

# Introduction to TorchPhysics

Deep Learning Library for Differential Equations

Janek Gödeke, Nick Heilenkötter, Tom  
Freudenberg  
Renningen, 20.11.2025

# Goals of TorchPhysics-Workshop

**Overall:** Usage of TorchPhysics for

- 💡 Physics-informed DL
  - 💡 Data-based DL
- }
- for ODEs / PDEs / Parameter identification
- 💡 Track 1:** Function Learning with PINNs    **Track 2:** Operator Learning

# Goals of TorchPhysics-Workshop

**Overall:** Usage of TorchPhysics for

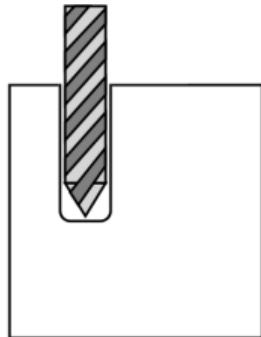
- 💡 Physics-informed DL
  - 💡 Data-based DL
- }
- for ODEs / PDEs / Parameter identification
- 💡 Track 1:** Function Learning with PINNs    **Track 2:** Operator Learning

**Today:**

- 💡 Learning solution functions of PDEs with PINNs
- 💡 Track 1:** Time-dependent domains    **Track 2:** Introduction to Operator Learning

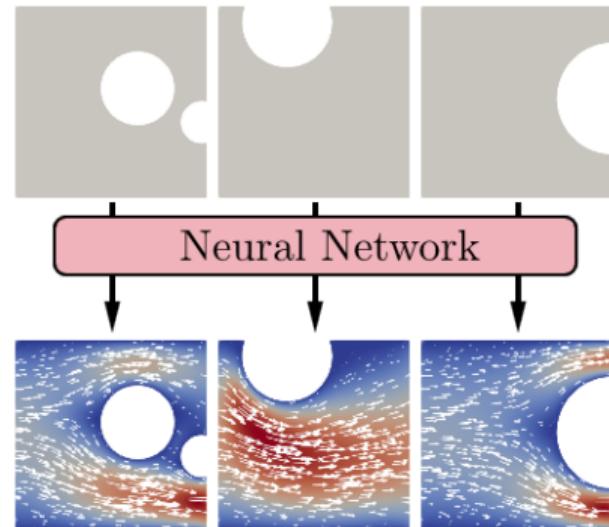
# Example from **Track 1**

- Solve a simplified drilling problem
- Heat equation on a **time dependent domain**



## Example from Track 2

- Compute the flow around an obstacle
  - Learn the solver:
    - **Input:** description of domain
    - **Output:** flow field inside the domain fulfilling the Stokes equations
- **One network for many solutions!**



## Recall: Neural Networks

# What is a Neural Network (NN)?

- Parameterized function

$$u : \Theta \times \mathbb{R}^d \longrightarrow \mathbb{R}^n$$

$$(\theta, x) \longmapsto u(\theta, x)$$

parameter space  $\Theta \subseteq \mathbb{R}^p$

- **Notation:**  $u_\theta(x) := u(\theta, x)$

# What is a Neural Network (NN)?

- Parameterized function

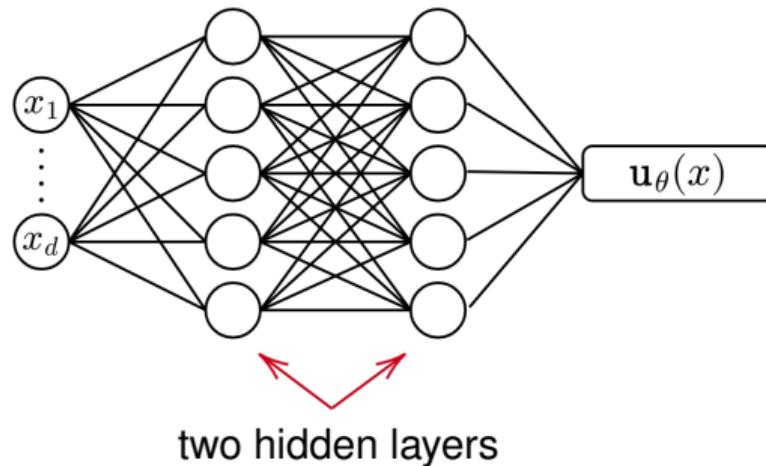
$$u : \Theta \times \mathbb{R}^d \rightarrow \mathbb{R}^n$$

$$(\theta, x) \longmapsto u(\theta, x)$$

parameter space  $\Theta \subseteq \mathbb{R}^p$

- **Notation:**  $u_\theta(x) := u(\theta, x)$

## Fully Connected NN:

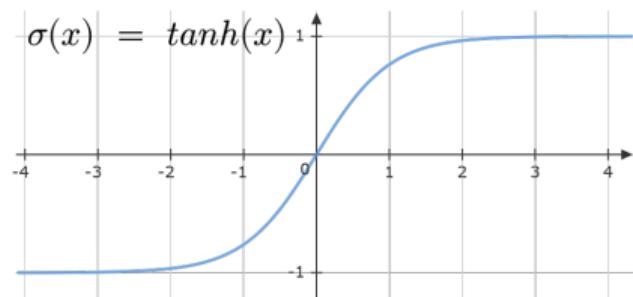


# Transformations by Neurons

- ... of  $u_\theta$  are functions of the form

$$\begin{aligned}\mathbb{R}^I &\longrightarrow \mathbb{R} \\ x &\longmapsto \sigma(a^T x + b)\end{aligned}$$

- Vector  $a$  and bias  $b$  belong to parameters  $\theta$
- Activation function  $\sigma : \mathbb{R} \rightarrow \mathbb{R}$



# Training NNs with Data

**Given:** Data points  $(x_1, u_1), \dots, (x_k, u_k)$

**Goal:** Adjust parameters  $\theta \in \Theta$

$$u_\theta(x_i) \approx u_i$$

# Training NNs with Data

**Given:** Data points  $(x_1, u_1), \dots, (x_k, u_k)$

**Goal:** Adjust parameters  $\theta \in \Theta$

$$u_\theta(x_i) \approx u_i$$

**How?** Minimize (data-)loss

$$\ell(\theta) := \frac{1}{k} \sum_{i=1}^k |u_\theta(x_i) - u_i|^2$$

by gradient descent algorithms

# Training NNs with Data

**Given:** Data points  $(x_1, u_1), \dots, (x_k, u_k)$

**Goal:** Adjust parameters  $\theta \in \Theta$

$$u_\theta(x_i) \approx u_i$$

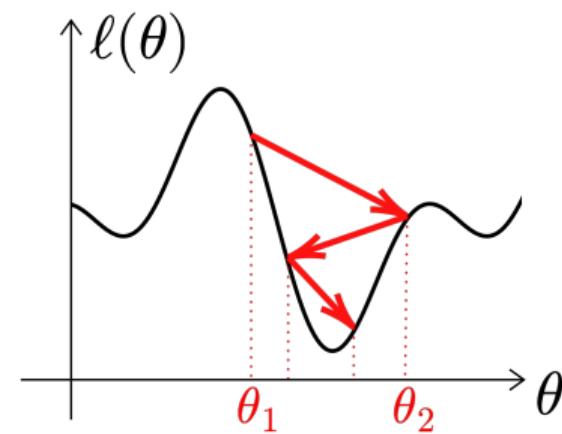
**How?** Minimize (data-)loss

$$\ell(\theta) := \frac{1}{k} \sum_{i=1}^k |u_\theta(x_i) - u_i|^2$$

by gradient descent algorithms

**Gradient descent step:** E.g.,

$$\theta_{j+1} = \theta_j - \text{lr} \cdot \nabla \ell(\theta_j)$$

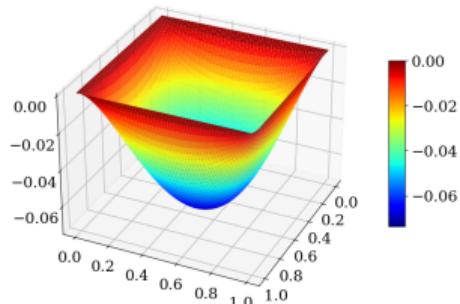


# Deep Learning for PDEs

# Function Learning Tasks

Poisson equation:

$$\begin{aligned}\Delta u(x) &= f(x) \quad \text{for } x \in \Omega, \\ u(x) &= c \quad \text{for } x \in \partial\Omega.\end{aligned}$$

