UNIVERSITÀ DEGLI STUDI FIRENZE

DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

DISIT DISTRIBUTED SYSTEMS AND INTERNET TECHNOLOGIES LAB
DISTRIBUTED DATA INTELLIGENCE AND TECHNOLOGIES LAB

https://www.snap4City.org

SNAP4CITY KM4CITY

NeuroSymbolic Artificial Intelligence at Scale

Marco Fanfani, marco.fanfani@unifi.it

https://www.disit.org/

Parte: 2.4 (2025-26) Physics-Informed Neural Networks
PINN Framework

Physics-Informed Neural Networks (PINNs)

Solving Partial Differential Equations through Scientific Machine Learning

PINN Framework

The PINN Framework: An Overview

The Core Objective:

- **Network Goal:** Train a neural network N that maps coordinates $(\mathbf{x}, t)$ to physical states $\mathbf{s}$.
- **Physics Integration:** Exploit governing equations directly, reducing reliance on massive labeled datasets.
- **Surrogate Solution:** Construct a surrogate $\hat{y}(\mathbf{x}, t;\ \theta) = \mathbf{s}$, i.e. a proxy of the numerical solvers, optimized to respect physical laws.

Generalized PINN Architecture

The Computational Pipeline:

1. **Inputs:** Spatial coordinates (x, y, z) and time (t).
2. **Neural Network:** Fully connected layers produce estimated output $\hat{y}$.
3. **AD Module:** Computes first/second-order derivatives w.r.t. input coordinates.
4. **Loss Component:** Derivatives are fed into PDEs to calculate physical residuals.

Transforming Simulation into Optimization

- The network parameters are determined by minimizing a composite loss function:

$$L(\Theta) = w_f L_f + w_{ic} L_{ic} + w_{bc} L_{bc}$$

- Components:
  - L_f **Physics Loss:** Enforces the governing **differential equations**.
  - L_{ic} / L_{bc}**:** Ensures solution respects **initial/boundary** conditions.
  - **Optimizers:** Algorithms like Adam (global) and L-BFGS (local) minimize the residuals.

Mathematical Formulation of a PDE Problem

- To solve a physical problem, we must define:
  - **Governing Equations:** The specific PDE (e.g., Navier-Stokes, Heat Eq).
  - **Domain (Ω):** The physical space where the phenomenon occurs.
  - **Initial Conditions (IC):** The state of the system at $t = 0$.
  - **Boundary Conditions (BC):** Constraints on edges (e.g., No-slip walls).

The Synergy of NN and AD

Why they work together:

- Neural networks are **differentiable by design** when using smooth activation functions
  - Because the network is continuous and smooth, you can mathematically ask, "If I change the input x by a tiny amount, how much does the output u change?"
  - Since physics is defined by changes (rates of change, acceleration, gradients), a network that is "differentiable by design" is naturally equipped to represent physical systems.

- AD provides exact partial **derivatives at machine precision**, avoiding the truncation and round-off errors of finite difference methods. Also, no mesh required!

By combining the smooth network with exact AD, you create **a direct bridge between the model's guess and the laws of physics**. If the network predicts a velocity field for water, you can instantly use AD to check if that field satisfies mass conservation. If it doesn't, the *physics error* is sent back into the network to correct the weights.

The Physics Residual Mechanism

Penalizing Non-Physical Behavior:

- The **Residual Network** calculates how much the network's output deviates from the governing PDE.
  - In other word... the Residual Network is the implementation of the AD on the specific PDE of the physical problem
  - The Residual Net is not trained, it only compute the residual
- **Virtual Evaluation**: Because AD works at any point, the physics can be evaluated at "virtual points" (collocation points) where no training data exists .
- **Penalty System**: If the predicted fields violate the physics (e.g., non-zero divergence in a velocity field), the loss function increases, penalizing the network and guiding it back toward a physically consistent state .

The Multi-Objective Loss Function

The Core Logic:

- PINN training is a **multi-objective optimization problem**.

$$L(\Theta) = w_f L_f + w_{ic} L_{ic} + w_{bc} L_{bc}$$

- Weighted Summation: weights (w_i) determine the relative importance of physics residuals *vs* boundary constraints.
- Finding the right weights can be difficult

The Physics Residual Term L_f

1. Feed $(\mathbf{x}, t)$ to get predicted states $\hat{y}$.
2. Use AD to compute derivatives (e.g., $\frac{\partial \hat{y}}{\partial t}$, $\nabla^2 \hat{y}$).
3. Substitute into PDE to calculate residual f.
   - L_f is the Mean Squared Error (MSE) evaluated over N_f collocation points. Usually in the order of thousands scattered over the domain

$$L_f = \frac{1}{N_f} \sum_{i=1}^{N_f} \| f(\mathbf{x}_i, t_i; \hat{y}, \boldsymbol{\nabla} \hat{y}) \|^2$$

PDE as residual function

To properly use the PDEs into the residual function f, the PDE must be transformed into a "penalty" for the neural network.

1. **Zero-Equation Formulation**: the first step is to rearrange the governing PDE so that all terms are moved to one side, setting the entire expression equal to zero

$$\rho\left(\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u}\cdot\nabla\mathbf{u}\right) = -\nabla p + \mu\,\nabla^2\mathbf{u} + \mathbf{f}$$

$$\Downarrow$$

$$\mathbf{r} := \rho\left(\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u}\cdot\nabla\mathbf{u}\right) + \nabla p - \mu\,\nabla^2\mathbf{u} - \mathbf{f} = \mathbf{0}$$

PDE as residual function

To properly use the PDEs into the residual function f, the PDE must be transformed into a "penalty" for the neural network.

2. **Definition of the residual function**: The function f represents the physics residual. It measures how much the neural network's current approximation, $\hat{u}(x, t; \theta)$ violates the physical law at any given point

$$f\left(x, t, \frac{\partial \hat{y}}{\partial x}, \frac{\partial \hat{y}}{\partial t}, \ldots; \Psi\right) = 0$$

Input

Partial derivative of output

Physical parameters, e.g., viscosity, Re, ...

PDE as residual function

To properly use the PDEs into the residual function f, the PDE must be transformed into a "penalty" for the neural network.

1. **Forward pass**: use input (x, t) (i.e., collocation points) to obtain the prediction $\hat{y}$

$$f\left(x, t, \frac{\partial \hat{y}}{\partial x}, \frac{\partial \hat{y}}{\partial t}, \ldots; \Psi\right) = 0$$

PDE as residual function

To properly use the PDEs into the residual function f, the PDE must be transformed into a "penalty" for the neural network.

2. Use **AD** to compute the partial derivative of output w.r.t. the input

$$f\left(x, t, \frac{\partial \hat{y}}{\partial x}, \frac{\partial \hat{y}}{\partial t}, \ldots; \Psi\right) = 0$$

PDE as residual function

To properly use the PDEs into the residual function f, the PDE must be transformed into a "penalty" for the neural network.

$$f\left(x, t, \frac{\partial \hat{y}}{\partial x}, \frac{\partial \hat{y}}{\partial t}, \ldots; \Psi\right) = 0$$

3. **Compose the f function** and obtain the residual for each collocation point

Enforcing Initial Conditions L_{ic}

- Defines the state at $t = t_0$
- This term serves as an "anchor," ensuring the transient solution evolves from the correct starting point
- Constraint: $\hat{y}(x, t_0)$ must match initial state $h(x)$.

$$L_{ic} = \frac{1}{N_{ic}} \sum_{i=1}^{N_{ic}} \|\hat{y}(x_i, t_0) - h(x_i)\|^2$$

Enforcing Boundary Conditions L_{bc}

- Boundary conditions (BCs) define how the system interacts with its environment at the domain edges:
- Types of Enforced BCs:
  - **Dirichlet:** Fixed values (e.g., constant inlet velocity).
  - **Neumann:** Fixed derivatives (e.g., zero pressure gradient at an outlet).
  - **No-Slip:** Forcing velocity to zero at solid walls (fundamental for Navier-Stokes)

$$L_{bc} = \frac{1}{N_{bc}} \sum_{i=1}^{N_{bc}} \|\mathcal{B}\hat{y}(x_i, t_i) - g(x_i, t_i)\|^2$$

Enforcing Boundary Conditions L_{bc}

- Boundary conditions (BCs) define how the system interacts with its environment at the domain edges:

$\mathcal{B}$ is the boundary condition operator. It defines the mathematical rule or constraint that must be applied to the neural network's prediction ($\hat{y}$) at the domain boundaries ($\partial\Omega$).
E.g., for no-slip, $\mathcal{B}$ is an operator that select the velocities from the output $\hat{y}$.

