta}; \mathcal{T}_{g_i}\right) = \frac{1}{\left|\mathcal{T}_{g_i}\right|} \sum_{\mathbf{x} \in \mathcal{T}_{g_i}} \left|\frac{\partial f}{\partial x_i}\right|^2$$

# Neural Solver

- Novel Architectures
  - Boundary encoding [11] (use hard boundary constraints)

$$\begin{aligned} \mathcal{F}(u)(\boldsymbol{x}) &= 0, x \in \Omega, \\ u(\boldsymbol{x}) &= g(\boldsymbol{x}), x \in \partial\Omega. \end{aligned} \quad \longrightarrow \quad \begin{gathered} u(\boldsymbol{x}) = v(\boldsymbol{x}) + D(\boldsymbol{x})y(\boldsymbol{x}). \\ D(\boldsymbol{x}) = 0, x \in \partial\Omega. \end{gathered}$$

  - Feature Embedding [12] (e.g. Fourier features)

$$\begin{aligned} \gamma(\boldsymbol{x}) = (&\sin(2\pi \boldsymbol{b}_1^T \cdot \boldsymbol{x}), \cos(2\pi \boldsymbol{b}_1^T \cdot \boldsymbol{x}), \ldots, \\ &\sin(2\pi \boldsymbol{b}_m^T \cdot \boldsymbol{x}), \cos(2\pi \boldsymbol{b}_m^T \cdot \boldsymbol{x})). \end{aligned}$$

  - Adaptive activation functions [13]…

# Neural Solver

- Novel Architectures
  - Sequential Architectures [14]
    - Solves time-dependent PDEs and uses LSTM architectures

$$\mathcal{L}_{\text{reg}} = \left\| \frac{u_{i+1} - u_i}{\Delta t} - F\left(u_i, \frac{\partial u_i}{\partial x_1}, \dots \frac{\partial u}{\partial x_d}, \dots, \theta\right) \right\|^2$$

  - Convolutional Architectures [14]
    - Replace spatial differentiation with numerical ones and use CNNs

$$D_1 \approx \frac{1}{h^2} \begin{pmatrix} 1 & -2 & 1 \end{pmatrix}$$

$$D_2 \approx \frac{1}{h^2} \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

→

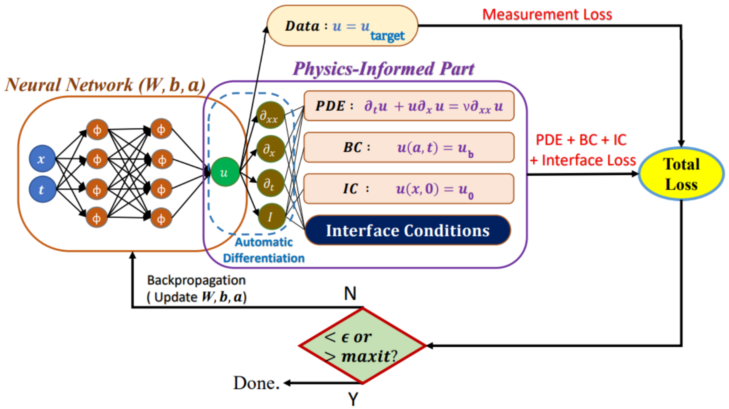
$$\Delta u(x, y) \approx D_2 * U$$

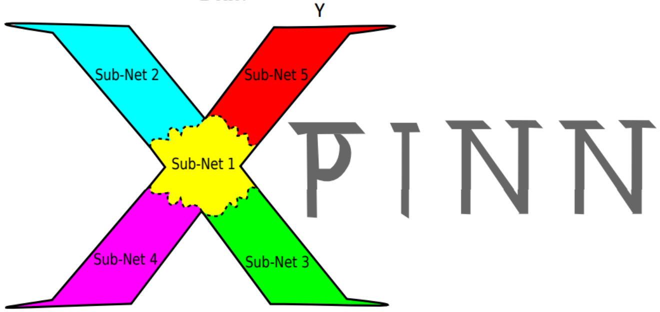
# Neural Solver

- Novel Architectures—Domain Decomposition
  - XPINNs [15]: use $K$ subnets for $K$ subdomains
  - Loss functions:

$$\mathcal{L} = \sum_{k=1}^{K} (\lambda_r^k \mathcal{L}_r^k + \lambda_b^k \mathcal{L}_b^k + \lambda_i^k \mathcal{L}_i^k) + \sum_{m=1}^{M} \lambda_I^m \mathcal{L}_I^m.$$

