



TOBIAS BUCK

27.9.2024 TOBIAS.BUCK@IWR.UNI-HEIDELBERG.DE

SCIENTIFIC ML

OUTLINE OF TODAY

1. Scientific ML
 - A. Neural ODEs
 - B. Symbolic Regression
 - C. Operator Learning
 - D. Physics Informed Neural Networks
 - E. Hamiltonian Neural Networks

WHERE TO FIND THE LECTURE MATERIAL



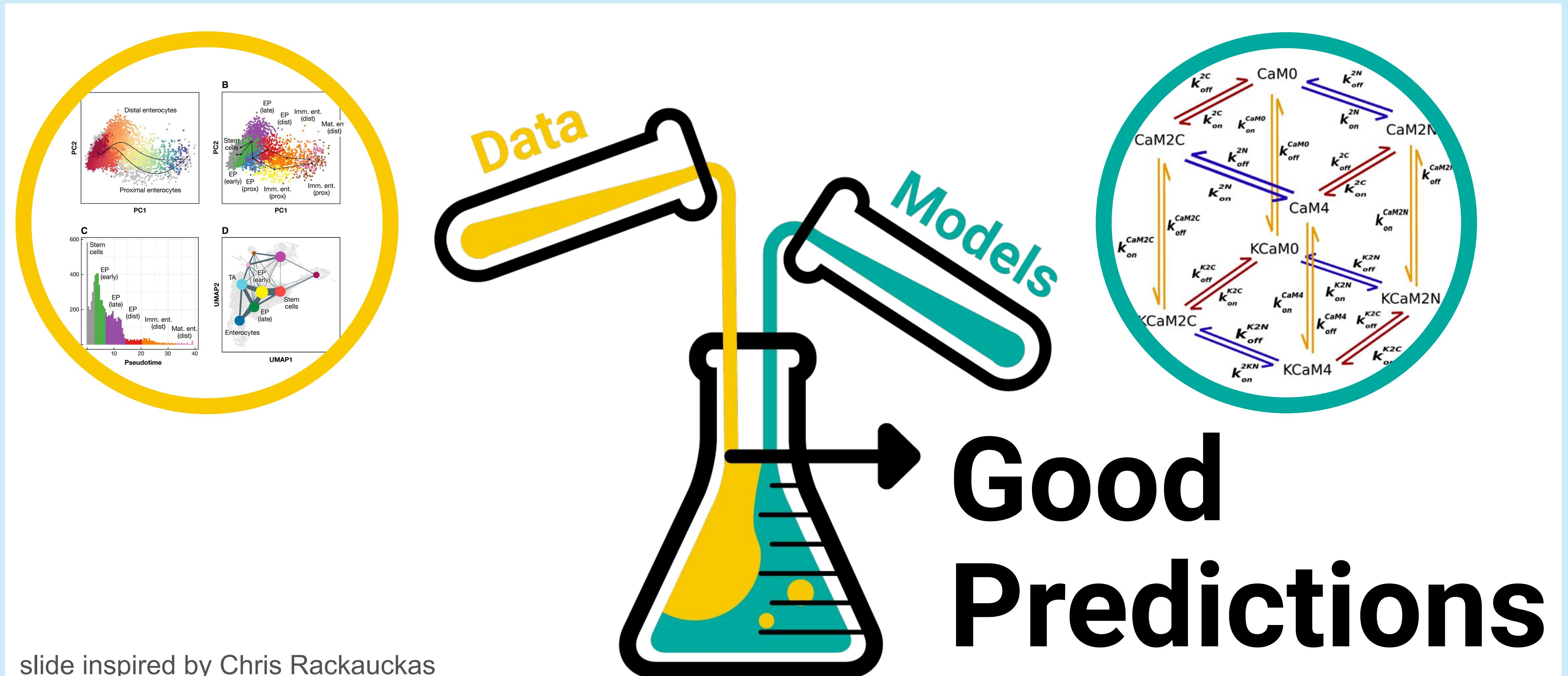
<https://github.com/TobiBu/graddays>

SCIENTIFIC MACHINE LEARNING

»Scientific machine learning is the burgeoning field combining techniques of machine learning into traditional scientific computing and mechanistic modeling.« [e. g. Zubov et al, 2021]

SCIENTIFIC MACHINE LEARNING IS MODEL-BASED DATA-EFFICIENT ML

How do we simultaneously use both sources of knowledge?



**Core Point of SciML:
the more structure that is encoded into the
model, the less there is to learn and the more
data efficient the algorithm is!**

Symbolic regression

MOTIVATION

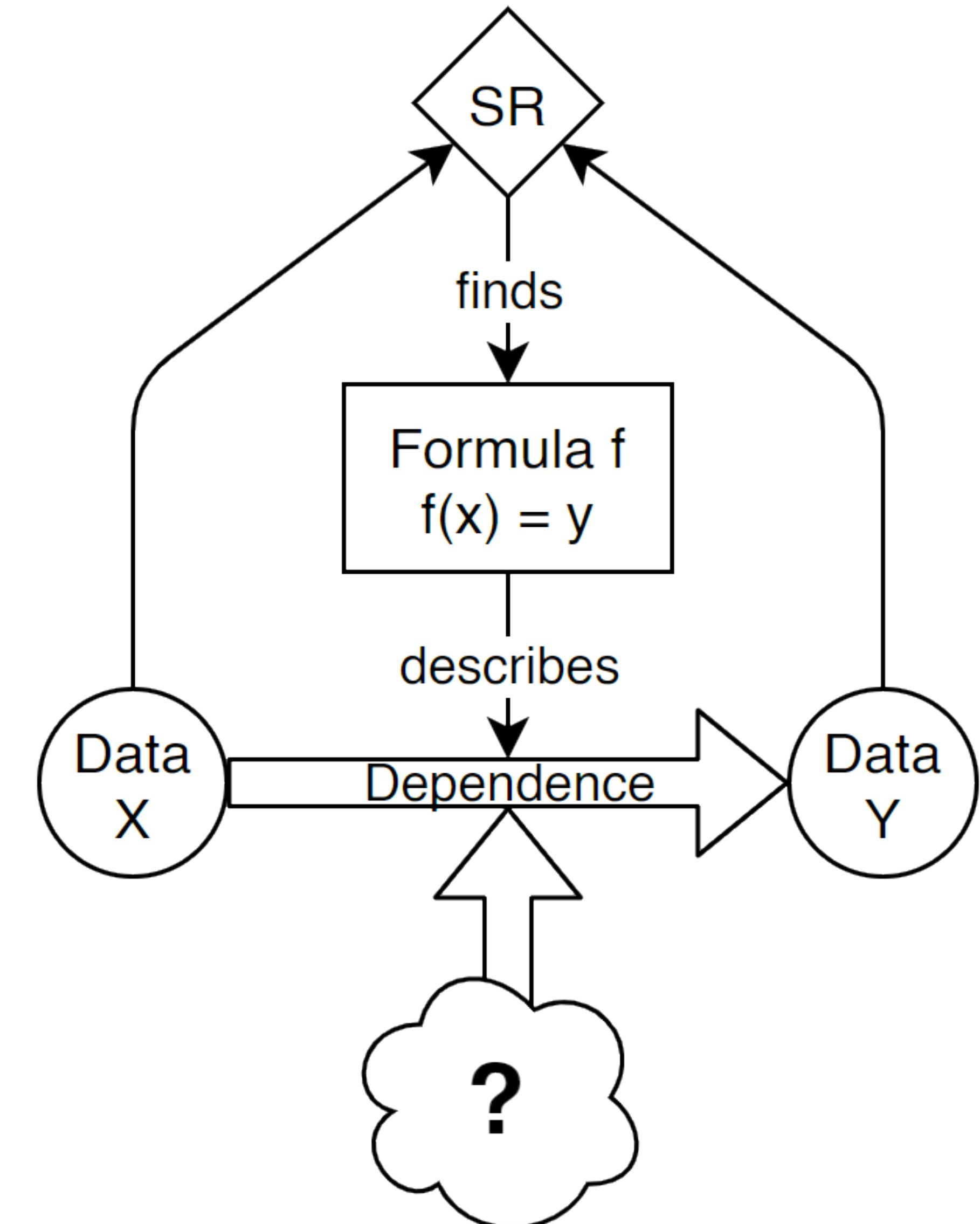
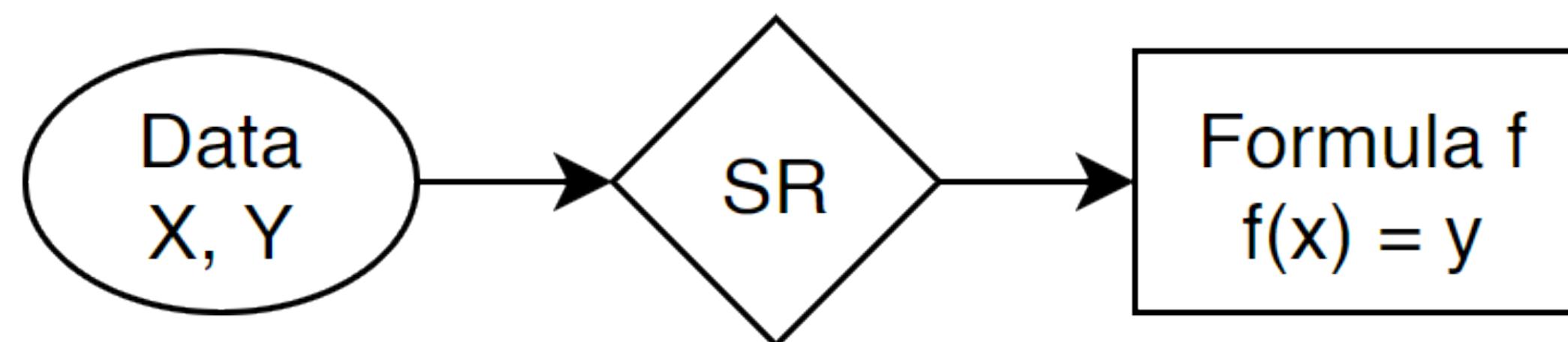
- ▶ Given: Data $\{(\vec{x}_i, y_i) | i \in \{1, \dots, N\}\}$
$$\overbrace{x_{i,1}, \dots, x_{i,d}}$$
- ▶ Wanted: Formula $f: f(x) = y$

MOTIVATION

- ▶ Given: Data $\{(\vec{x}_i, y_i) | i \in \{1, \dots, N\}\}$

$\overbrace{x_{i,1}, \dots, x_{i,d}}$

- ▶ Wanted: Formula $f: f(x) = y$



MOTIVATION

How does y depend on x ?

x	y
0	0
1	1
2	4
3	9
4	16
5	25

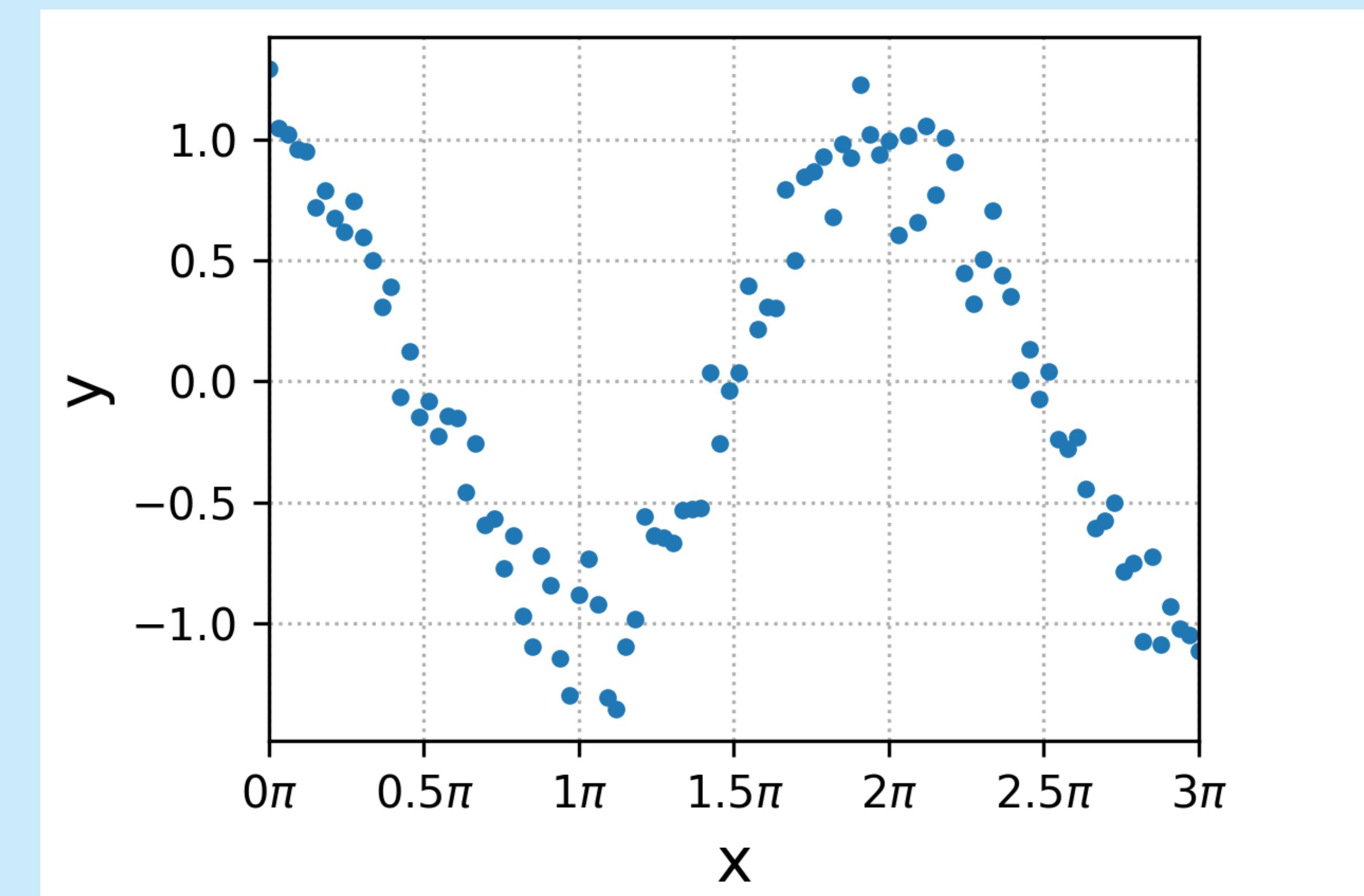
x_1	x_2	y
0	0	0
3	6	9
4.5	5.5	10
0	123	123
99	1	100
5	25	30

MOTIVATION

How does y depend on x ?

x	y
0	0
1	1
2	4
3	9
4	16
5	25

x_1	x_2	y
0	0	0
3	6	9
4.5	5.5	10
0	123	123
99	1	100
5	25	30



FINDING FORMULAS

What is a formula?

- ▶ functions
- ▶ arguments
 - ▶ functions
 - ▶ input x_i
 - ▶ constants

$$f((x_1, x_2)) = \sin(a * x_1) + \exp\left(\frac{x_2}{b}\right)$$

$$+(\sin(*(\mathbf{a}, x_1)), \exp(/(x_2, b))))$$

$$F_1 = \{+, -, *, /, \sin, \cos\} \rightarrow \tan(.) = /(\sin(.), \cos(.))$$

$$F_2 = \{+, -, *, /, \cos, \tan\} \rightarrow \sin(.) = *(\tan(.), \cos(.))$$

IDENTIFYING THE „BEST“ FORMULA: ACCURACY VS. COMPLEXITY

Which „metric“ to use?

Want a number for quantification

1. Accuracy: $|y - \hat{y}| = |y - f(x)| \rightarrow \text{Loss}$
2. Length/complexity of the formula

► Taylor series of $f(x)$: $\sum_{n=0}^N \frac{f^{(n)}(a)}{n!} (x - a)^n$

► Overfitting + Noise:

True solution: $y = \sin(x)$

Lowest loss: $f(x) = 1.005 * \sin(0.008 + 0.998 * x)$

→ favor shorter formulas, punish longer formulas

$$L_{\text{total}} = c_1 \underbrace{\mathcal{L}(f, \text{data})}_{\text{Accuracy}} + c_2 \underbrace{\mathcal{L}(f)}_{\text{Complexity}}$$

SYMBOLIC REGRESSION AS A CLASSIFICATION PROBLEM

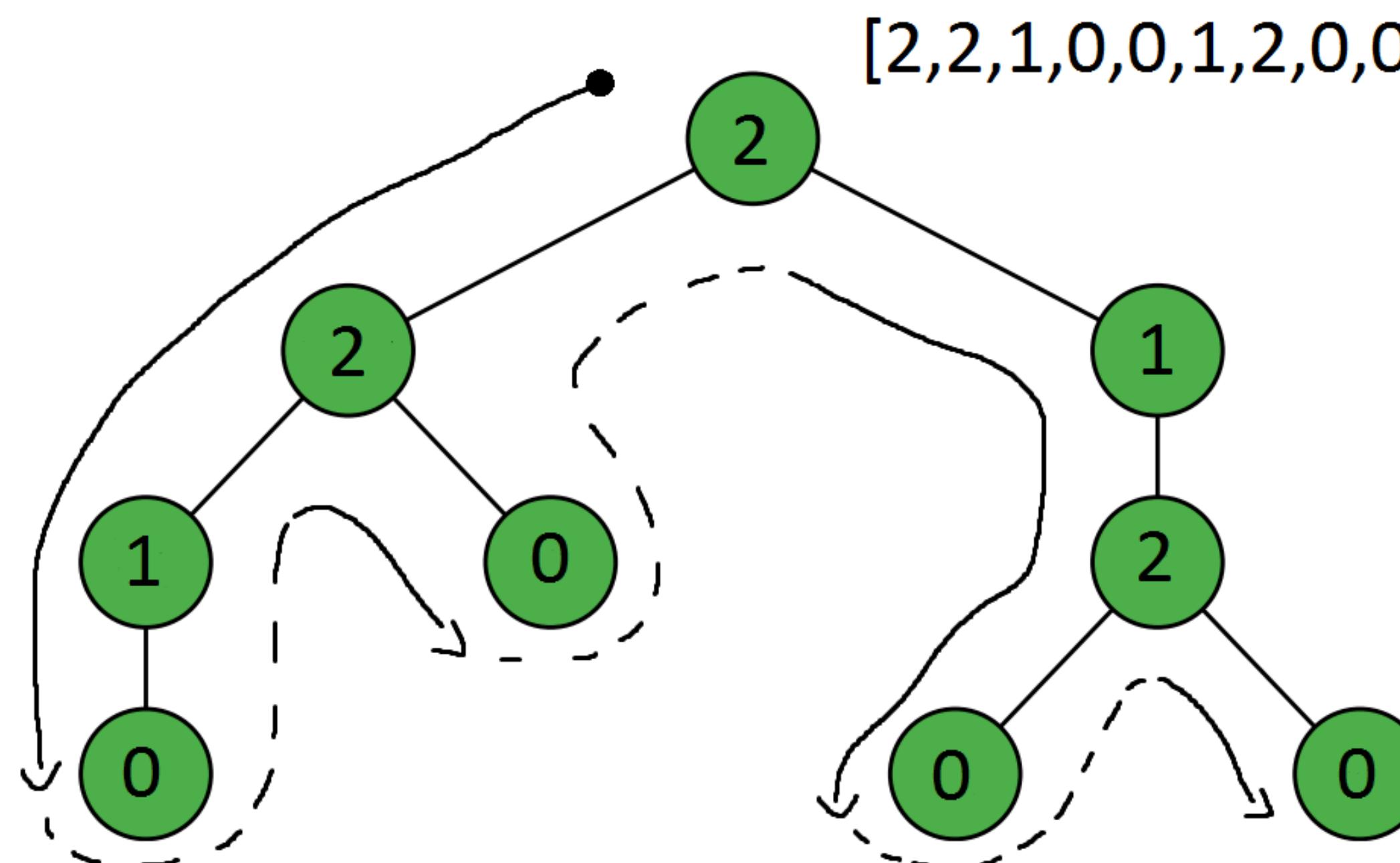
NP-hard

supervised learning task

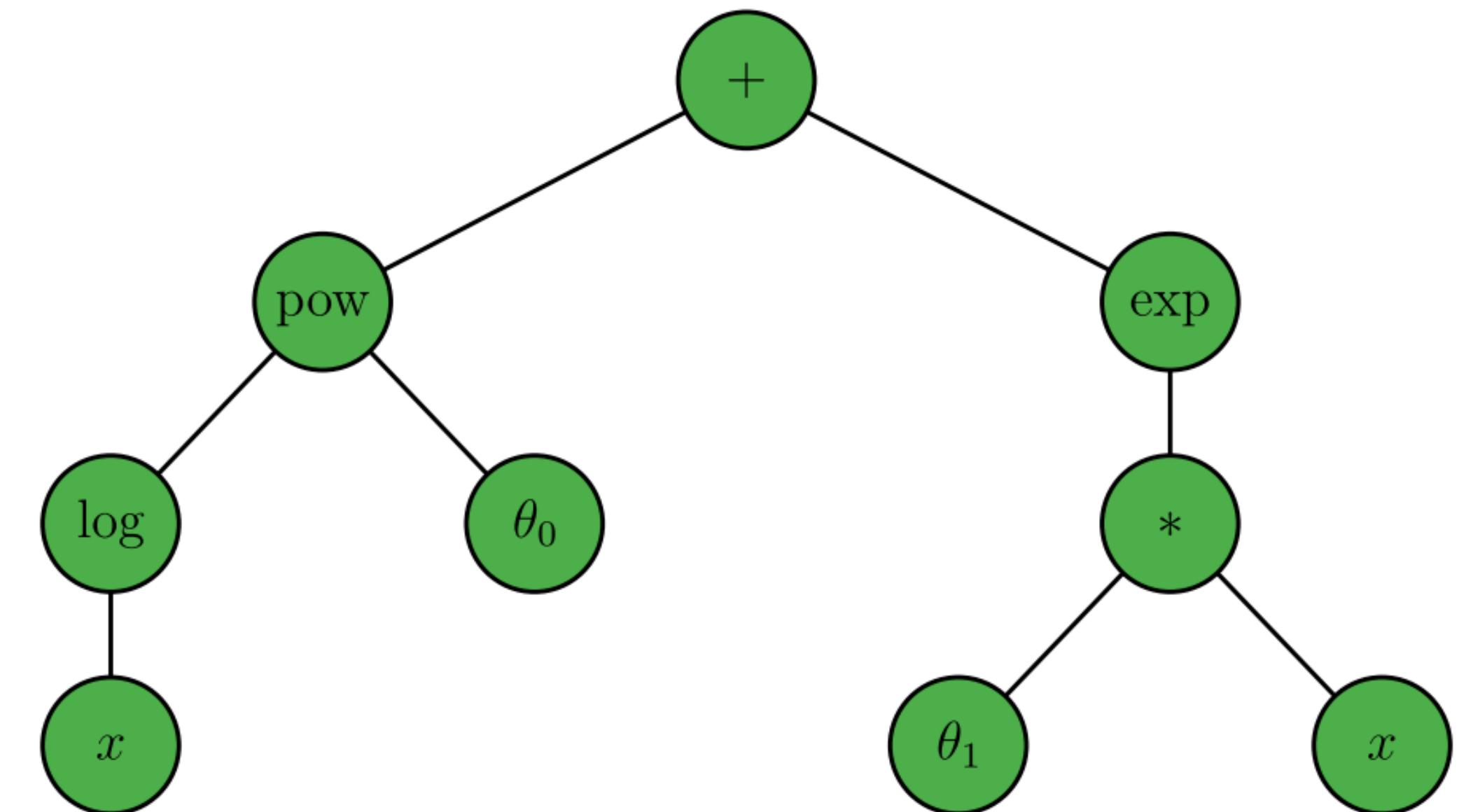
Multi-object optimisation (\rightarrow Pareto-optimal solution)