$g(x_i, t_i)$ is the value $\mathcal{B}\hat{y}$ must match at the boundaries ($\partial\Omega$). E.g. for no-slip, $g(x_i, t_i) = 0$

$$L_{bc} = \frac{1}{N_{bc}} \sum_{i=1}^{N_{bc}} \| \mathcal{B}\hat{y}(x_i, t_i) - g(x_i, t_i) \|^2$$

The Hybrid Nature of PINNs

- In addition to physical and constraints losses, PINN can also use **labeled data** as traditional NN learning, **fusing data-driven learning with physical constraints**

$$L(\Theta) = w_f L_f + w_{ic} L_{ic} + w_{bc} L_{bc} + w_d L_d$$

- Labelled data provides a "head start," guiding the optimizer toward the global minimum more quickly than physics alone
- Labelled data are hard to obtain from real world measurements
- Sometimes classical CFD can be used to obtain some examples

Full PINN Architecture

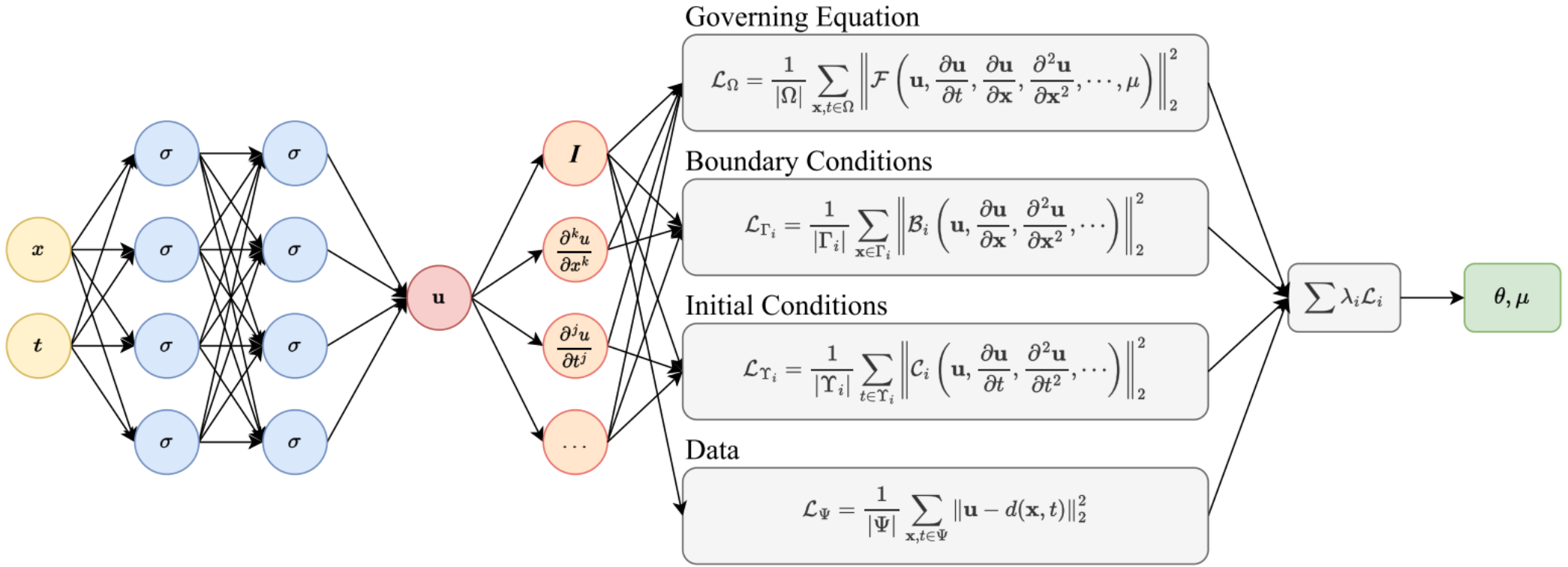


Image from *R. Bischof, M.A. Kraus, "Multi-Objective Loss Balancing for Physics-Informed Deep Learning," Computer Methods in Applied Mechanics and Engineering, Volume 439, 2025, 117914, ISSN 0045-7825, https://doi.org/10.1016/j.cma.2025.117914*.

Challenges in Loss Optimization

Problem of composing the PINN loss:

- **Linear Scalarization**: L_f (from physics) and L_{ic}, L_{bc} (from known constraint) or loss from additional examples may have **vastly different numerical scales**. The loss with the highest value will dominate the optimization.

Challenges in Loss Optimization

Problem of composing the PINN loss:

- **Numerical Stiffness**: Stiffness describes a situation in which the optimization algorithm struggles to find the minimum because the different terms of the loss "pull" in opposite directions or have gradients with incompatible magnitudes.
  - Conflicting gradients: The gradient that reduces L_f could dramatically increase L_{ic} or L_{bc}. This creates a very narrow and steep valley in the landscape of the loss function.
  - Differential operators: Because physics is imposed by derivatives (often high-order), the term L_f is inherently more complex to optimize than a simple mean squared error on the data/constraints.

Challenges in Loss Optimization

Automatic Weighting: To avoid manual hyperparameter tuning, advanced schemes like **ReLoBraLo (Relative Loss Balancing with Random Lookback)** are used to dynamically adjust w_i based on the gradient history of each term.

Result: A balanced loss ensures the network satisfies the PDE without sacrificing boundary accuracy

ReLoBraLo

- **Historical Comparison**: ReLoBraLo looks back at the performance of each sub-loss in previous iterations.
- **Relative Magnitude**: It adjusts the weight of a specific loss term relative to the others based on how quickly (or slowly) that term is converging.
- **Adaptive Nature**: This "dynamic weight strategy" is essential for the convergence of complex systems like the Navier-Stokes equations

ReLoBraLo

Example: suppose to have this loss

$$L = w_f L_f + w_{bc} L_{bc}$$

- The network quickly learns that the velocity at the walls must be zero (L_{bc} goes down quickly). However, the physical residue of the equations of momentum (L_f) remains very high because calculating gradients with AD is computationally difficult.
- **ReLoBraLo intervention**: The algorithm compares the historical trend. Note that L_{bc} decreased by 90%, while L_f only 2%.
- **Weight adjustment**: ReLoBraLo automatically increments w_f (e.g. from 1 to 20) and reduces w_{bc}
- The optimizer (e.g. Adam) receives a **much stronger signal from the physical term**. This "forces" the network to stop focusing only on the edges and start correctly modelling the dynamics of the fluid within the domain.
- This avoid trial-and-error hyperparameter search, as the **network self-balances** during the process

ReLoBraLo

- How it works: given multi-objective loss as

$$L(t) = \sum_{i=1}^{N} \lambda_i(t) \cdot L_i(t)$$

1. **Relative Balance Calculation**: It calculates the relative improvement of each term i between two time steps (t and t')

$$\lambda_{bal,i}(t, t') = N \frac{\exp\left(\frac{L_i(t)}{\tau L_i(t')}\right)}{\sum_{j=1}^{N} \exp\left(\frac{L_j(t)}{\tau L_j(t')}\right)}$$

ReLoBraLo

- How it works: given multi-objective loss as

$$L(t) = \sum_{i=1}^{N} \lambda_i(t) \cdot L_i(t)$$

2. **Historical Update with Random Lookback**: It incorporates a Bernoulli random variable (ρ) to decide whether to look back at the previous step or the initial state

$$\lambda_{hist,i}(t) = \rho\lambda_i(t-1) + (1-\rho)\lambda_{bal,i}(t,0)$$

ReLoBraLo

- How it works: given multi-objective loss as

$$L(t) = \sum_{i=1}^{N} \lambda_i(t) \cdot L_i(t)$$

1. **Final Scaling Update**: An exponential decay integrates the historical trend with the immediate relative progress

$$\lambda_i(t) = \alpha \lambda_{hist,i}(t) + (1 - \alpha)\lambda_{bal,i}(t, t-1)$$

ReLoBraLo

- τ (**Temperature**): Influences the "sharpness" and magnitude of the scalings.
  - $\tau \to \infty$: Weights become uniform (all $\lambda_i = 1$).
  - $\tau \to 0$: The system assign all weight to the term with the lowest relative progress and zero to others.
- $E[\rho]$ (**Expected Saudade**): Determines the frequency of looking back to the very start of training ($L_i(0)$)
  - $E[\rho] \approx 1$: Minimum "saudade"; the model mostly looks at the last value in history.
  - Lower $E[\rho]$: Frequent full lookbacks help the model "remember" the total progress made since $t = 0$, which is particularly helpful for escaping local minima in complex PDEs
- α (**Exponential Decay Rate**): Controls the balance between past information and current updates.
  - High α (0.9 to 0.999): Provides a smoother curve by giving more weight to past loss statistics, which helps "remember" deteriorations longer.
  - Low α: Increases stochasticity and the model's flexibility to react to sudden changes

$$L(t) = \sum_{i=1}^{N} \lambda_i(t) \cdot L_i(t)$$

1. $\lambda_{bal,i}(t, t') = N \dfrac{\exp\left(\frac{L_i(t)}{\tau L_i(t')}\right)}{\sum_{j=1}^{N} \exp\left(\frac{L_j(t)}{\tau L_j(t')}\right)}$
2. $\lambda_{hist,i}(t) = \rho\lambda_i(t-1) + (1-\rho)\lambda_{bal,i}(t, 0)$
3. $\lambda_i(t) = \alpha\lambda_{hist,i}(t) + (1-\alpha)\lambda_{bal,i}(t, t-1)$

ReLoBraLo

- Let's imagine a PINN with m=2 (two terms: PDE and Boundary Conditions) and suppose that between two iterations the relative progress (ri=Li(t)/Li(t')) is:
  - Term 1 (PDE): r1=0.8 (improving well).
  - Term 2 (BC): r2=1.1 (is getting worse or stalled).

- Case 1: High Temperature (**τ=10**) - Uniform Balance: If **we set a high τ**, the exponent becomes very small, flattening the differences:
  - Exponent 1: 0.8/10=0.08⟹e^0.08≈1.083
  - Exponent 2: 1.1/10=0.11⟹e^0.11≈1.116
  - λbal,2=2·(1.116/(1.083+1.116))≈1.015
  - λbal,1=2·(1.083/(1.083+1.116))≈0.985
  - Result: The weights are almost equal (~1.0). The network treats terms almost the same despite one getting worse.