  - $\mathcal{L}_I^m$ : Interface condition
    - Continuity of physical quantities
    - Conservation/continuity of other variables flow or gradients



# Neural Solver

- Summary of existing methods for neural solver

| | Method | Description | Representatives |
|---|---|---|---|
| Neural Solver | Loss Reweighting | Grad Norm | GradientPathologiesPINNs [43] |
| | | NTK Reweighting | PINNsNTK [44] |
| | | Variance Reweighting | Inverse-Dirichlet PINNs [45] |
| | Novel Optimization Targets | Numerical Differentiation | DGM [46], CAN-PINN [47], cvPINNs [48] |
| | | Variantional Formulation | vPINN [49], hp-PINN [50], VarNet [51], WAN [52] |
| | | Regularization | gPINNs [53], Sobolev Training [54] |
| | Novel Architectures | Adaptive Activation | LAAF-PINNs [55], [56], SReLU [57] |
| | | Feature Preprocessing | Fourier Embedding [58], Prior Dictionary Embedding [59] |
| | | Boundary Encoding | TFC-based [60], CENN [61], PFNN [62], HCNet [63] |
| | | Sequential Architecture | PhyCRNet [64], PhyLSTM [65] AR-DenseED [66], HNN [67], HGN [68] |
| | | Convolutional Architecture | PhyGeoNet [69], PhyCRNet [64], PPNN [70] |
| | | Domain Decomposition | XPINNs [71], cPINNs [72], FBPINNs [73], Shukla et al. [74] |
| | Other Learning Paradigms | Transfer Learning | Desai et al. [75], MF-PIDNN [76] |
| | | Meta-Learning | Psaros et al. [77], NRPINNs [78] |

TABLE 2: An overview of variants of PINNs. Variants of PINNs include loss reweighting, novel optimization targets, novel architectures and other techniques such as meta-learning.

# Neural Operator

- Learning an operator $\tilde{G}: \Theta \times \Omega \to \mathbb{R}^m$

$$\min_{w \in W} \left\| G_w(\theta)(\boldsymbol{x}) - \tilde{G}(\theta)(\boldsymbol{x}) \right\|$$

  where $\theta \in \Theta$ is control/design parameters, $G_w$ is the trained neural model.

- Training dataset
  - Data points: $\{\tilde{G}(\theta_i)(\boldsymbol{x}_j)\}$
  - Collocation points: $\{(\theta_i, \boldsymbol{x}_j)\}$ (physics-informed loss)
- Categories
  - Direct Methods, Green's Function learning, Grid-based Operator Learning, Graph-based Operator Learning.

# Direct Methods

- Directly parameterize $\tilde{G}\colon \Theta \times \Omega \rightarrow \mathbb{R}^m$ as a neural network, following the format in *Universal Approximation Theorem of Operator*

| Category & Formulation | Rrepresentative | Description |
|---|---|---|
| **Direct Methods** <br> $G_w(\theta)(\boldsymbol{x}) = b_0 + \sum_{k=1}^{p} b_k(\theta) t_k(\boldsymbol{x})$ | DeepONet [153] | Parameterize $b_k$ and $t_k$ with neural networks, which are trained with supervised data. |
| | Physics-informed DeepONet [154] | Train DeepONet with a combination of data and physics-informed losses. |
| | Improved Architectures for DeepONet [155], [156] | Including modified network structures (see Eq. (99)), input transformation ($\boldsymbol{x} \mapsto (\boldsymbol{x}, \sin(\boldsymbol{x}), \cos(\boldsymbol{x}), \dots)$), POD-DeepONet (see Eq. (101)), and output transformation (see Eq. (102) and Eq. (103)). |
| | Multiple-input DeepONet [157] | A variant of DeepONet taking multiple various parameters as input, i.e., $\tilde{G}\colon \Theta_1 \times \Theta_2 \times \cdots \times \Theta_n \rightarrow Y$. |
| | Pre-trained DeepONet for Multi-physics [158], [159] | Model a multi-physics system with several pre-trained DeepONets serving as building blocks. |
| | Other Variants | Including Bayesian DeepONet [160], multi-fidelity DeepONet [161], and MultiAuto-DeepONet [162]. |