EXHAUSTIVE SEARCH

- ▶ Brute force
- ▶ Operators
 - ▶ functions & arguments
 - ▶ Types: nullary = 0, unary = 1, binary = 2
- ▶ Tree as operator list
- ▶ Complexity k = length of operator list



$$(\log(x))^{\theta_0} + \exp(\theta_1 x) = [+,\text{pow},\log,x,\theta_0,\exp,*,\theta_1,x]$$

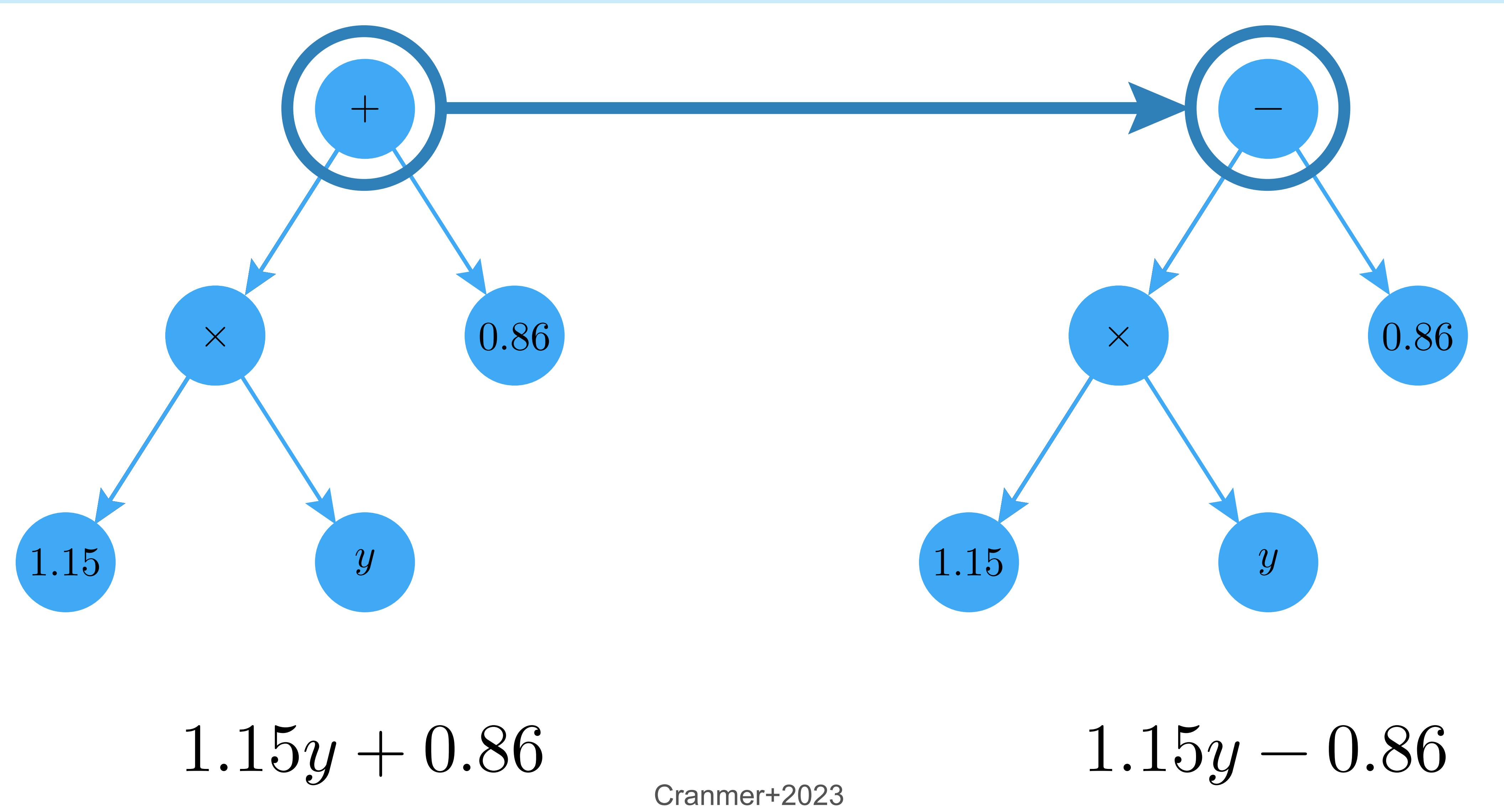


GENETIC PROGRAMMING

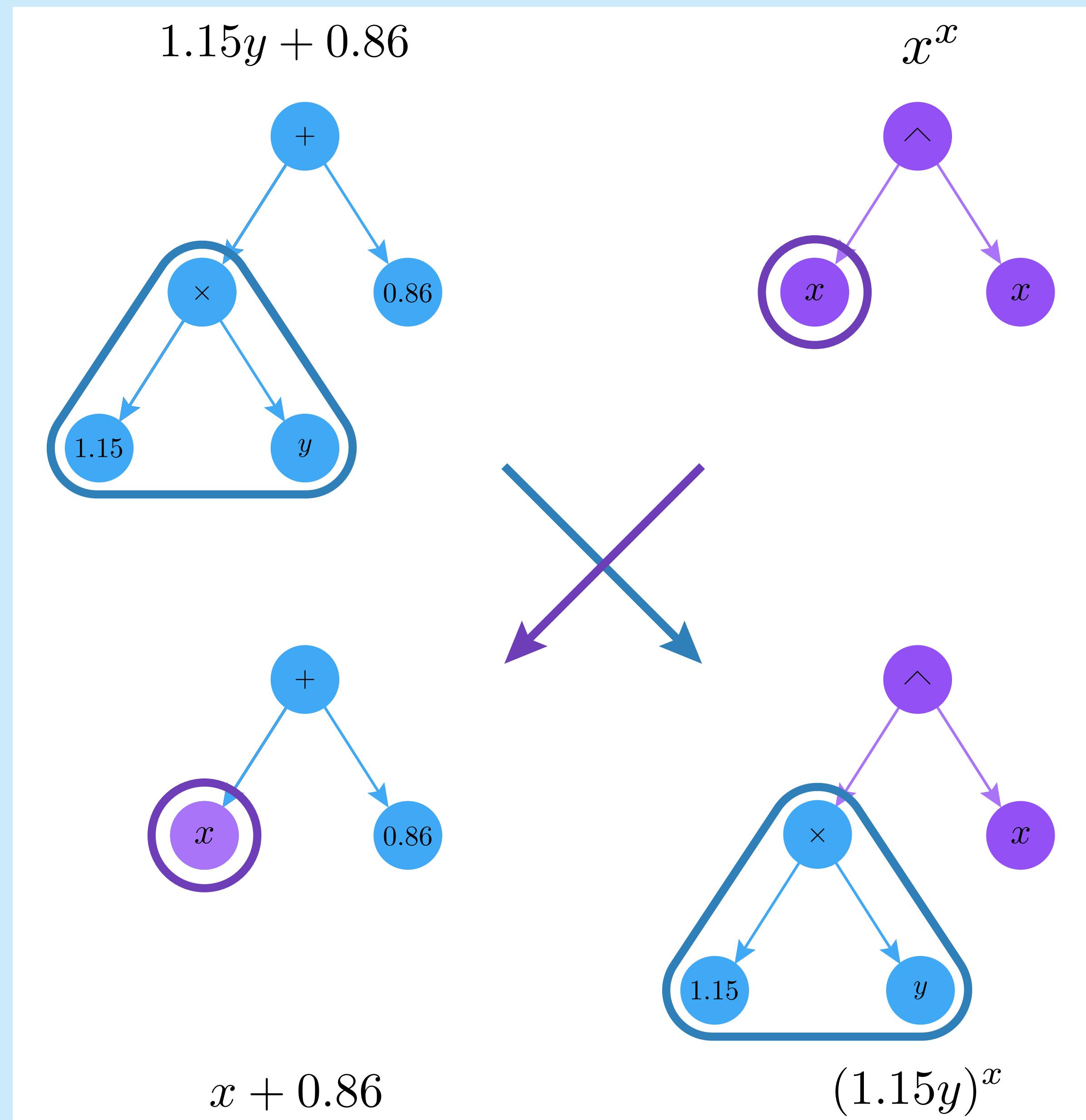
Basic algorithm:

- ▶ initialize a population of “individuals”
- ▶ repeat the evolution process:
 1. randomly choose a subset → competitors in “tournament”
 2. evaluated fitness of competitors
 3. select a winner
 4. replace the weakest/oldest individual with mutation of winner
- ▶ return fittest individual
- splitting allows for parallelization
- “Tournament selection”

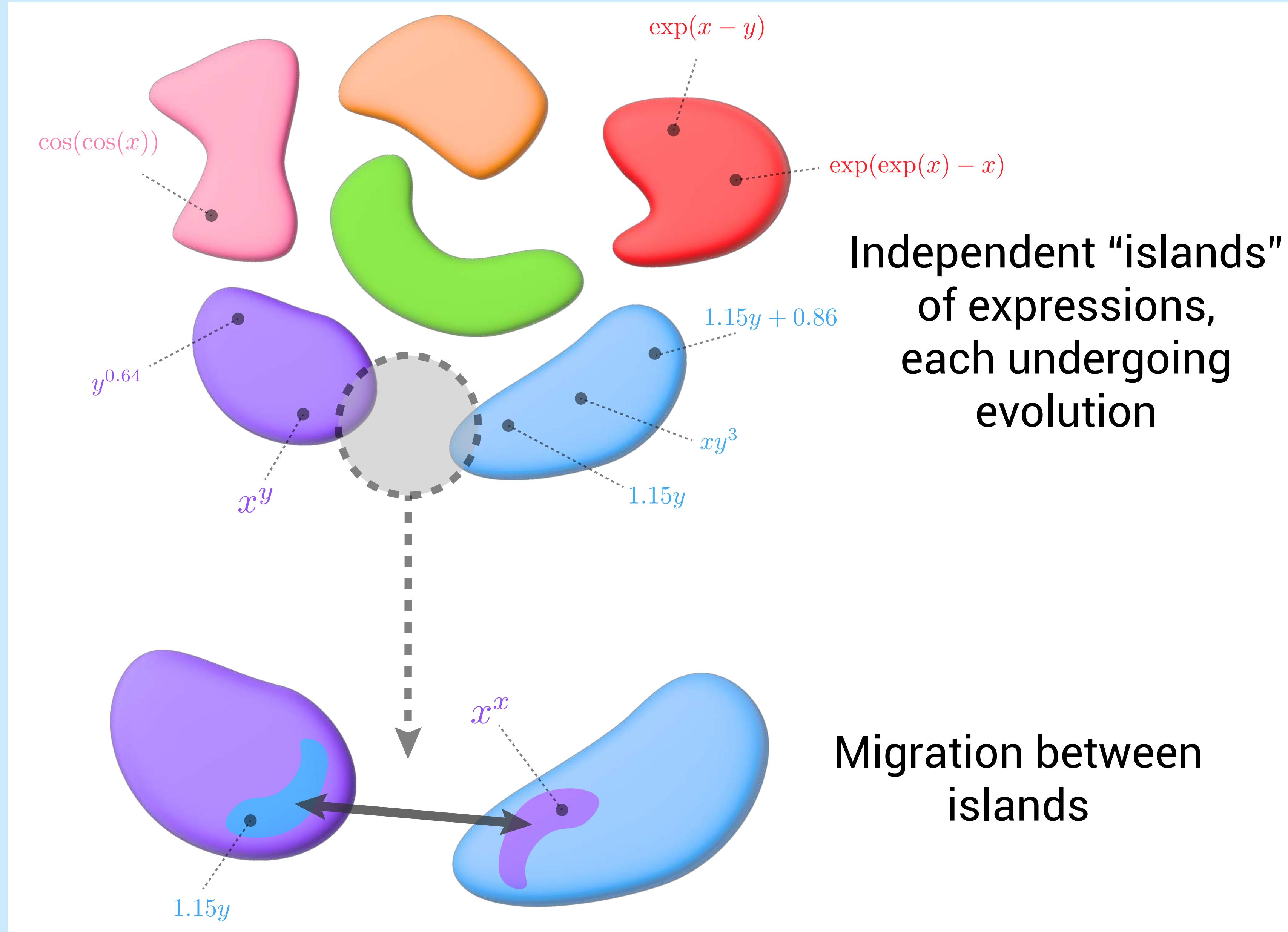
GENETIC PROGRAMMING: MUTATION



GENETIC PROGRAMMING: CROSSOVER



GENETIC PROGRAMMING: MIGRATION



GENETIC PROGRAMMING: PYSR

- ▶ performs SR using genetic algorithm
- ▶ open source, backend in julia
- ▶ 2nd place in SRBench 2022
- ▶ evolve-simplify-optimize-loop
- ▶ custom operators (broken power laws, dilogarithm)
- ▶ returns best functions of several complexities



PROS AND CONS OF GENETIC PROGRAMMING

- + exploration of huge search space
- + speedup by parallel processing
- + solution diversity
- problem should be low-dimensional
- can only optimize over discrete spaces
- exploration is probabilistic
- premature convergence

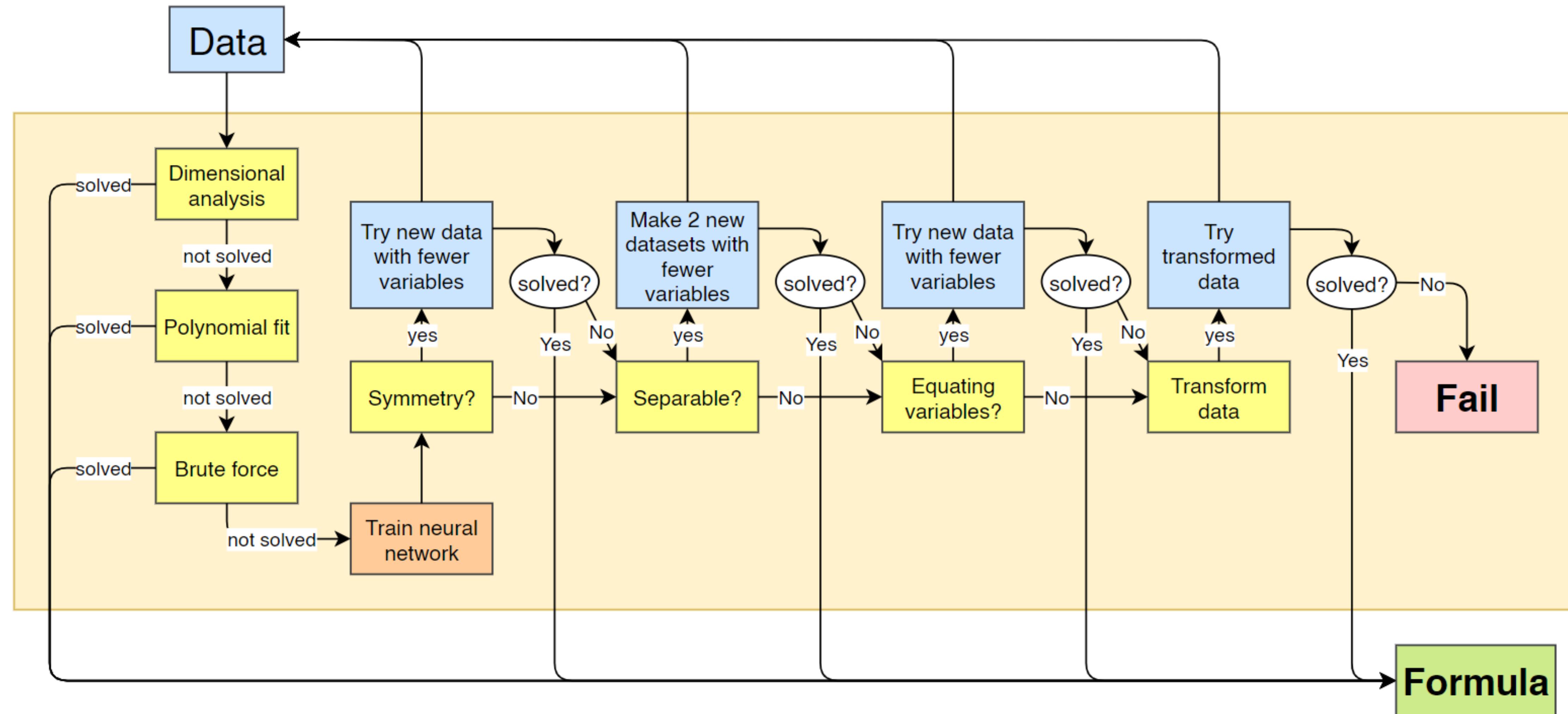
AI-FEYNMAN — SOLVE MYSTERIES

<https://github.com/SJ001/AI-Feynman>

- ▶ [2] one of the most cited paper about SR
- ▶ Use properties that often show up in physical/scientific equations
 - ▶ Units
 - ▶ Low order polynomial elements
 - ▶ Symmetry
 - ▶ Separability...
- ▶ Neural network for data interpolation

AI-FEYNMAN — SOLVE MYSTERIES

Algorithm



SR TRANSFORMER

Obtain function from DL model?

- ▶ idea: predict formulas from samples with a deep learning model
- ▶ 2022: “End-to-end symbolic regression with transformers” [6]

arXiv:2204.10532v1 [cs.LG] 22 Apr 2022

End-to-end symbolic regression with transformers

Pierre-Alexandre Kamienny^{*†1,2}, Stéphane d’Ascoli^{††1,3}, Guillaume Lample¹ and François Charton¹

¹Meta AI, Paris

²ISIR MLIA, Sorbonne Université, Paris

³Department of Physics, Ecole Normale Supérieure, Paris

Abstract

Symbolic regression, the task of predicting the mathematical expression of a function from the observation of its values, is a difficult task which usually involves a two-step procedure: predicting the “skeleton” of the expression up to the choice of numerical constants, then fitting the constants by optimizing a non-convex loss function. The dominant approach is genetic programming, which evolves candidates by iterating this subroutine a large number of times. Neural networks have recently been tasked to predict the correct skeleton in a single try, but remain much less powerful.

In this paper, we challenge this two-step procedure, and task a Transformer to directly predict the full mathematical expression, constants included. One can subsequently refine the predicted constants by feeding them to the non-convex optimizer as an informed initialization. We present ablations to show that this end-to-end approach yields better results, sometimes even without the refinement step. We evaluate our model on problems from the SRBench benchmark and show that our model approaches the performance of state-of-the-art genetic programming with several orders of magnitude faster inference.

Introduction

Inferring mathematical laws from experimental data is a central problem in natural science; having observed a variable y at n points $\{x_i\}_{i \in \mathbb{N}_n}$, it implies finding a function f such that $y_i \approx f(x_i)$ for all $i \in \mathbb{N}_n$. Two types of approaches exist to solve this problem. In *parametric statistics* (PS), the function f is defined by a small number of parameters that can directly be estimated from the data. On the other hand, *machine learning* (ML) techniques such as decision trees and neural networks select f from large families of non-linear functions by minimizing a loss over the data. The latter relax the assumptions about the underlying law, but their solutions are more difficult to interpret, and tend to overfit small experimental data sets, yielding poor extrapolation performance.

Symbolic regression (SR) stands as a middle ground between PS and ML approaches: f is selected from a large family of functions, but is required to be defined by an interpretable analytical expression. It has already proved extremely useful in a variety of tasks such as inferring physical laws [1, 2].

SR is usually performed in two steps. First, predicting a “skeleton”, a parametric function using a pre-defined list of operators – typically, the basic operations ($+$, \times , \div) and functions (sqrt , exp , \sin , etc.). It determines the general shape of the law up to a choice of constants, e.g. $f(x) = \cos(ax + b)$. Then, the constants in the skeleton (a, b) are estimated using optimization techniques, typically the Broyden–Fletcher–Goldfarb–Shanno algorithm (BFGS).

^{*}pakamienny@fb.com

[†]stephane.dascoli@gmail.com

^{††}Equal contribution.

SR TRANSFORMER: PROS AND CONS

- + fast inference
- + prediction of simple equations
 - training data only contains simple equations
- selection of operators is fixed
- still worse than GP
 - could initialize evolutionary method with transformer results

WHERE TO FIND THE LECTURE MATERIAL



<https://github.com/TobiBu/graddays>

Operator Learning

UNIVERSAL APPROXIMATION THEOREM FOR NONLINEAR OPERATORS

UAT - Operators

Chen & Chen, Universal Approximation to Nonlinear Operators..., 1995

Let $G: V \rightarrow W, u \mapsto G(u)$ be a non-linear continuous Operator.

$G(u)(y)$ denotes the value of $G(u)$ at point y .