- Case 2: Low Temperature (**τ=0.1**) - Aggressive Balance: If **we lower τ**, the system becomes much more sensitive to differences in progress:
  - Exponent 1: 0.8/0.1=8⟹e8≈2.981
  - Exponent 2: 1.1/0.1=11⟹e11≈59.874
  - λbal,2=2·(59,874/(2,981+59,874))≈1,905
  - λbal,1=2·(2.981/(2.981+59.874))≈0.095

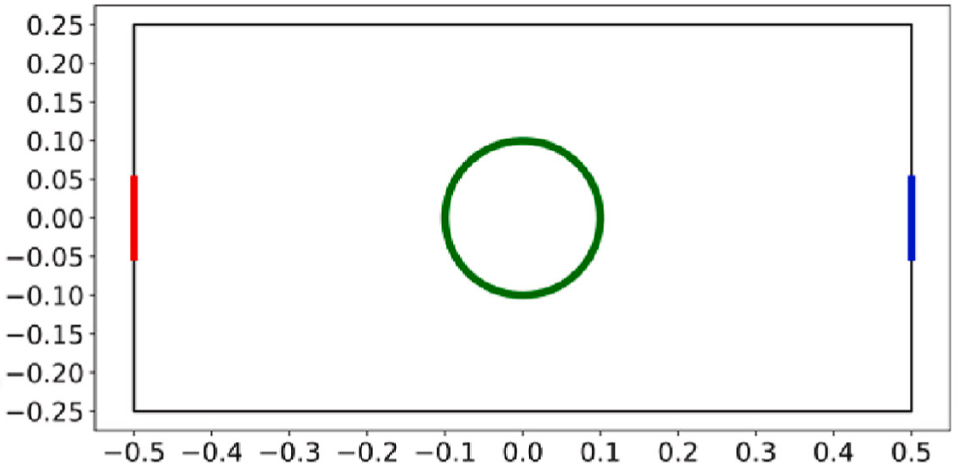
$$\lambda_{bal,i}(t,t') = N \frac{\exp\left(\frac{L_i(t)}{\tau L_i(t')}\right)}{\sum_{j=1}^{N} \exp\left(\frac{L_j(t)}{\tau L_j(t')}\right)}$$

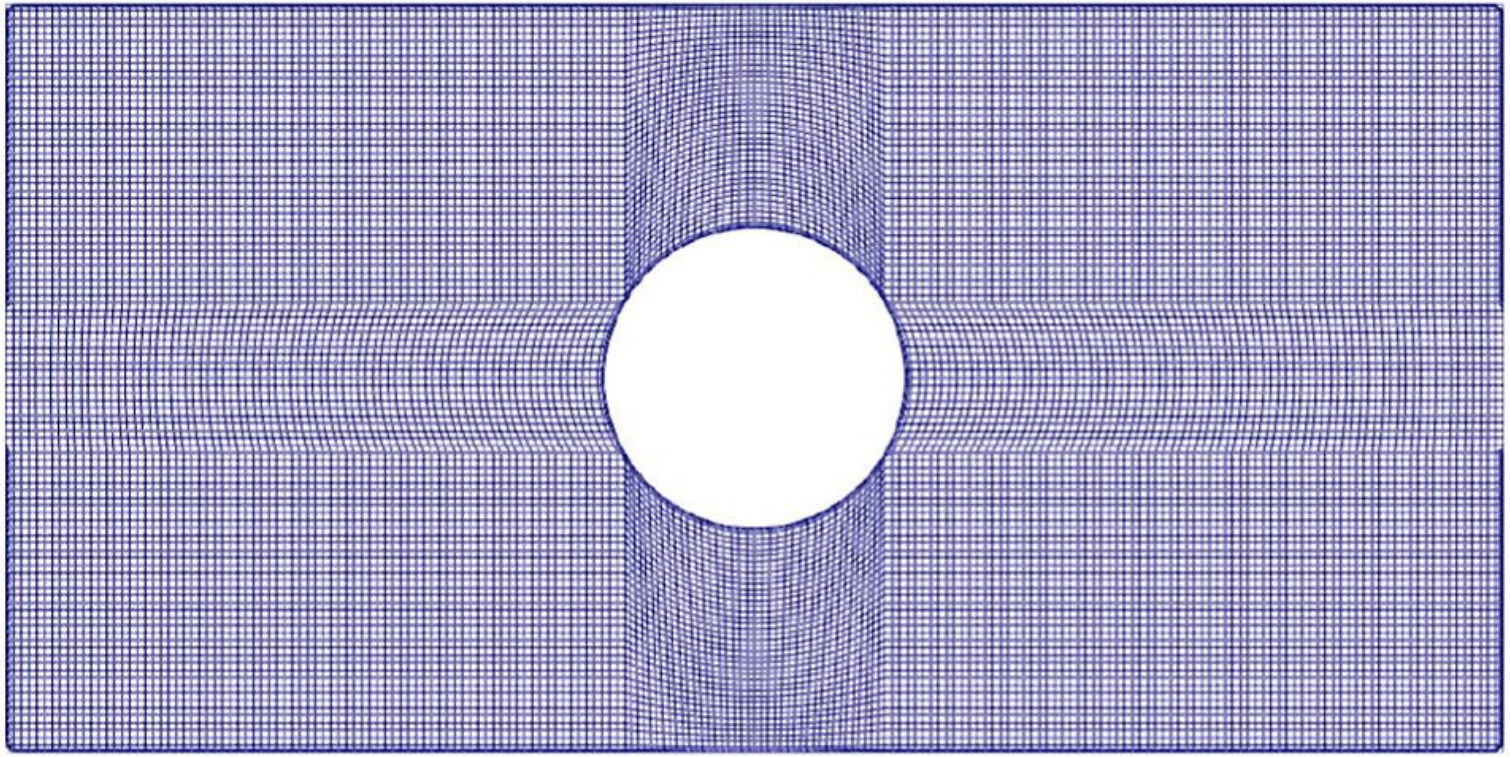
The Mesh-Free Paradigm

Breaking the Grid:

- Traditional CFD methods (FEM/FVM) are *grid-based*, requiring a rigid connectivity of nodes and cells.
- PINN Approach: A "mesh-free" strategy that bypasses the need for manual mesh generation.
- Solutions are **approximated directly in the continuous space-time domain** (or more precisely on a set of sampled points) rather than on discrete elements.
- This significantly **reduces human effort** when dealing with irregular or complex evolving geometries, such as industrial pieces inside a channel

CFD Grid Example





Collocation Points: The 'Virtual' Sensors

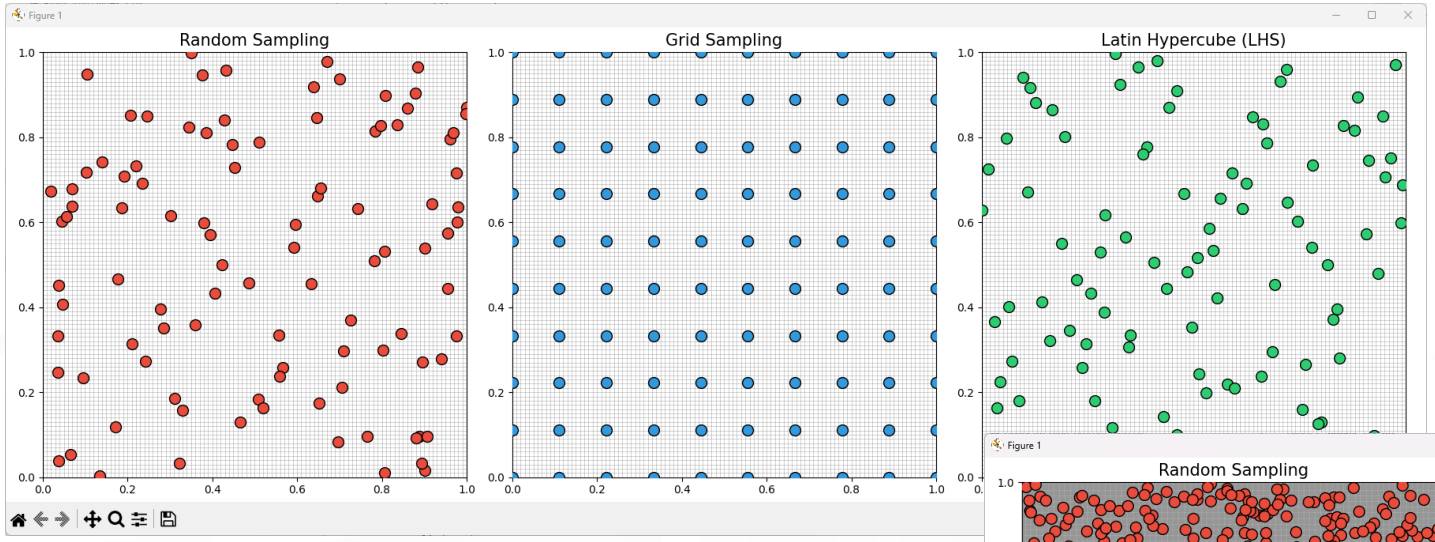
- Instead of fixed mesh nodes, PINNs use scattered collocation points (N_f)
- These are spatial and temporal coordinates $(\mathbf{x}, t)$ where the PDE residual is evaluated.
- The training process forces the network to "obey" physical laws at these specific points throughout the domain.
- Points can be sampled anywhere—inside the domain volume or strictly on the boundaries to satisfy constraints.

