Embodied Vision

Physical Modeling: Learning-based Dynamics

Tsung-Wei Ke

Spring 2025

Physical Modeling We Have Covered

- We have covered mass-spring system, position-based dynamics, incremental potential contact and material point methods

- Pros:
  - Physically accurate if parameters are well tuned

- Cons:
  - Need to manually select a lot of parameters. For instance, connection pattern and stiffness for mass-spring system; material type, boundary conditions, E, nu for MPM…
  - These frameworks lack of generalization, each of which is only good for certain scenarios

Learn Physically Accurate Dynamics without the Need for Parameter Tuning but Generalize Well

- Ideas:
  - Pure data-driven approaches: Fit models on large-scale data
  - Hybrid approaches: Data-driven supervision and physic-based regularization

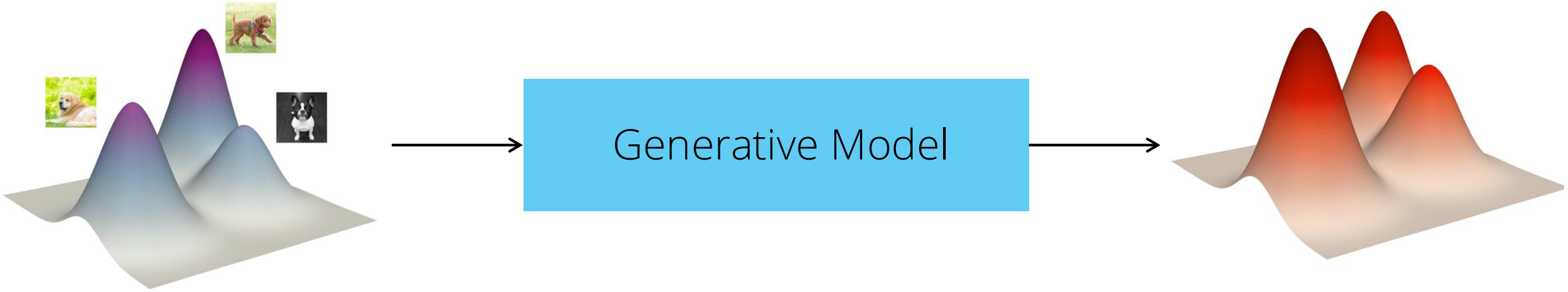
Content

- Pure Data-driven Approaches
- Hybrid Approaches
  - Physics-Informed Diffusion Model
  - Physics-Informed Neural Network
  - Neural ODE
  - Lagrangian Network / Hamiltonian Network
  - Neural Operator
  - Fourier Neural Operator

Content

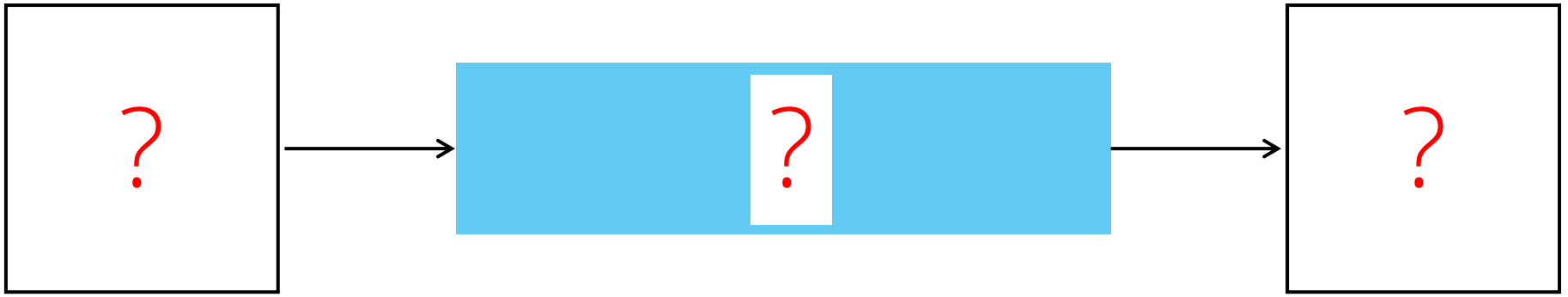
- Pure Data-driven Approaches
- Hybrid Approaches
  - ➢Physics-Informed Diffusion Model
  - ➢Physics-Informed Neural Network
  - ➢Neural ODE
  - ➢Lagrangian Network / Hamiltonian Network
  - ➢Neural Operator
  - ➢Fourier Neural Operator

Pure Data-Driven Approaches



- All you need is to collect a large-scale dataset, and fit generative models on the dataset

Generative Modeling for Simulation



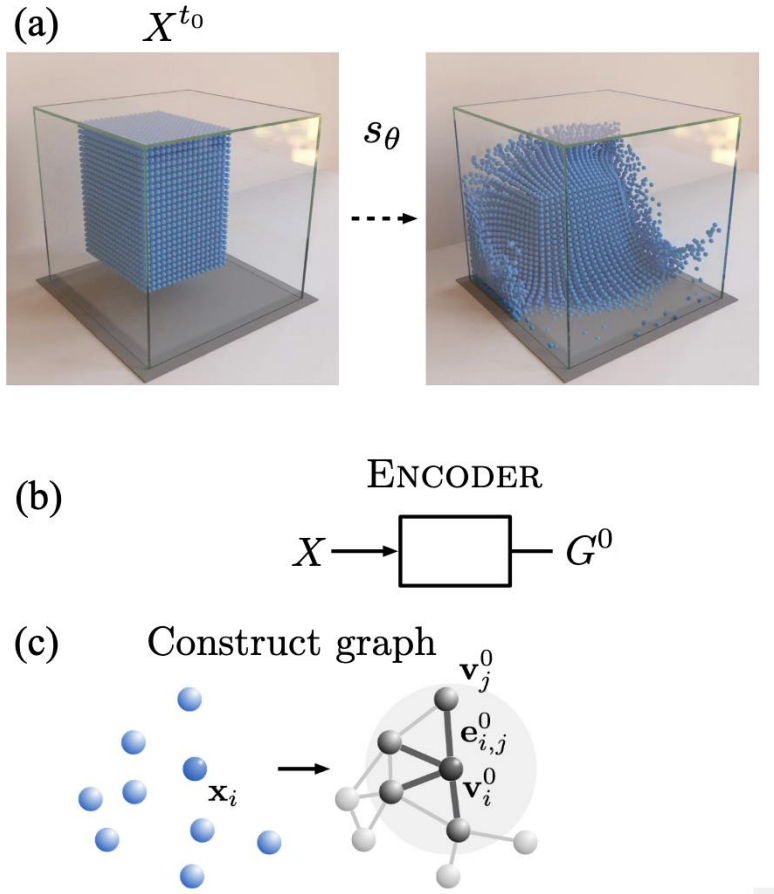
- What are the input representations?
- What are the model architectures?
- What are the output representations?
- How to obtain ground truths?

Learning to Simulate Complex Physics with Graph Networks

Alvaro Sanchez-Gonzalez [*1] **Jonathan Godwin** [*1] **Tobias Pfaff** [*1] **Rex Ying** [*1 2] **Jure Leskovec** [2]
Peter W. Battaglia [1]

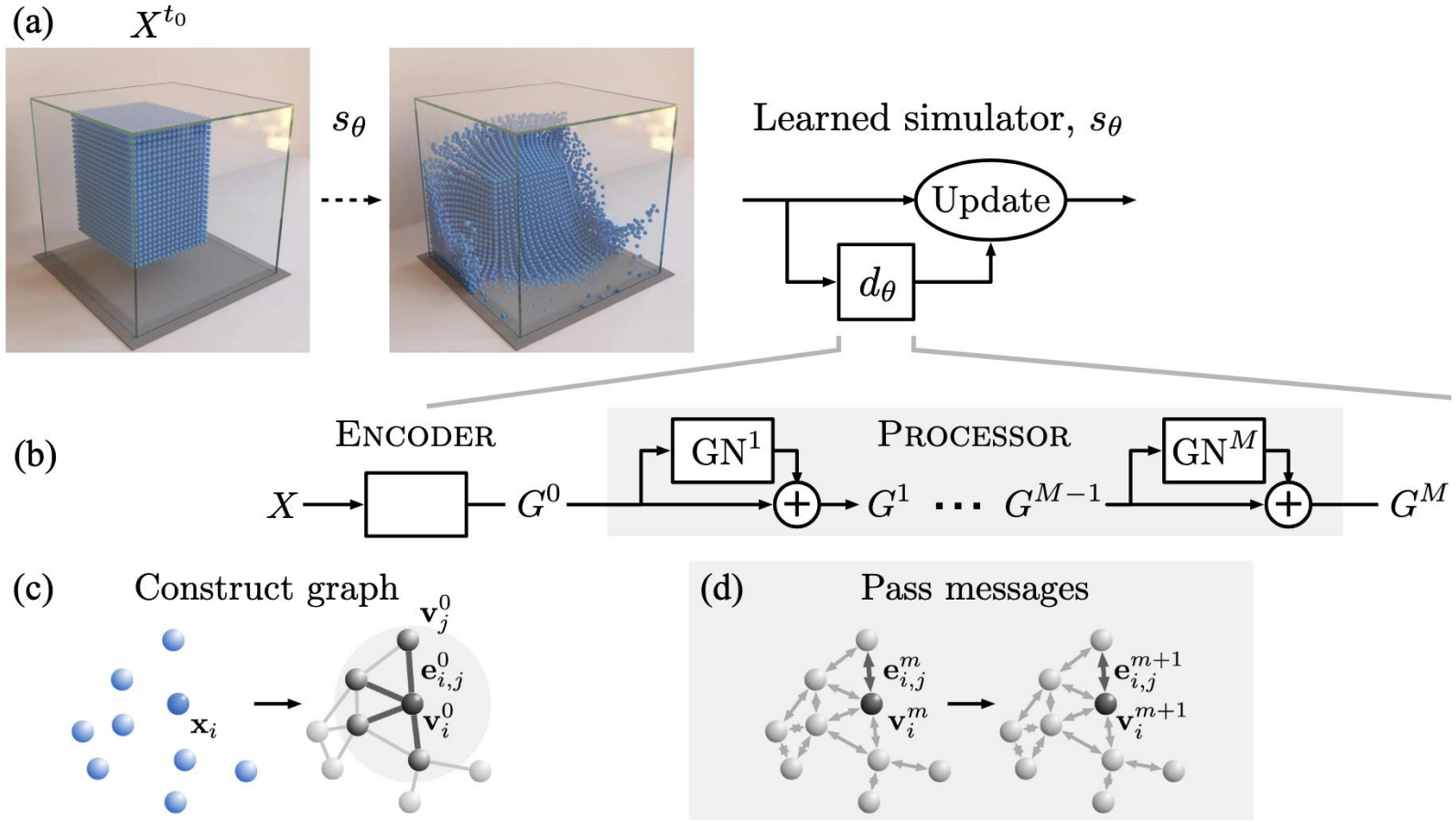
- What are the input representations?
  - Particles
- What are the model architectures?
  - Graph neural networks
- What are the output representations?
  - Particle position / velocity …
- How to obtain ground truths?
  - Generate with PBD, MPM, …