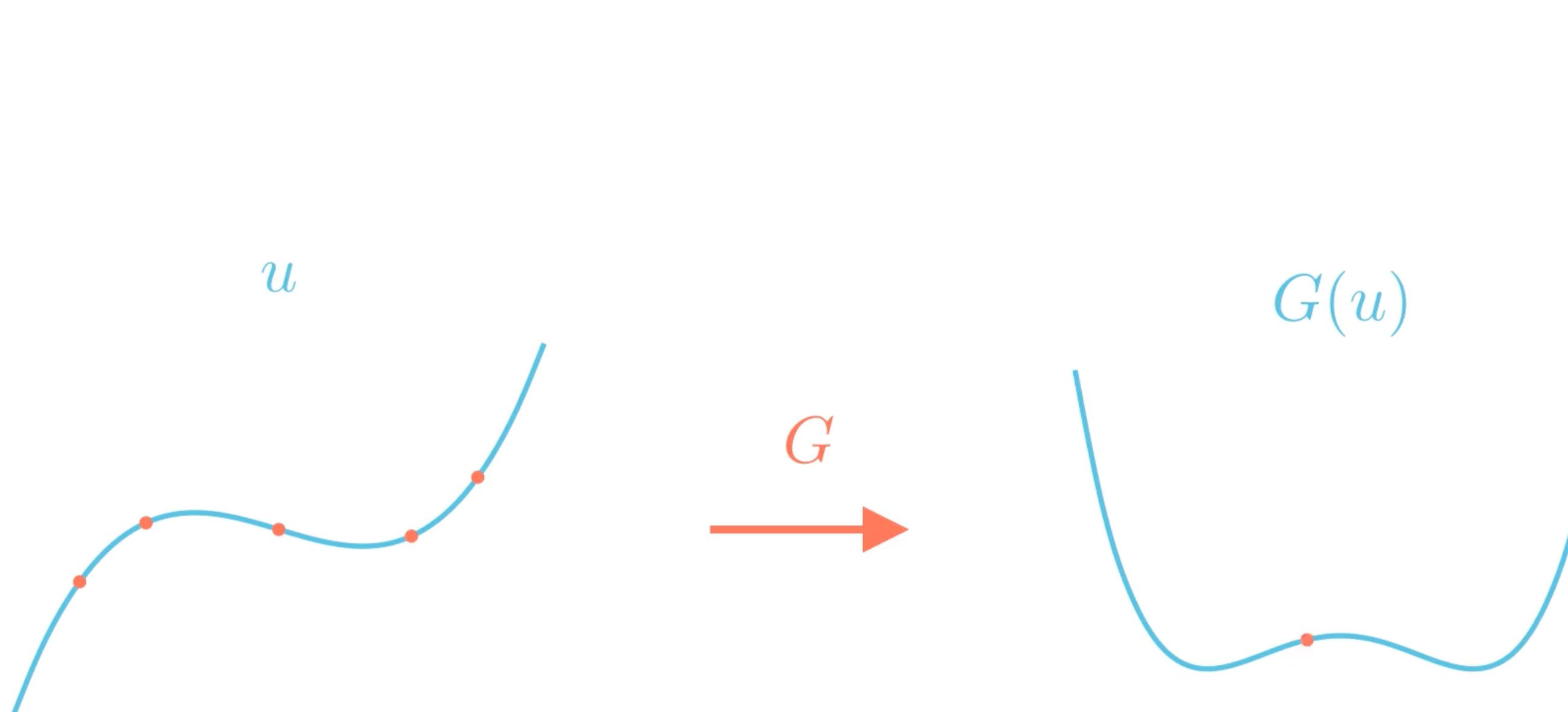
There is a function $f_{c,\xi,\theta,\zeta}(x, y)$ such that for every $\epsilon > 0$

$$|G(u)(y) - f_{c,\xi,\theta,\zeta}(x, y)| < \epsilon$$

for all $u \in V, y \in Y$ ($G(u): Y \rightarrow Z$).

$$f_{c,\xi,\theta,\zeta}(x, y) = \sum_{k=1}^p \sigma(w_k \cdot y + \zeta_k) \sum_{i=1}^n c_i^k \cdot \sigma\left(\sum_{j=1}^m \xi_{ij}^k \cdot u(x_j) + \theta_i^k\right)$$

OPERATOR LEARNING: INPUT AND OUTPUT

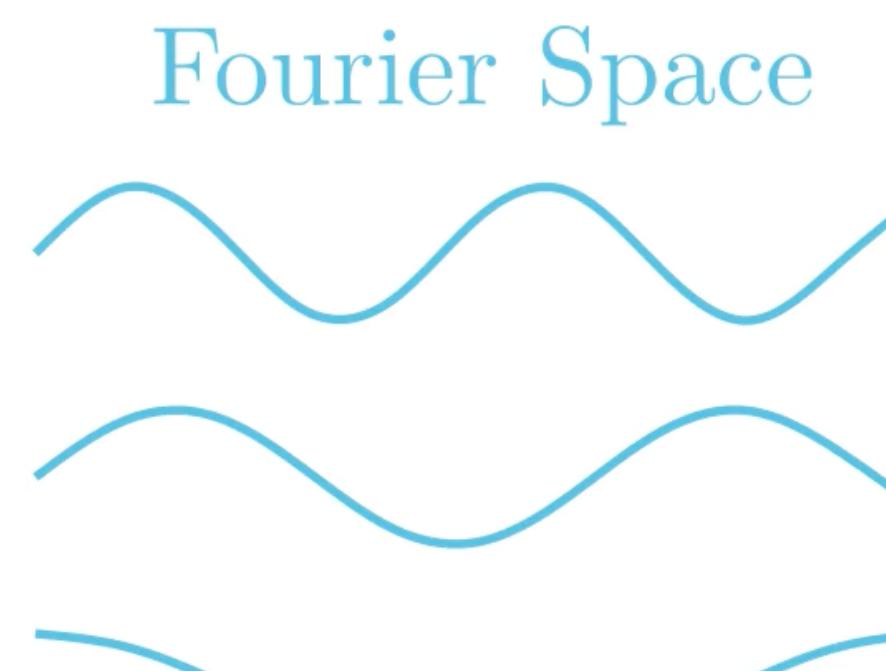


Problem: Can not put a function into an NN...

Solution: "Sensors"
→ Use function values $u(x_i)$ at locations x_i

More flexibility: "Arbitrary" output location y

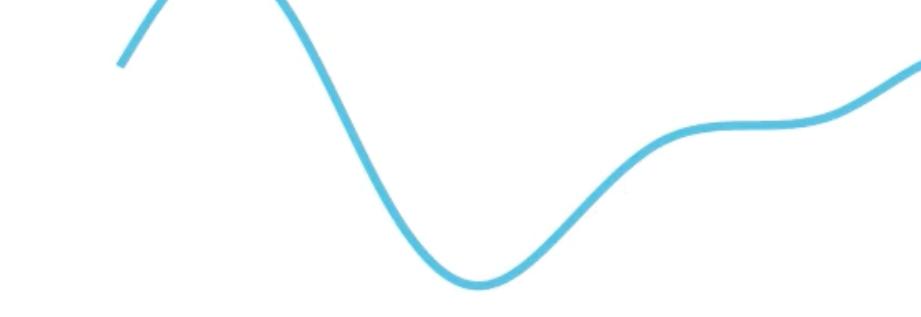
INTERMISSION: GAUSSIAN RANDOM FIELDS



$$\sum \sin(\omega_i x + \phi_i)$$

Red arrow pointing from the Fourier Space section to the Real Space section.

Real Space



$$u: \mathbb{R}^p \rightarrow \mathbb{R}^d$$

$$[u(x_1), u(x_2), \dots, u(x_m)]$$

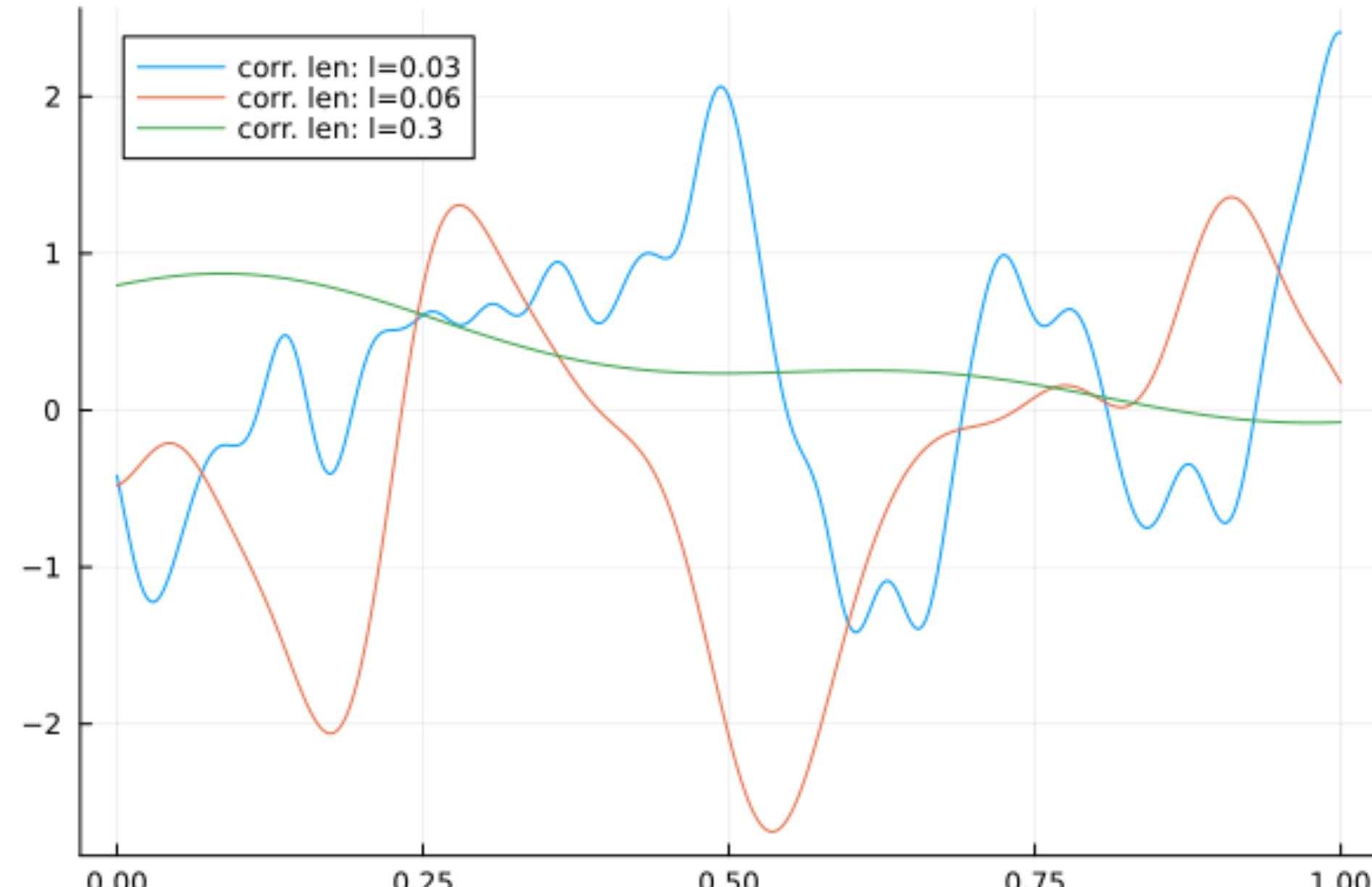
1. Fourier Space
randomly sample phase φ_i



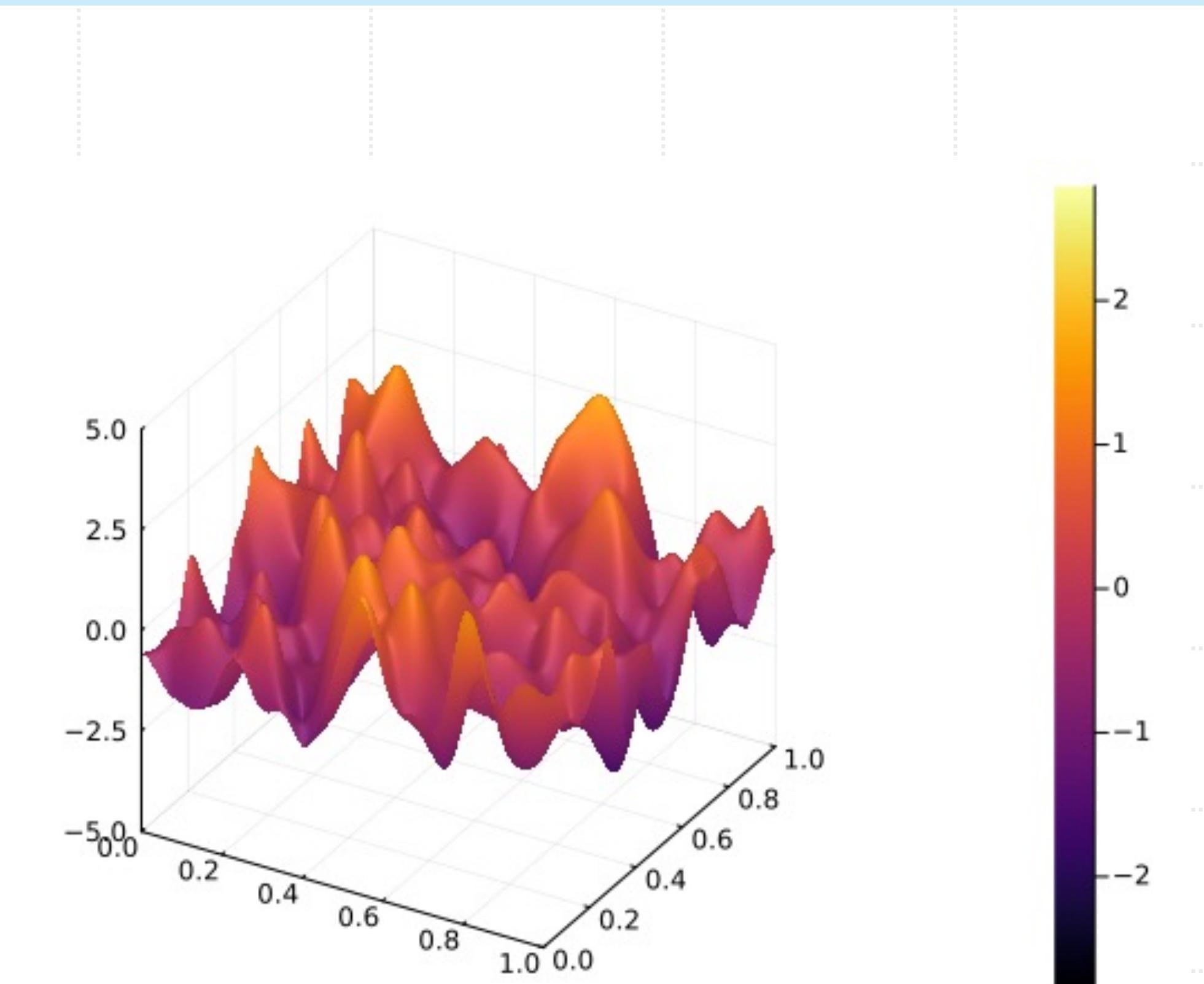
2. Real Space
generate random function
Wavelength \rightarrow Pattern size

INTERMISSION: GAUSSIAN RANDOM FIELDS

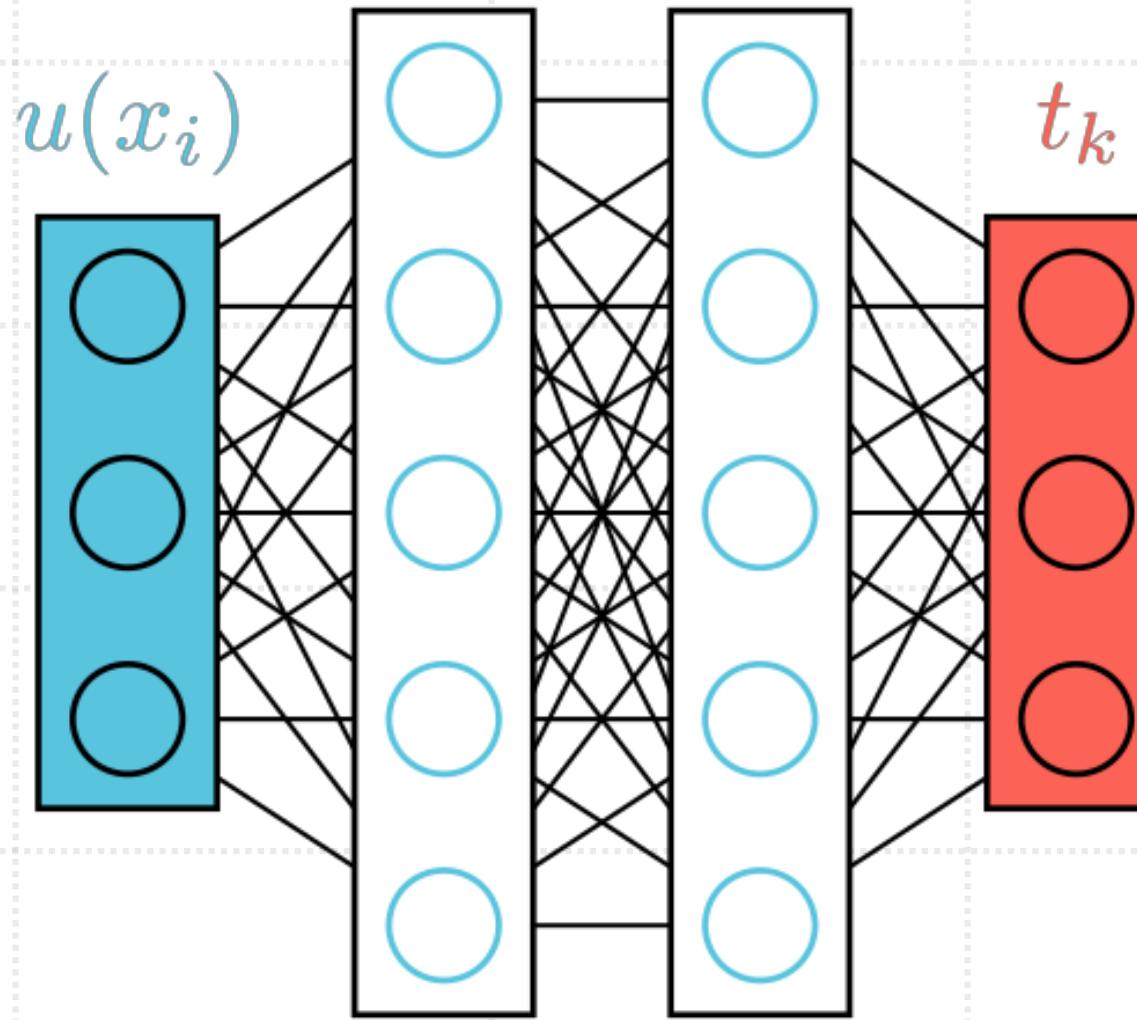
1D



2D



DEEP OPERATOR ARCHITECTURE



Idea: Use inductive bias
→ 2 Separate Neural Networks

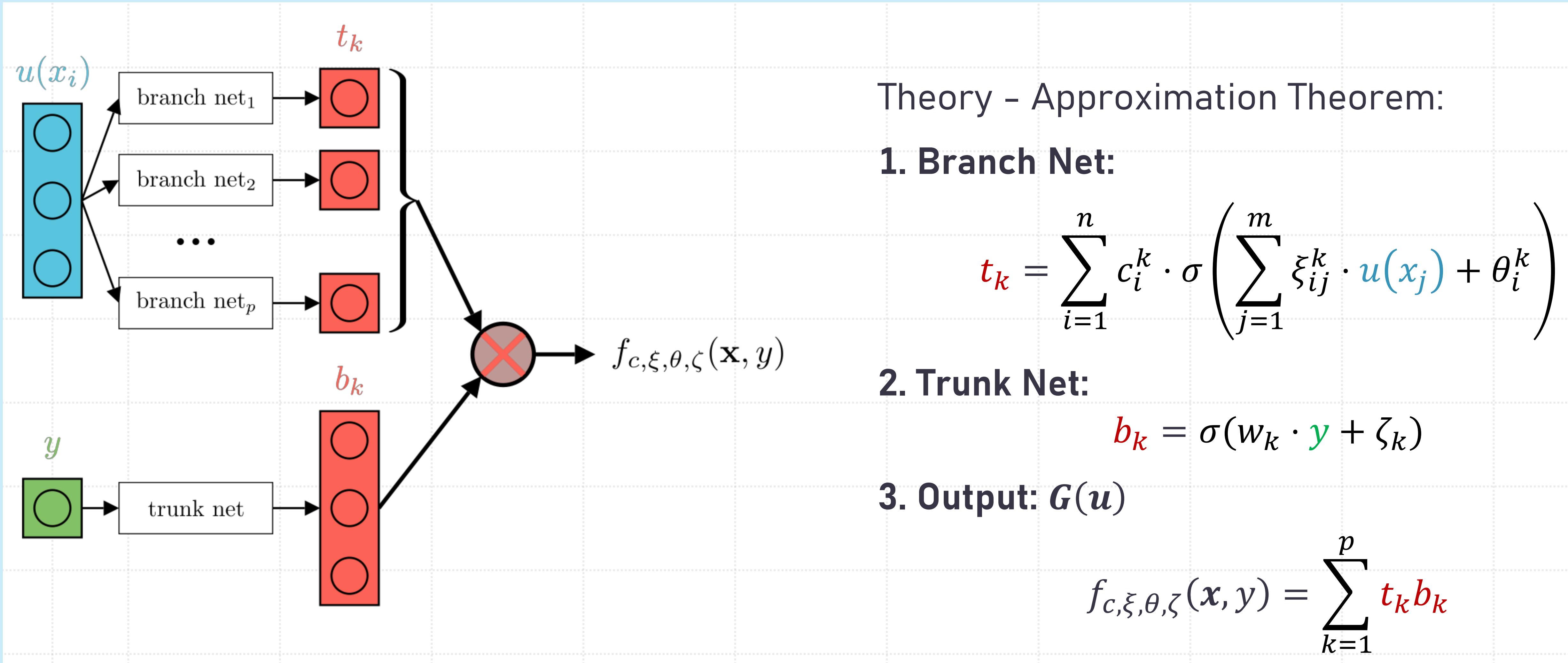
1. NN: “Branch Net”