# Green's Function learning

- We are interested when $\theta$ is a *function* ($\tilde{G}: f \mapsto u$)

$$\mathcal{F}_L[u] = f, \qquad \boldsymbol{x} \in \Omega$$
$$\mathcal{B}_L[u] = g, \qquad \boldsymbol{x} \in \partial\Omega$$

where $\mathcal{F}_L$ and $\mathcal{B}_L$ are two linear operators

- Represent the solution via Green's function

$$u(\boldsymbol{x}) = \int_\Omega \mathcal{G}(\boldsymbol{x}, \boldsymbol{y}) f(\boldsymbol{y}) d\boldsymbol{y} + u_{\text{homo}}(x)$$

where $\mathcal{G}(\boldsymbol{x}, \boldsymbol{y})$ is parameterized by NN (**double dimension**)

| | | |
|---|---|---|
| **Green's Function Learning** | Methods for Linear Operators [163], [164] | Parameterize $\mathcal{G}$ and $u_{\text{homo}}$ with neural networks, which are trained with supervised data (and possibly physics-informed losses). |
| $G_w(\theta)(\boldsymbol{x}) = \int_\Omega \mathcal{G}(\boldsymbol{x}, \boldsymbol{y})\theta(\boldsymbol{y})\mathrm{d}\boldsymbol{y} + u_{\text{homo}}(\boldsymbol{x})$, where $\theta$ is a function $\theta = v(\boldsymbol{x})$ | Methods for Nonlinear Operators [165] | Discretize the PDEs and use trainable mappings to linearize the target operator, where Green's function formula is subsequently applied to construct the approximation. |

# Grid/Graph-based Methods

- Grid-based operator learning (image-to-image)

$$\tilde{G}: \{u(\boldsymbol{x}_i)\} \mapsto \{v(\boldsymbol{x}_i)\}$$

- Graph-based Methods (graph-to-graph)

$$\tilde{G}: \text{node features} \mapsto \text{node features}$$

| | | |
|---|---|---|
| **Grid-based Operator Learning**<br><br>$G_w(\theta) = \{u(\boldsymbol{x}_i)\}_{i=1}^N$, where $\{u(\boldsymbol{x}_i)\}_{i=1}^N$ and $\theta = \{v(\boldsymbol{x}_i)\}_{i=1}^N$ are discretizations of input and output functions in some **grids** | Convolutional Neural Network [69], [166] | A convolutional neural network is utilized to approximate such an image-to-image mapping, where the loss function is based on supervised data (and possibly physics-informed losses). |
| | Fourier Neural Operator [167] | Several Fourier convolutional kernels are incorporated into the network structure, to better learn the features in the frequency domain. |
| | Neural Operator with Attention Mechanism [168], [169], [170] | The attention mechanism is introduced to the design of the network structure, to improve the abstraction ability of the model. |
| **Graph-based Operator Learning**<br><br>$G_w(\theta) = \{u(\boldsymbol{x}_i)\}_{i=1}^N$, where $\{u(\boldsymbol{x}_i)\}_{i=1}^N$ and $\theta = \{v(\boldsymbol{x}_i)\}_{i=1}^N$ are discretizations of input and output functions in some **graphs** | Graph Kernel Network [171] | A graph kernel network is employed to learn such a graph-based mapping. |
| | Multipole Graph Neural Operator [172] | The graph kernel is decomposed into several multi-level sub-kernels, to capture multi-level neighboring interactions. |
| | Graph Neural Opeartor with Autogressive Methods [173] | Extend graph neural operators to time-dependent PDEs. |