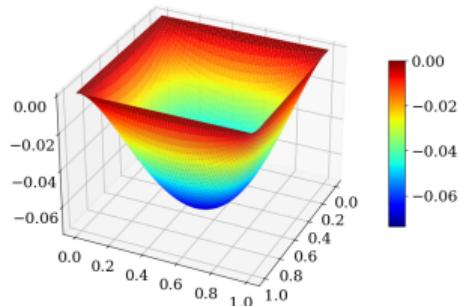


**Fig.:** Learned solution for  $f = 1$ ,  
 $c = 0$

# Function Learning Tasks

Poisson equation:

$$\begin{aligned}\Delta u(x) &= f(x) \quad \text{for } x \in \Omega, \\ u(x) &= c \quad \text{for } x \in \partial\Omega.\end{aligned}$$



**Fig.:** Learned solution for  $f = 1$ ,  
 $c = 0$

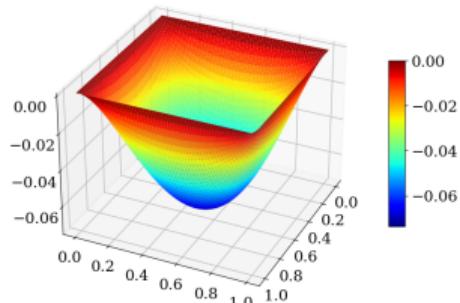
- 1) Find solution  $u$  for **fixed** parameter  $c \in \mathbb{R}$   
function  $f : \Omega \rightarrow \mathbb{R}$

Neural network  $u_\theta(x) \approx u(x)$

# Function Learning Tasks

Poisson equation:

$$\begin{aligned}\Delta u(x) &= f(x) \quad \text{for } x \in \Omega, \\ u(x) &= c \quad \text{for } x \in \partial\Omega.\end{aligned}$$



**Fig.:** Learned solution for  $f = 1$ ,  
 $c = 0$

- 1) Find solution  $u$  for **fixed** parameter  $c \in \mathbb{R}$  function  $f : \Omega \rightarrow \mathbb{R}$

Neural network  $u_\theta(x) \approx u(x)$

- 2) Find solutions  $u_c$  for **all**  $c \in [a, b]$

Neural network  $u_\theta(x, c) \approx u_c(x)$

→ **Track 1** tomorrow (parameter study)

# Operator Learning Task

Poisson equation:

$$\begin{aligned}\Delta u(x) &= \mathbf{f}(x) \quad \text{for } x \in \Omega, \\ u(x) &= c \quad \text{for } x \in \partial\Omega.\end{aligned}$$

3) Find solutions  $u_f$  for **many functions**  $\mathbf{f}: \Omega \rightarrow \mathbb{R}$

Neural network  $u_\theta(x, \mathbf{f}) \approx u_f(x)$

e.g., polynomials, sin, cos, etc.

# Operator Learning Task

Poisson equation:

$$\begin{aligned}\Delta u(x) &= \mathbf{f}(x) \quad \text{for } x \in \Omega, \\ u(x) &= c \quad \text{for } x \in \partial\Omega.\end{aligned}$$

3) Find solutions  $u_f$  for **many functions**  $\mathbf{f}: \Omega \rightarrow \mathbb{R}$

Neural network  $u_\theta(x, \mathbf{f}) \approx u_f(x)$

e.g., polynomials, sin, cos, etc.

**Problem:** Input of NNs must be from  $\mathbb{R}^n$

→ **Track 2** of this workshop

# Today's Starting Point

Function Learning with PINNs

- 1) Find solution  $u$  for **fixed** parameter  $c \in \mathbb{R}$   
function  $f : \Omega \rightarrow \mathbb{R}$

Poisson equation:

$$\begin{aligned}\Delta u(x) &= f(x) \quad \text{for } x \in \Omega, \\ u(x) &= c \quad \text{for } x \in \partial\Omega.\end{aligned}$$

Neural network  $u_\theta(x) \approx u(x)$

# Today's Starting Point

Function Learning with PINNs

- 1) Find solution  $u$  for **fixed** parameter  $c \in \mathbb{R}$   
function  $f : \Omega \rightarrow \mathbb{R}$

Poisson equation:

$$\begin{aligned}\Delta u(x) &= f(x) \quad \text{for } x \in \Omega, \\ u(x) &= c \quad \text{for } x \in \partial\Omega.\end{aligned}$$

Neural network  $u_\theta(x) \approx u(x)$

- 💡 Get in touch with physics-informed DL
- 💡 Get along with structure of TORCHPHYSICS

# Physics-Informed Neural Networks (PINNs)

# PINNs

 **Physics-informed neural networks: A deep learning framework for solving  
(...) partial differential equations**

Authors: Raissi, Perdikaris, Karniadakis  
Journal of Computational Physics, 2019

# PINNs

 **Physics-informed neural networks: A deep learning framework for solving (...) partial differential equations**

Authors: Raissi, Perdikaris, Karniadakis  
Journal of Computational Physics, 2019

- "**Physics-Informed**": Include PDE into loss function
- No data is required  
(combination might be beneficial, though)

# PINNs - Main Idea

- Find solution  $\mathbf{u} : \Omega \subset \mathbb{R}^n \rightarrow \mathbb{R}^m$  of

$$\begin{aligned}\mathcal{N}[\mathbf{u}](x) &= 0 \quad \text{for } x \in \Omega, \\ \mathcal{B}[\mathbf{u}](x) &= 0 \quad \text{for } x \in \partial\Omega.\end{aligned}$$

- E.g.  $\mathbf{u} : (0, 1) \times (0, 1) \rightarrow \mathbb{R}$

$$\begin{aligned}\mathcal{N}[\mathbf{u}](x) &= \Delta \mathbf{u}(x) - f(x) \quad \text{for } x \in \Omega, \\ \mathcal{B}[\mathbf{u}](x) &= \mathbf{u}(x) - c \quad \text{for } x \in \partial\Omega.\end{aligned}$$

# PINNs - Main Idea

- Find solution  $\mathbf{u} : \Omega \subset \mathbb{R}^n \rightarrow \mathbb{R}^m$  of

$$\begin{aligned}\mathcal{N}[\mathbf{u}](x) &= 0 \quad \text{for } x \in \Omega, \\ \mathcal{B}[\mathbf{u}](x) &= 0 \quad \text{for } x \in \partial\Omega.\end{aligned}$$

- E.g.  $\mathbf{u} : (0, 1) \times (0, 1) \rightarrow \mathbb{R}$

$$\begin{aligned}\mathcal{N}[\mathbf{u}](x) &= \Delta \mathbf{u}(x) - f(x) \quad \text{for } x \in \Omega, \\ \mathcal{B}[\mathbf{u}](x) &= \mathbf{u}(x) - c \quad \text{for } x \in \partial\Omega.\end{aligned}$$