Step 1: Encode a Graph of Particles



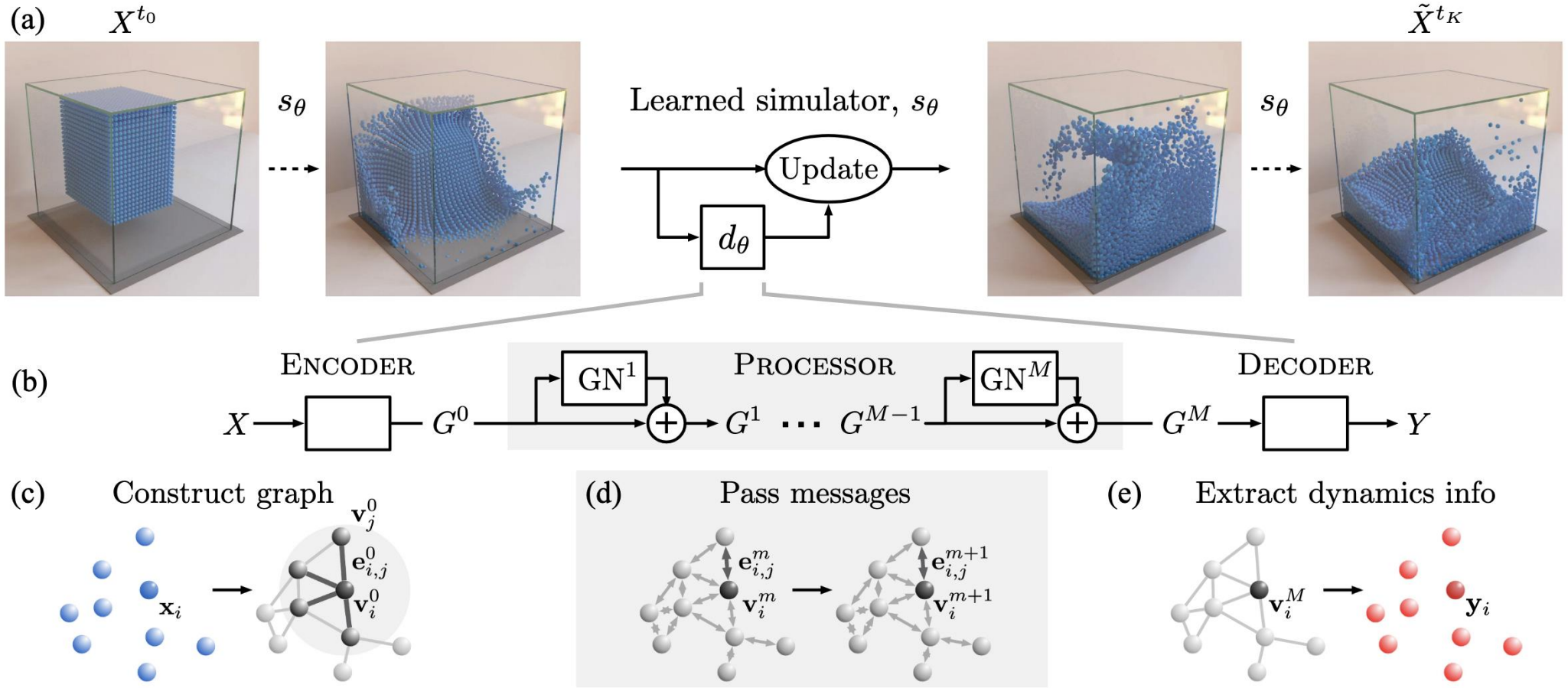
Learning to Simulate Complex Physics with Graph Networks. Sanchez-Gonzalez et al.

Step 2: Contextualize the Graph with Graph Neural Network



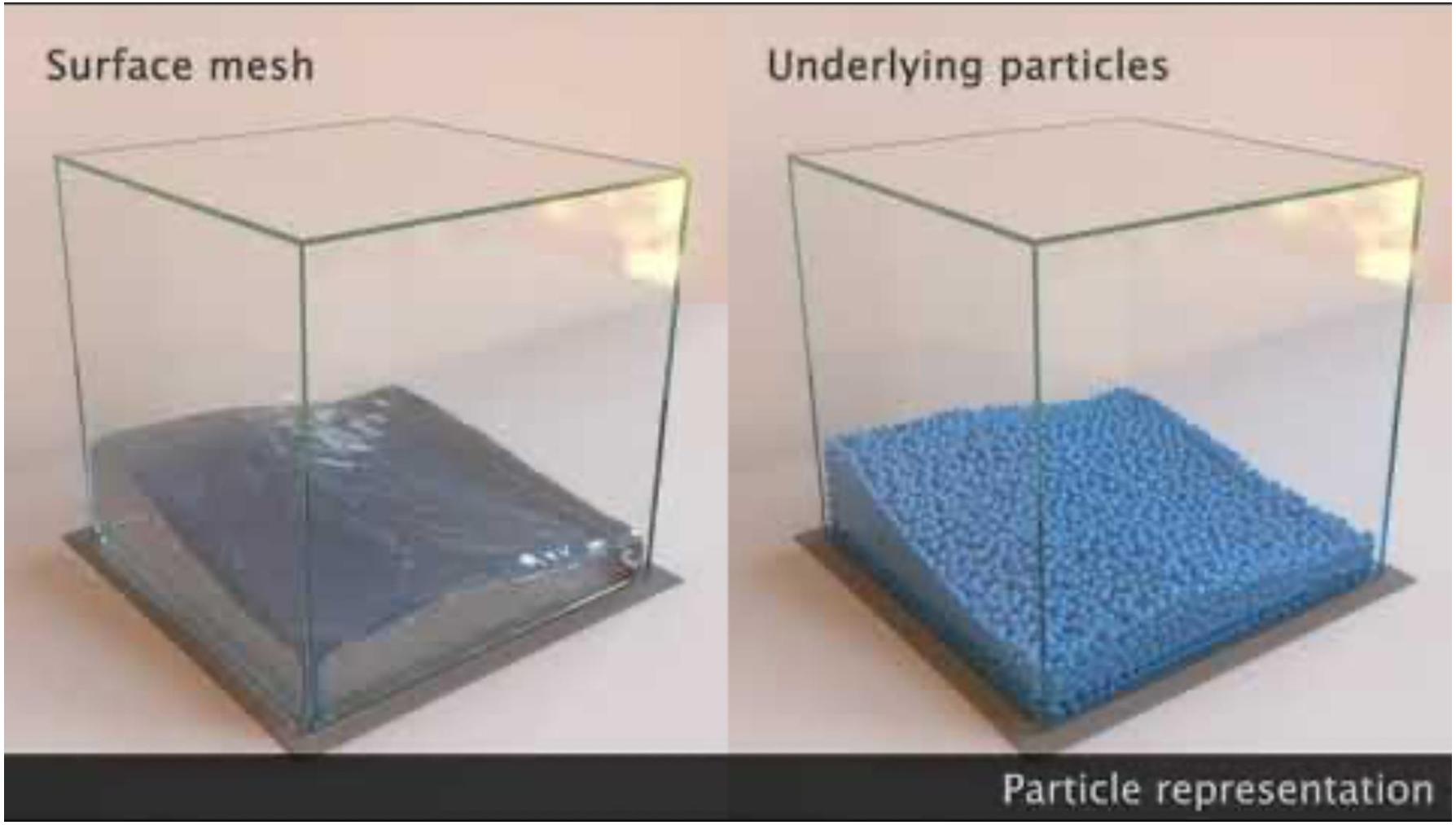
Learning to Simulate Complex Physics with Graph Networks. Sanchez-Gonzalez et al.

Step 3: Decode Particle Motion



Learning to Simulate Complex Physics with Graph Networks. Sanchez-Gonzalez et al.

Results



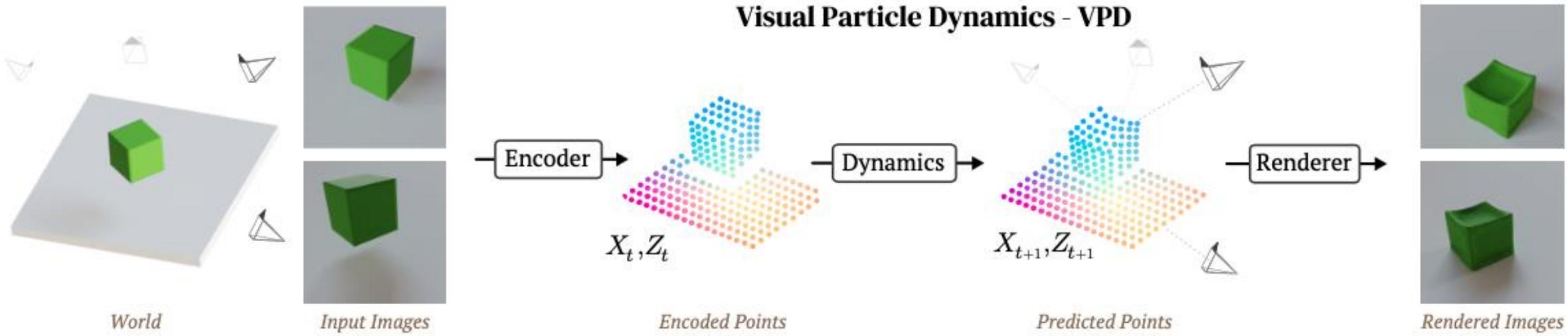
Learning to Simulate Complex Physics with Graph Networks. Sanchez-Gonzalez et al.