Sampling Strategies

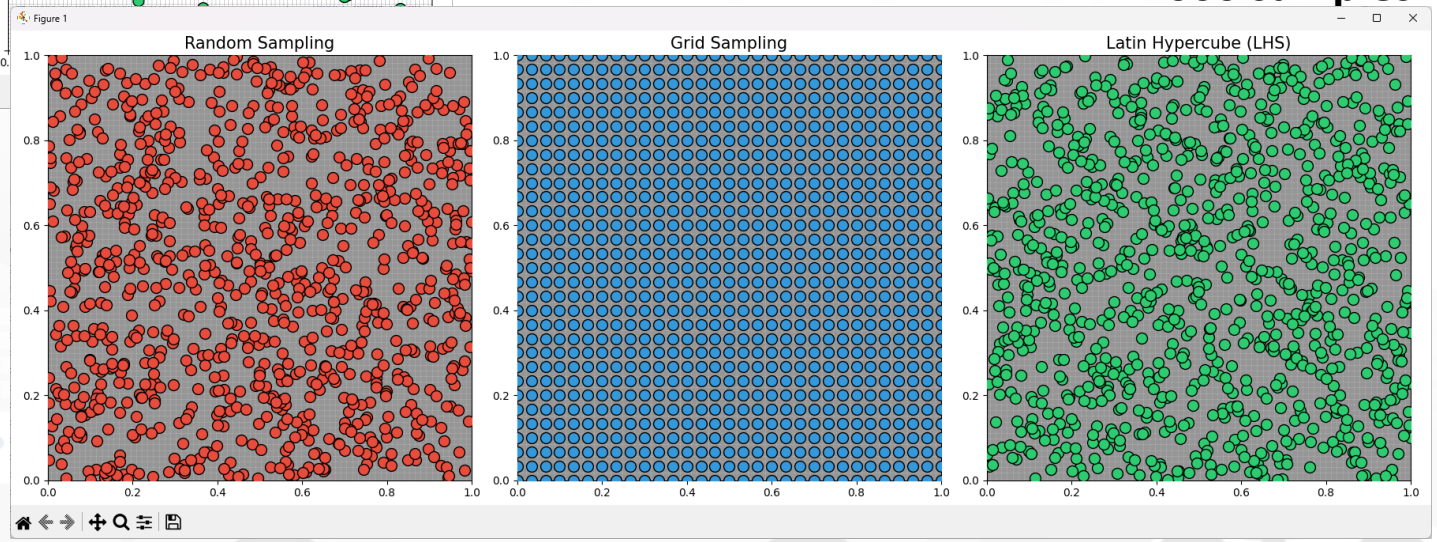
- Sampling strategies can be used:
  - Uniform sampling
  - Grid sampling
  - Latin Hyper-Cube (LHC)
  - etc...
- Informed sampling: more critical areas in the domain requires more points, e.g.
  - boundaries
  - discontinuities

Sampling Strategies

100 samples



1000 samples



Adaptive and Dynamic Resampling

Optimizing Point Distribution:

- Static point sets may fail to capture **transient waves** or **sharp discontinuities** like shock waves.
- **Dynamic Resampling**: Points are shifted or re-sampled during training to focus on areas where the physics residual is highest.
- **Multi-Resolution Learning**: A strategy where training starts with a low number of points to learn coarse features and gradually increases density to capture fine details.
  - This "curriculum-style" approach can significantly accelerate convergence toward a stable steady-state solution

The Trade-off: Accuracy vs. Cost

Balancing Resources:

- **Accuracy**: Increasing the number of collocation points generally reduces the Mean Absolute Error (MAE).
- **Time**: More points increase the computational burden per epoch, leading to longer training times.
- A "Mid-level" point density often provides the best compromise: slight accuracy drops for massive gains in training speed.
  - *In experiments carried out at the DISIT lab we found that reducing points can decrease training time from 7 hours to less than 3 hours while maintaining acceptable engineering tolerance*

Solving Forward Problems with PINNs

Definition of the **Forward Problem**:

- In a forward (or direct) problem, the **governing physical laws** (PDEs) and their **parameters** (e.g., viscosity, conductivity) are fully **known**.
- The goal is to **determine the hidden state** $u(\mathbf{x}, t)$ of the system across the entire domain.
- The PINN Role: The neural network acts as a **continuous solver** that approximates the solution by minimizing the residuals of the known equations.
- Unlike traditional solvers that discretize the domain, **PINNs provide a functional approximation** of the physics.

The Objective: Data-Driven Solutions

How PINNs Solve PDEs:

- **Input-Output Mapping**: The network maps coordinates $(\mathbf{x}, t)$ directly to the predicted physical variables $\hat{u}$
- **Minimal Data Requirement**: While traditional ML requires massive datasets, a PINN forward solver can often function with **only boundary and initial condition data**, using the physics residual to "fill in" the rest of the domain.
- This makes PINNs **highly effective** for surrogate modelling in complex scenarios.

Inverse Problems (Physics Discovery)

Forward vs. Inverse Problems:

- **Forward Problem**: Computes the system's hidden state (e.g., velocity or pressure) given fixed model parameters and boundary/initial conditions.
- **Inverse Problem**: Aims to discover unknown physical parameters (e.g., viscosity, stiffness) or entirely unknown states from noisy, sparse, or incomplete experimental measurements.
- Physics-Informed Advantage: In the PINN framework, the governing Partial Differential Equation (PDE) acts as a **powerful regularizer, enabling the network to "fill in the gaps"** where data is missing.

Inverse Problems (Physics Discovery)

- Inverse problems in Physics-Informed Neural Networks (PINNs) encompass two closely related tasks:
  - Parameter Identification
  - State Reconstruction

Inverse Problems: Parameter Identification

Finding **specific constants** λ within a PDE (e.g., e.g., viscosity, thermal conductivity)

- Traditional CFD Approach:
  - Iterative Solving: Requires **wrapping the simulator in an optimization loop**. One must guess a parameter, run a full simulation, compare it to data, and repeat.
  - Adjoint Methods: This is a more sophisticated traditional approach where an "adjoint version" of the code is written to calculate the gradient of the error with respect to the input parameters. However, adjoint codes are notoriously **difficult to implement and maintain** for complex fluid dynamics software.
- The PINN Approach:
  - Trainable Variables: Unknown parameters λ are treated exactly like neural network weights (θ). They are **optimized simultaneously during training**.
  - Differentiable Solvers: PINNs leverage Automatic Differentiation (AD) to compute the exact gradient of the error with respect to the unknown physical parameters, **making discovery significantly more efficient than trial-and-error**.

Inverse Problems: State Reconstruction

Also known as Hidden Fluid Mechanics (HFM), this task involves **inferring hidden quantities like 3D velocity and pressure fields from simple visualizations**, such as the spatiotemporal movement of smoke or dye or heat (a "passive scalar").

- **Mechanism**: The PINN is trained on the visualization data while being constrained by the Navier-Stokes equations. It "reads between the lines" to find the quantitative velocity and pressure fields **that are mathematically consistent with the observed trace**.

Why State Reconstruction is More Complex

State reconstruction represents a **much more difficult challenge** than simple parameter identification:

- Function vs. Scalar: Parameter identification seeks to find a single constant value. State reconstruction seeks to **discover an entire spatiotemporal field** (a function of space and time).
- Indirect Observations: You do not measure the target variables (velocity/pressure) directly. Instead, **you observe their effects on a passive scalar** and must work backward to find the causes.
- Multiphysics Coupling: It requires solving **coupled nonlinear PDEs simultaneously**. E.g.: the network must satisfy both the transport equation of the smoke/dye and the momentum/mass conservation of the fluid to yield a valid solution.
- Dimensionality: Reconstructing a **3D field over time** significantly **increases computational complexity** compared to identifying a material constant.

Simultaneous learning of states and parameters

- In PINNs, unknown PDE parameters (μ or λ) are treated as trainable variables, just like neural network weights.
- When reconstructing the state (e.g., deriving velocity and pressure from smoke images), the network must satisfy the Navier-Stokes equations.
- If the viscosity is unknown, the network will optimize it along with the velocity fields to ensure that the physical residue of the PDE is close to zero.

Does State Reconstruction include Parameter Estimation?

- Technically, they are two distinct tasks that are often solved together in the same inverse problem:
  - Parameter Identification: Focuses on finding a constant (e.g. viscosity coefficient ν).
  - State Reconstruction: Focuses on the reconstruction of an entire space-time field (e.g. pressure and velocity in 3D) starting from indirect or scattered data.
- In the practice of PINNs, **state reconstruction is such a flexible process that it often inherently includes parameter estimation**.
- If the reconstructed state is to be physically consistent, the parameters governing the physical state must be correct; therefore, the network "discovers" them to minimize the total loss.

Practical Example: Air Flow from Heat Measurements

Industrial Scenario (e.g., Autoclaves or Heat Exchangers):

- Input Data: A point cloud of temperature measurements $T(x, y, z, t)$ acquired over time, which may be sparse or noisy.
- Hidden Latent Variables: The velocity field $[u, v, w]$ and the pressure field p, which are not measured directly.
- The Physical "Bridge": The PINN uses the **heat transport equation** as the primary constraint linking temperature changes to fluid movement:

$$\underbrace{\frac{\partial T}{\partial t} + \mathbf{u} \cdot \nabla T}_{\text{Advection}} = \underbrace{\alpha \nabla^2 T}_{\text{Diffusion}}$$

- Result: By minimizing this residual **alongside Navier-Stokes**, the network "discovers" the velocity and pressure fields that are mathematically consistent with the observed heat distribution.

Architectural Complexity: Multi-Output Systems

As we move to state reconstruction, the PINN architecture evolves from a simple scalar mapper into a **complex multi-variable system**:

- Shared Multi-Output Network:
  - Input: Spatial and temporal coordinates (x, y, z, t).
  - Multiple Outputs: The network must simultaneously predict 5 or more variables: $[\hat{T}, \hat{u}, \hat{v}, \hat{w}, \hat{p}]$.
  - Shared Hidden Layers: These outputs typically **share the initial layers** to capture intrinsic correlations between the scalar transport and the fluid dynamics.
  - **Parameters as trainable variables**: In an inverse problem, parameters such as viscosity (ν), Reynolds number (Re) are not fixed constants, but are treated as optimizable variables like the weights of the network θ
- The Automatic Differentiation (AD) Graph:
  - The AD system must compute complex cross-derivatives of all the outputs w.r.t. the inputs.