- Sample points  $x_i^{\mathcal{N}} \in \Omega$  and  $x_j^{\mathcal{B}} \in \partial\Omega$
- Train network  $\mathbf{u}_\theta$  minimizing the PDE-loss

$$\frac{1}{N_{\mathcal{N}}} \sum_{i=1}^{N_{\mathcal{N}}} \|\mathcal{N}[\mathbf{u}_\theta](x_i^{\mathcal{N}})\|^2 + \frac{1}{N_{\mathcal{B}}} \sum_{j=1}^{N_{\mathcal{B}}} \|\mathcal{B}[\mathbf{u}_\theta](x_j^{\mathcal{B}})\|^2$$

# PINNs - Main Idea

- Find solution  $\mathbf{u} : \Omega \subset \mathbb{R}^n \rightarrow \mathbb{R}^m$  of

$$\begin{aligned}\mathcal{N}[\mathbf{u}](x) &= 0 \quad \text{for } x \in \Omega, \\ \mathcal{B}[\mathbf{u}](x) &= 0 \quad \text{for } x \in \partial\Omega.\end{aligned}$$

- E.g.  $\mathbf{u} : (0, 1) \times (0, 1) \rightarrow \mathbb{R}$

$$\begin{aligned}\mathcal{N}[\mathbf{u}](x) &= \Delta \mathbf{u}(x) - f(x) \quad \text{for } x \in \Omega, \\ \mathcal{B}[\mathbf{u}](x) &= \mathbf{u}(x) - c \quad \text{for } x \in \partial\Omega.\end{aligned}$$

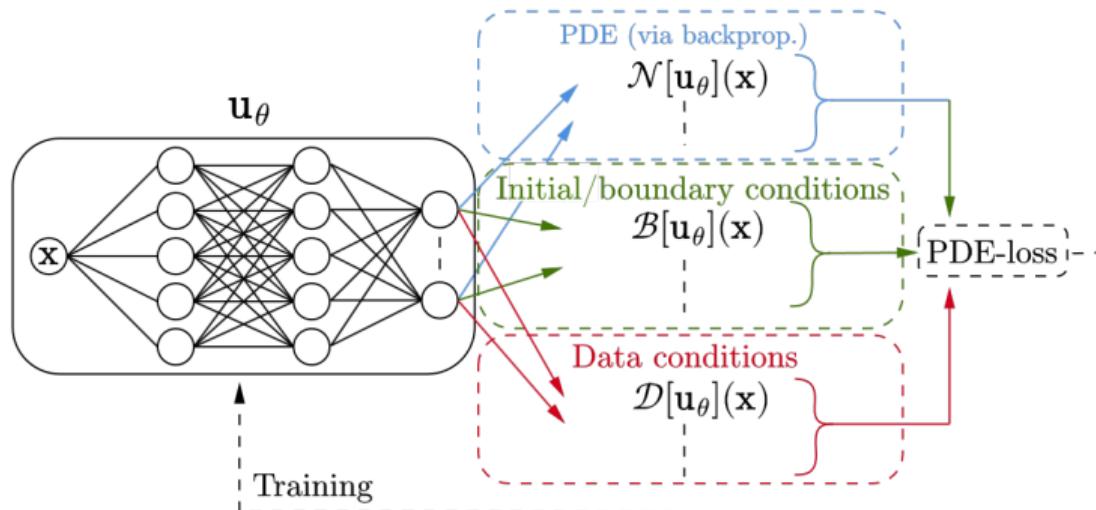
- Sample points  $x_i^{\mathcal{N}} \in \Omega$  and  $x_j^{\mathcal{B}} \in \partial\Omega$
- Train network  $\mathbf{u}_\theta$  minimizing the PDE-loss

Data available?

$$\frac{1}{N_{\mathcal{N}}} \sum_{i=1}^{N_{\mathcal{N}}} \|\mathcal{N}[\mathbf{u}_\theta](x_i^{\mathcal{N}})\|^2 + \frac{1}{N_{\mathcal{B}}} \sum_{j=1}^{N_{\mathcal{B}}} \|\mathcal{B}[\mathbf{u}_\theta](x_j^{\mathcal{B}})\|^2 + \frac{1}{N_D} \sum_{k=1}^{N_D} \|\mathbf{u}_\theta(x_k^D) - \mathbf{u}_k\|^2$$

# PINN-Approach

Schematic depiction



$$u_\theta = \arg \min_u \frac{\omega_{\mathcal{N}}}{N_{\mathcal{N}}} \sum_{i=0}^{N_{\mathcal{N}}} \|\mathcal{N}[u](x_i^{\mathcal{N}})\|^2 + \frac{\omega_{\mathcal{B}}}{N_{\mathcal{B}}} \sum_{i=0}^{N_{\mathcal{B}}} \|\mathcal{B}[u](x_i^{\mathcal{B}})\|^2 + \frac{\omega_{\mathcal{D}}}{N_{\mathcal{D}}} \sum_{i=0}^{N_{\mathcal{D}}} \|\mathcal{D}[u](x_i^{\mathcal{D}})\|^2$$

# Requirements for Physics-Informed Training

One needs to:

- Compute differential operators of  $u_\theta$ , e.g.,  $\Delta u_\theta$  for  $\mathcal{N}[\mathbf{u}_\theta] = \Delta u_\theta - f$   
→ Enables PDE-loss computation

$$\text{PDE-loss: } \frac{1}{N_{\mathcal{N}}} \sum_{i=1}^{N_{\mathcal{N}}} \|\mathcal{N}[\mathbf{u}_\theta](x_i^{\mathcal{N}})\|^2 + \frac{1}{N_{\mathcal{B}}} \sum_{j=1}^{N_{\mathcal{B}}} \|\mathcal{B}[\mathbf{u}_\theta](x_j^{\mathcal{B}})\|^2$$

# Requirements for Physics-Informed Training

One needs to:

- Compute differential operators of  $u_\theta$ , e.g.,  $\Delta u_\theta$  for  $\mathcal{N}[\mathbf{u}_\theta] = \Delta u_\theta - f$   
→ Enables PDE-loss computation

$$\text{PDE-loss: } \frac{1}{N_{\mathcal{N}}} \sum_{i=1}^{N_{\mathcal{N}}} \|\mathcal{N}[\mathbf{u}_\theta](x_i^{\mathcal{N}})\|^2 + \frac{1}{N_{\mathcal{B}}} \sum_{j=1}^{N_{\mathcal{B}}} \|\mathcal{B}[\mathbf{u}_\theta](x_j^{\mathcal{B}})\|^2$$

- Differentiate PDE-loss w.r.t. network parameters  $\theta$

# Requirements for Physics-Informed Training

One needs to:

- Compute differential operators of  $u_\theta$ , e.g.,  $\Delta u_\theta$  for  $\mathcal{N}[\mathbf{u}_\theta] = \Delta u_\theta - f$ 
  - Enables PDE-loss computation

$$\text{PDE-loss: } \frac{1}{N_{\mathcal{N}}} \sum_{i=1}^{N_{\mathcal{N}}} \|\mathcal{N}[\mathbf{u}_\theta](x_i^{\mathcal{N}})\|^2 + \frac{1}{N_{\mathcal{B}}} \sum_{j=1}^{N_{\mathcal{B}}} \|\mathcal{B}[\mathbf{u}_\theta](x_j^{\mathcal{B}})\|^2$$