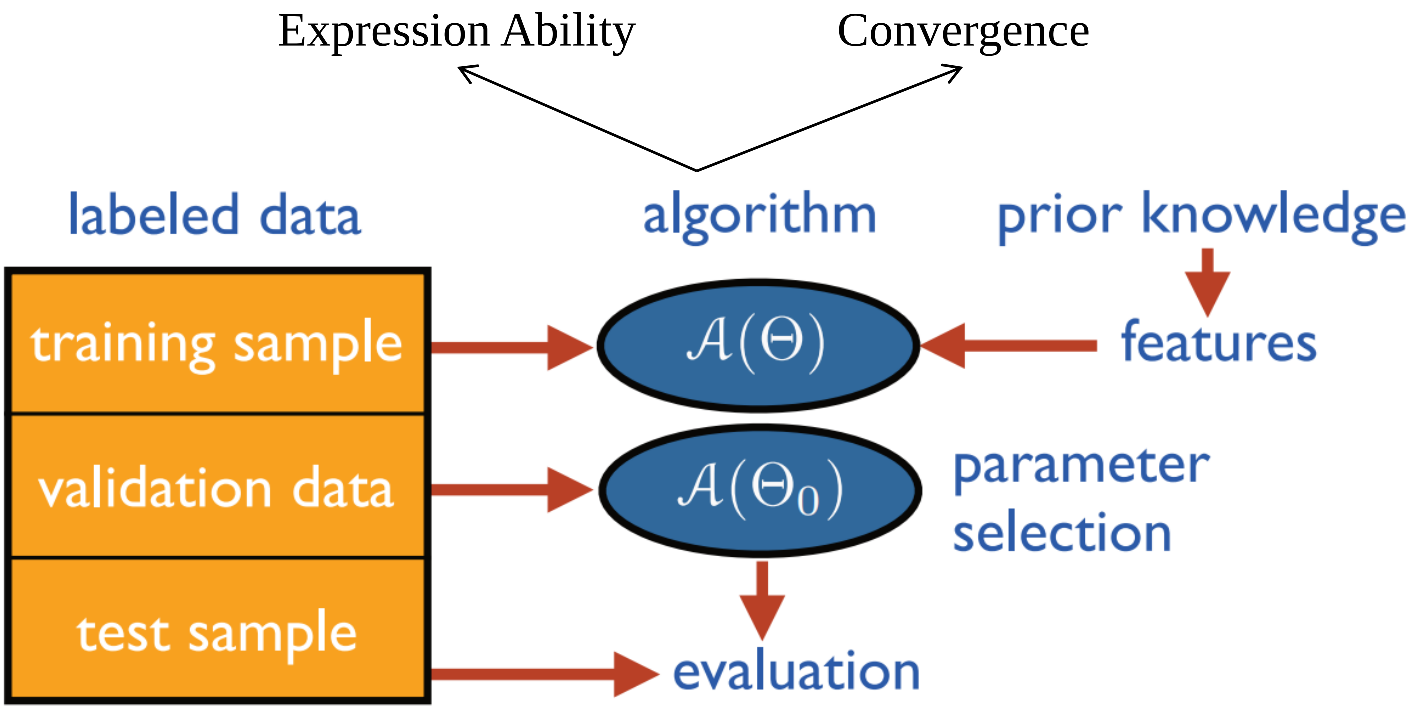
# Neural Operator

- Open Challenges
  - Incorporating physical priors
    - Generalizability↑, Data Demand↓
    - Close integration of physics knowledge and models,
    in addition to *physics-informed loss functions*
  - Reducing the cost of gathering datasets (**major**)
    - Large design space $\Theta$, complex geometry $\Omega$
    - High cost of data generating
  - Developing large pre-trained models
  - Modeling real-world physical systems

# Neural Operator

- Open Challenges
  - Incorporating physical priors
  - Reducing the cost of gathering datasets (**major**)
  - Developing large pre-trained models
    - Handle so many downstream tasks
    - A possible to reduce data cost and training overhead
  - Modeling real-world physical systems
    - From idealized experiments  to real-world ones
    - It may be helpful to borrow from the field of numerical computing
    - Efficiently employed in industrial scenarios, e.g., optimization, simulation, etc.

# Theory in PIML




Error Estimation

# Expression Ability

- It is well known that multi-layer neural networks are universal approximators, i.e., they can approximate any measurable function to arbitrary accuracy.
- A major concern in PIML is to approximate neural operator.
- One-layer neural networks can approximate any operators [16, 17]. (DeepONet)
- Next question: how many nodes do we need?
- Wide, shallow neural networks may need exponentially many neurons to obtain similar expression ability with deep, narrow ones [18].

# Expression Ability

- [18] takes a first step for providing an upper bound of the width of the deeper neural networks for approximating operators.