Learning 3D Particle-based Simulators from RGB-D Videos

William F. Whitney,* Tatiana Lopez-Guevara,* Tobias Pfaff, Yulia Rubanova, Thomas Kipf, Kimberly Stachenfeld, Kelsey R. Allen
Google DeepMind

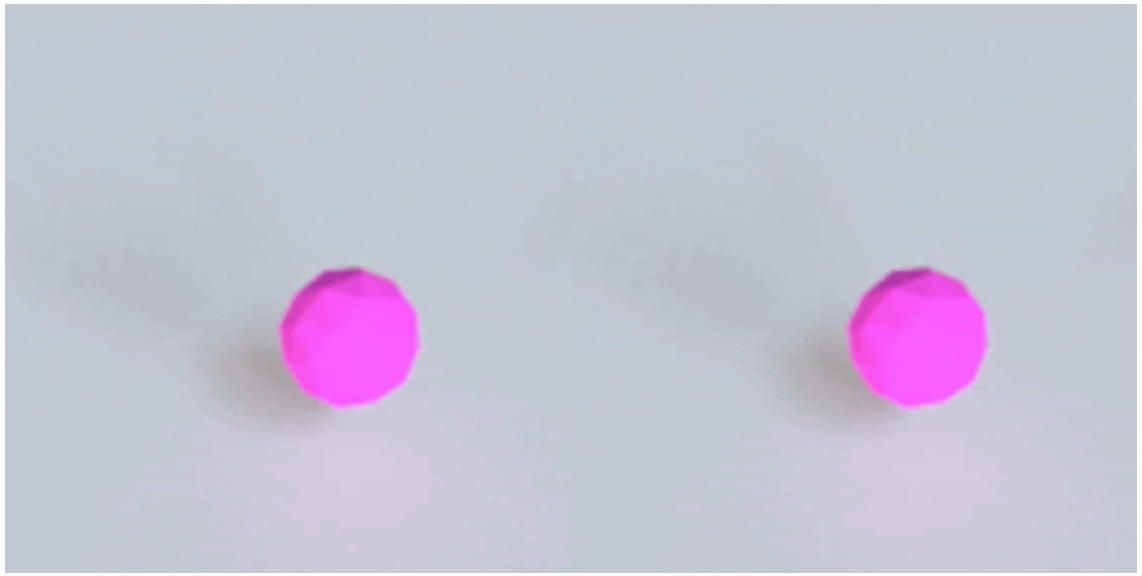
- What are the input representations?
  - ➢ Pixel particles
- What are the model architectures?
  - ➢ Graph neural networks
- What are the output representations?
  - ➢ Particle position / velocity …
- How to obtain ground truths?
  - ➢ RGB-D videos

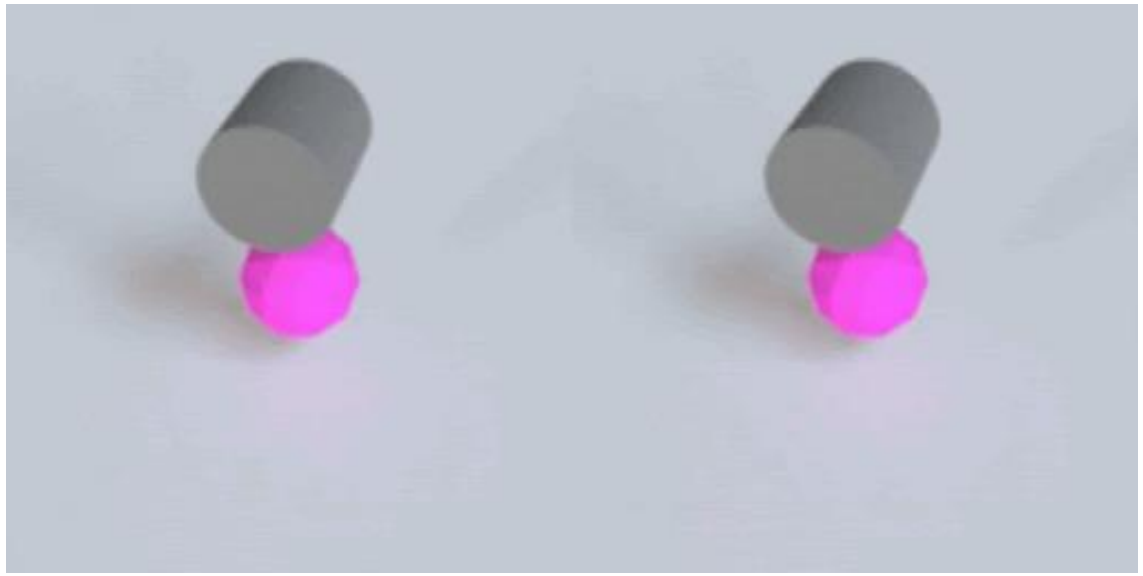
Similar Frameworks Supervised with Rendering Loss



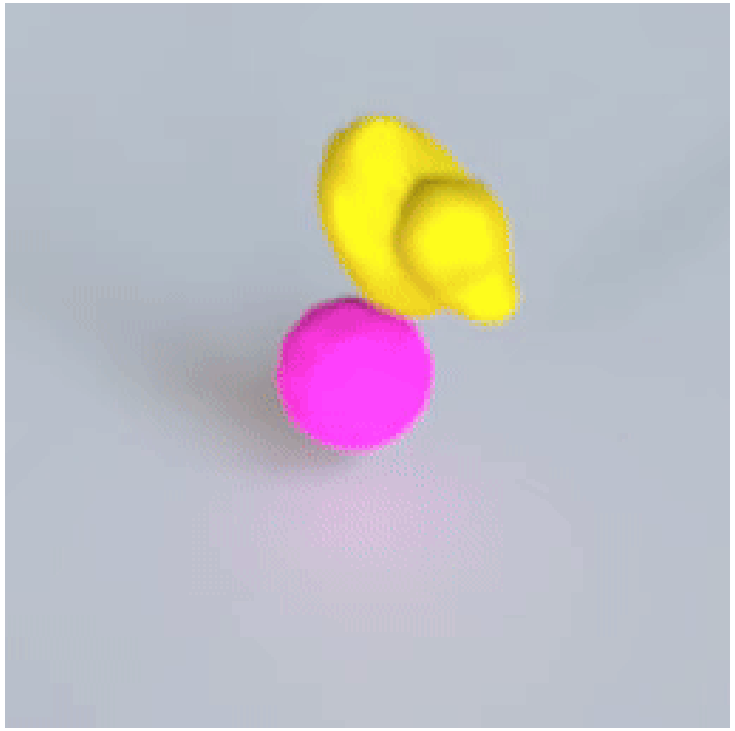
Learning 3D Particle-based Simulators from RGB-D Videos. Whitney et al.