- Differentiate PDE-loss w.r.t. network parameters  $\theta$

**TORCHPHYSICS does this for us!**

- 💡 Makes use of PyTorch's autograd functionality

# PINNs - Advantages

Compared to classical methods

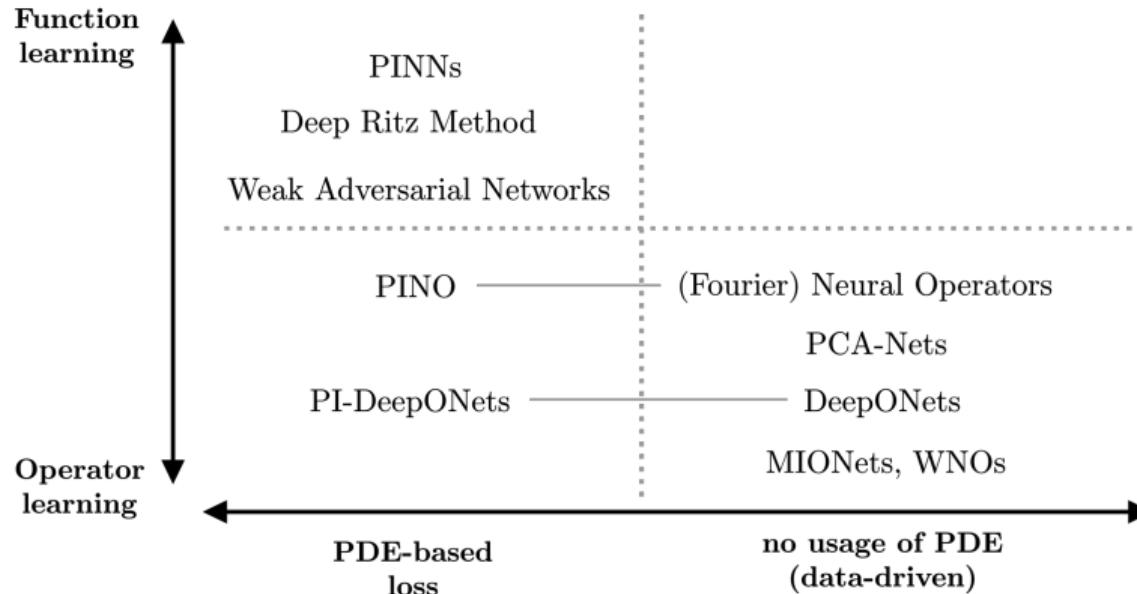
- Grid/mesh independent → more flexible & saving more memory efficient
- General approach: e.g., applicable to non-linear PDEs
- Learning parameter dependencies  $u_\theta(x, c) \approx u_c(x)$
- Extension to parameter identification problems is easy to implement

# PINNs - Disadvantages

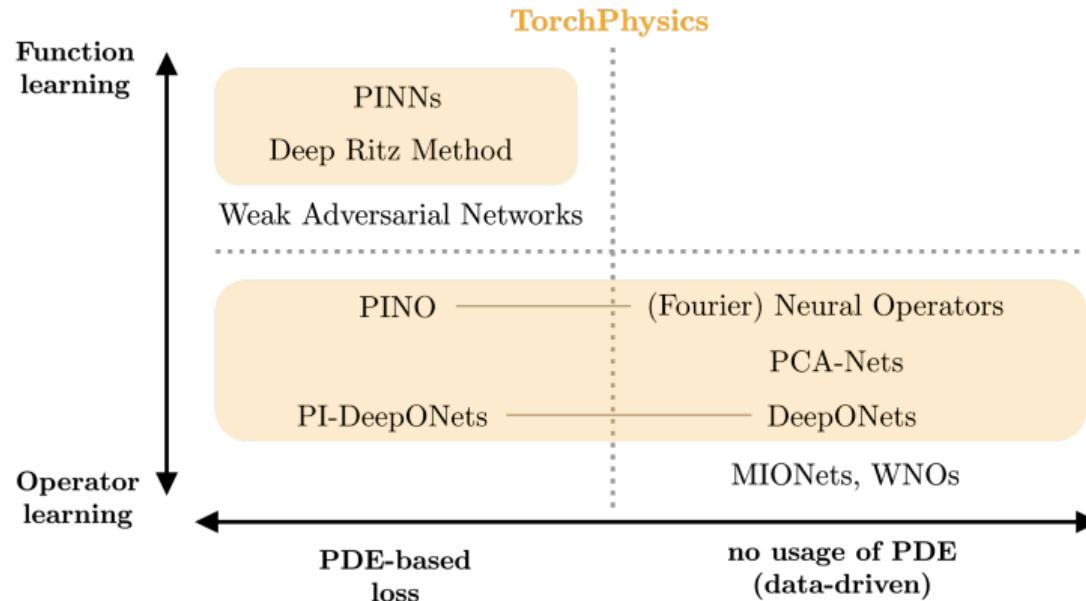
Compared to classical methods

- No convergence theory
- Error not arbitrarily small
- Much slower for solving a single PDE
- Often trial and error for finding good hyperparameters, NN architectures

# Overview of DL-Methods for Differential Equations



# Overview of DL-Methods for Differential Equations



# TorchPhysics

# Initiation of TORCHPHYSICS



- 2021: Robert Bosch GmbH got interested in PINNs
- Technical applications:  
Car parts, electronics, etc.
- Recent applications:
  - Injection molding
  - Electrochemical Systems
  - Capillary surfaces



© factum

# Initiation of TORCHPHYSICS

- 2021: Robert Bosch GmbH got interested in PINNs
- Technical applications:  
Car parts, electronics, etc.
- Recent applications:
  - Injection molding
  - Electrochemical Systems
  - Capillary surfaces
- Student project: Deep Learning library for PDEs
- Main Developers: Nick Heilenkötter & Tom Freudenberg



© factum

# The Toolbox TORCHPHYSICS

- Open-Source on GitHub
- Build upon  PyTorch<sup>1</sup>

 [boschresearch / torchphysics](#) Public

OSS library that implements deep learning methods for partial differential equations and much more

 [boschresearch.github.io/torchphysics/](#)

 Apache-2.0 license

 451 stars     45 forks     Branches     Tags  
 Activity

---

<sup>1</sup>Paszke et al., *PyTorch: An Imperative Style, High-Performance Deep Learning Library*, 2019

# The Toolbox TORCHPHYSICS

- Open-Source on GitHub
- Build upon  PyTorch<sup>1</sup>
- **Goal:** User-friendly for everyone with basic mathematical knowledge

 [boschresearch / torchphysics](#) Public

OSS library that implements deep learning methods for partial differential equations and much more

 [boschresearch.github.io/torchphysics/](https://boschresearch.github.io/torchphysics/)

 Apache-2.0 license

 451 stars     45 forks     Branches     Tags  
 Activity

---

<sup>1</sup>Paszke et al., *PyTorch: An Imperative Style, High-Performance Deep Learning Library*, 2019

# The Toolbox TORCHPHYSICS

- Open-Source on GitHub
- Build upon  PyTorch<sup>1</sup>
- **Goal:** User-friendly for everyone with basic mathematical knowledge
- TORCHPHYSICS provides:
  - Clean documentation, detailed tutorials ([website torchphysics.de](http://torchphysics.de))
  - Modular and extendable structure
  - Intuitive transfer of maths into code

 [boschresearch / torchphysics](#) Public

OSS library that implements deep learning methods for partial differential equations and much more

 [boschresearch.github.io/torchphysics/](#)