- Future work:

- design more effective architecture to approximate operators with fewer nodes is significant for designing more stable and effective algorithms
- analyze the expression ability of other architectures

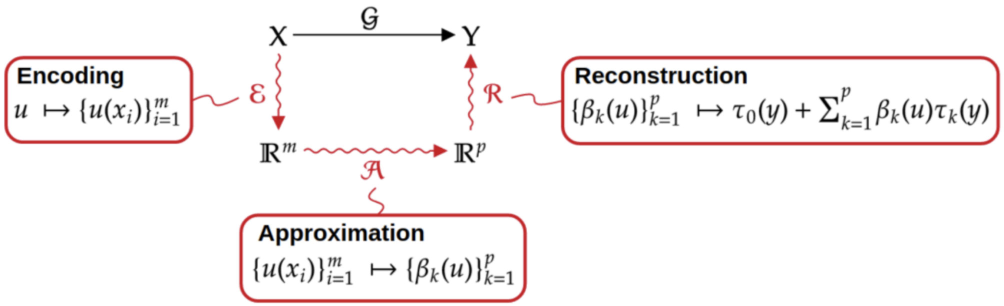
# Convergence

- evaluate the algorithm: whether it converges and its convergence speed
- combine optimization and PDEs
- current with little research: PINNs [19], neural operator [20], deep ritz methods [21]

# Error Estimation

- There are different kinds of error: approximation error (the target loss), generalization error (generalize to unseen samples) ...

- [22] first analyzes the approximation error and generalization error of DeepONet



# Theory in PIML

- Future work:
- design more effective architecture to approximate operators with fewer nodes is significant for designing more stable and effective algorithms
- analyze the expression ability of other architectures
- analyze the convergence of PINNs for different kinds of PDEs for designing more efficient architectures and algorithms

# Inverse Problem (Inverse Design)

- To optimize or discover unknown parameters of a physical system, including scientific discovery, shape optimization, optimal control, etc.

$$\min_{\theta\in\Theta} \mathcal{J}(u(\boldsymbol{x};\theta),\theta),$$
$$s.t.\ \mathcal{P}(u;\theta)(\boldsymbol{x}) = 0.$$

- Traditional methods
  - SQP, Adjoint PDE
  - infeasible in large-scale problems
  - heavy computational cost
  - non-differentiable physical process

# Inverse Design

- Solving inverse design usually involves multiple steps
  - e.g., simulation, evaluation, configuration
- System Simulation & Evaluation
  - Neural Surrogates
  - PINN, Neural Operator, Neural Simulator
- Neural Representation
- Design Prediction
- Data Generation
- …

# Neural Surrogates

With PINN

- Directly extend PINN to inverse design[26]

$$\mathcal{L} = \lambda_p \mathcal{L}_{PINN} + \lambda_d \mathcal{J}$$

  - imbalance training objectives
- PINN with hard constraints (h-PINN)[23]
  - imposing hard constraints with the penalty method and the augmented Lagrangian method
- Bi-level optimization framework[24]

$$\begin{aligned} \min_{\theta} \quad & \mathcal{J}(w^*, \theta) \\ s.t. \quad & w^* = \arg\min_w \mathcal{L}_{PINN}(w, \theta). \end{aligned}$$

# Neural Surrogates

With Neural Operator

- Trained differentiable neural operator predicting the state variables[25]

$$w^* = \arg\min_{w \in W} \mathcal{L}_{operator},$$

$$\min_{\alpha} \mathcal{J}(G_{w^*}(\theta_\alpha)(\boldsymbol{x}), \theta_\alpha),$$

  - using various models,e.g., DeepONet[27],Autoencoder[28]

With Neural Simulators

- Differentiable simulators mapping parameters to the values of interest to avoid numerical simulations[29]

# Other Methods

- Neural Representation
  - parameterize the parameters/configurations with neural network to achieve more highly-detailed and continuous representations[30]
- Design Prediction
  - map the desired targets to the design parameters[31]
- Data Generation
  - generate novel samples with superior performance using generative models like VAE[32]

# Open Problems and Challenges

- Neural Surrogate Modeling
  - balance of multiple loss terms and training convergence for physics-informed methods
  - large demand of data for operator and simulator training
- Large Scale Application
  - potential challenges like curse of dimensions, computational complexity in large scale scenarios
- Other Directions
  - using neural networks in other steps of inverse design besides simulation