Trainable parameters

- Variables, not layers: Instead of being transformations of data (like a dense layer), **parameters are "trainable scalars"** that appear directly in differential equations within physics loss
- Automatic Differentiation (AD): The AD system calculates the gradients of the loss function with respect to everything that is marked as trainable. Then, calculate **∂L/∂θ (for weights)** and **∂L/∂λ (for physical parameters)** at the same time
- Dual optimizer strategy: Since network weights and physical parameters can be orders of magnitude very different, **it is often preferred to use two separate optimizers** to update them independently, ensuring more stable convergence

In summary, the network "discovers" the correct physics by adjusting these parameters until its predictions satisfy both the observed data and the laws expressed by the PDE

Architectural Complexity: Multi-Output Systems

Reconstructing hidden states creates a loss function with many "competing" terms:

- Data Loss (L_{data}): The Mean Squared Error between predicted temperature and the sensor measurements.
- Heat Loss (L_{heat}): The residual of the heat transport equation (must drive toward zero).
- Navier-Stokes Loss (L_{NS}): Residuals for momentum and mass conservation, forcing the velocity and pressure fields to be physically realistic.
- The Balancing Challenge: With so many terms, the optimization becomes difficult → use adaptive schemes like ReLoBRaLo

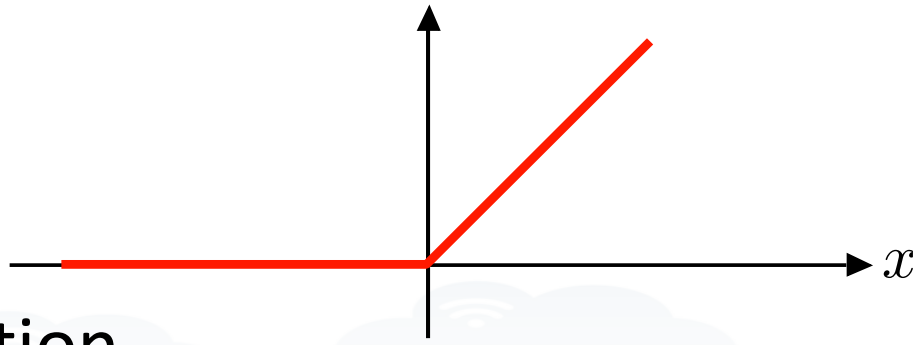
The Crucial Role of Activation Functions

Enabling **Higher-Order Derivatives**:

- In standard Deep Learning, activation functions primarily provide non-linearity for data fitting.
- **The PINN Difference**: Because PINNs rely on Automatic Differentiation (AD) to calculate PDE residuals, the activation functions must be **smooth** and **continuously differentiable**.
- If an activation function is not sufficiently smooth (e.g., C^2 continuity for second-order PDEs), the physical residuals cannot be accurately computed, leading to training failure.

Why ReLU is Often Unsuitable for PINNs

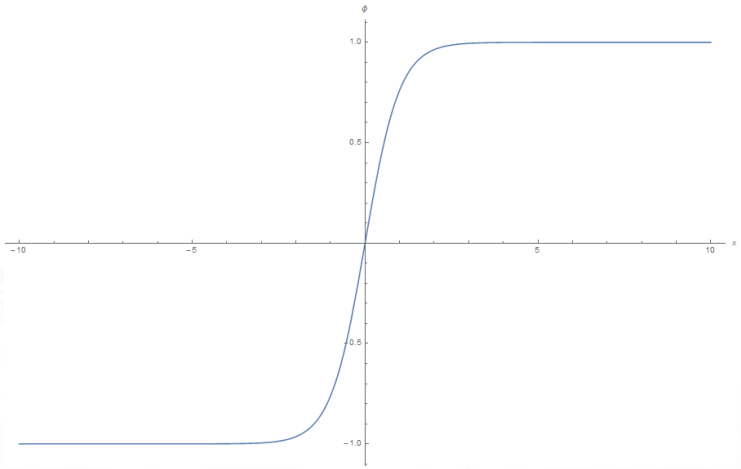
The Limits of Piecewise Linearity:



- **ReLU** (Rectified Linear Unit) is the "standard" in many ML tasks but poses significant risks for PINNs.
- ReLU is a **non-smooth**, **piecewise linear** function that is **not differentiable at zero**.
- For PDEs like Navier-Stokes that require second-order derivatives (Laplacian Δu), the second derivative of ReLU is **zero everywhere except at the origin, where it is undefined**.
- Using ReLU can lead to inaccurate, suboptimal solutions or a **complete inability to learn** the underlying physics

Smooth Alternatives

- **Hyperbolic Tangent** (Tanh): A traditional choice for PINNs due to its infinite differentiability and smooth "S" shape.



- Limitation: Tanh is **bounded**, which can lead to **vanishing gradient** problems in deep networks or when modelling large dynamics.

Smooth Alternatives

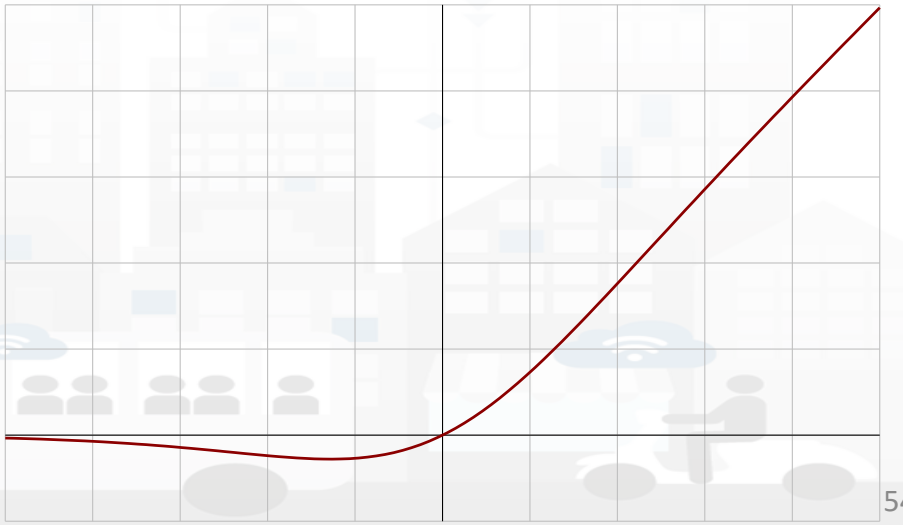
- **SiLU/Swish:**

$$\textbf{GELU: } \mathbf{f(x)} = \frac{\boldsymbol{x}}{\mathbf{1} + \boldsymbol{e}^{-\alpha x}}$$

$$\mathbf{f(x)} = \boldsymbol{xP(X \leq x)} \approx \mathbf{0.5}\boldsymbol{x}\left(\mathbf{1} + \tanh\left[\sqrt{\mathbf{2}/\boldsymbol{\pi}}\left(\boldsymbol{\alpha x} + \mathbf{0.044715}(\boldsymbol{\alpha x})^{\mathbf{3}}\right)\right]\right)$$

  - P is the Gaussian function

- Advantage: It is **smooth** and **unbounded** for positive values and provides better gradients for complex physical representations.



Adaptive Activation Functions

- Standard activations are static. Adaptive activations introduce **learnable parameters** (α) that allow the network to adjust the slope or "aggressivity" of the function during training.
- Acceleration: These functions are proven to **accelerate convergence** in deep PINN architectures.
- In our tests **Adaptive GELU** achieved the best trade-off between high accuracy and faster training times for velocity estimates.

Optimization

- The loss landscape for PINNs is often **highly non-convex** and "stiff," making it difficult for standard optimizers to reach the global minimum without **getting stuck in poor local minima**.
- Modern PINN training typically employs a **hybrid approach**:
  - start with a first-order optimizer for global search
  - switch to a second-order optimizer for local refinement

Optimization - Global Search

- As first-order optimizer for global search **Adam** can be used in the initial phase of training to provide a **fast coarse optimization**
- Strengths:
  - Stochasticity: Handles noisy gradients effectively, which is common when using randomly sampled collocation points.
  - Resampling: Unlike second-order methods, Adam allows for dynamic resampling of collocation points (e.g., via Latin Hypercube Sampling) in every epoch to improve generalization.
- Limitations: It can struggle to achieve high-precision convergence (machine precision) and often plateaus as it nears the minimum.

Optimization - Local Refinement

- As second-order optimizer for local refinement L-BFGS can be used
- It is typically activated after Adam has sufficiently lowered the loss to refine the results and reach high accuracy.
- Key Characteristics:
  - It generally requires the collocation points to be fixed (no resampling) throughout its optimization steps.
  - It provides significantly higher accuracy for smooth functions, "polishing" the solution found by Adam.
  - It is computationally more expensive and memory-intensive per iteration compared to Adam.

Optimization - Best practices

- Learning Rate Decay: In the Adam phase, multiplicative factor reductions (e.g., 0.1) are often applied when progress plateaus for several thousand steps.
- Collocation Sampling: Re-sampling during the Adam phase helps the network "see" the entire domain, improving its representation of the PDE.
- Initial Loss Training: For stiff systems (e.g., Allen-Cahn), it can be beneficial to train on initial conditions alone for several thousand epochs before introducing the full physics loss.
- Early Stopping: Monitoring validation progress is critical to prevent overfitting to specific point sets and to stop training once global progress halts.