 Apache-2.0 license

 451 stars     45 forks     Branches     Tags  
 Activity

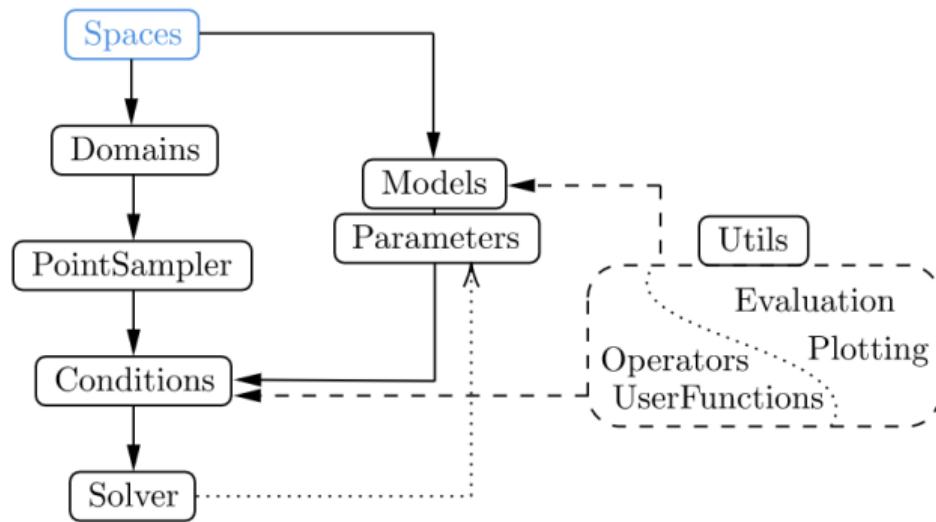
---

<sup>1</sup>Paszke et al., *PyTorch: An Imperative Style, High-Performance Deep Learning Library*, 2019

## TorchPhysics - Structure

# TORCHPHYSICS

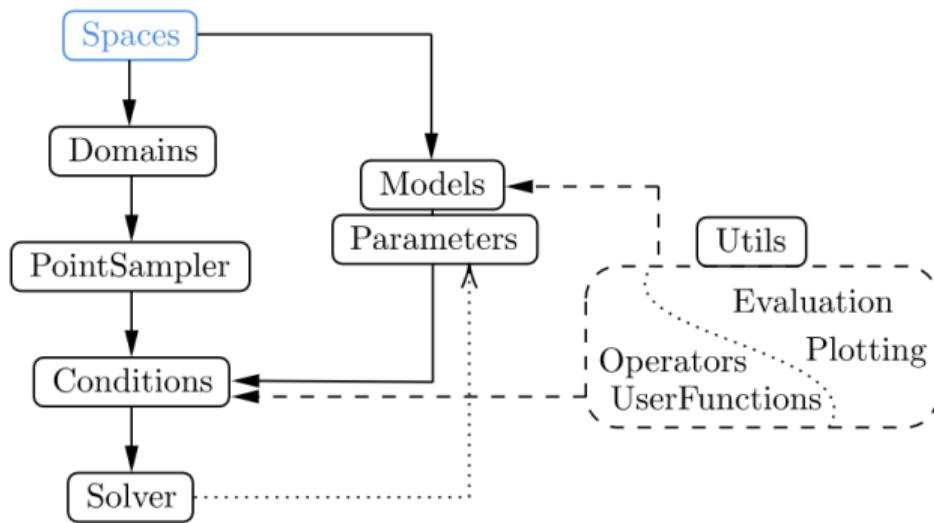
## Spaces



# TORCHPHYSICS

## Spaces

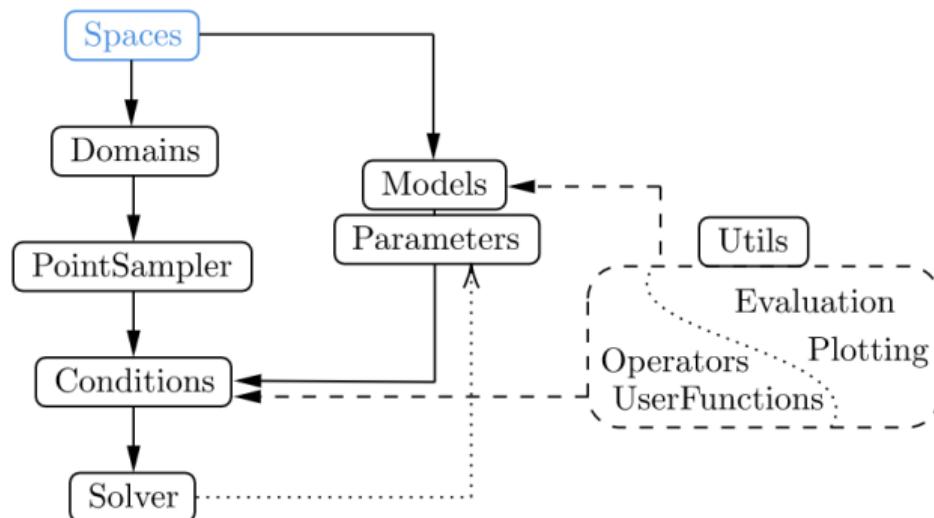
Example:  $\Omega = (0, 1) \times (0, 1)$



$$\begin{aligned}\Delta u(x) &= f(x), && \text{for } x \in \Omega, \\ u(x) &= c, && \text{for } x \in \partial\Omega.\end{aligned}$$

# TORCHPHYSICS

## Spaces



Example:  $\Omega = (0, 1) \times (0, 1)$

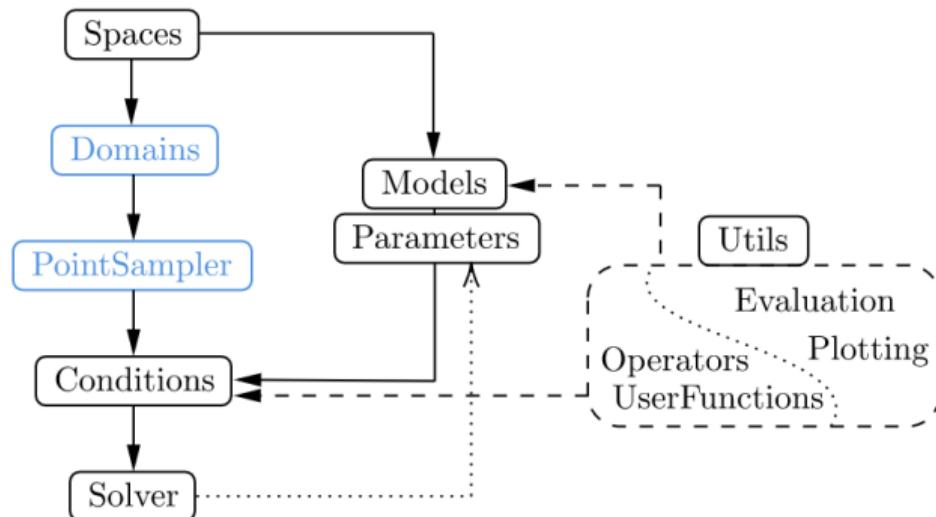
$$\Delta u(x) = f(x), \quad \text{for } x \in \Omega, \\ u(x) = c, \quad \text{for } x \in \partial\Omega.$$

## TORCHPHYSICS Spaces:

- Names for in-/output variables  $x, u$
- Order of function inputs does not matter!