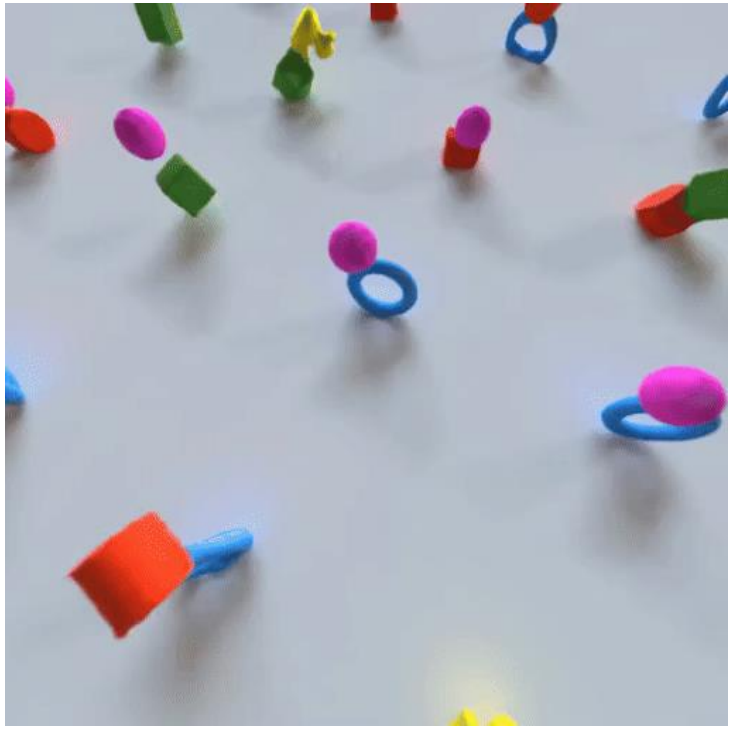
Results





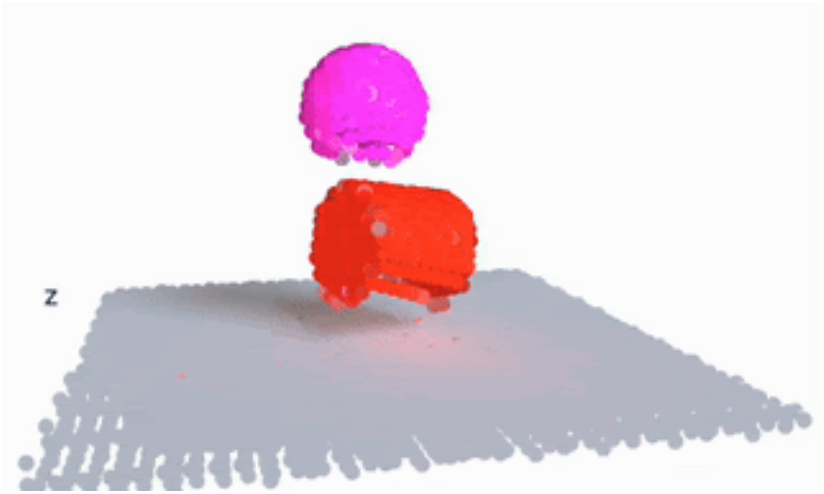
Results



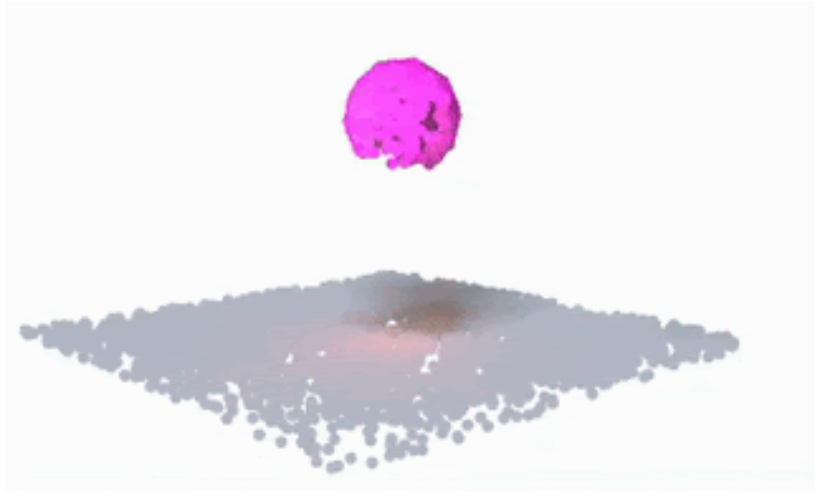


Results: Scene Editing

Origin scene



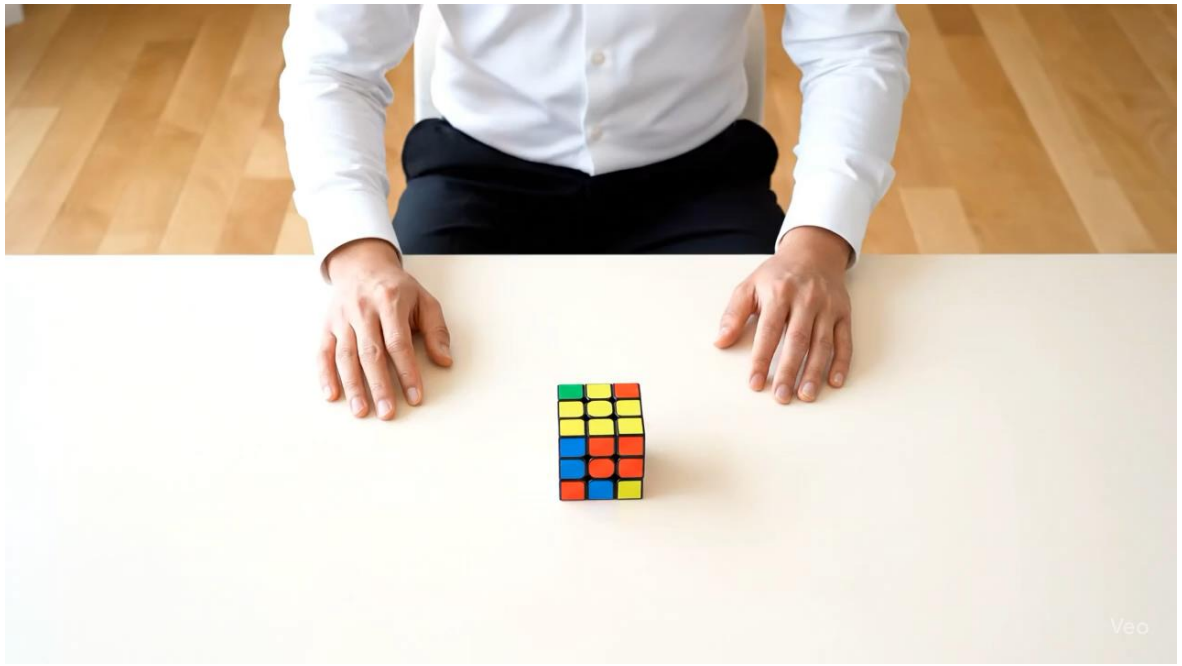
Delete cylinder



Delete floor

However, Pure Data-Driven Approaches May Not Work We Learn the Same Lesson from Video Generative Models

Video generated by Veo3

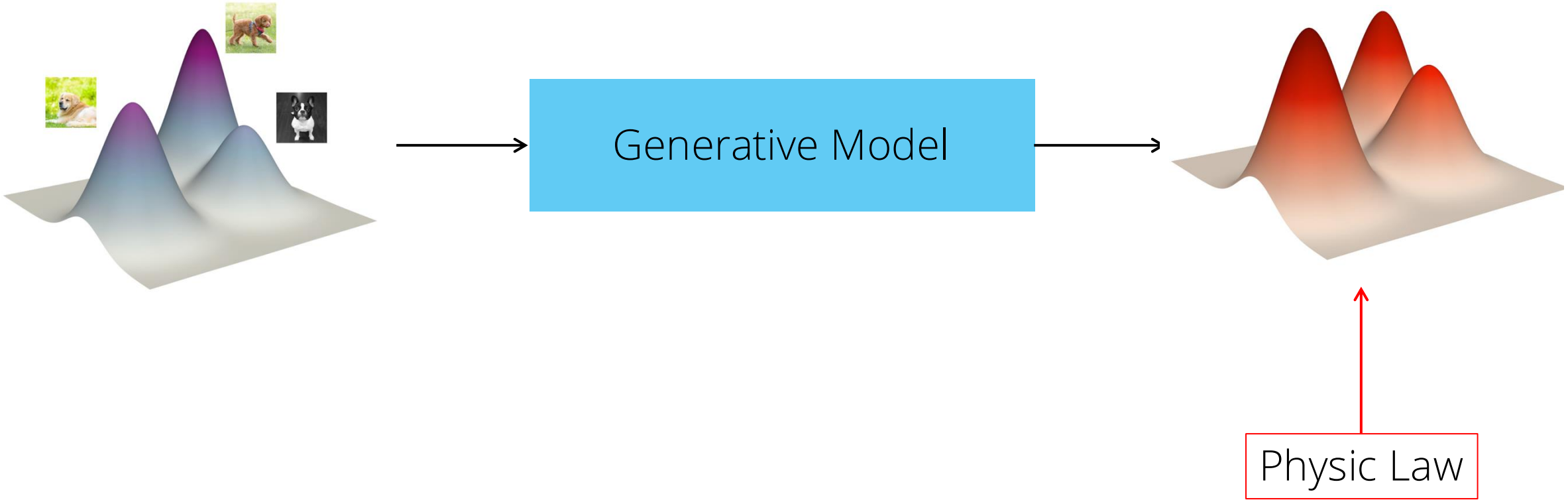


Video generated by Veo3

Content

- Pure Data-driven Approaches
- Hybrid Approaches
  - ➢Physics-Informed Diffusion Model
  - ➢Neural ODE
  - ➢Lagrangian Network / Hamiltonian Network
  - ➢Physics-Informed Neural Network
  - ➢Neural Operator
  - ➢Fourier Neural Operator

Physic-based Regularization



Content

- Pure Data-driven Approaches
- Hybrid Approaches
  - ➢Physics-Informed Diffusion Model
  - ➢Physics-Informed Neural Network
  - ➢Neural ODE
  - ➢Lagrangian Network / Hamiltonian Network
  - ➢Neural Operator
  - ➢Fourier Neural Operator

PHYSICS-INFORMED DIFFUSION MODELS

Jan-Hendrik Bastek
Dept. of Mechanical and Process Eng.
ETH Zurich
Zurich, Switzerland
jbastek@ethz.ch

WaiChing Sun
Dept. of Civil Eng. and Eng. Mechanics
Columbia University
New York, NY, USA
wsun@columbia.edu

Dennis M. Kochmann
Dept. of Mechanical and Process Eng.
ETH Zurich
Zurich, Switzerland
dmk@ethz.ch

- Enforce samples generated from the learned distribution conform to the known physical constraints

We Can Formulate Physic Laws as a set of PDEs

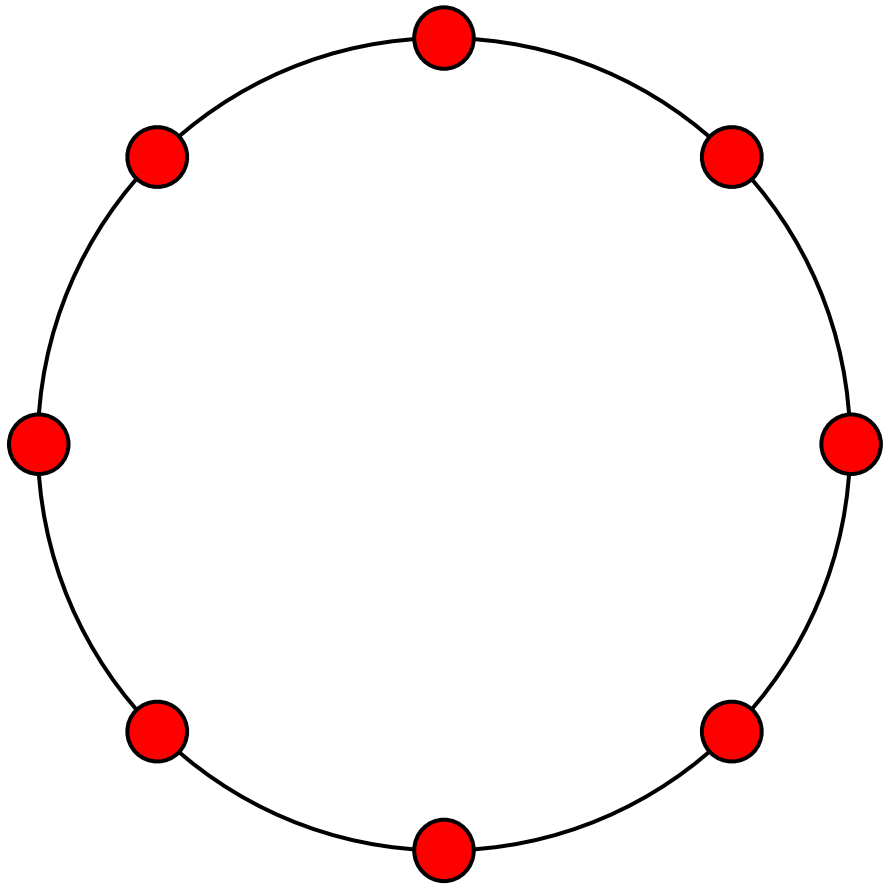
- Constraints:

$$\boldsymbol{\mathcal{F}}[\boldsymbol{u}(\boldsymbol{\xi})] = \mathbf{0}, \quad \boldsymbol{\xi} = (\xi_1, \xi_2, \cdots, \xi_d)^\intercal \in \Omega, \quad \boldsymbol{u} = (u_1(\boldsymbol{\xi}), u_2(\boldsymbol{\xi}), \cdots, u_c(\boldsymbol{\xi}))^\intercal \in \mathbb{R}^c,$$

where $\boldsymbol{\mathcal{F}}$ denote differential operators, $\boldsymbol{\xi}$ denote elements in domain $\boldsymbol{\Omega}$ and $\boldsymbol{u}$ denote the solution field satisfies the set of PDES