Spectral Bias

- Neural networks exhibit a "spectral bias," meaning they naturally **prioritize learning low-frequency** components of a function first.
- In many PDEs, such as those governing high-speed fluid flows or acoustics, **critical information exists in high-frequency** details (e.g., sharp gradients or complex waves).
- Standard Multi-Layer Perceptrons (MLPs) often **struggle to resolve these fine-scale physical structures**, leading to stalled training or physically incomplete solutions despite a decreasing loss.

The Modified Fourier Network (MFN)

- MFN is specifically designed to **overcome spectral bias** and **capture high-frequency functions** more effectively than vanilla MLPs.
- It projects low-dimensional input coordinates into a **high-dimensional feature space** using sinusoidal encoding functions.
- Key Components:
  - Input Encoding Layer (F): Maps spatial/temporal inputs to Fourier features.
  - Transformation Layers: Two additional layers (W_{T1}, W_{T2}) that further process these features to improve convergence and final accuracy.

The Modified Fourier Network (MFN)

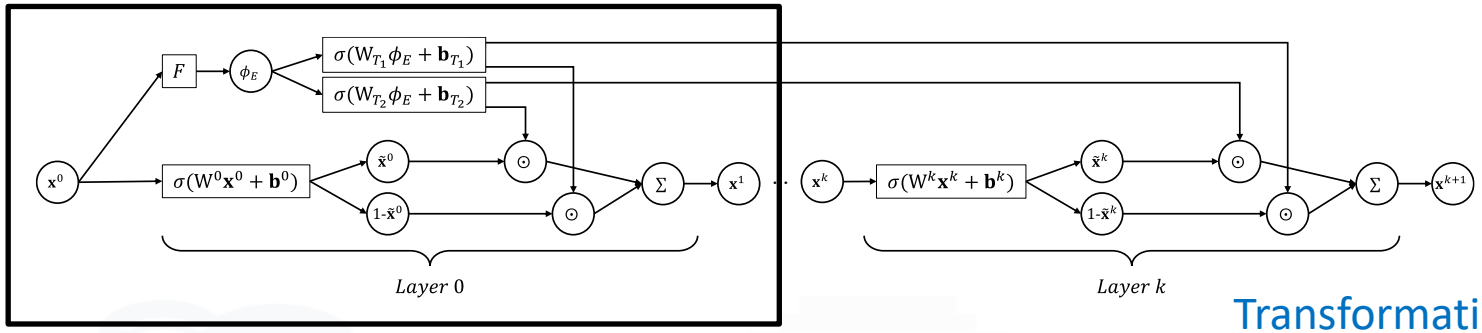
Encoding and Transformation:

- Fourier Encoding: The input x_0 is transformed into a frequency set

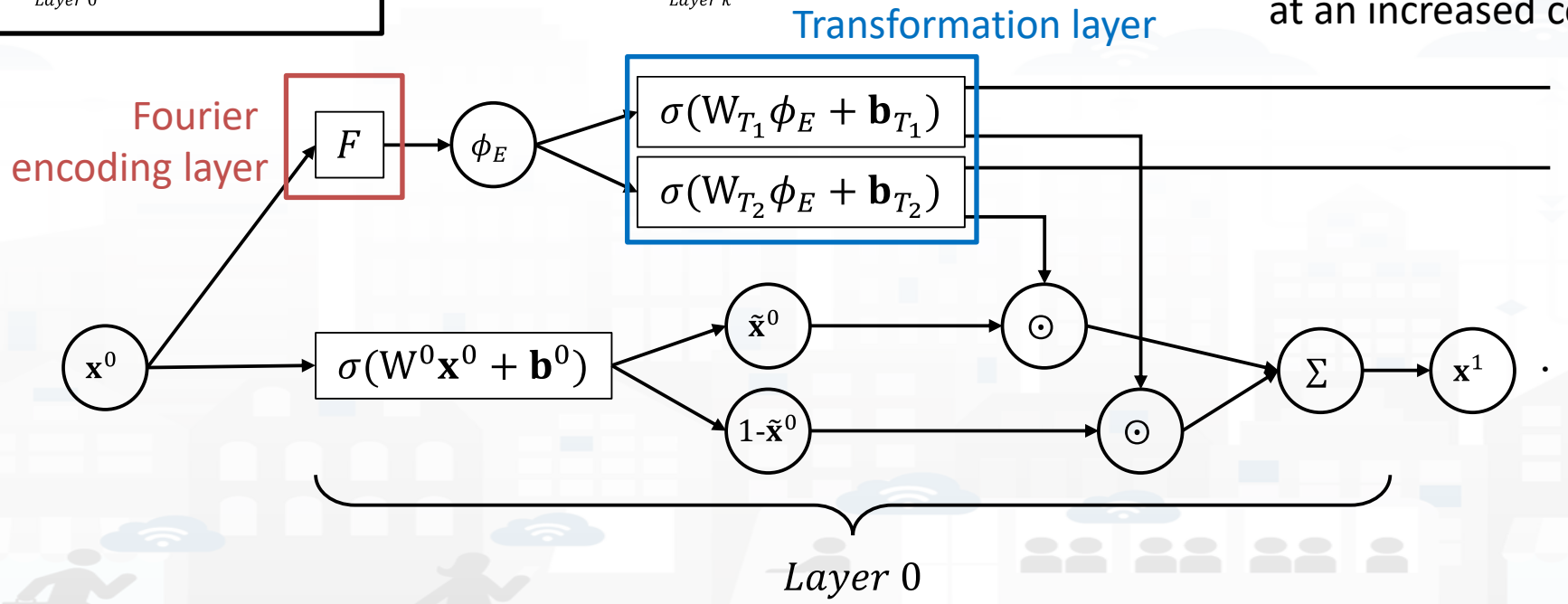
$$\phi_E = [\sin(2\pi\mathbf{f} \times \mathbf{x}_0) ; \cos(2\pi\mathbf{f} \times \mathbf{x}_0)]^T$$

- Unlike fixed encoding, the frequency parameters ($\mathbf{f}$) are learned during training, allowing the network to adapt to the specific scales of the PDE.
- The outputs of the standard MLP layers are element-wise multiplied ($\odot$) with the outputs of the transformation layers, combining global trends with high-frequency refinement.

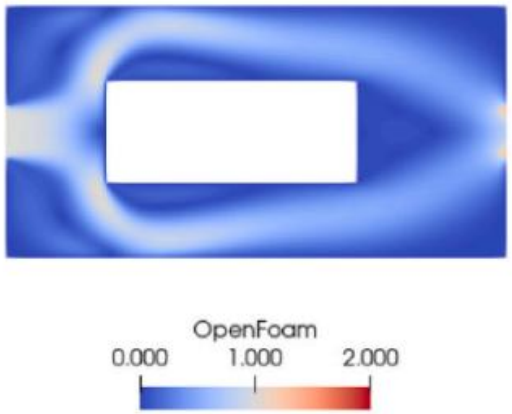
The Modified Fourier Network (MFN)

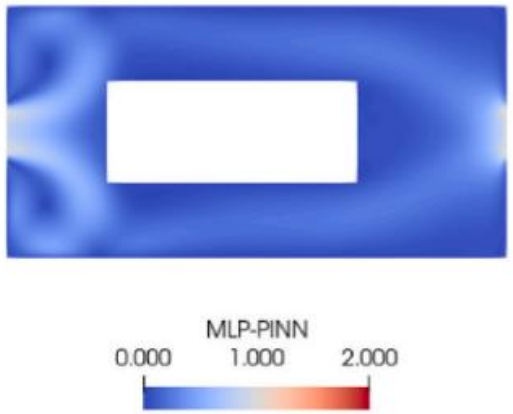


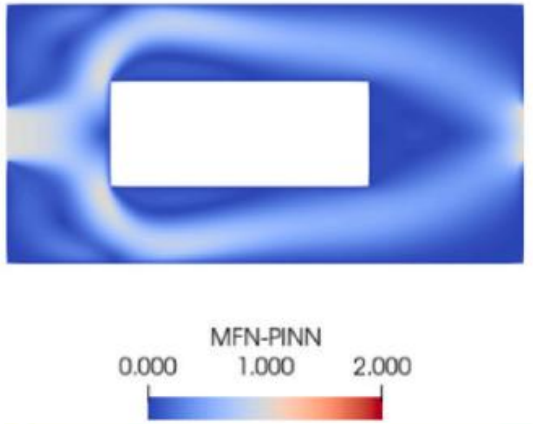
- The transformation is computed once and applied in each layer.
- Is also possible to have a transformation layer for each layer, at an increased computational cost

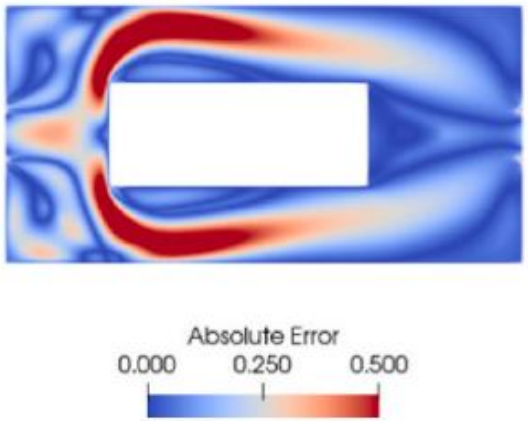


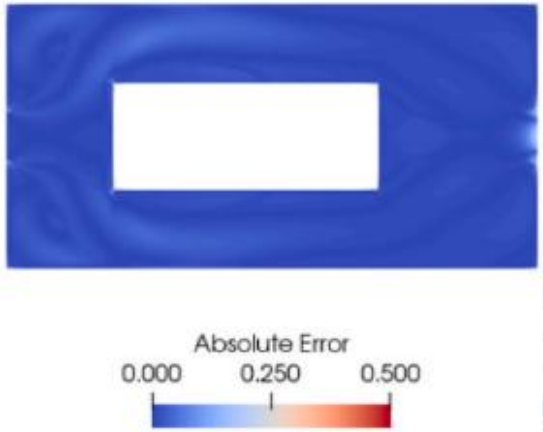
MLP vs MFN











Experimental validation

See Botarelli, Fanfani, Nesi, Pinelli, *Using Physics-Informed neural networks for solving Navier-Stokes equations in fluid dynamic complex scenarios*, Engineering Applications of Artificial Intelligence, 2025
https://doi.org/10.1016/j.engappai.2025.110347