# Computer Vision and Graphics

- Traditional Visual Tasks (Classification & Detection)
  - knowledge of symmetry, such as equivariance to rotation[33]
- Motion and Pose Analysis/Physical Scene Understanding
  - knowledge of mechanics and kinematics, such as motion constraints[34], Hamiltonian canonical equations[35]
- Computer graphics
  - knowledge of rendering, such as classical volume rendering equations[36]

# Reinforcement Learning

- Goal
  - to interact with an unknown world to maximize reward, with/without the learning of world models
- Policy Training
  - use knowledge to design rewards for specific goals, such as adaptive mesh refinement[37]
- Model Training
  - use knowledge to learn a better world model, such as equations of continuous dynamics[38]
- Exploration Guiding
  - use knowledge to restrict exploration to safe regions, such as logical sandboxes[39]

# Open Problems and Challenges

- Better Description of Physical World
  - to learn meaningful representations from visual observations
  - to find formulated representations of intuitive physics
- Generic Modeling of Physical Tasks
  - to deal with new tasks from proper expert knowledge instead of case-by-case design
- Solving High-Dimensional Problems in RL
- Guaranteeing Safety in Complex, Uncertain Environments

# Conclusions

- Open challenges
- From methodological perspective
  - Standardized dataset
  - Better algorithms for inference and optimization
  - Scalable algorithms for intuitive physics in real world
- From tasks perspective
  - Better methods for neural simulation
  - Inverse problems
  - More applications in real world CV/RL

Thank you!

# References


[1] DGM: A deep learning algorithm for solving partial differential equations
[2] The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems
[3] Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations
[4] Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators
[5] Fourier neural operator for parametric partial differential equations
[6] Amortized finite element analysis for fast pde-constrained optimization
[7] Understanding and mitigating gradient pathologies in physics-informed neural networks
[8] Efficient Training of Physics-Informed Neural Networks via Importance Sampling
[9] Variational physics-informed neural networks for solving partial differential equations
[10] Gradient-enhanced physics-informed neural networks for forward and inverse PDE problems
[11] Neural-network methods for boundary value problems with irregular boundaries
[12] On the eigenvector bias of fourier feature networks: From regression to solving multi-scale pdes with physics-informed neural networks
[13] Adaptive activation functions accelerate convergence in deep and physics-informed neural networks
[14] PhyCRNet: Physics-informed convolutional-recurrent network for solving spatiotemporal PDEs
[15] Extended physics-informed neural networks (XPINNs) : A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equation
[16] Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems
[17] Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems
[18] Arbitrary-depth universal approximation theorems for operator neural networks

# References

[19] On the convergence of physics informed neural networks for linear second-order elliptic and parabolic type PDEs
[20] Convergence rates for learning linear operators from noisy data
[21] Uniform convergence guarantees for the deep Ritz method for nonlinear problems
[22] Error estimates for deeponets: A deep learning framework in infinite dimensions
[23] 31.Physics-informed neural networks with hard constraints for inverse design
[24] Bi-level Physics-Informed Neural Networks for PDE Constrained Optimization using Broyden's Hypergradients
[25] Fast PDE-constrained optimization via self-supervised operator learning
[26] Optimal control of pdes using physics-informed neural networks
[27] Amortized synthesis of constrained configurations using a differentiable surrogate
[28] Solving pde-constrained control problems using operator learning
[29] Iterative surrogate model optimization (ismo): an active learning algorithm for pde constrained optimization with deep neural networks
[30] Neural reparameterization improves structural optimization
[31] Inverse design of airfoil using a deep convolutional neural network
[32] Data-driven topology design using a deep generative model
[33] Rotationally equivariant 3d object detection
[34] Physical inertial poser (pip): Physics-aware real-time human motion tracking from sparse inertial sensors,
[35] Hamiltonian generative networks
[36] Nerf: Representing scenes as neural radiance fields for view synthesis
[37] Reinforcement learning for adaptive mesh refinement
[38] Differentiable physics models for real-world offline model-based reinforcement learning
[39] Safe reinforcement learning via formal methods: Toward safe control through proof and learning