- Boundary conditions: $\boldsymbol{\mathcal{B}}[\boldsymbol{u}(\boldsymbol{\xi})] = \mathbf{0}, \quad \boldsymbol{\xi} \in \partial\Omega$
- We can compose both factors into a residual operator: $\boldsymbol{\mathcal{R}}(\boldsymbol{x_0}) := \begin{bmatrix} \boldsymbol{\mathcal{F}}[\boldsymbol{x_0}] \\ \boldsymbol{\mathcal{B}}[\boldsymbol{x_0}] \end{bmatrix}$
- To enforce physical constraints, we minimize the residual $\mathcal{R}(\boldsymbol{x_0})$ of sample $\boldsymbol{x_0}$

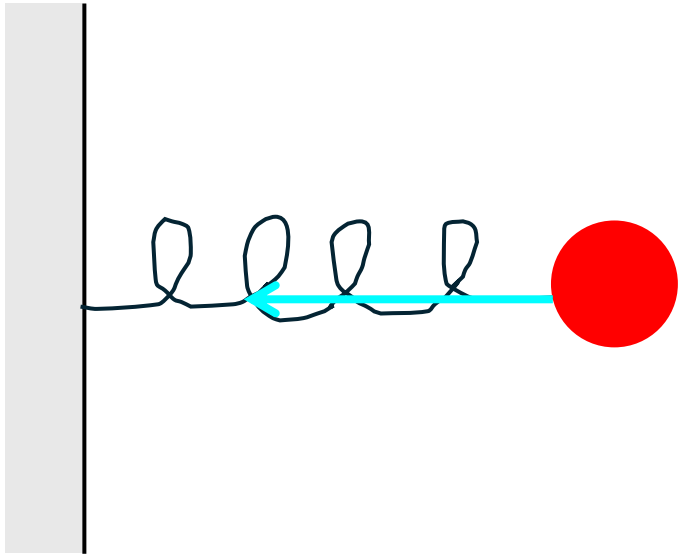
Example: Position Points on a Circle



- Boundary conditions:

$$(x - x_c)^2 + (y - y_c)^2 = r^2$$

Example: Spring Motion



- Physic law:

$$m\ddot{x} = -kx$$

- Boundary conditions:

$$(\mathrm{x} - \mathrm{x_c})^2 + (\mathrm{y} - \mathrm{y_c})^2 = \mathrm{r}^2$$

Integrate into the Diffusion Model

- To maintain the probabilistic perspective of generative models, we can introduce the residuals as *virtual observables*

$$q_{\mathcal{R}}(\hat{\boldsymbol{r}}|\boldsymbol{x}_0) = \mathcal{N}(\hat{\boldsymbol{r}}; \boldsymbol{\mathcal{R}}(\boldsymbol{x}_0), \sigma^2 \boldsymbol{I})$$

clean data sample

Integrate into the Diffusion Model

- To maintain the probabilistic perspective of generative models, we can introduce the residuals as *virtual observables*

$$q_{\mathcal{R}}(\hat{\boldsymbol{r}}|\boldsymbol{x}_0) = \mathcal{N}(\hat{\boldsymbol{r}}; \boldsymbol{\mathcal{R}}(\boldsymbol{x}_0), \sigma^2 \boldsymbol{I})$$

- The likelihood of *virtual observables*:

$$p_\theta(\hat{\boldsymbol{r}}) = \int p_\theta(\hat{\boldsymbol{r}}, \boldsymbol{x}_0)\, \mathrm{d}\boldsymbol{x}_0 = \int q_{\mathcal{R}}(\hat{\boldsymbol{r}}|\boldsymbol{x}_0) p_\theta(\boldsymbol{x}_0)\, \mathrm{d}\boldsymbol{x}_0 = \mathbb{E}_{\boldsymbol{x}_0 \sim p_\theta(\boldsymbol{x}_0)} q_{\mathcal{R}}(\hat{\boldsymbol{r}}|\boldsymbol{x}_0).$$

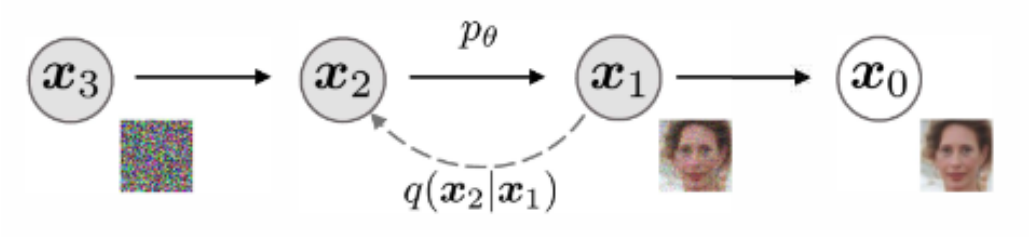
- Generating a physically correct sample is to maximize the likelihood of *virtual observables*:

$$\arg\max_\theta \mathbb{E}_{\hat{\boldsymbol{r}}} \left[\log p_\theta(\hat{\boldsymbol{r}})\right] = \arg\max_\theta \mathbb{E}_{\boldsymbol{x}_0 \sim p_\theta(\boldsymbol{x}_0)} \left[\log q_{\mathcal{R}}(\hat{\boldsymbol{r}} = \boldsymbol{0}|\boldsymbol{x}_0)\right].$$

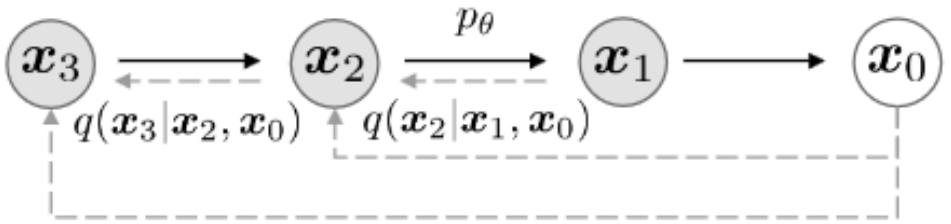
Sample from the learned distribution

Step 1: Estimate $\mathbf{x}_0^*$

- Idea 1: accelerate sampling with DDIM



DDPM (markovian)



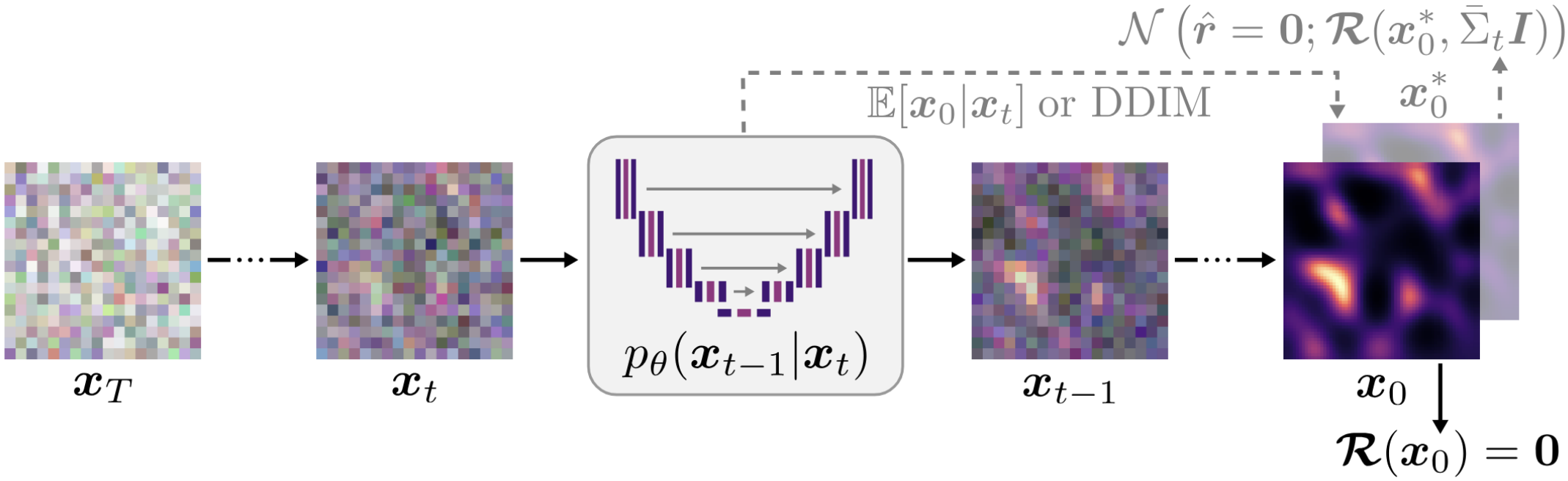
DDIM (non-markovian)