The Challenge of Long-Time Integration

- Solving evolution equations (like Navier-Stokes or Allen-Cahn) over large temporal horizons is **notoriously difficult** for vanilla PINNs.
- As the time interval increases, the loss landscape becomes increasingly "stiff" and non-convex.
- Global training often results in the network **failing to converge** or producing solutions that satisfy boundary conditions but **ignore the underlying transient physics**

Why Global Temporal Training Fails

- In a global approach, the network attempts to **solve the entire time span simultaneously**.
- However, physical systems are **inherently causal**—the state at t=10 depends strictly on the state at t=0
- Errors in early time steps **propagate and amplify**, making it nearly impossible for the optimizer to find a valid solution for later time steps
- Without labelled data at every point in time, a global search often falls into "trivial" local minima that satisfy the PDE only superficially

The Time-Windowing Approach

Solution: **Temporal Domain Decomposition**

- Instead of one global training session, the total time interval $[0, T]$ is divided into **smaller, consecutive segments or "windows"** $[t_0, t_1], [t_1, t_2\ ], \ldots, [t_{n-1}, t_n\ ]$.
- The PINN is trained to solve the physics within the first window before moving to the next.
- The predicted solution at the end of window n **serves as the initial condition (IC)** for the start of window $n + 1$

The Time-Windowing Approach

- A model M_0 trained on the first window is not discarded. Instead, its weights are **used to initialize** (and then fine-tune) the model M_1 for the successive window.
- This "warm start" allows the network to carry over learned spatial features and global trends, reducing the number of epochs needed for subsequent windows.

Benefits of Time-Windowing

- By focusing on short durations, the **optimization task is simplified**, preventing the divergence of errors as time proceeds.
- Experimental evidence shows that fine-tuning between windows can **achieve a 2x speed-up** compared to training each segment from scratch.

The Causality Pathology in PINNs

- Standard PINNs minimize **all temporal residuals simultaneously** across the entire domain.
- Neural Tangent Kernel (NTK) analysis reveals that PINNs are biased toward approximating solutions at later times before resolving initial conditions.
- This violation of temporal **causality causes PINNs to get trapped in erroneous local minima**, especially in chaotic or highly non-linear systems like the Allen-Cahn equation.

Causal Training

- The standard loss is replaced with a **weighted version**:

$$L_r(\theta) = \sum w_i \, L_r(t_i, \theta)$$

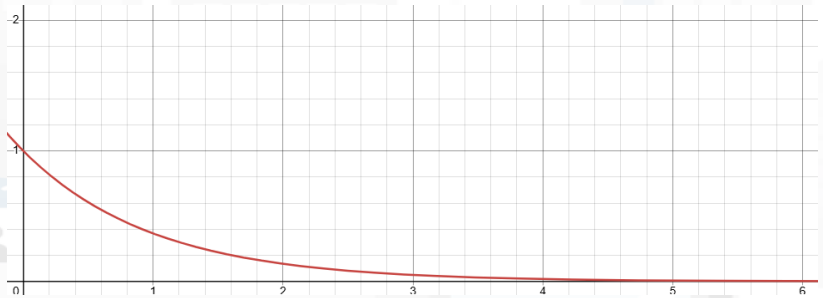
- **Causal Logic**: The weights (w_i) ensure that the residual at time t_i is not minimized unless all previous residuals ($t < t_i$) have already decreased to a small value
- This encourages the network to **learn the solution sequentially**, respecting how information naturally propagates in time.

The Weighting Mechanism

- Weight formula

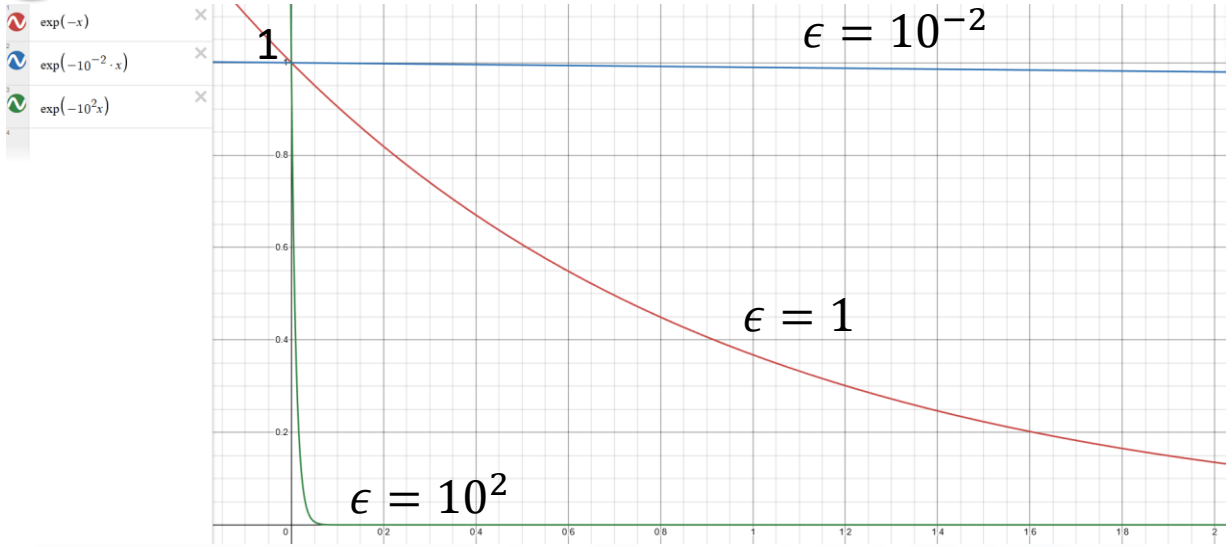
$$w_i = \exp\left(-\epsilon \sum_{k=1}^{i-1} L_r(t_k, \theta)\right)$$

- Weights are inversely exponentially proportional to the cumulative loss of all preceding time steps



The Weighting Mechanism

- ϵ is the hyper-parameter that determines the "slope" or strictness of the causality constraint
- To avoid tedious tuning, the algorithm uses an increasing sequence of values (e.g., 10^{-2} to 10^{2})
- Gradually increasing ϵ allows the network to **first find a coarse global solution before strictly enforcing the temporal causal structure**



Convergence and Stopping Criterion

- At the start of training, all weights except w_0 are near zero. As causality is satisfied, **weights gradually "activate"** and converge toward 1.
- Training can be terminated once the minimum weight exceeds a threshold (e.g., $\min w_i > 0.99$)
- This criterion provides a **quantitative measure of convergence** and prevents overfitting, leading to state-of-the-art accuracy in chaotic systems

Domain Decomposition in PINNs

- Standard (monolithic) PINNs struggle with **extremely large domains**, complex **non-regular geometries**, or highly **heterogeneous media** where physics properties change abruptly.
- The global computational domain Ω is decomposed into several smaller, discrete sub-domains Ω_i
- Instead of one large model, each sub-domain is **solved independently** using its own separate, often shallower, neural network.
- Each fragment can employ a **neural network with unique parameters** (depth, width, or optimization approach) specifically tailored to the local dynamics of that region

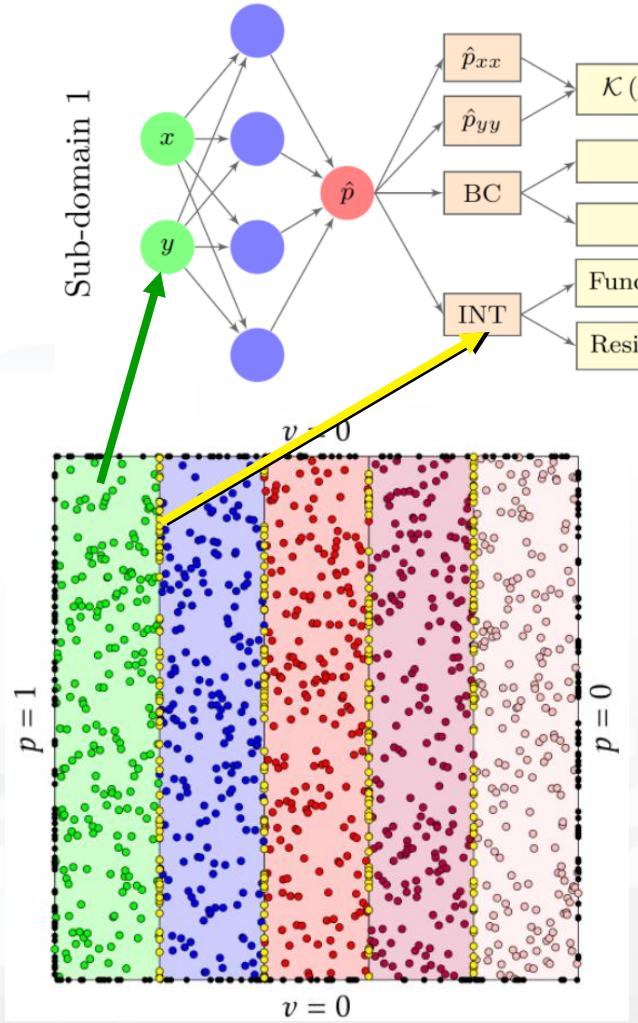
Interface Conditions and "Stitching"