$u(x) \rightarrow$ encode input space

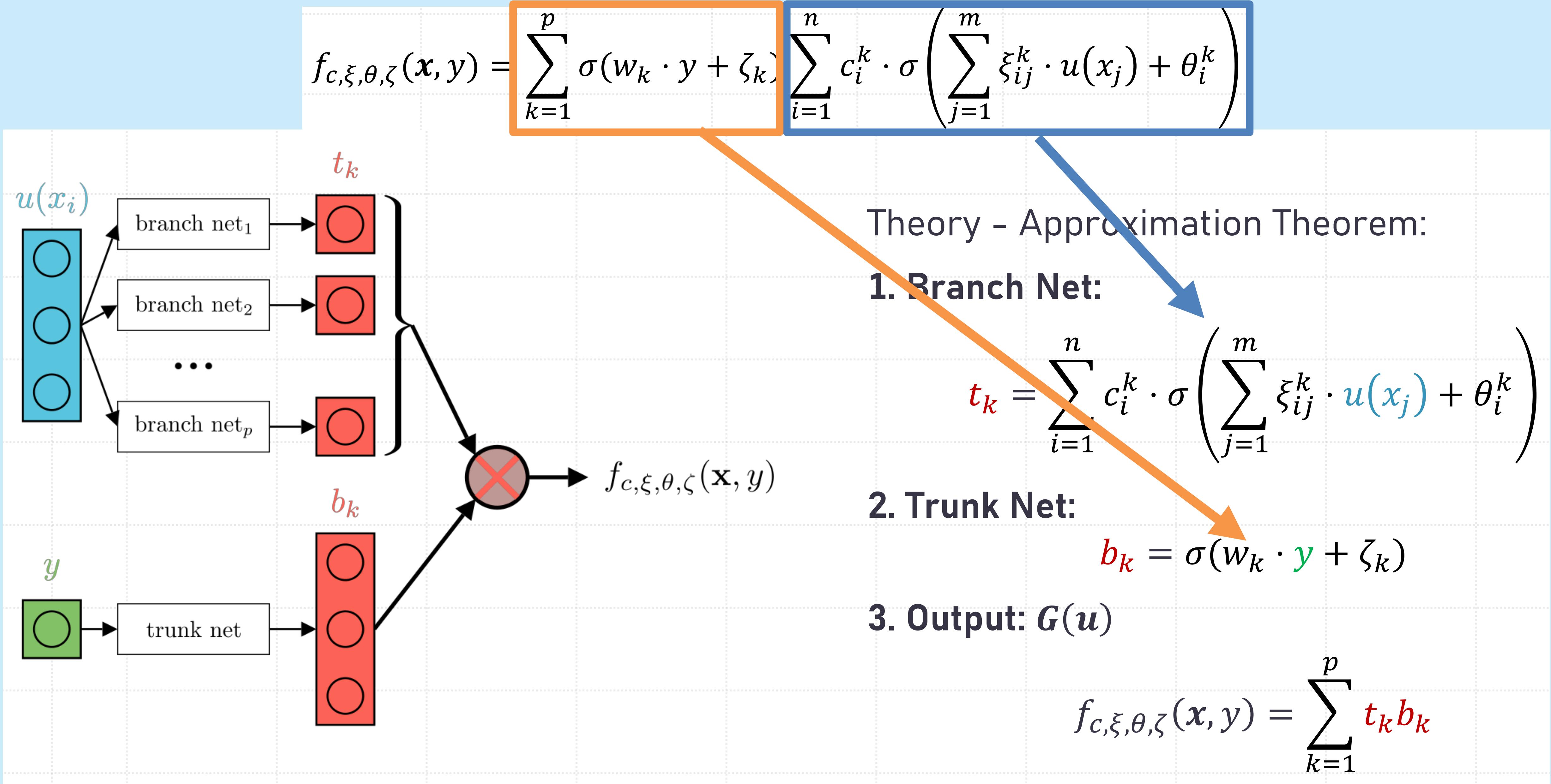
2. NN: “Trunk Net”

$y \rightarrow$ encode domain of output function

DEEP OPERATOR ARCHITECTURE



DEEP OPERATOR ARCHITECTURE

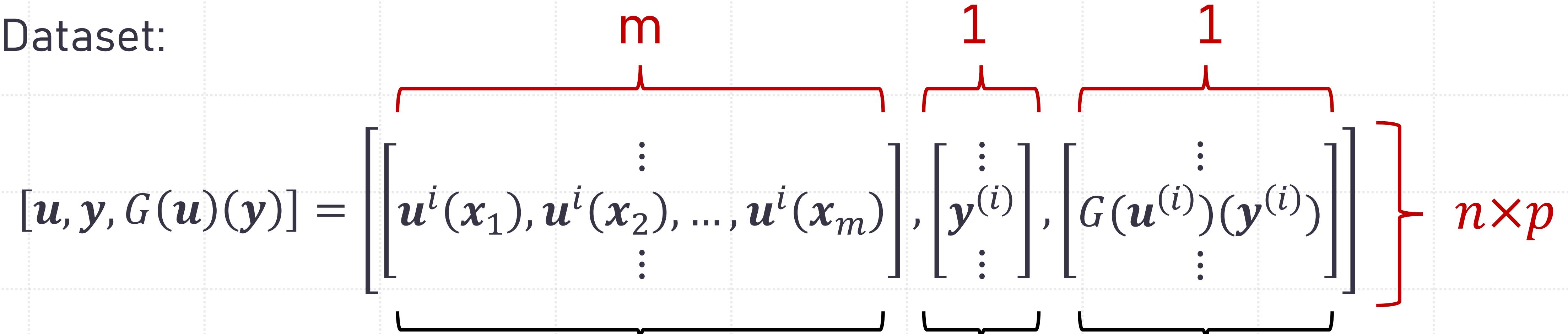


DEEP OPERATOR TRAINING

- Training Dataset:

$$[\mathbf{u}, \mathbf{y}, G(\mathbf{u})(\mathbf{y})] = \left[\begin{array}{c} \mathbf{u}^i(x_1), \mathbf{u}^i(x_2), \dots, \mathbf{u}^i(x_m) \\ \vdots \\ \mathbf{u}^i(x_m) \end{array}, \begin{array}{c} \mathbf{y}^{(i)} \\ \vdots \\ \mathbf{y}^{(i)} \end{array}, \begin{array}{c} G(\mathbf{u}^{(i)})(\mathbf{y}^{(i)}) \\ \vdots \\ G(\mathbf{u}^{(i)})(\mathbf{y}^{(i)}) \end{array} \right] \quad n \times p$$

function values
→ Branch Loc. Target Output
→ Trunk



- Can evaluate $G(u)(y)$ at different Locations y (for same u)
 - Nr. reuses of $\mathbf{u}^{(i)}$ → Hyperparameter p

DEEP OPERATOR TRAINING

Using Simulated Data:

1. Generate Input Functions $u(x)$
 - e.g. Gaussian Random Fields
 2. Numerically solve Problem: $G(u)(y)$
 3. Train NN on Data
- DeepONet has learned Operator



Real Data:

1. Data preparation
 2. Train NN on Data
- DeepONet has learned Operator

DEEP OPERATOR LEARNING: APPLICATION EXAMPLE**

- DeepONet applied to 1D Heat Diffusion System:

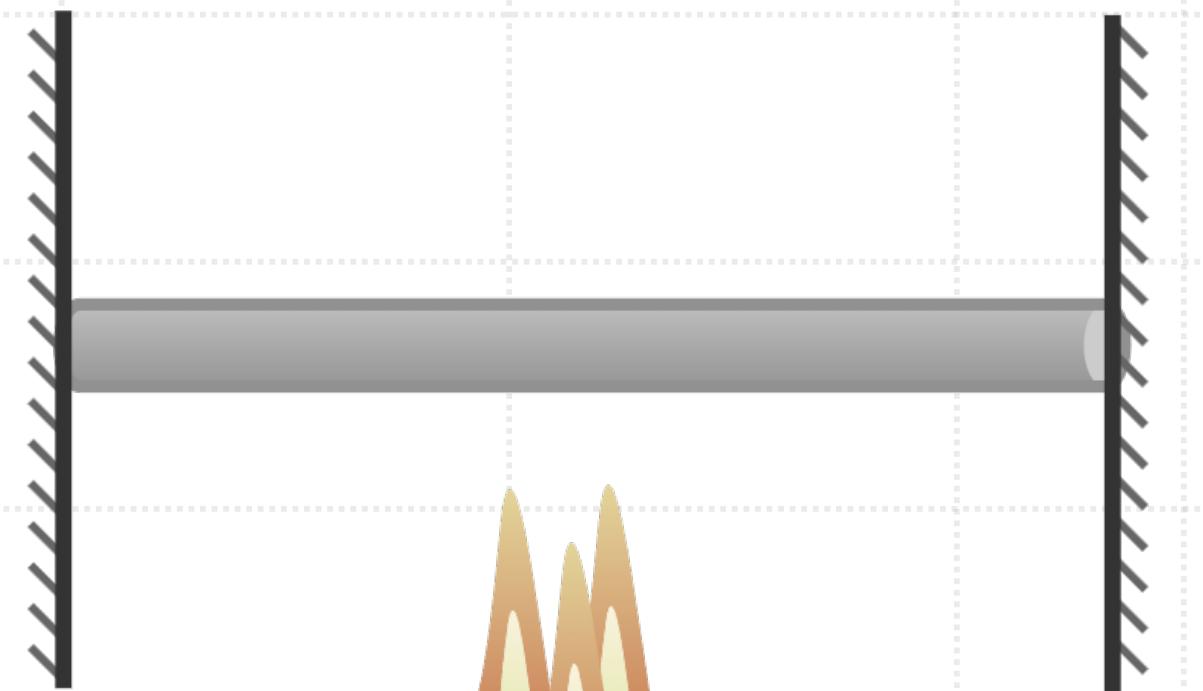
$$\partial_t s = D \partial_x^2 s + u(x), \quad (x, t) \in [0, 1] \times [0, 1]$$

1) 2)

1. Diffusion Term: Heat spreads out over t

2. Source Term: our input function $u(x): \mathbb{R} \rightarrow \mathbb{R}$

Predict: Temperature Dynamics: $s(x, t): \mathbb{R}^2 \rightarrow \mathbb{R}$

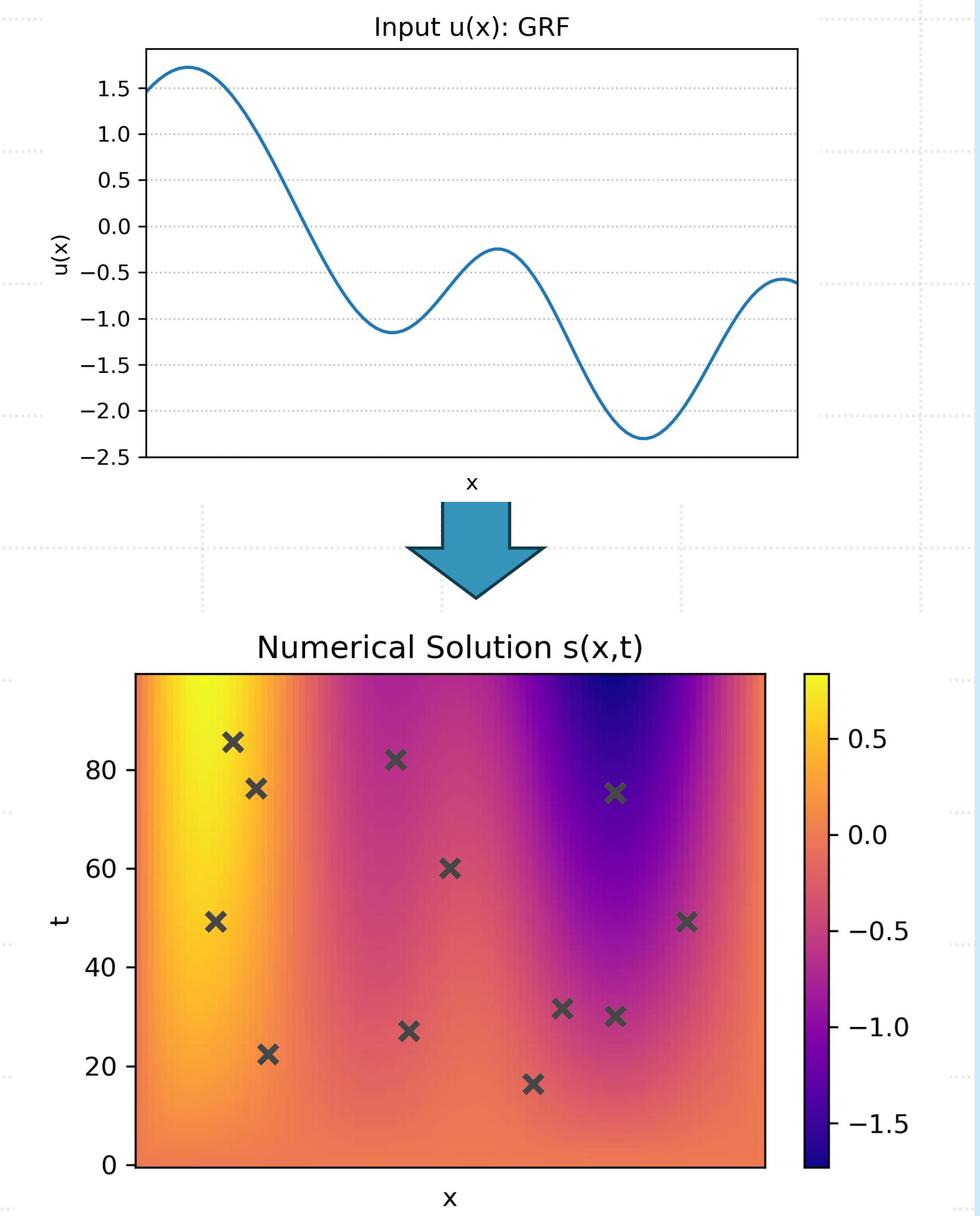


DEEP OPERATOR LEARNING: APPLICATION EXAMPLE

Code Example - Process

Using  DeepXDE: [5]

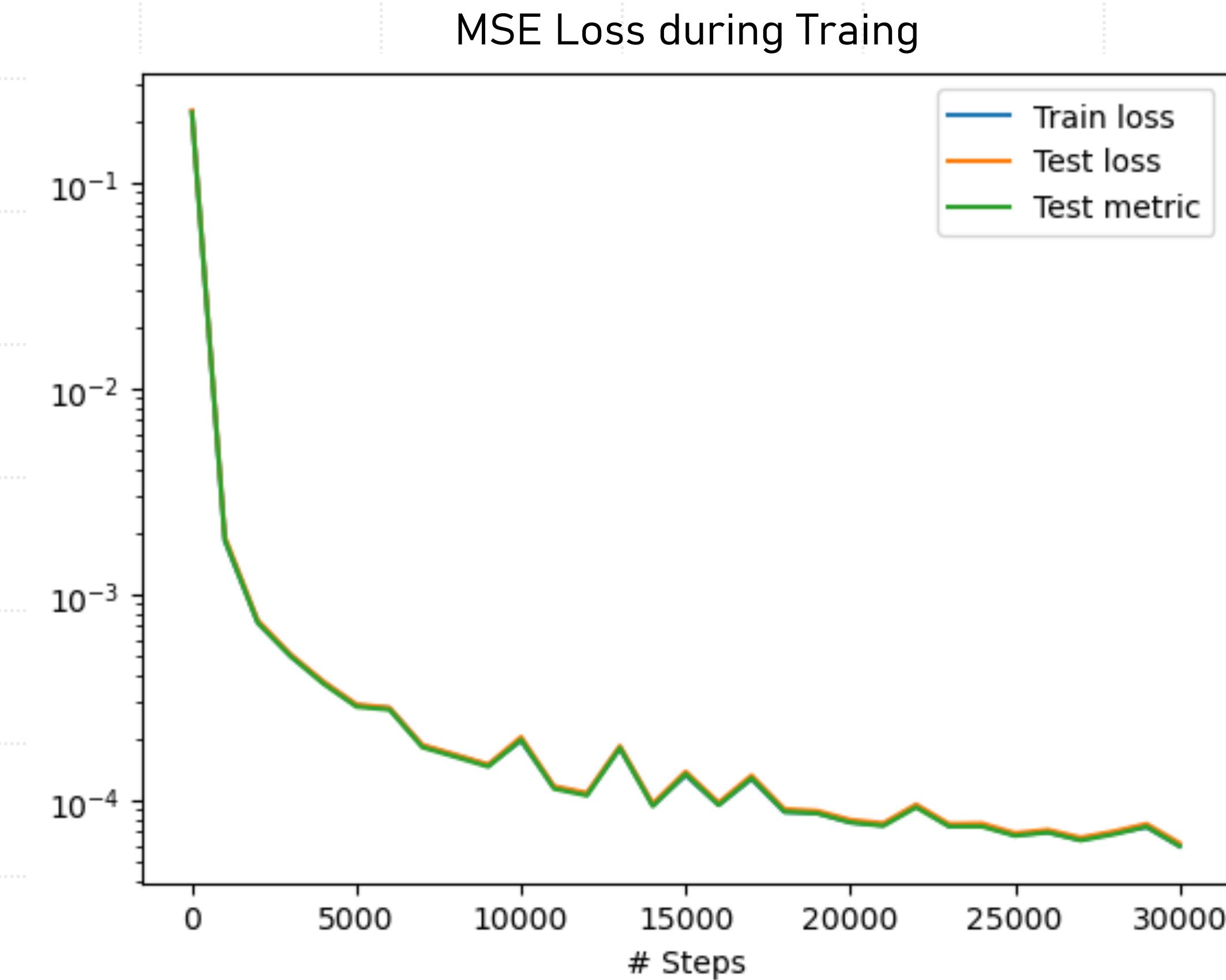
1. Work on 100×100 Grid (x, t)
 - 100 space pts. = sensors
2. Get $u(x)$ from GRF
 - Length Scale: $l = 0.15$
3. Solve: $\partial_t s = D \partial_x^2 s + u(x)$
4. Sample $s(x, t)$ at 100 locations $y = (x, t)$
→ Train Model on Datapoints



DEEP OPERATOR LEARNING: APPLICATION EXAMPLE

Code Example - Training

- 2 hidden Layers \times 40 Neurons
- 30k Train Samples
- 30k Test Samples
- ~6h Training