# TORCHPHYSICS

## Domains

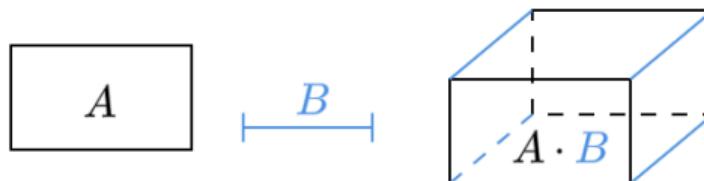


# Domains

- Basic geometries implemented:
  - Point, Interval, Parallelogram, Circle, ...
- Complex domains via logical operators:

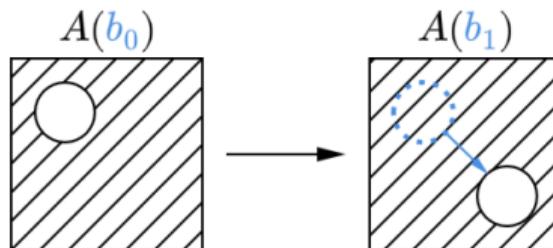


- Higher dimensional domains via Cartesian product:



# Domains

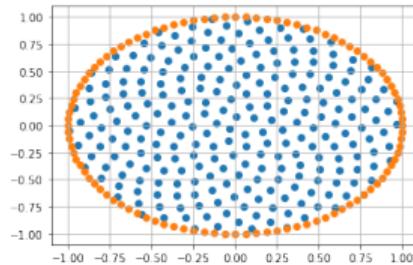
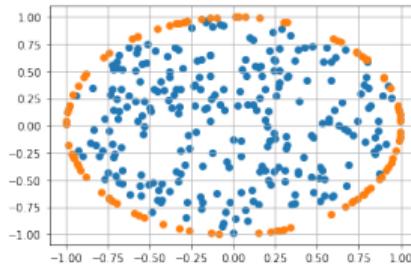
- Domain  $A(b)$  may depend on point  $b$  from domain  $B$   
(e.g. time-dependent)



- Property `.boundary` returns the boundary as a new Domain object

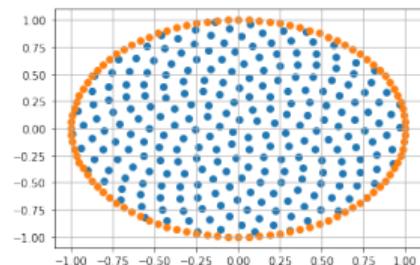
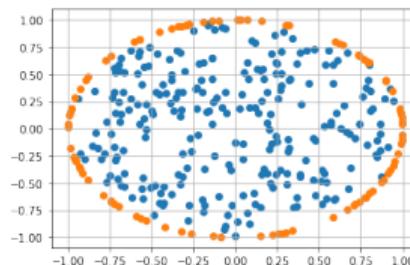
# PointSampler

- Creation of training/validation points inside of the domains
- Different types of sampling:
  - `RandomUniformSampler`, `GridSampler`, `GaussianSampler`,  
`AdaptiveRejectionSampler`, ...



# PointSampler

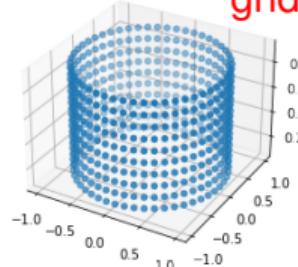
- Creation of training/validation points inside of the domains
- Different types of sampling:
  - `RandomUniformSampler`, `GridSampler`, `GaussianSampler`,  
`AdaptiveRejectionSampler`, ...
- Can be combined as desired



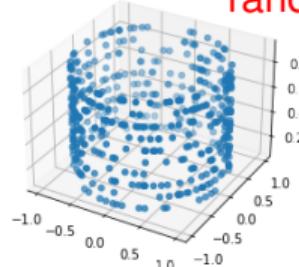
# PointSampler

Flexible domain sampling

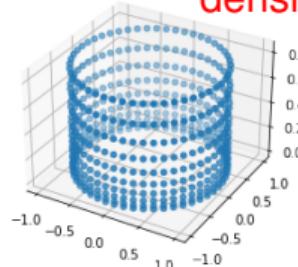
grid



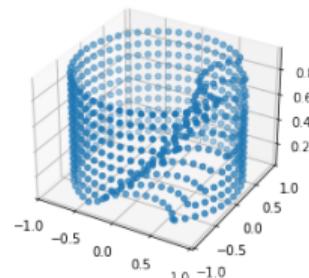
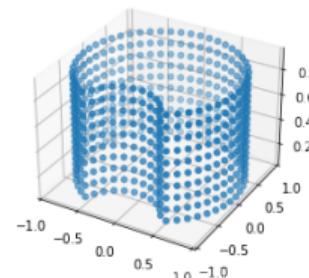
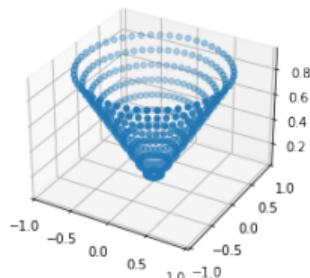
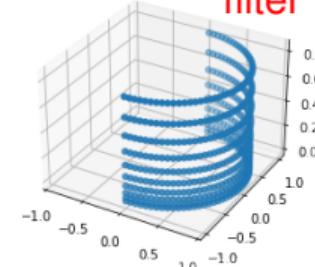
random



density



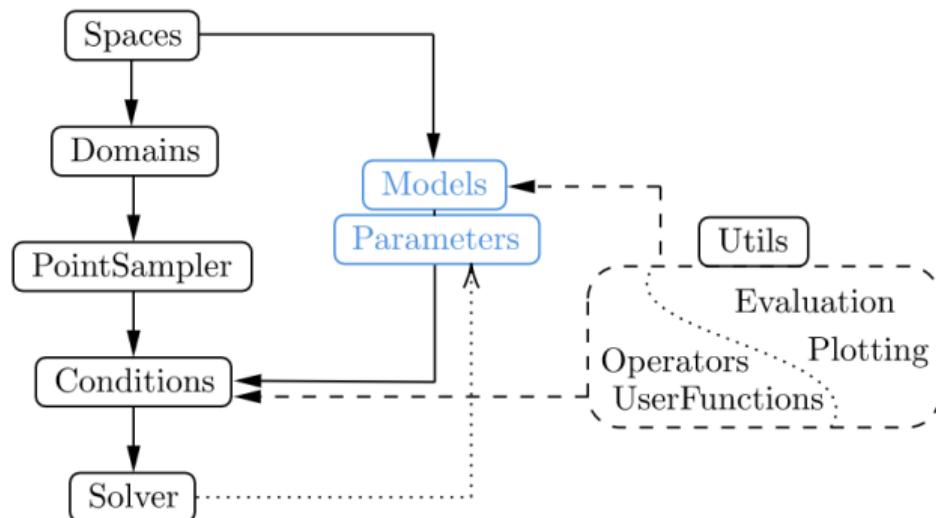
filter



time-dependent domain

# TORCHPHYSICS

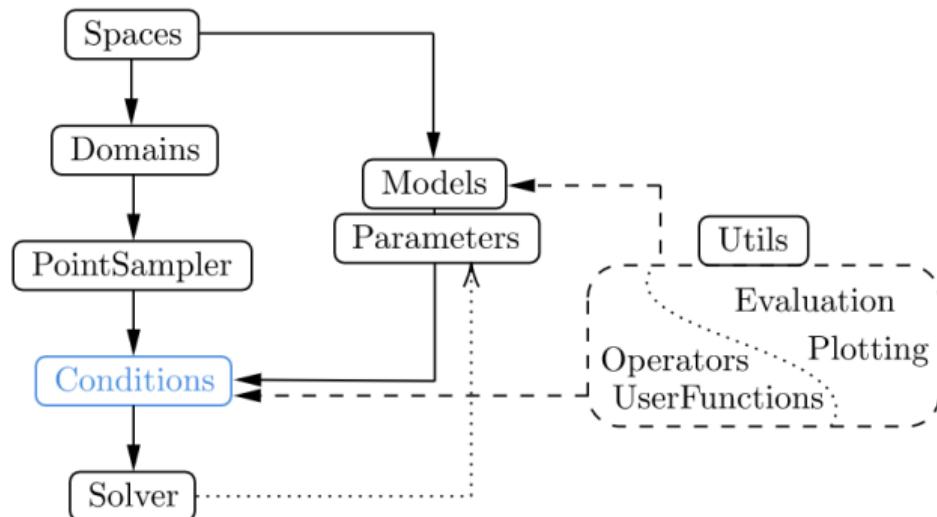
## Neural Networks



- Load any PyTorch model
- Pre-implemented architectures:  
FCN, ResNet, DeepONet,...