- To create a continuous global solution, the independent PINNs **must communicate across their shared boundaries**.
- Subdomains are "stitched" together by adding specific penalty terms to the total loss function, for example:
  - **Interface Function Continuity** (IF): Forces the **predicted values** (e.g., pressure or velocity) to match at the boundary comparing those of a domain with the average values of the neighbour domains
  - **Interface Residual Continuity** (IR): Ensures that the **physics residuals** (the errors based on the derivatives) are consistent as they propagate between subdomains.
- To compute the interface terms, you **need data from neighbouring domains**

Interface Conditions and "Stitching"

- The global domain is decomposed into multiple sub-domains, each governed by its own separate PINN architecture
- The subdomains are "stitched" together by **minimizing a total loss function**
- The final objective function is the **summation of all individual subdomain losses** (including each interface stitching components)
- This unified training ensures that the solution is **physically consistent across the entire domain**

XPINN Domain Decomposition



$$\mathrm{L}^{(s)} = \mathrm{W}_{\mathrm{BC}}\, MSE^{(s)}_{\mathrm{BC}} + \mathrm{W}_{\mathrm{PDE}}\, MSE^{(s)}_{\mathrm{PDE}} + \mathrm{W}_{\mathrm{IF}}\, MSE^{(s)}_{\mathrm{IF}} + \mathrm{W}_{\mathrm{IR}}\, MSE^{(s)}_{\mathrm{IR}}$$

LOSS$^{(2)}$

LOSS$^{(3)}$

LOSS$^{(4)}$

LOSS$^{(5)}$

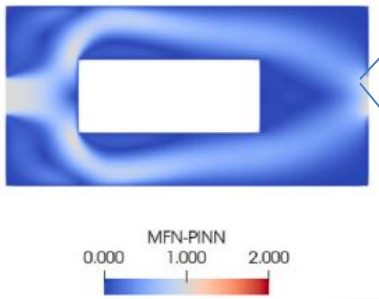
$$\mathrm{L} = \sum_{s=1}^{5} \mathrm{L}^{(s)}$$

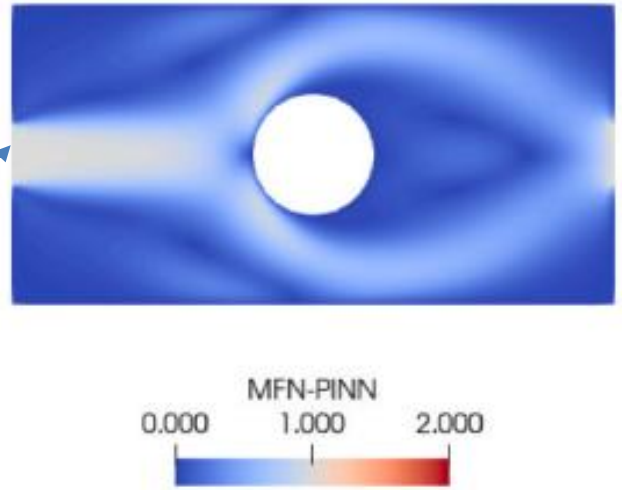
Images from Alhubail, Ali, et al. "Extended physics-informed neural networks for solving fluid flow problems in highly heterogeneous media." International petroleum technology conference. IPTC, 2022.

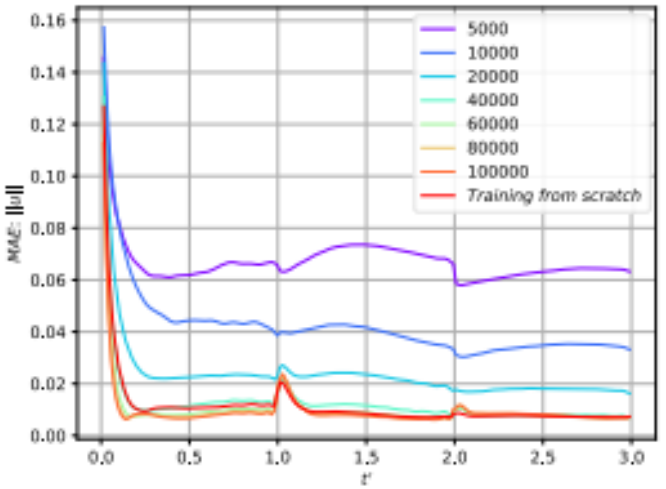
Finetuning

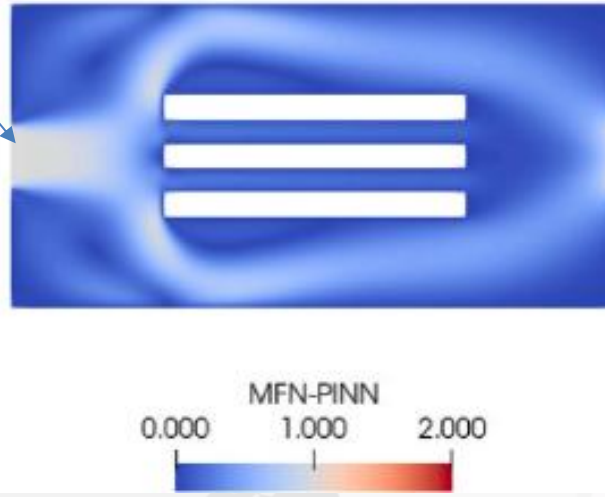
- Finetuning can work not only in the time-windowing approach, but also in a more general sense
- A PINN trained to solve a PDE on a given domain/geometry can be used to initialize a novel PINN working on a different (similar) domain

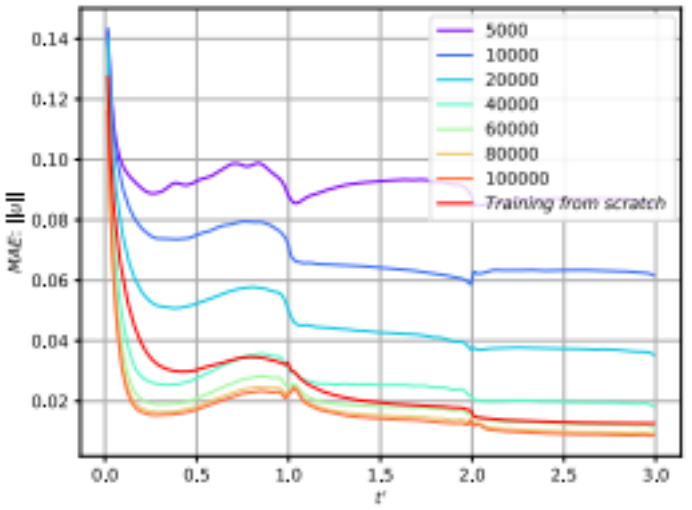
Finetuning


MFN-PINN
0.000
1.000
2.000


MFN-PINN
0.000
1.000
2.000


5000
10000
20000
40000
60000
80000
100000
Training from scratch
MAE: ||u||
t'


MFN-PINN
0.000
1.000
2.000


5000
10000
20000
40000
60000
80000
100000
Training from scratch
MAE: ||u||
t'

Multi-resolution Training

- A strategy where the **number of collocation points varies across different stages or time windows** of the simulation.
- Training early "transient" phases with a low density of points and increasing the density for the final "steady-state" windows.
- Unlike traditional CFD solvers where the mesh must remain fixed, **PINNs allow point density to be adjusted dynamically** between training phases
- Different time windows can **employ different collocation point densities** (Multi-resolution training), using fewer points for early transient phases and more for complex steady-state regions.

Parameterized Learning

- Providing geometry in a **parameterized manner** (ranges instead of fixed dimensions) to train a single PINN for multiple setups.
- Initial training is longer and more computationally expensive (e.g., 150,000 epochs), but the resulting model **solves an entire class of problems**
- E.g.: in an autoclave production line, a single parameterized model can replace 10 individual trainings, resulting in an overall speed-up

Software Ecosystem for SciML

- Solving PDEs with deep learning requires specialized tools that go beyond standard ML libraries like TensorFlow or PyTorch.
- Key Capabilities:
  - Support for Automatic Differentiation (AD).
  - Physics-informed loss function builders.
  - Optimization strategies (Adam + L-BFGS).
  - Scalability for high-performance computing (GPUs/TPUs).
- Main solutions: **DeepXDE** and **NVIDIA Modulus**

DeepXDE

Versatile Library for Differential Equations:

- A widely used deep learning library specifically designed for solving differential equations.
- Strengths:
  - Supports a vast array of PDE types (ODEs, PDEs, fPDEs, integro-differential equations)
  - Includes built-in modules for complex geometries and various boundary conditions.
  - Ideal for benchmarking new algorithms and testing rapid prototypes.
- Can run on multiple backends (TensorFlow, PyTorch, JAX, PaddlePaddle).

NVIDIA Modulus

- An AI-accelerated multi-physics simulation framework designed for industrial-scale problems.
- Core Features:
  - Physics-ML Focused: Optimized for high-fidelity digital twins and complex engineering systems.
  - Scalability: Specifically engineered to leverage multi-GPU nodes and massive datasets.
  - High-Order Accuracy: Includes specialized architectures (like modified MLPs) to handle sharp gradients and high-frequency physics.
- Application: Used extensively in fluid mechanics, electromagnetics, and manufacturing