DEEP OPERATOR LEARNING: APPLICATION EXAMPLE I

source term: $u(x) = \sin(2\pi x)$

Numerics

-

One full
simulation

Runtime: 9.18ms

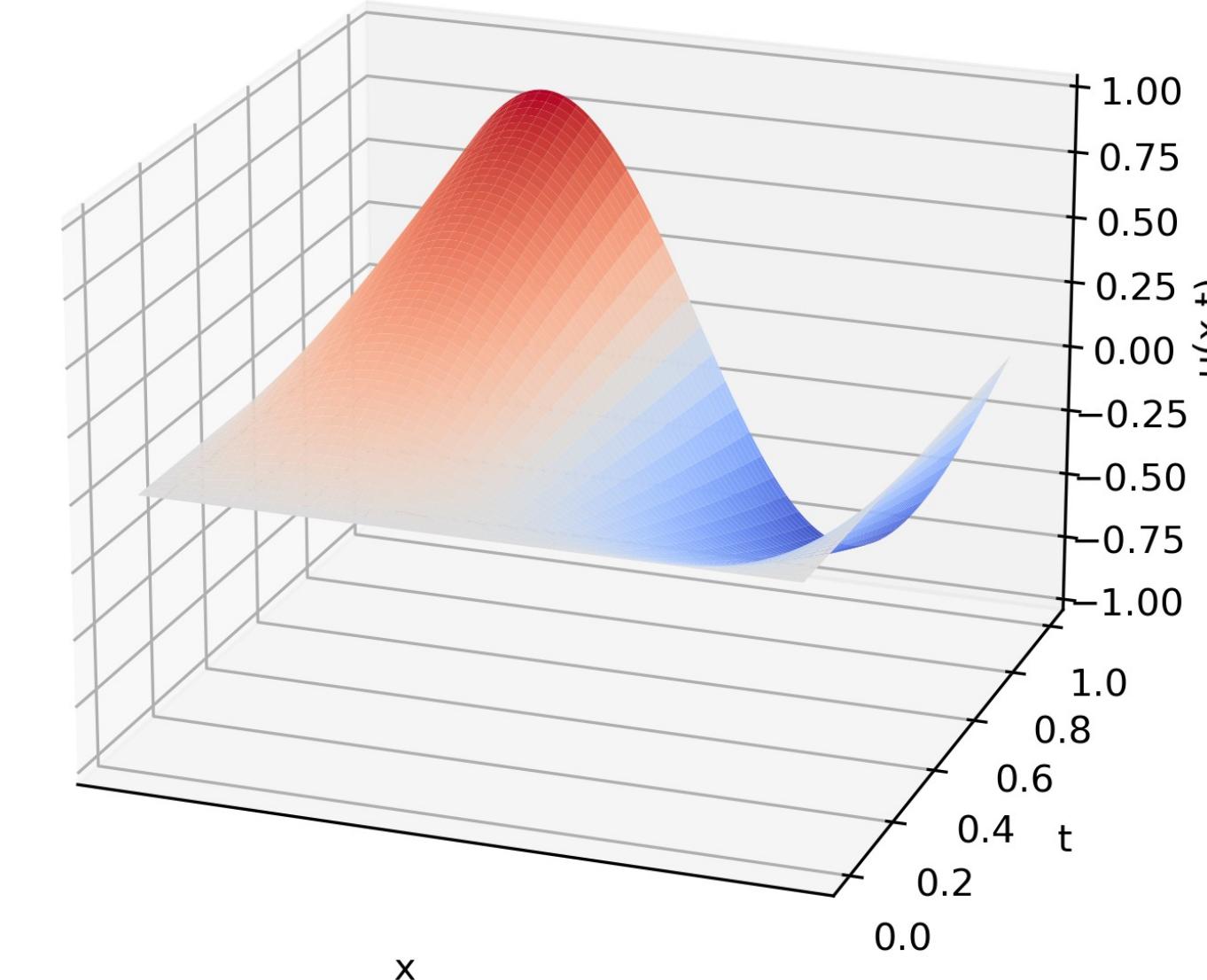
DeepONet

Error: $\sim 10^{-3}$

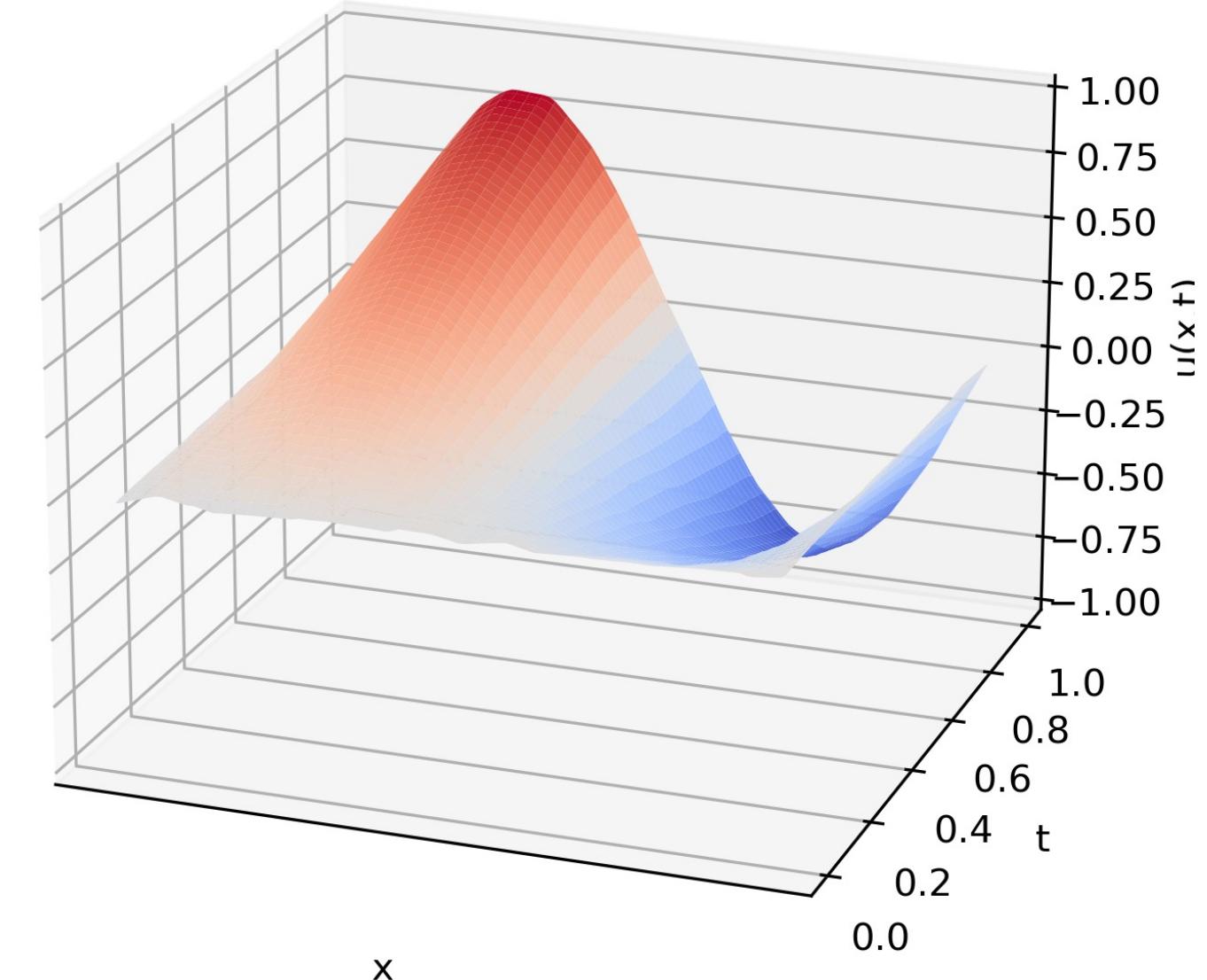
NN inferences

Time: 2.86ms

Numerical Solution



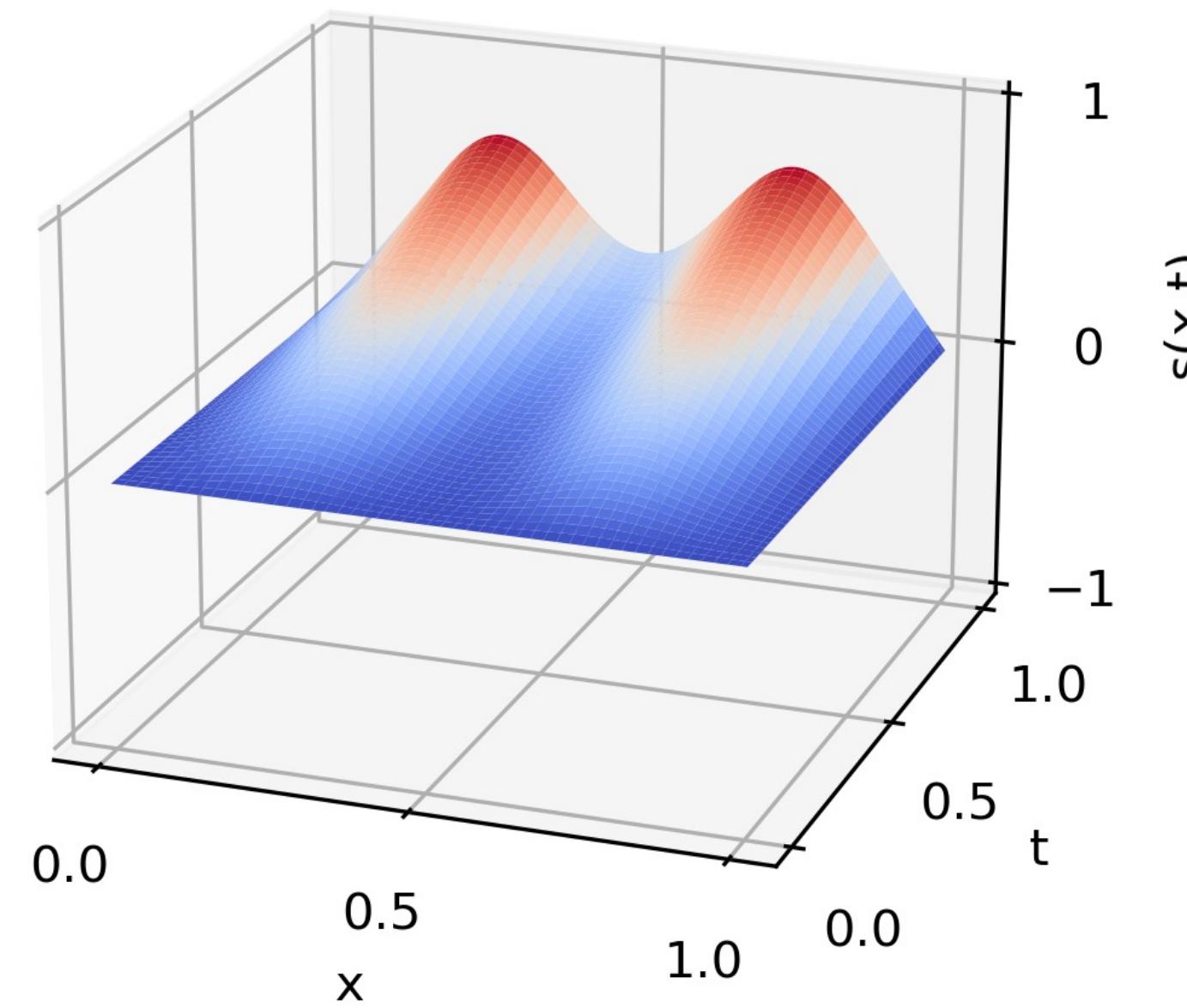
DeepONet Output



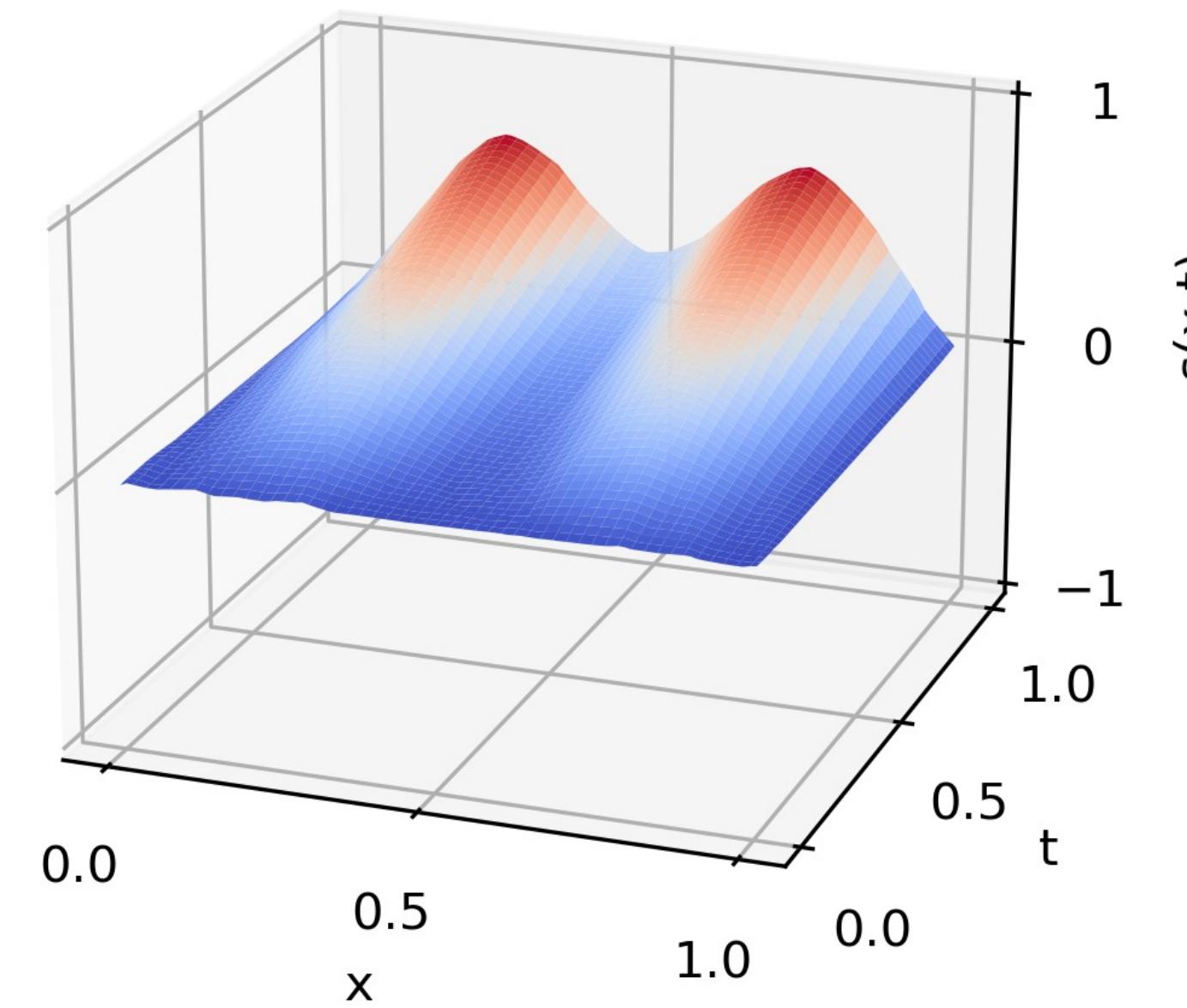
DEEP OPERATOR LEARNING: APPLICATION EXAMPLE II

source term: $u(x)$: Double gaussian

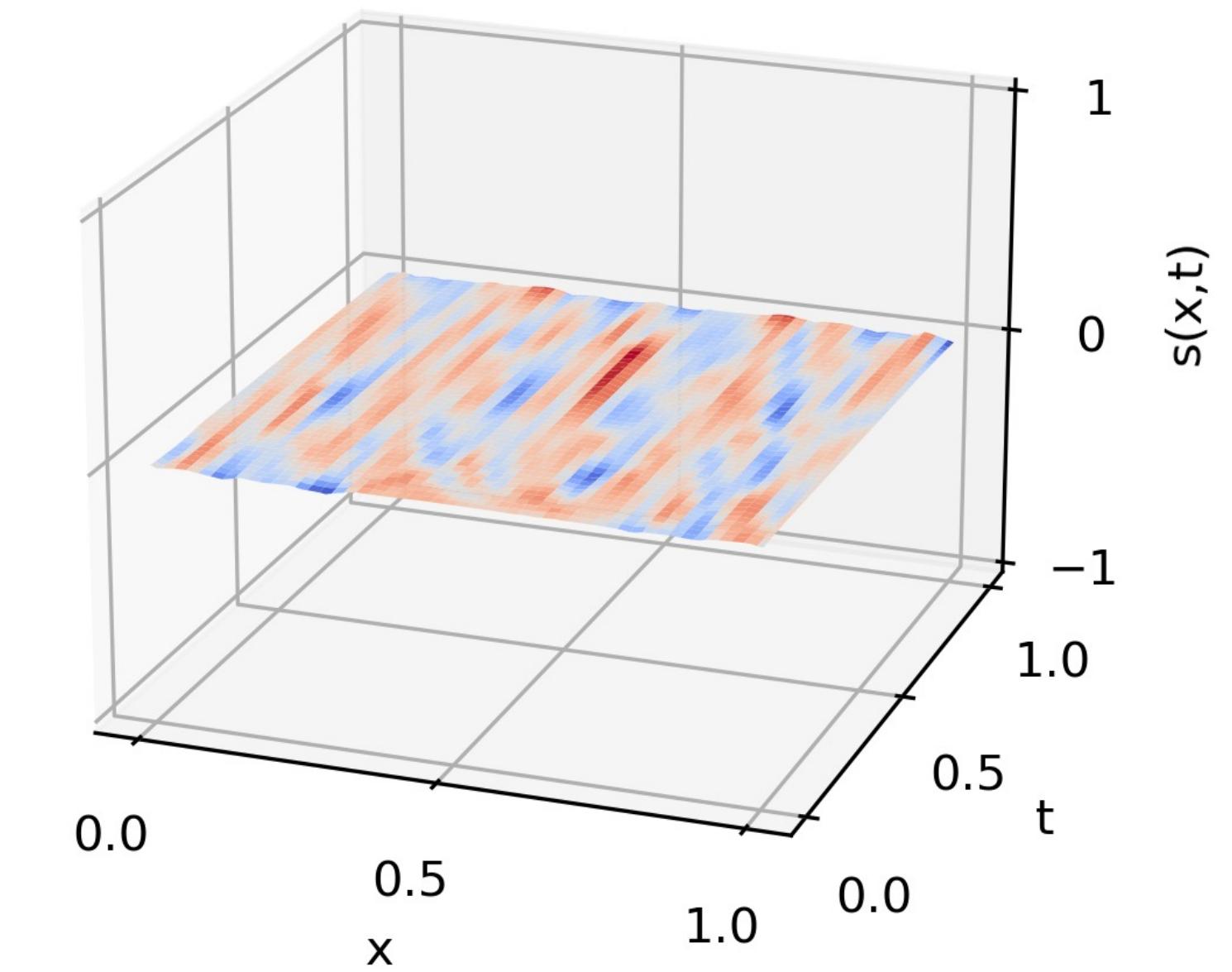
Numerical Solution



DeepONet Output



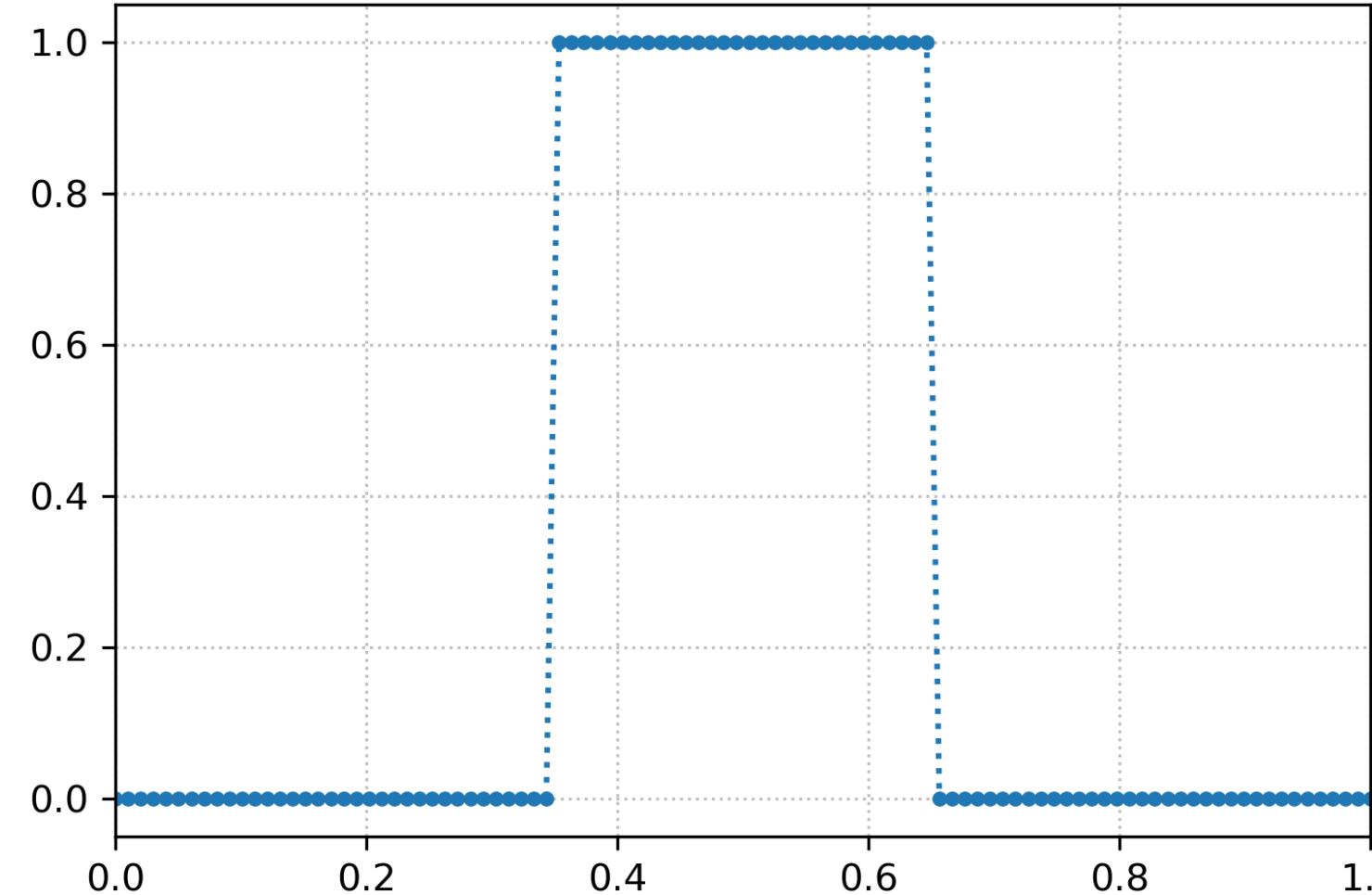
Pointwise Error



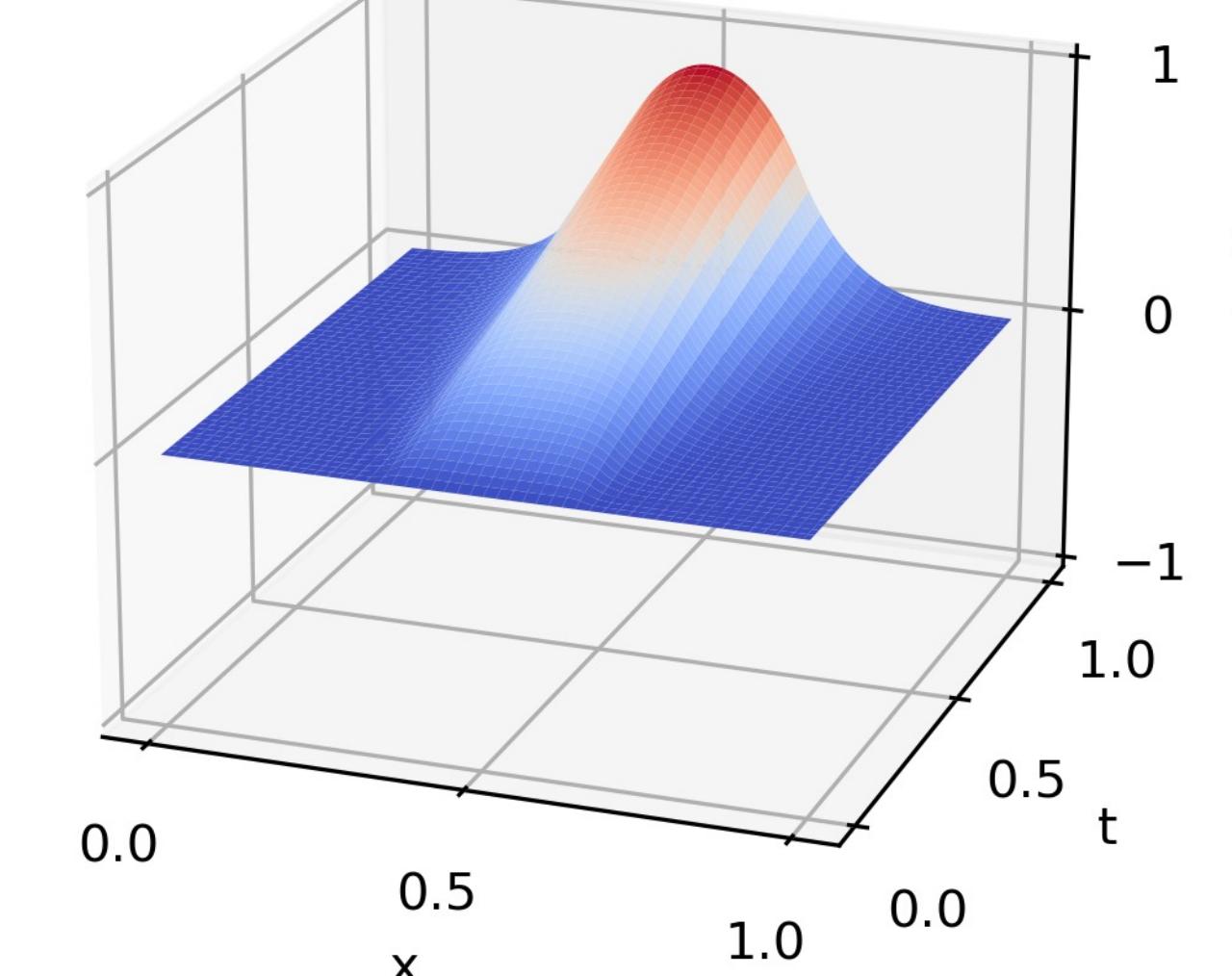
DEEP OPERATOR LEARNING: APPLICATION EXAMPLE III