- Idea 2: estimate directly from $\mathbf{x_t}$

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I})$$

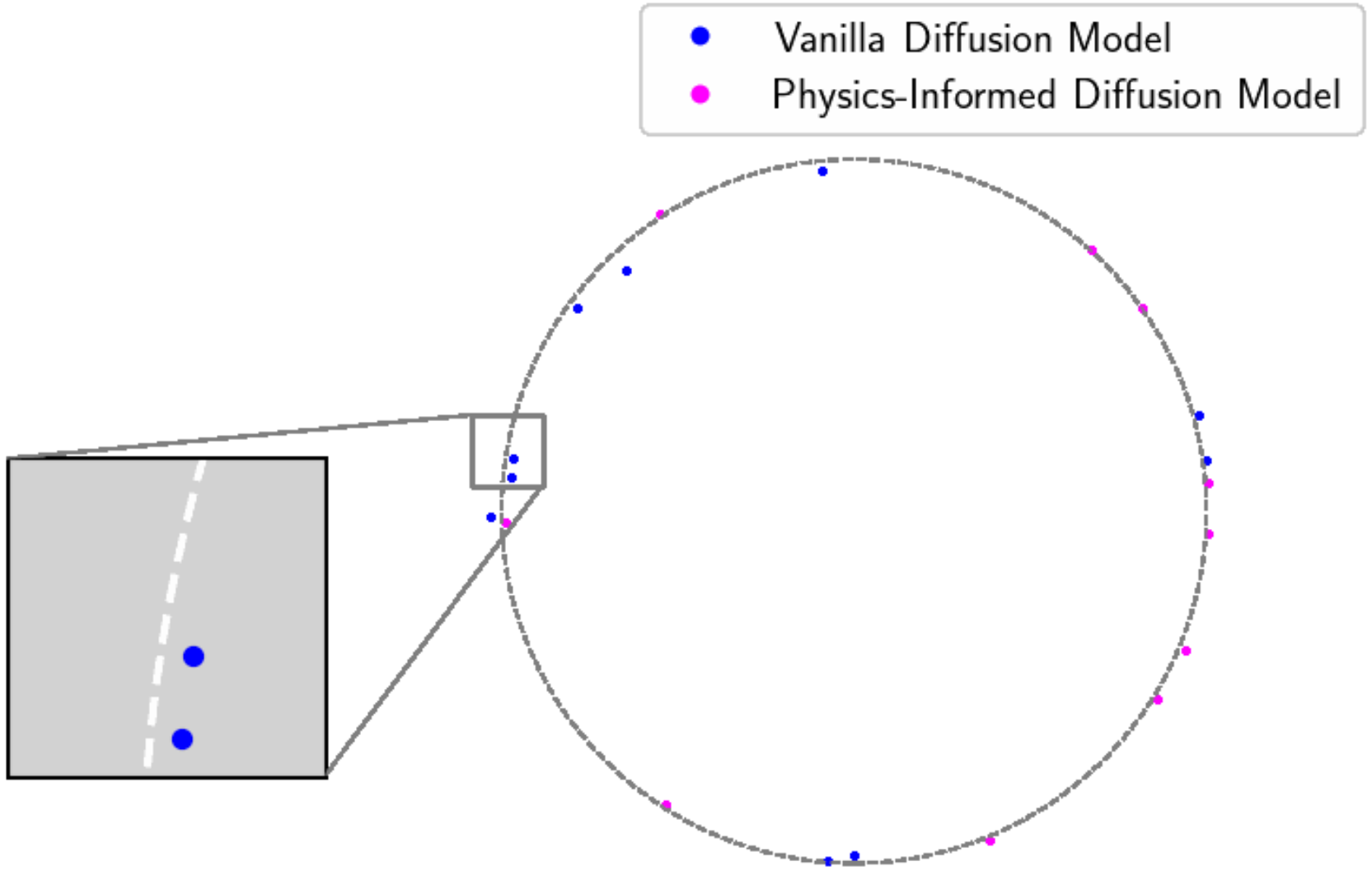
$$\mathbf{x}_0 \approx \hat{\mathbf{x}}_0 = \left(\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t)\right)/\sqrt{\bar{\alpha}_t}$$

Step 2: Minimize Residual Loss

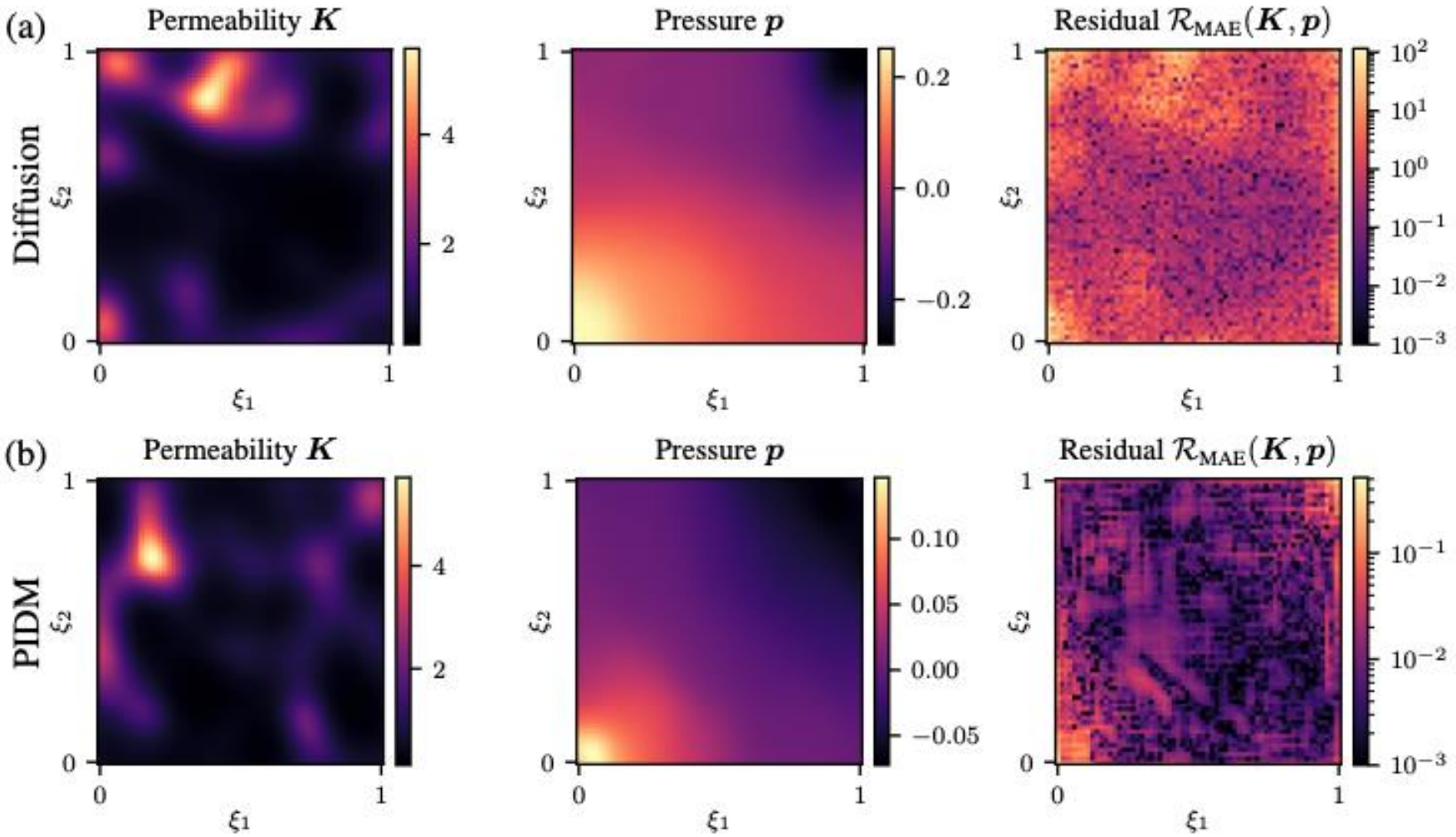


$$L_{\text{PIDM}}(\theta)=\mathbb{E}_{t\sim[1,T],\boldsymbol{x}_{0:T}\sim q(\boldsymbol{x}_{0:T})}\left[\lambda_t\|\boldsymbol{x}_0-\hat{\boldsymbol{x}}_0(\boldsymbol{x}_t,t)\|^2+\frac{1}{2\bar{\Sigma}_t}\|\mathcal{R}(\boldsymbol{x}_0^*(\boldsymbol{x}_t,t))\|^2\right]$$

Results


Vanilla Diffusion Model
Physics-Informed Diffusion Model

Results


(a)
Permeability $\boldsymbol{K}$
Pressure $\boldsymbol{p}$
Residual $\mathcal{R}_{\text{MAE}}(\boldsymbol{K}, \boldsymbol{p})$
Diffusion
(b)
PIDM
ξ_1
ξ_2

Content

- Pure Data-driven Approaches
- Hybrid Approaches
  - ➢Physics-Informed Diffusion Model
  - ➢Physics-Informed Neural Network
  - ➢Neural ODE
  - ➢Lagrangian Network / Hamiltonian Network
  - ➢Neural Operator
  - ➢Fourier Neural Operator

A More General Formulation of Physic-Informed Neural Network (PINN)

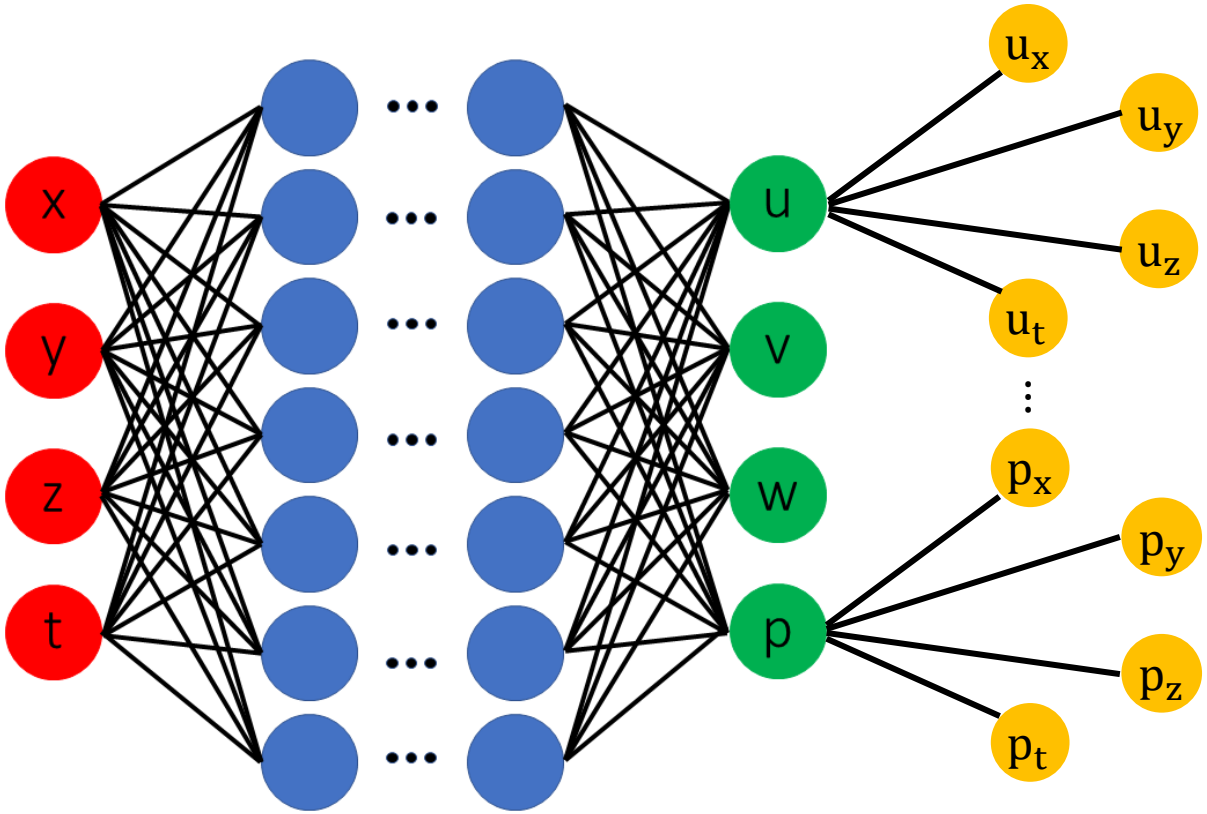
- Core idea of physics-informed diffusion model is to impose constraints based on physic laws on the model output
- A general form of PDEs:

$$u_t + \mathcal{N}[u] = 0,\ x \in \Omega,\ t \in [0, T],$$

where u is the solution, $\mathcal{N}[\cdot]$ denotes a non-linear differential operator.

- Idea: learn a neural network predicting solutions of PDEs to bypass time-consuming numerical simulation

PINN



Naviers-Stokes loss

$$\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}+\frac{\partial w}{\partial z}-0$$

$$\frac{\partial u}{\partial t}+u\frac{\partial u}{\partial x}+v\frac{\partial u}{\partial y}+w\frac{\partial u}{\partial z}+\frac{\partial P}{\partial x}-\frac{1}{Re}(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}+\frac{\partial^2 u}{\partial z^2})=0$$

$$\frac{\partial v}{\partial t}+u\frac{\partial v}{\partial x}+v\frac{\partial v}{\partial y}+w\frac{\partial v}{\partial z}+\frac{\partial P}{\partial y}-\frac{1}{Re}(\frac{\partial^2 v}{\partial x^2}+\frac{\partial^2 v}{\partial y^2}+\frac{\partial^2 v}{\partial z^2})=0$$

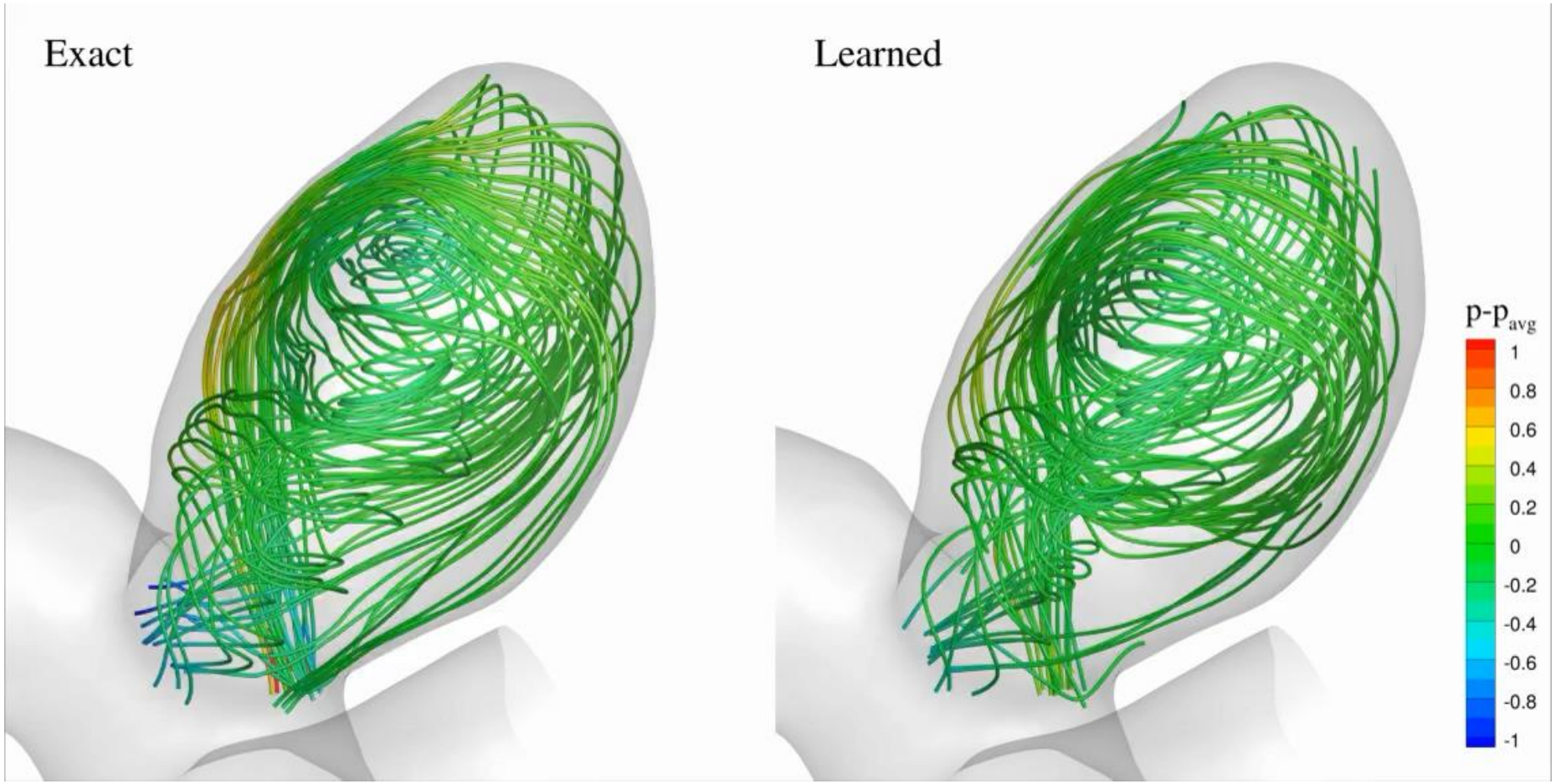
$$\frac{\partial w}{\partial t}+u\frac{\partial w}{\partial x}+v\frac{\partial w}{\partial y}+w\frac{\partial w}{\partial z}+\frac{\partial P}{\partial z}-\frac{1}{Re}(\frac{\partial^2 w}{\partial x^2}+\frac{\partial^2 w}{\partial y^2}+\frac{\partial^2 w}{\partial z^2})=0$$

Experimental data loss

$$\|\vec{\mathbf{V}}-\vec{\mathbf{V}}_{exp}\|^2=0$$

Image source: wikipedia

Results



Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. Raissi et al.

Content

- Pure Data-driven Approaches
- Hybrid Approaches
  - ➢Physics-Informed Diffusion Model
  - ➢Physics-Informed Neural Network
  - ➢Neural ODE
  - ➢Lagrangian Network / Hamiltonian Network
  - ➢Neural Operator
  - ➢Fourier Neural Operator

ODE Forms the Foundation of Simulation

- For example, positions are integrals of velocity

$$\frac{d\mathrm{x}(\mathrm{t})}{d\mathrm{t}} = \mathrm{v}(\mathrm{t}) \Leftrightarrow \mathrm{x}(\mathrm{t}) = \mathrm{x}(0) + \int_0^{\mathrm{t}} \mathrm{v}(\mathrm{t})\mathrm{dt}$$

- Idea: specify an ordinary differential equation (ODE) by a neural network

$$\frac{d\mathrm{z}(\mathrm{t})}{d\mathrm{t}} = \mathrm{f}(\mathrm{z}(\mathrm{t}), \mathrm{t}, \theta) \Leftrightarrow \mathrm{z}(\mathrm{t}_1) = \mathrm{z}(\mathrm{t}_0) + \int_{\mathrm{t}_0}^{\mathrm{t}_1} \mathrm{f}(\mathrm{z}(\mathrm{t}), \mathrm{t}, \theta)\mathrm{dt}$$

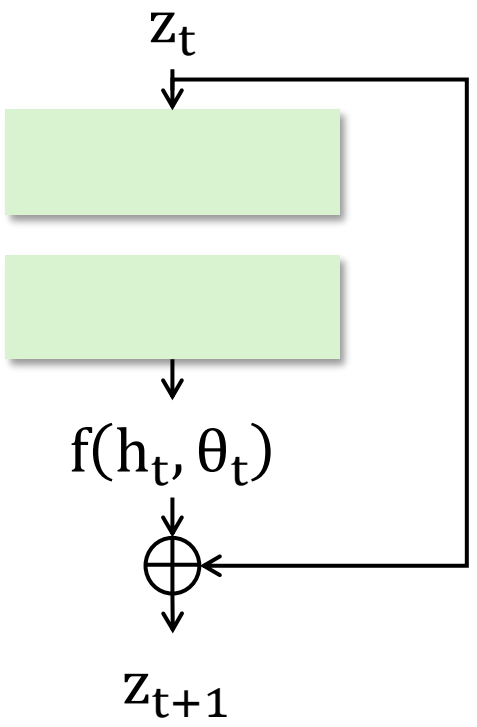
- How to learn the neural network?