# TORCHPHYSICS

## Conditions



# Conditions

Core of TORCHPHYSICS

- Different types, e.g. PINNCondition
- Represents one mathematical condition, e.g.

$$\Delta u(x) = f(x) \text{ in } \Omega \quad \text{or} \quad u(x) = c \text{ at } \partial\Omega$$

# Conditions

Core of TORCHPHYSICS

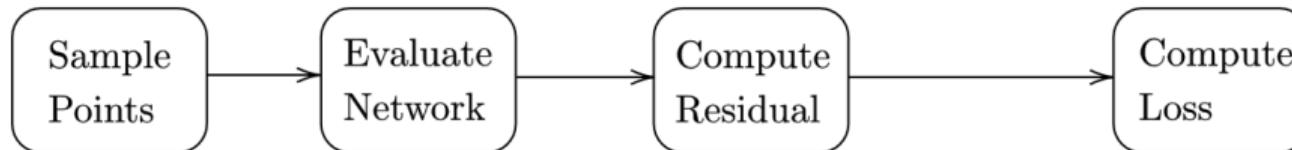
- Different types, e.g. PINNCondition
- Represents one mathematical condition, e.g.

$$\Delta u(x) = f(x) \text{ in } \Omega \quad \text{or} \quad u(x) = c \text{ at } \partial\Omega$$

- Pre-implemented DifferentialOperators available,  
e.g. Laplacian  $\Delta$

# Conditions

## Workflow



### PINNCondition

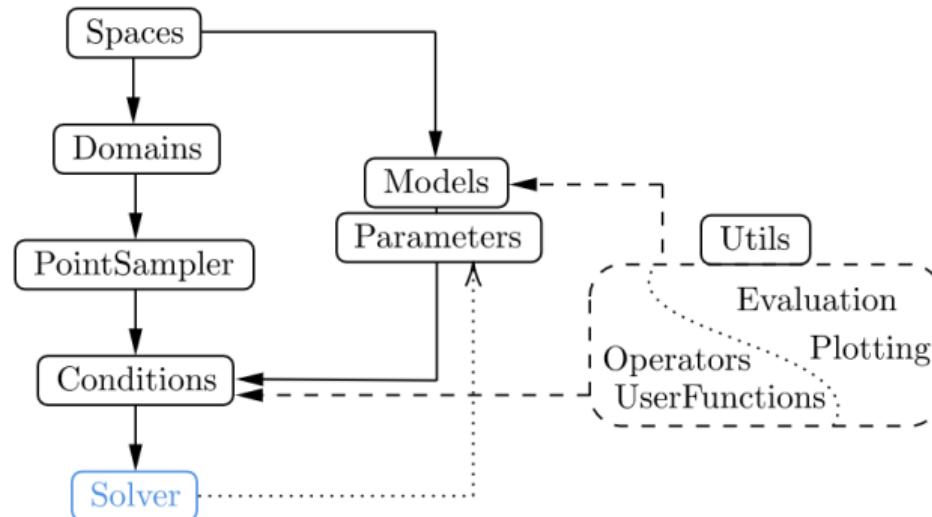
$$x_i \in \Omega \quad u_\theta(x_i) \quad \Delta u_\theta(x_i) - f(x_i) \quad \sum_i |\Delta u_\theta(x_i) - f(x_i)|^2$$

### DataCondition

$$(x_i, u_i) \in \Omega \times \mathbb{R} \quad u_\theta(x_i) \quad u_\theta(x_i) - u_i \quad \sum_i |u_\theta(x_i) - u_i|^2$$

# TORCHPHYSICS

## Solver

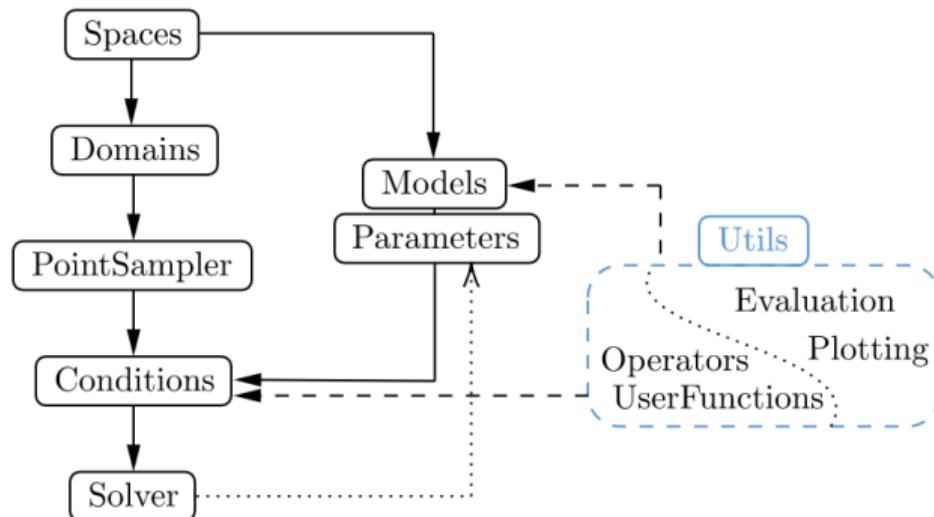


- Collects all conditions  
→ overall loss
- Flexible choice optimization algorithm
-  PyTorch Lightning<sup>2</sup>  
e.g. monitor individual conditions

<sup>2</sup> Falcon et al., *PyTorch Lightning*, 2019

# TORCHPHYSICS

## Utils

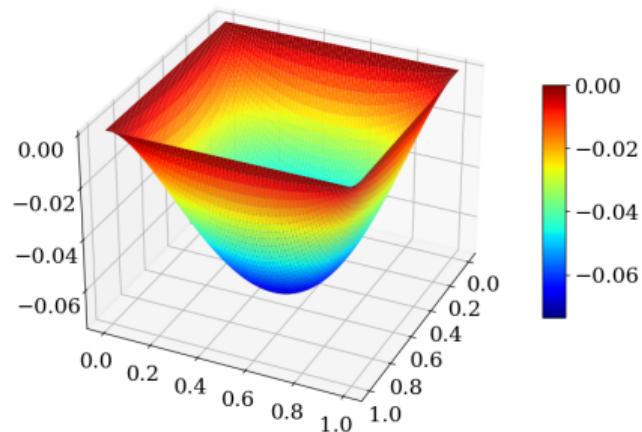


- Provides differential operators  $\Delta, \nabla, \text{div}, \dots$
- Simplifies visualization