source term: $u(x)$: Square Wave

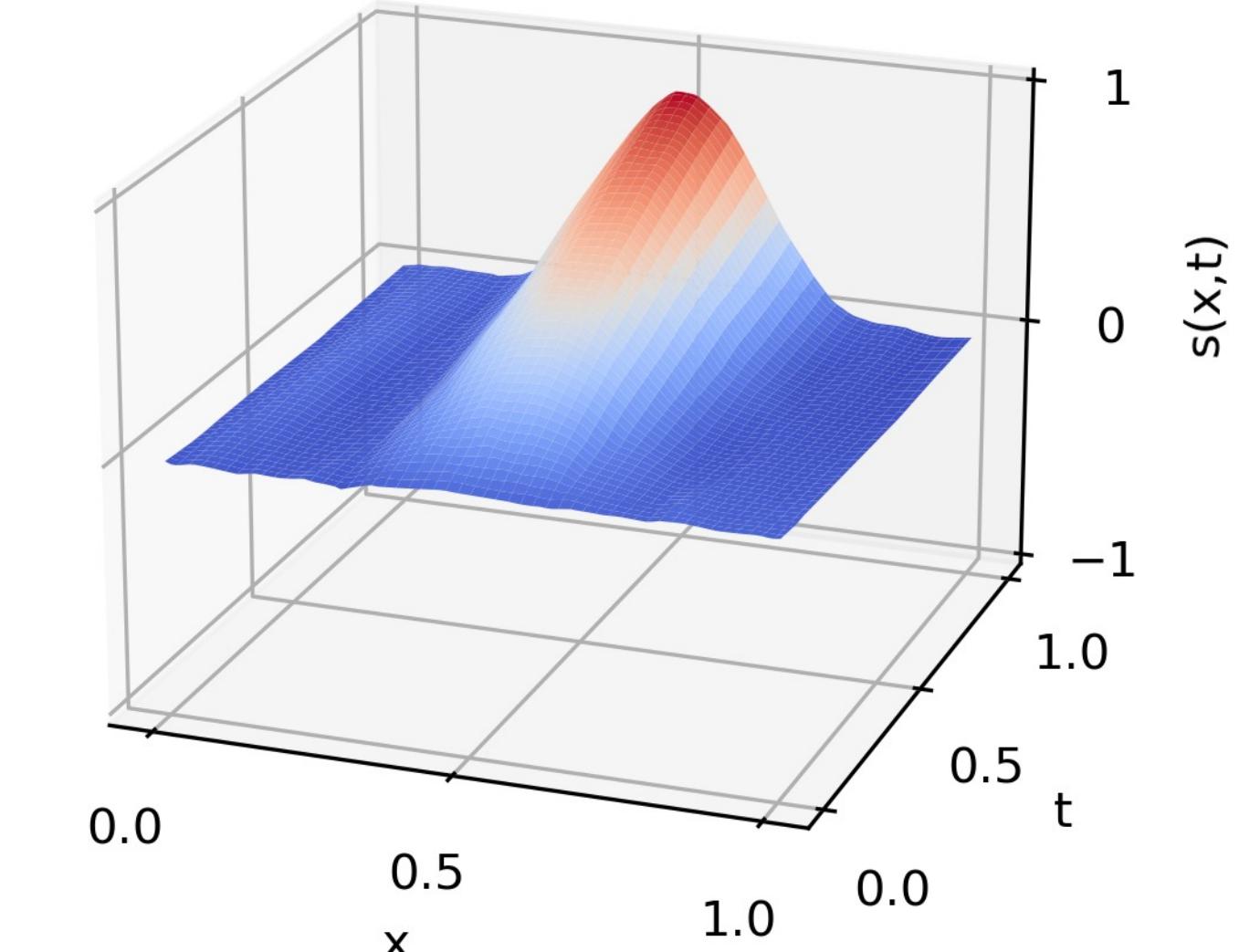
Input $u(x)$: Square Wave



Numerical Solution



DeepONet Output



DISCUSSION: WHEN TO USE DEEPONET?

Whenever there is a continuous operator!

Poisson:	$\nabla^2 V(x) = \rho(x)$	$G : \rho(x) \rightarrow V(x)$
Diffusion-Reaction:	$D * \nabla^2 s - \partial_t s = ks^2 + u(x)$	$G : u(x) \rightarrow s(x, t)$
QM Time Evolution:	$\psi(t) = e^{-iHt/\hbar} \psi_0$	$G : \psi_0 \rightarrow \psi(t)$

...ok, but when *should* I use DeepONet?

ADVANTAGES AND DISADVANTAGES OF DEEPONET

Pros

Learns the “solution space”
Fast and very efficient
High flexibility
Generalizes well
“Data-only” mode

Cons

No continuous output
“Curse of dimensionality”
Requires training and data generation
Dependency on sensor locations and input data

USE CASES FOR DEEPONETS

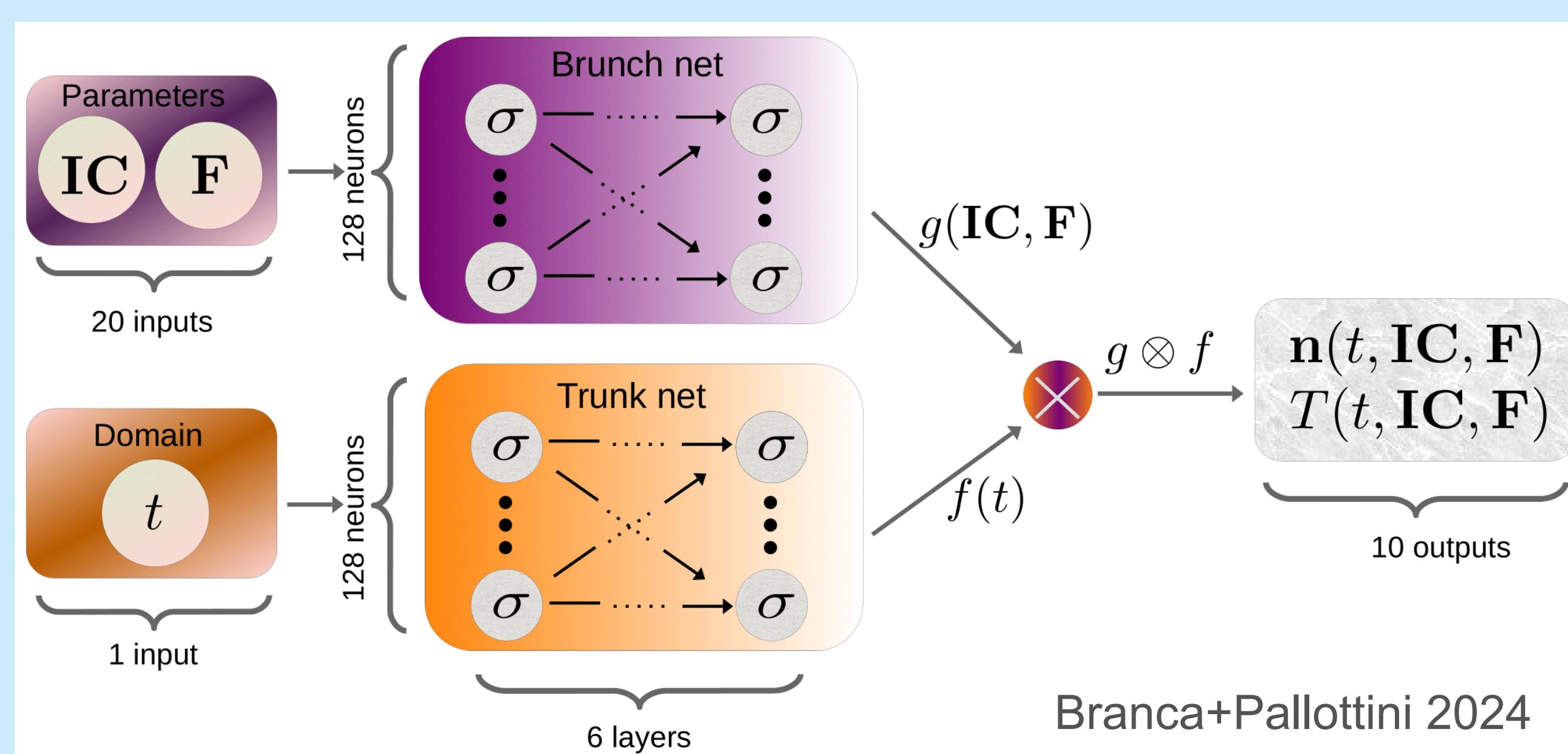
Two Scenarios:

1. Measurement data with unknown relation
(that is assumed to be representable by a continuous operator)
2. Known relation, obtaining outputs for many different “variants”/initial conditions **fast**

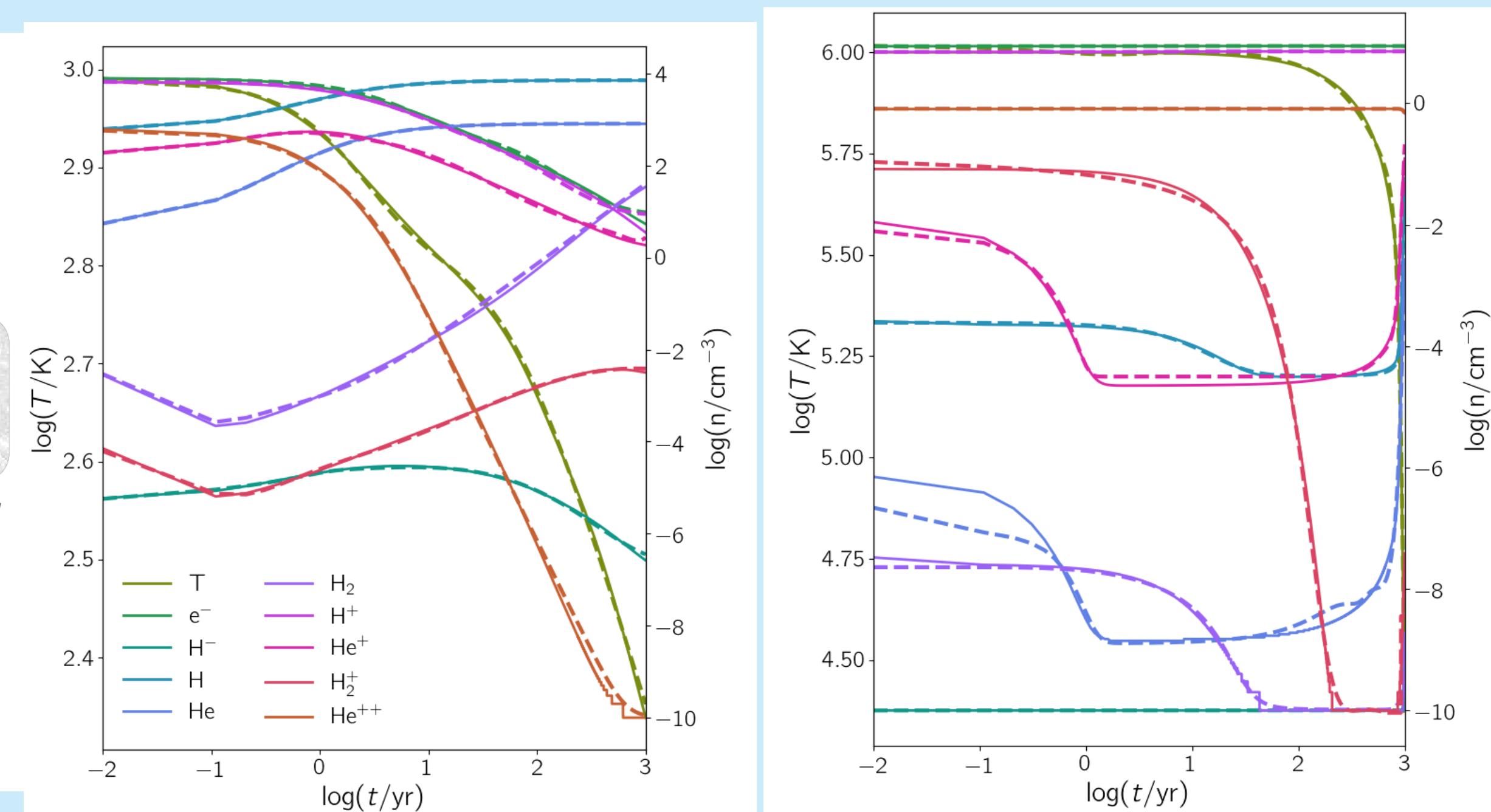
**Use DeepONet to replace repeated
(complex) numerical simulations**

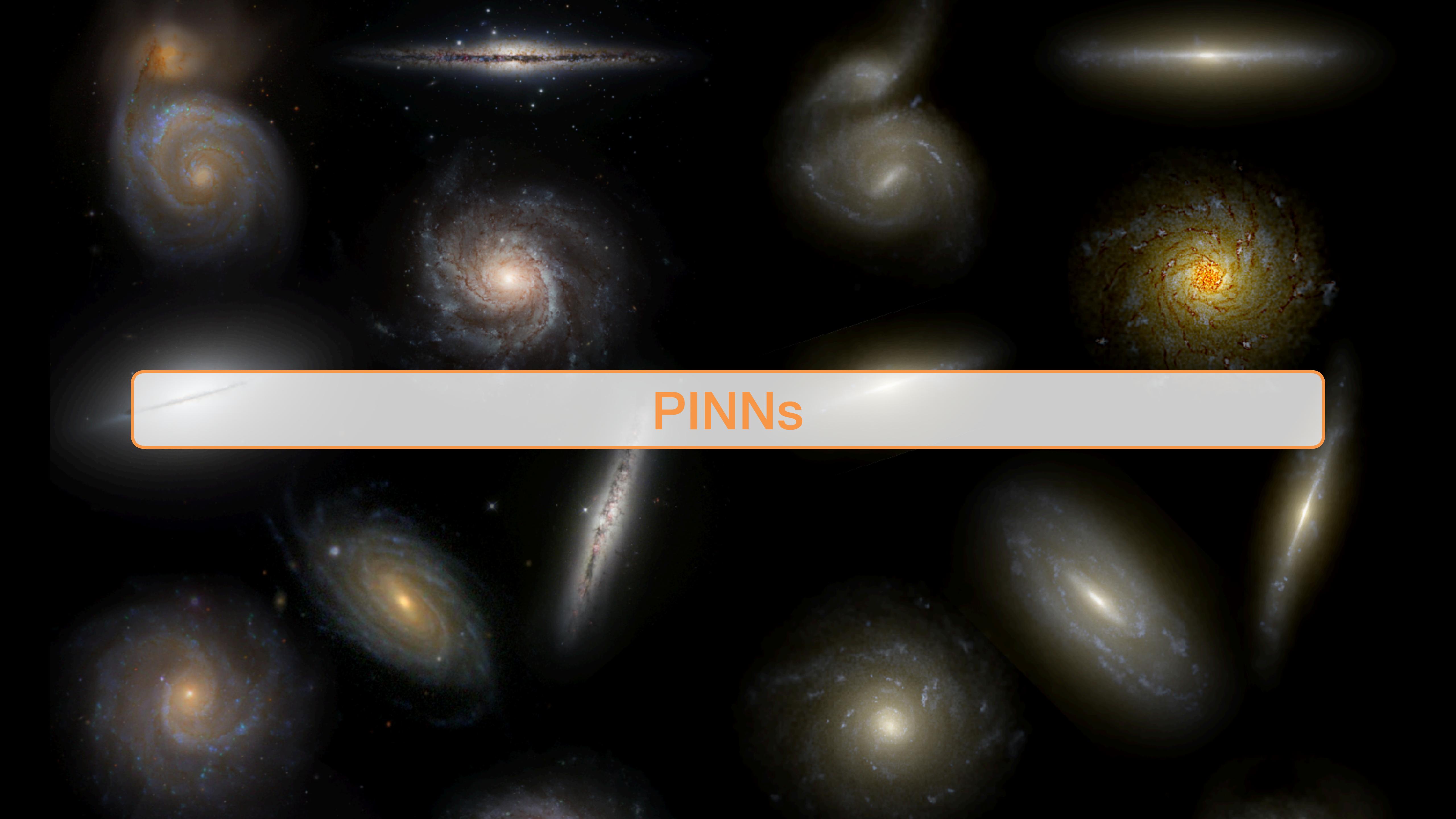
DEEP OPERATORS IN ASTROPHYSICS

- PPONet: Deep Operator Networks for Fast Prediction of Steady-State Solutions in Disk-Planet Systems (Mao+2023)
- Emulating the interstellar medium chemistry with neural operators (Branca+Pallottini 2024)
- CODES Benchmark for neuralODEs and Operator Learning for astrochemistry (Janssen,Sulzer+Buck 2024)



Branca+Pallottini 2024





PINNs

PINNS VS. NEURAL ODES

PINNs = Physics Informed Neural Networks

Neural ODE

$$\frac{dx}{dt} = f_\theta(t, x(t))$$

neural net r.h.s. of DE

PINN

$$x(t) = f_\theta(t, x_0)$$

neural net approx.
solution of DE

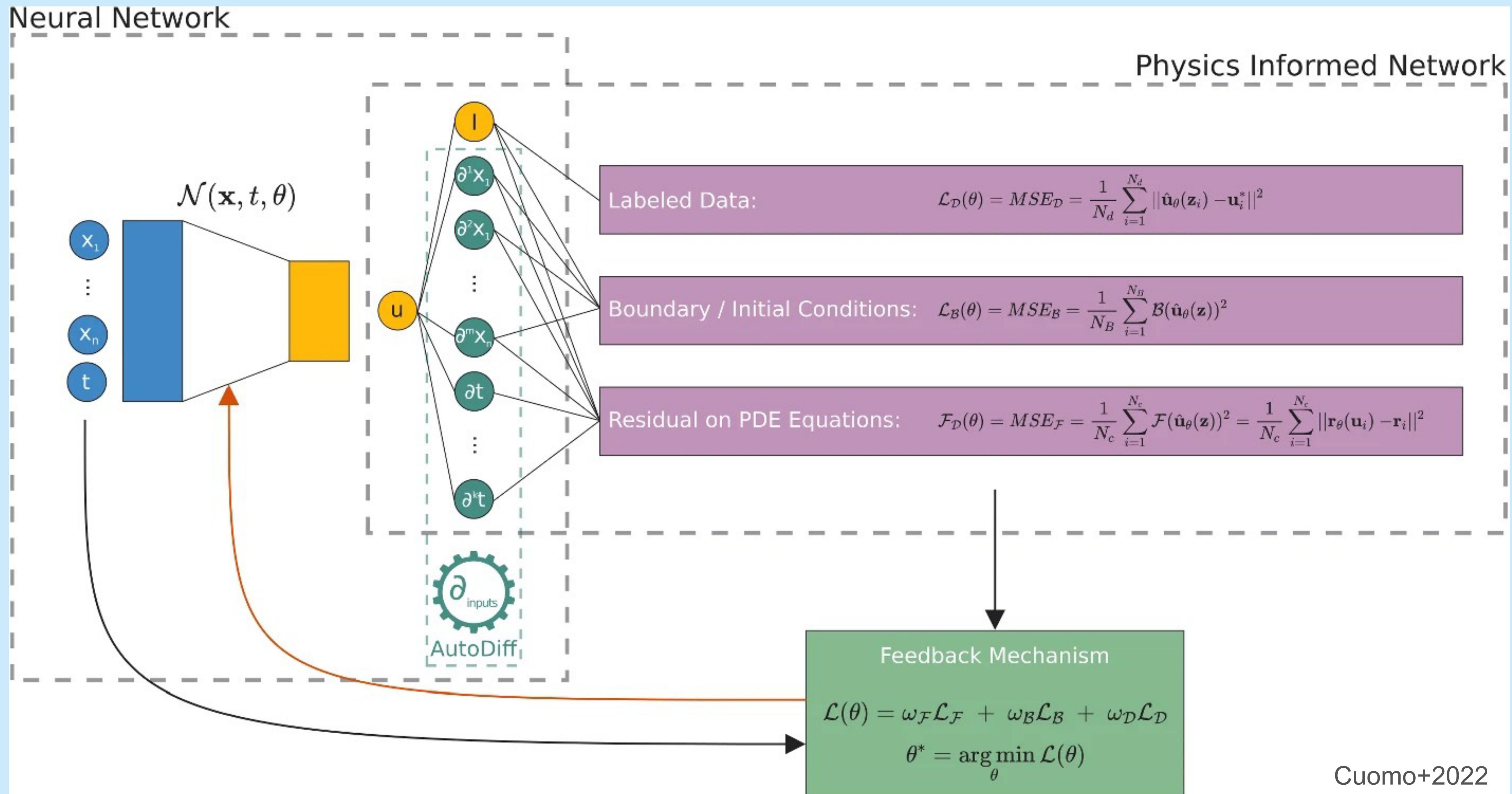
Hamiltonian Neural Net

$$\frac{dq}{dt} = \frac{\partial H_\theta(p, q)}{\partial p}$$

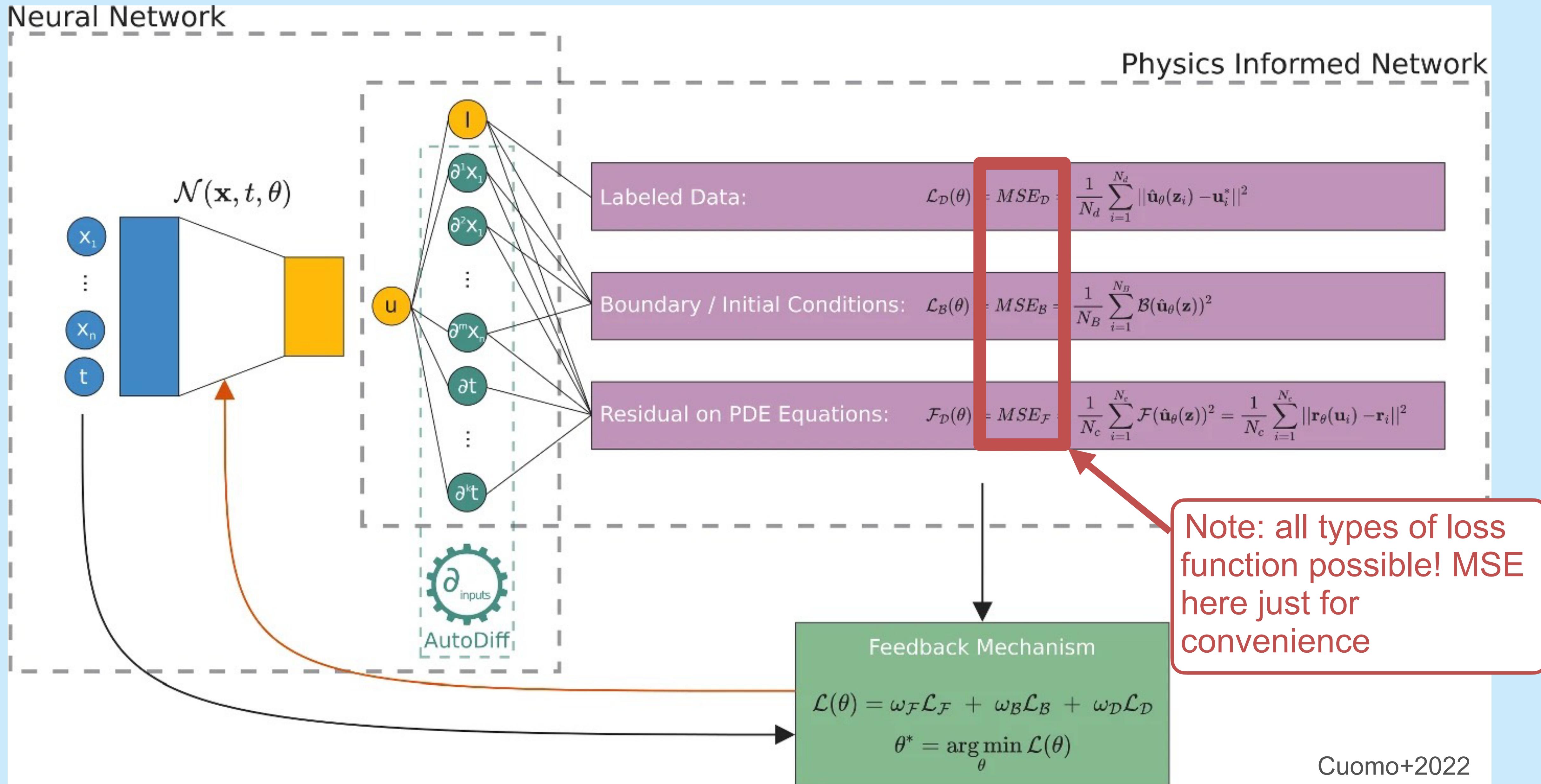
$$\frac{dp}{dt} = -\frac{\partial H_\theta(p, q)}{\partial q}$$

neural net represents a
Hamiltonian

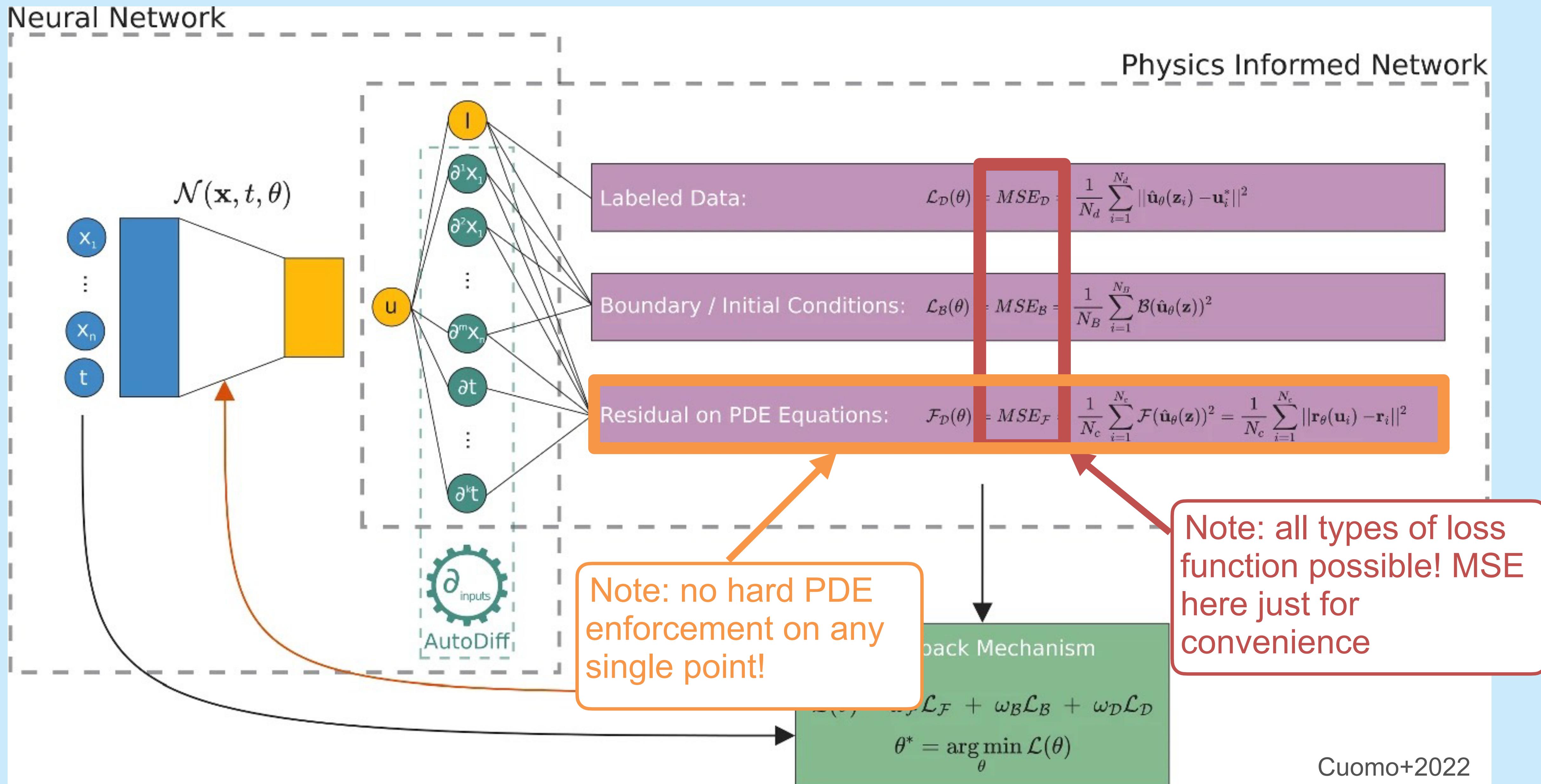
PINNS VS. NEURAL ODES



PINNS VS. NEURAL ODES



PINNS VS. NEURAL ODES



PHYSICS INFORMED NEURAL NETS

Loss function for PINNs

Differential Equation: $\mathcal{F}[u(x, y)] = f(x, y)$

Dataset: $(x_i, y_i, u_i); i = 1, \dots, N_{data}$

Collocation points: $(x_j, y_j); j = 1, \dots, N_c$

Initial Condition: (x_0, y_0, u_0)

$$L_{DiffEq} = \frac{1}{N_c} \sum_{j=1}^{N_c} (\mathcal{F}[u(x_j, y_j)] - f(x_j, y_j))^2$$

$$L_{total} = \omega_{data} \cdot L_{data} + \omega_{DiffEq} \cdot L_{DiffEq} + \omega_{IC} \cdot L_{IC}$$

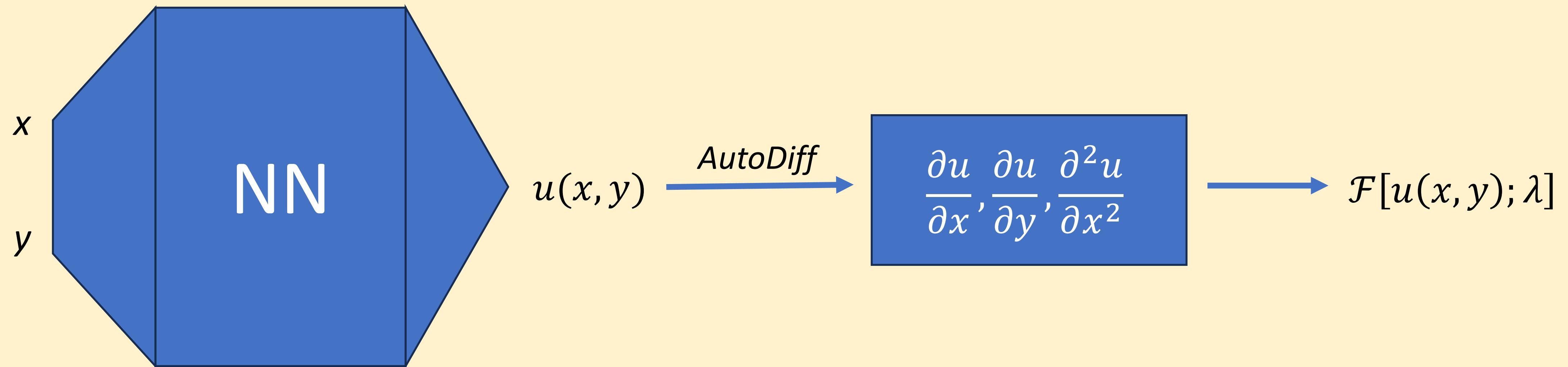
$$L_{data} = \frac{1}{N_{data}} \sum_{i=1}^{N_{data}} (u(x_i, y_i) - u_i)^2$$

$$L_{IC} = (u(x_0, y_0) - u_0)^2$$

PINNS: EXAMPLE

- Example: $\mathcal{F}[u(x, y); \lambda] = \frac{\partial u}{\partial x} + \lambda_1 \frac{\partial u}{\partial y} + \lambda_2 \frac{\partial^2 u}{\partial x^2}$ and $f(x, y) = x^2 + y$
 - In order to calculate \mathcal{F} , we need the derivatives: $\frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}, \frac{\partial^2 u}{\partial x^2}$
- Automatic differentiation

$$\mathcal{F}[u(x, y); \lambda] = f(x, y)$$
$$\lambda := \{\lambda_1, \lambda_2\}$$

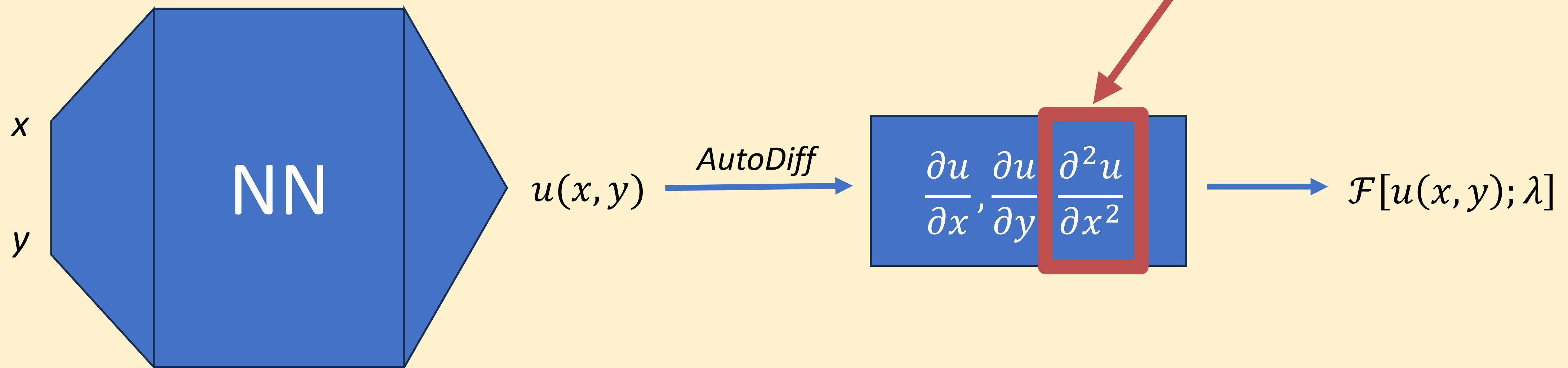


PINNS: EXAMPLE

- Example: $\mathcal{F}[u(x, y); \lambda] = \frac{\partial u}{\partial x} + \lambda_1 \frac{\partial u}{\partial y} + \lambda_2 \frac{\partial^2 u}{\partial x^2}$ and $f(x, y) = x^2 + y$
- In order to calculate \mathcal{F} , we need the derivatives: $\frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}, \frac{\partial^2 u}{\partial x^2}$
- Automatic differentiation

$$\mathcal{F}[u(x, y); \lambda] = f(x, y)$$
$$\lambda := \{\lambda_1, \lambda_2\}$$

Careful with RELU!
Vanishing Hessian



NEURAL ODES VS. PINNS

NDEs are a *modelling approach*:

- Learn θ in the model $\frac{dy}{dt}(t) = f_\theta(t, y(t))$, so that you match your data.
- Loss function: $\min_\theta \| y(t) - \text{data}(t) \|^2$

PINNs are a *numerical method*:

- Learn φ in $y(t) = y_\varphi(t)$, so that it matches the diffeq $\frac{dy}{dt}(t) = f(t, y(t))$.
- Loss function: $\min_\varphi \| \frac{dy}{dt}(t) - f(t, y(t)) \|^2$

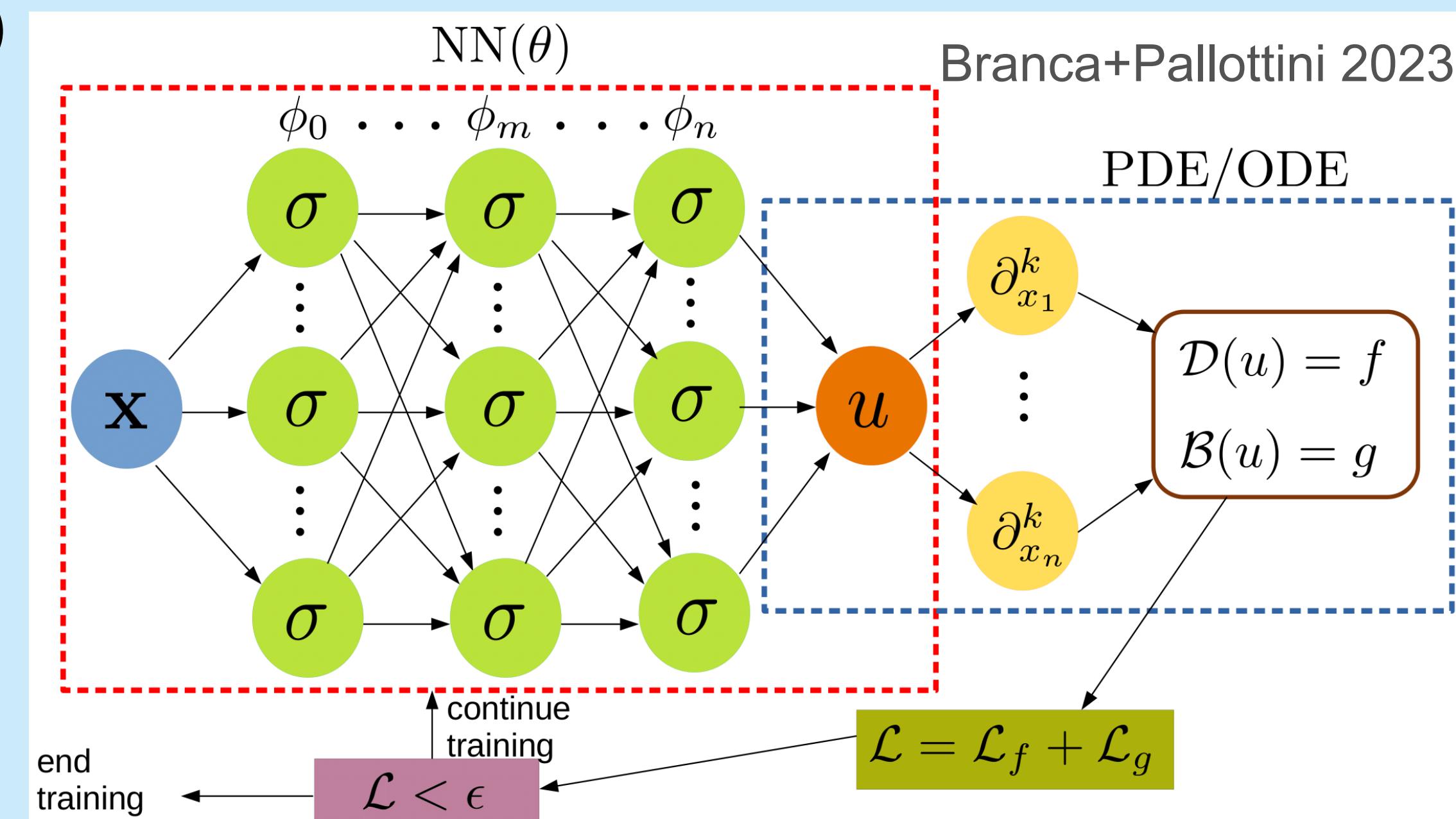
These terms have precise meaning, but the terms often muddled up, as sometimes both techniques are done at the same time!

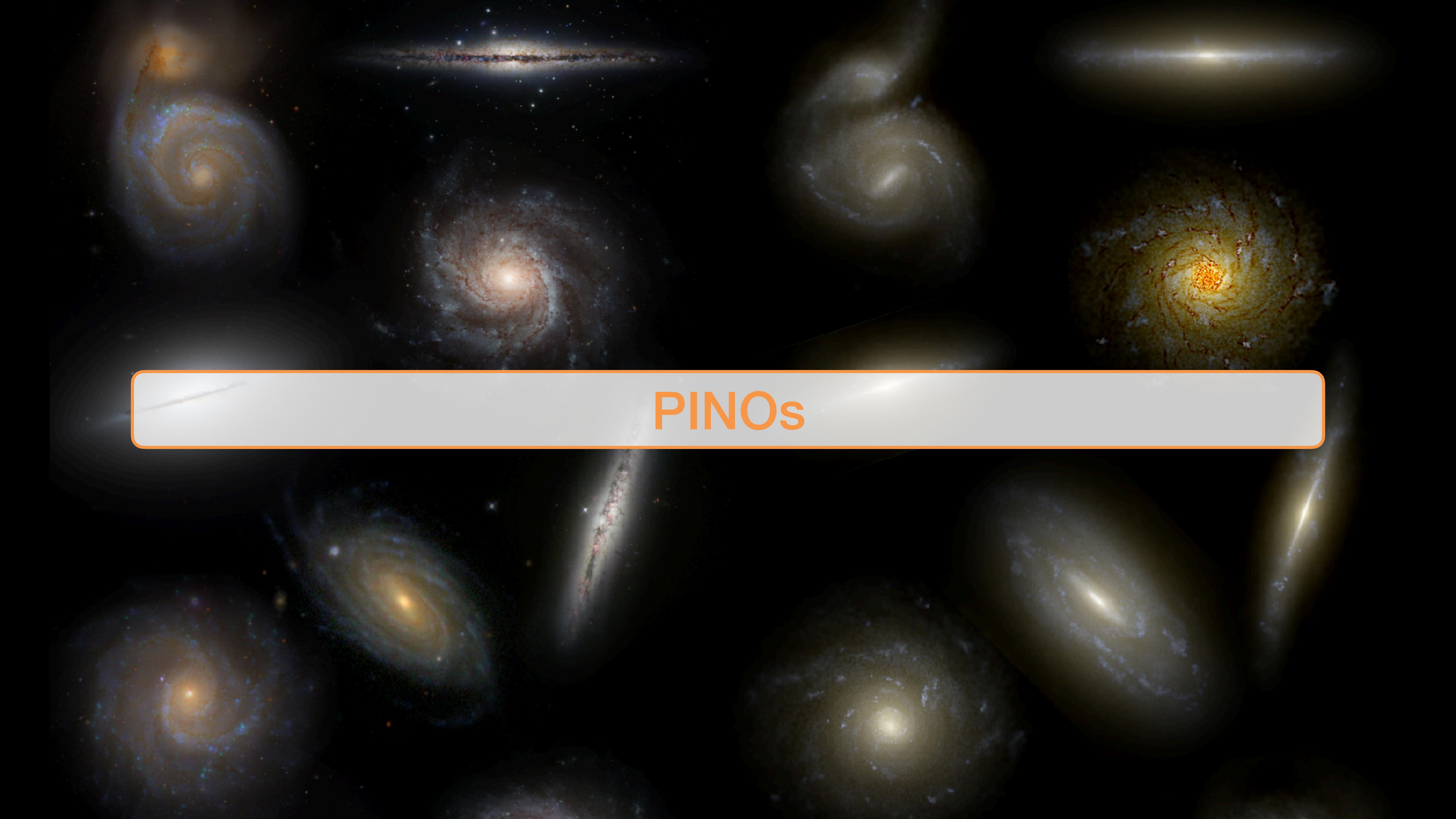
- Loss function: $\min_{\theta, \varphi} (\| y(t) - \text{data}(t) \|^2 + \| \frac{dy}{dt}(t) - f(t, y(t)) \|^2)$

PINNs are useful when traditional numerical methods fail (high-dimensional PDEs; nonlocal integral operators.). But mostly... just use traditional numerical methods.