## Examples

# Poisson Equation with PINNs

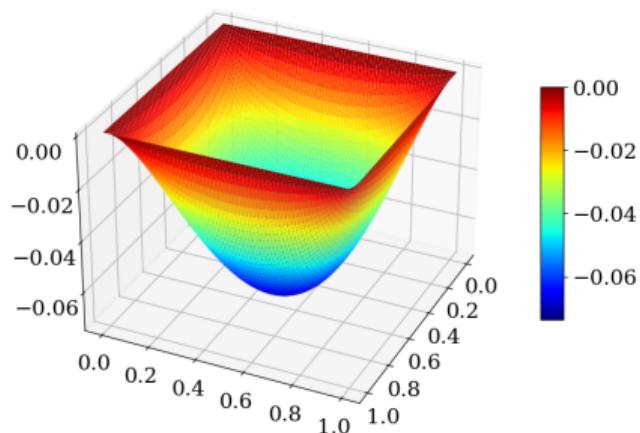
$$\begin{aligned}\Delta u(x) &= f(x) := 1, \quad \text{for } x \in \Omega \\ u(x) &= c := 0, \quad \text{for } x \in \partial\Omega\end{aligned}$$



# Poisson Equation with PINNs

$$\begin{aligned}\Delta u(x) &= f(x) := 1, \quad \text{for } x \in \Omega \\ u(x) &= c := 0, \quad \text{for } x \in \partial\Omega\end{aligned}$$

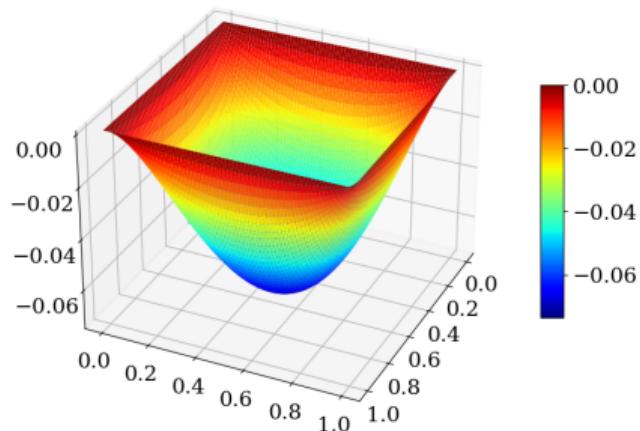
Only  $\approx 20$  lines of code!



- Formulation of PDE
- Training
- Visualization

# Poisson Equation with PINNs

$$\begin{aligned}\Delta u(x) &= f(x) := 1, \quad \text{for } x \in \Omega \\ u(x) &= c := 0, \quad \text{for } x \in \partial\Omega\end{aligned}$$



Only  $\approx 20$  lines of code!

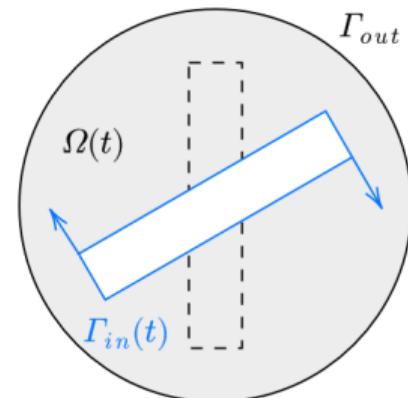
- Formulation of PDE
- Training
- Visualization

→ hands-on session later

# Time-Dependent Domain

PINNs for Navier-Stokes Example<sup>3</sup>

Bar heats up and rotates within some fluid:



---

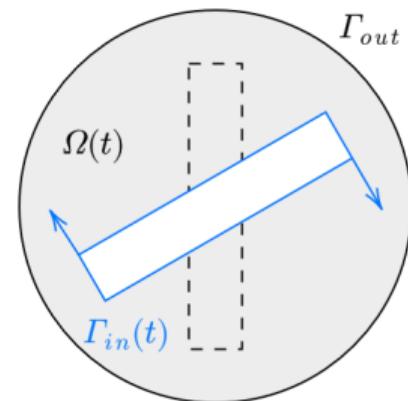
<sup>3</sup> Nganyu et al., *Deep Learning Methods for Partial Differential Equations [...]*, 2023

# Time-Dependent Domain

PINNs for Navier-Stokes Example<sup>3</sup>

Bar heats up and rotates within some fluid:

$$\begin{aligned}
 \partial_t u + (u \cdot \nabla) u &= \nu \Delta u - \nabla p, && \text{in } \Omega(t), \\
 \partial_t T + u \cdot \nabla T &= \lambda \Delta T, && \text{in } \Omega(t), \\
 \nabla \cdot u &= 0, && \text{in } \Omega(t), \\
 u(0, \cdot), p(0, \cdot), T(0, \cdot) &= 0, 0, 270 && \text{in } \Omega(0), \\
 u, T &= (0, 0), 270, && \text{on } \Gamma_{\text{out}}, \\
 u, T &= u_{\text{in}}(t), T_{\text{in}}(t), && \text{on } \Gamma_{\text{in}}.
 \end{aligned}$$

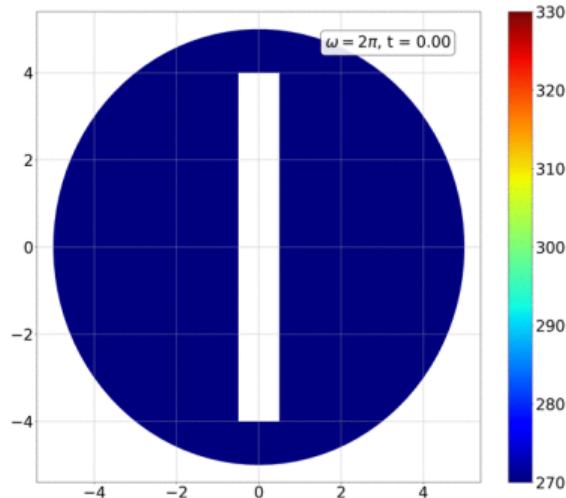



---

<sup>3</sup> Nganyu et al., *Deep Learning Methods for Partial Differential Equations [...]*, 2023

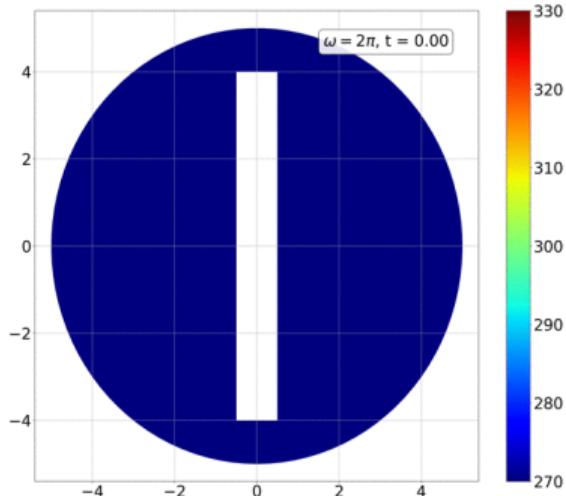
# Time-Dependent Domain

## Learned Temperature Field



# Time-Dependent Domain

## Learned Temperature Field



**Thank you for your attention!**

- Questions?

# Getting Ready for Hands-on Sessions...

## Email on November 18:

- Link to our GPU-servers: <server\_name>.math.uni-bremen.de
- Username: komsoXX
- Password: ...

# Getting Ready for Hands-on Sessions...

## Email on November 18:

- Link to our GPU-servers: <server\_name>.math.uni-bremen.de
- Username: komsoXX
- Password: ...

**Now: 5 min. break**