PINNS IN ASTROPHYSICS

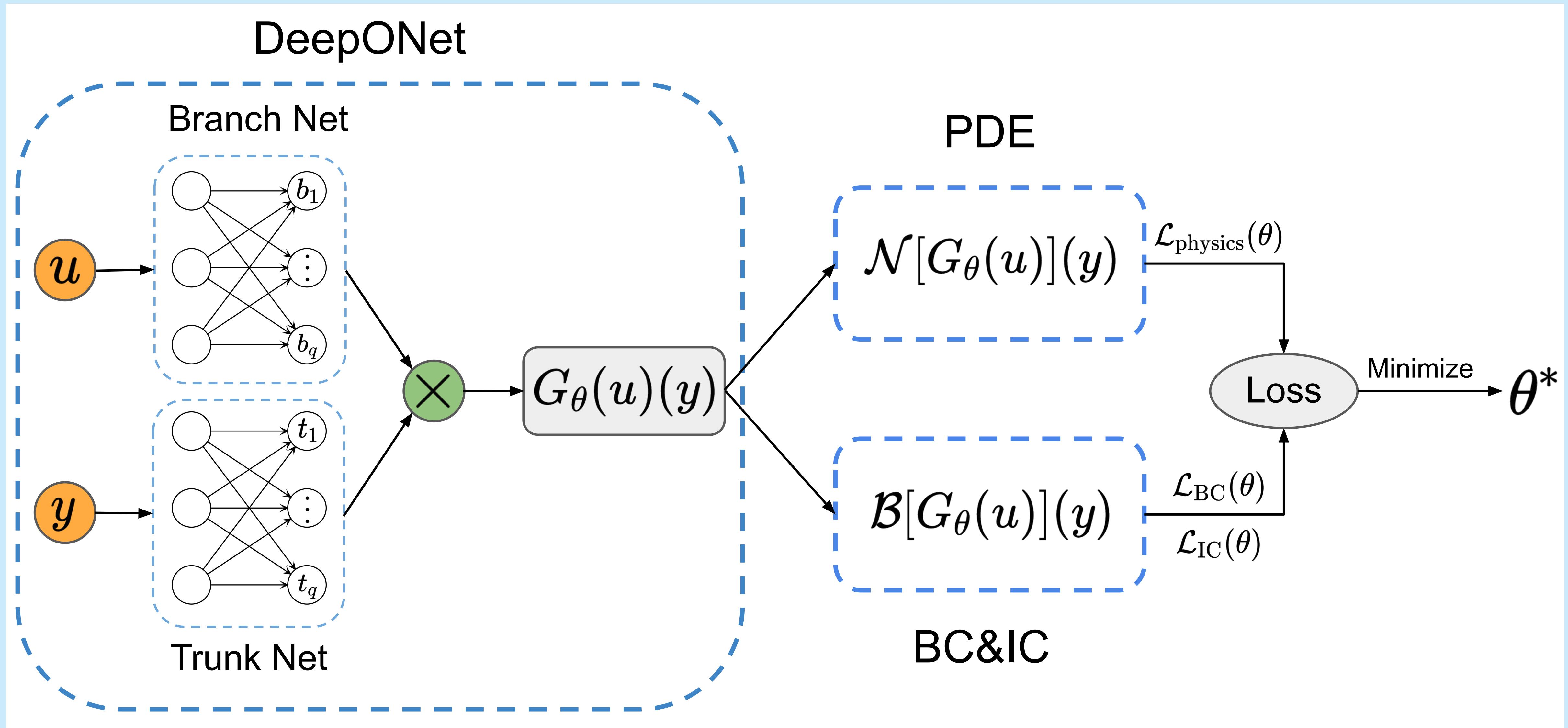
- Physics-informed neural networks for modeling astrophysical shocks (Moschou+2023)
- Probing the solar coronal magnetic field with physics-informed neural networks (Jarolim+2022)
- Physics-informed neural networks in the recreation of hydrodynamic simulations from dark matter (Dai+2023)
- Physics informed neural networks for simulating radiative transfer (Mishra+Molinaro 2021)
- Neural networks: solving the chemistry of the interstellar medium (Branca+Pallottini 2023)





PINOs

PHYSICS INFORMED DEEP OPERATOR



PHYSICS INFORMED DEEP OPERATOR

EXAMPLE: ANTI-DERIVATIVE OPERATOR

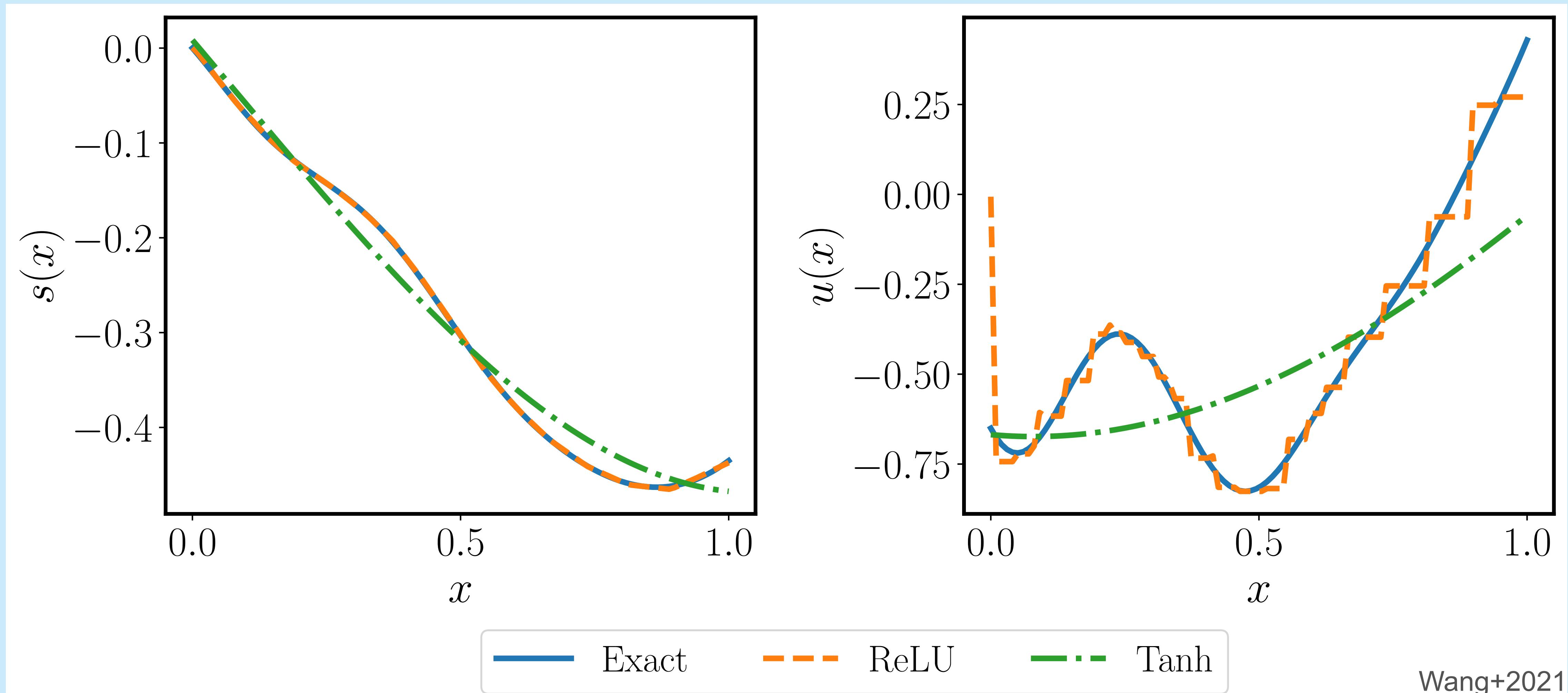
Consider this initial value problem with $s(0) = 0$

$$\frac{ds(x)}{dx} = u(x), \quad x \in [0, 1],$$

Goal is to learn the anti-derivative operator

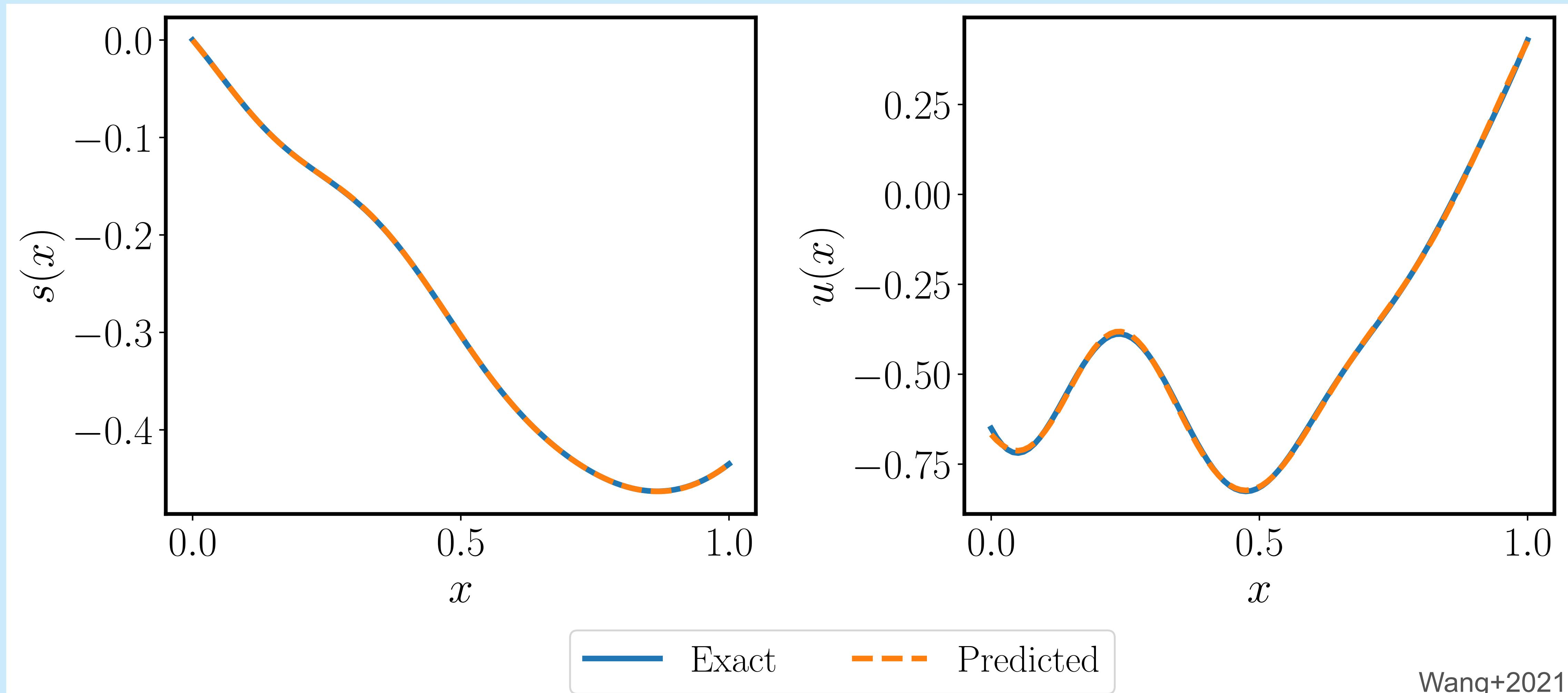
$$G : u(x) \longrightarrow s(x) = s(0) + \int_0^x u(t)dt, \quad x \in [0, 1].$$

EXAMPLE: ANTI-DERIVATIVE OPERATOR STANDARD DEEPONET



EXAMPLE: ANTI-DERIVATIVE OPERATOR

PINO



EXAMPLE: DIFFUSION REACTION SYSTEM

$$\frac{\partial s}{\partial t} = D \frac{\partial^2 s}{\partial x^2} + ks^2 + u(x), \quad (x, t) \in (0, 1] \times (0, 1],$$

With the following PDE residual

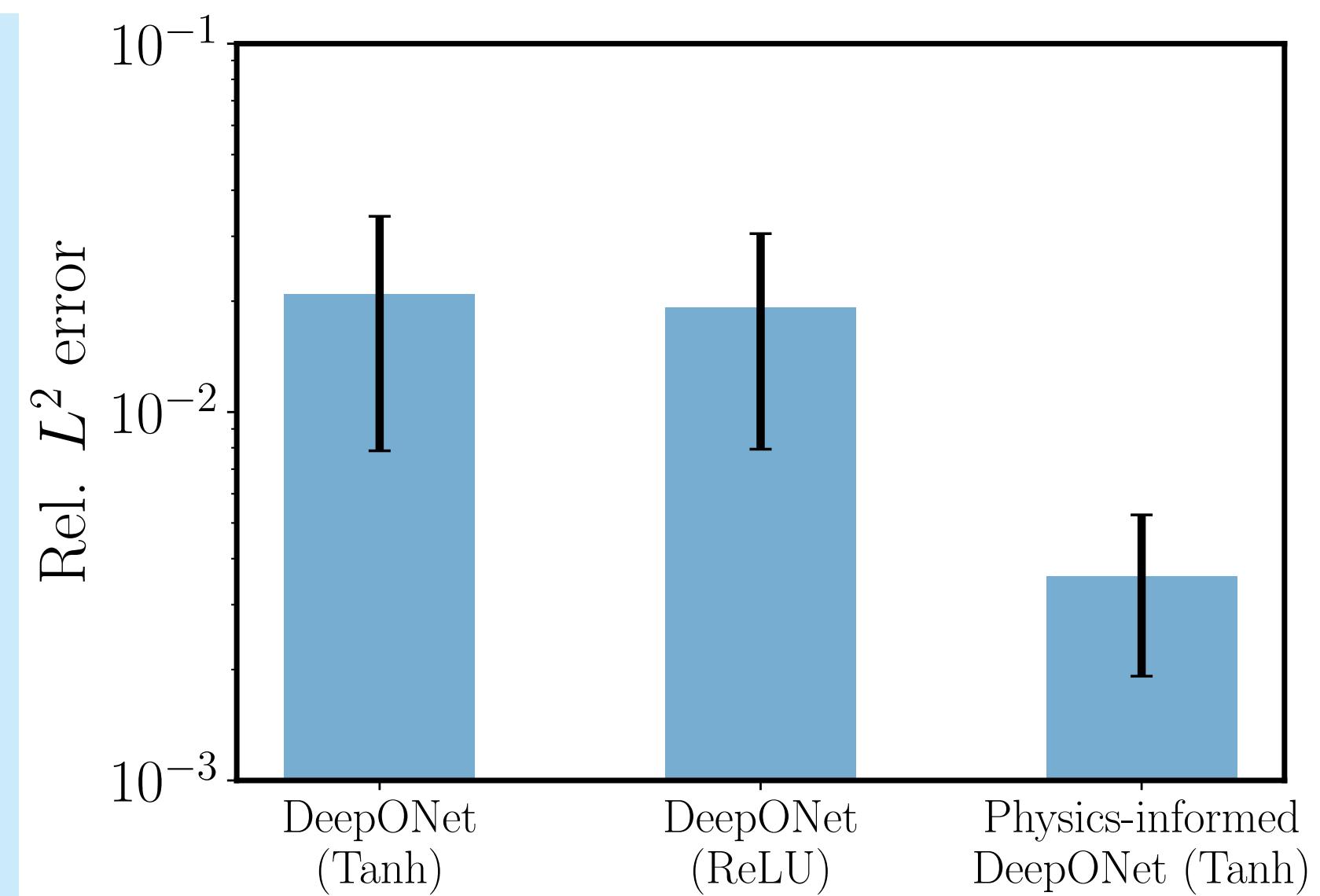
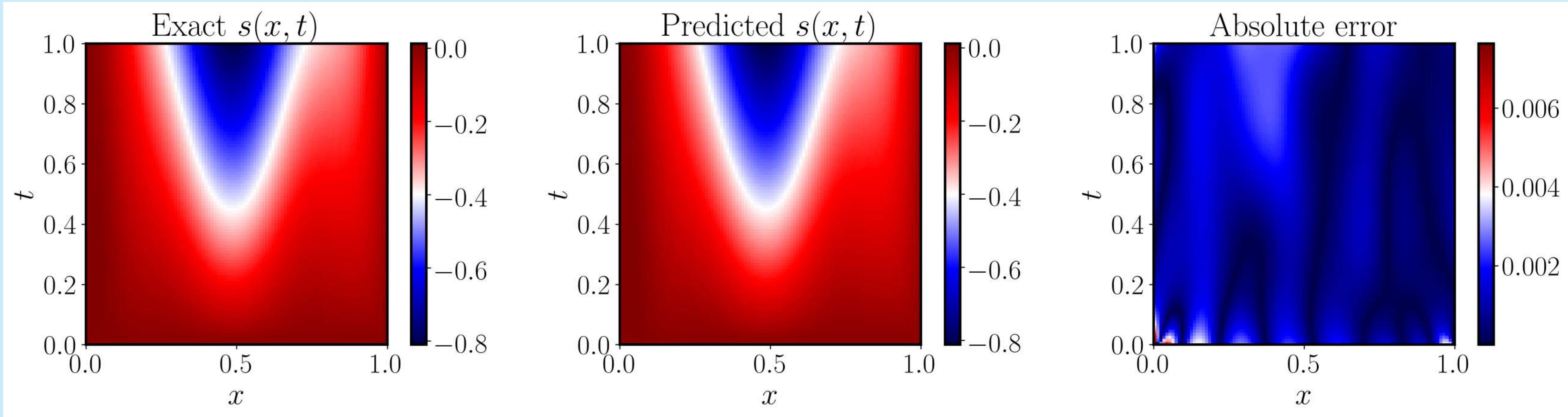
$$R_{\boldsymbol{\theta}}^{(i)}(x, t) = \frac{dG_{\boldsymbol{\theta}}(\mathbf{u}^{(i)})(x, t)}{dt} - D \frac{d^2G_{\boldsymbol{\theta}}(\mathbf{u}^{(i)})(x, t)}{dx^2} - k[G_{\boldsymbol{\theta}}(\mathbf{u}^{(i)})(x, t)]^2,$$

And total loss function

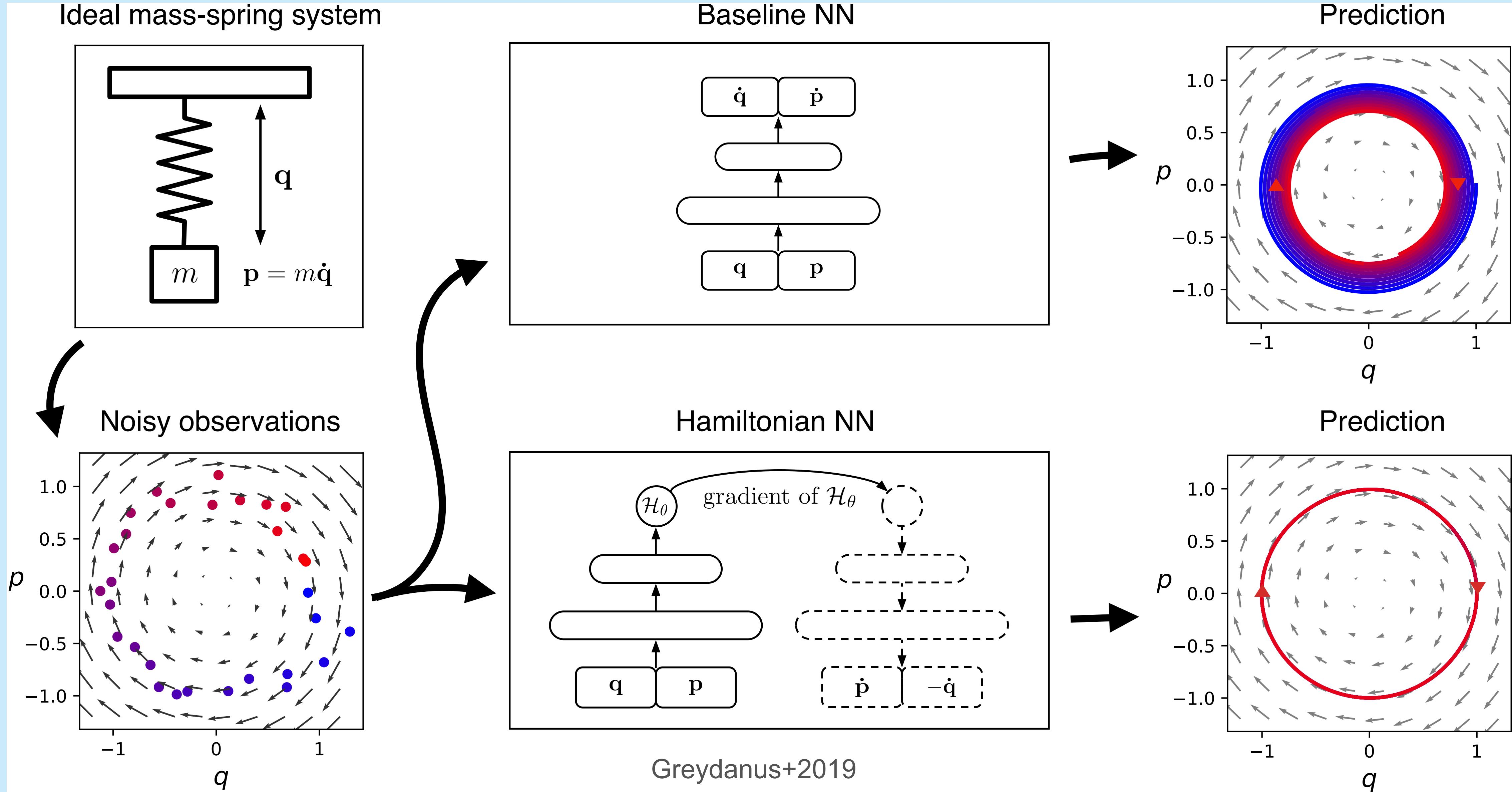
$$\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}_{\text{operator}}(\boldsymbol{\theta}) + \mathcal{L}_{\text{physics}}(\boldsymbol{\theta})$$

$$= \frac{1}{NP} \sum_{i=1}^N \sum_{j=1}^P \left| G_{\boldsymbol{\theta}}(\mathbf{u}^{(i)})(x_{u,j}^{(i)}, t_{u,j}^{(i)}) \right|^2 + \frac{1}{NQ} \sum_{i=1}^N \sum_{j=1}^Q \left| R_{\boldsymbol{\theta}}^{(i)}(x_{r,j}^{(i)}, t_{r,j}^{(i)}) - u^{(i)}(x_{r,j}^{(i)}) \right|^2.$$

EXAMPLE: DIFFUSION REACTION SYSTEM



HAMILTONIAN NEURAL NETWORKS



SUMMARY & CONCLUSION

Main take away:

scientific motivated inductive bias helps to be more robust, more data efficient
and better interpretable

My personal message:

Write better code!
Share more data!
Build more open-source software!

This will accelerate research cycles and lets you engage with peers early on!

WHERE TO FIND THE LECTURE MATERIAL



<https://github.com/TobiBu/graddays>

THIS IS YOUR MACHINE LEARNING SYSTEM?

YUP! YOU POUR THE DATA INTO THIS BIG
PILE OF LINEAR ALGEBRA, THEN COLLECT
THE ANSWERS ON THE OTHER SIDE.

WHAT IF THE ANSWERS ARE WRONG?

JUST STIR THE PILE UNTIL
THEY START LOOKING RIGHT.



Differentiable Simulators

DIFFERENTIABLE SIMULATORS