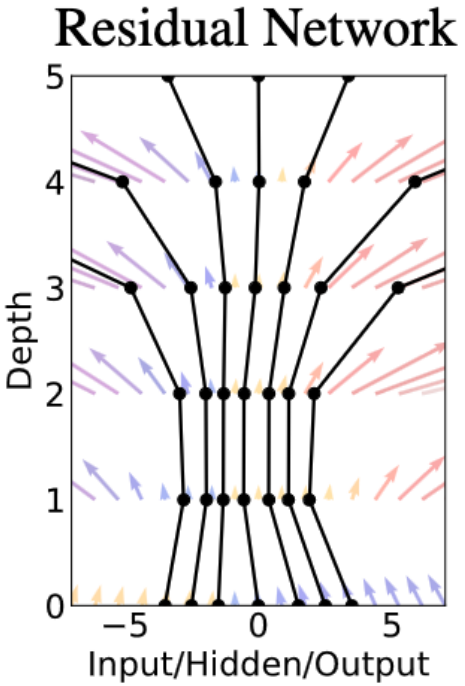
Parameterize Discretized ODE as Residual Network

- Parameterize discretized ODE:

$$z_{t+1} = z_t + f(h_t, \theta_t)$$

- Learning objective: reconstruct data at every time step t: $L = \sum_t \|z_t - \hat{z}_t\|^2$
- Problem:
  1. Requires a very deep network to handle long sequence
  2. Unable to simulate intermediate time steps



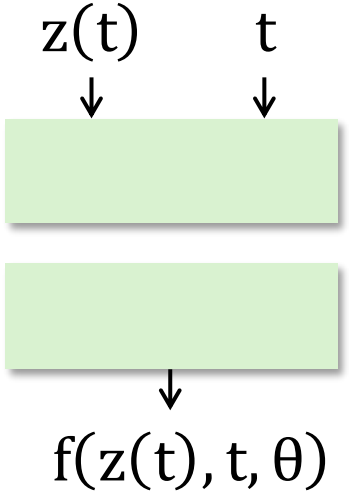


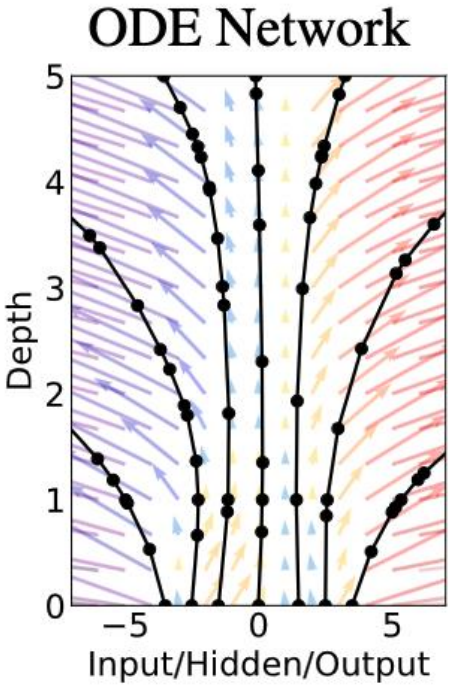
Parameterize Continuous ODE as Neural Network

- Parameterize continuous ODE:

$$z(t_1) = z(t_0) + \int_{t_0}^{t_1} f(z(t), t, \theta)dt$$

- Learning objective: reconstruct data at subsampled time step t': $L = \sum_{t'} \|z_{t'} - \hat{z}_{t'}\|^2$
- Problem:
  1. Gradients need to be back-propagated through time, which is memory inefficient





$$z(t + \Delta t) = z(t) + f(z(t), t, \theta)\Delta t$$
$$z(t + \Delta t + \Delta t) = z(t + \Delta t) + f(z(t + \Delta t), t + \Delta t, \theta)\Delta t$$
$$= z(t) + f(z(t), t, \theta)\Delta t + f(z(t) + f(z(t), t, \theta)\Delta t, t + \Delta t, \theta)\Delta t$$

Backpropagate Gradients Through Time

- Simplify continuous ODE with infinitesimal interval Δt

$$z(t + \Delta t) = z(t) + f(z(t), t, \theta)\Delta t$$

$$z(t + \Delta t + \Delta t) = z(t + \Delta t) + f(z(t + \Delta t), t + \Delta t, \theta)\Delta t$$

$$= z(t) + f(z(t), t, \theta)\Delta t + f(z(t) + f(z(t), t, \theta)\Delta t, t + \Delta t, \theta)\Delta t$$

Backpropagate Gradients Through Time

- Simplify continuous ODE with infinitesimal interval Δt

$$z(t + \Delta t) = z(t) + f(z(t), t, \theta)\Delta t$$

$$z(t + \Delta t + \Delta t) = z(t + \Delta t) + f(z(t + \Delta t), t + \Delta t, \theta)\Delta t$$

$$= z(t) + f(z(t), t, \theta)\Delta t + f(z(t) + f(z(t), t, \theta)\Delta t, t + \Delta t, \theta)\Delta t$$

- Gradients at $t + \Delta t + \Delta t$:

$$\frac{dL(t + \Delta t + \Delta t)}{d\theta} = \frac{dL(t + \Delta t + \Delta t)}{dz(t + \Delta t + \Delta t)} \frac{dz(t + \Delta t + \Delta t)}{d\theta}$$

Can we bypass gradient backpropagation through time?

Neural Ordinary Differential Equations

Ricky T. Q. Chen*, Yulia Rubanova*, Jesse Bettencourt*, David Duvenaud
University of Toronto, Vector Institute
{rtqichen, rubanova, jessebett, duvenaud}@cs.toronto.edu

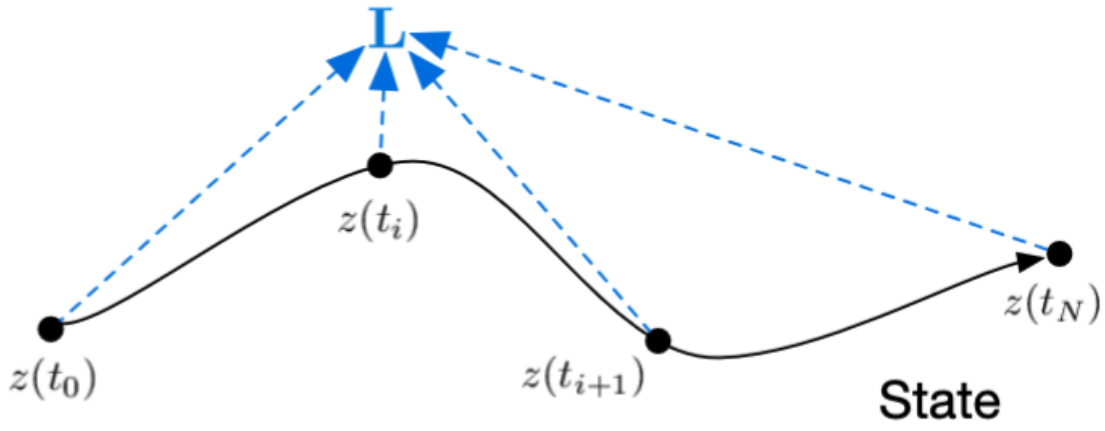
- Key idea: use the adjoint sensitivity method (Pontryagin et al., 1962) to bypass backpropagation through the ODE solver.
- The approach computes gradients by solving a second, augmented ODE backwards in time, and is applicable to all ODE solvers.

Idea: Integrate ODE Neural Network into any ODE Solver

- Forward pass remains the same. Define a scalar loss function $\mathbf{L}(\cdot)$

$$L(\mathbf{z}(t_1)) = L\left(\mathbf{z}(t_0) + \int_{t_0}^{t_1} f(\mathbf{z}(t), t, \theta)dt\right) = L\left(\mathbf{ODESolve}(\mathbf{z}(t_0), f, t_0, t_1, \theta)\right)$$

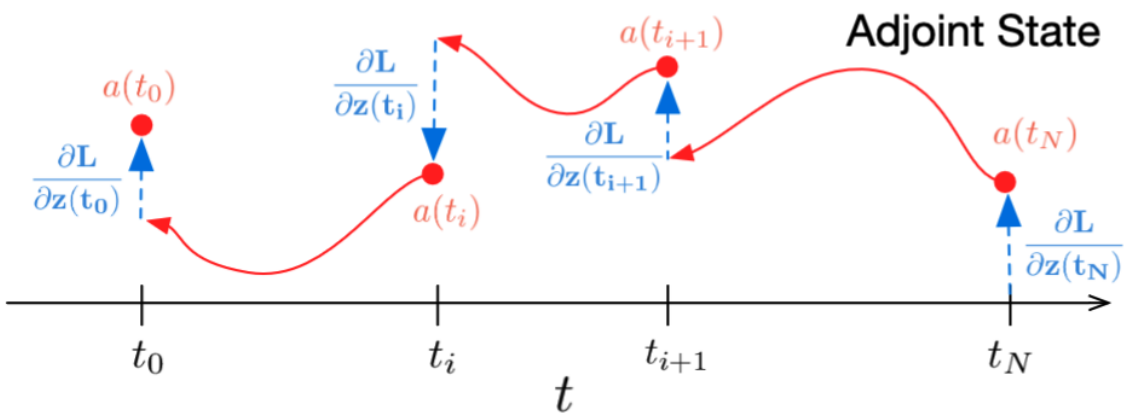
- Calculate loss at sampled time steps



Key Results

- First define adjoint: $\mathbf{a(t)} = \partial \mathbf{L}/\partial \mathbf{z(t)}$ which determines how the gradient of the loss depends on the hidden state $\mathbf{z(t)}$ at each instant
- Key result 1: the dynamics of $\mathbf{a(t)}$ can be given by another ODE

$$\frac{d\mathbf{a}(t)}{dt} = -\mathbf{a}(t)^{\mathsf{T}} \frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \mathbf{z}}$$



This solver must run backwards, starting from the initial value of $\partial \mathbf{L}/\partial \mathbf{z}(t_1)$.

We can simply recompute $\mathbf{z(t)}$ backwards in time together with the adjoint, starting from its final value $\mathbf{z}(t_1)$

Key Results

- First define adjoint: $\mathbf{a(t)} = \partial \mathrm{L} / \partial \mathbf{z(t)}$ which determines how the gradient of the loss depends on the hidden state $\mathbf{z(t)}$ at each instant
- Key result 1: the dynamics of $\mathbf{a(t)}$ can be given by another ODE

$$\frac{d\mathbf{a}(t)}{dt} = -\mathbf{a}(t)^{\mathsf{T}} \frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \mathbf{z}}$$

- Key result 2: computes the gradients with respect to parameter $\boldsymbol{\theta}$ requires evaluating a third integral, which depends on both $\mathbf{z(t)}$ and $\mathbf{a(t)}$:

$$\frac{dL}{d\theta} = -\int_{t_1}^{t_0} \mathbf{a}(t)^{\mathsf{T}} \frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \theta} dt$$

Proof of Result 1

$$\frac{d\mathbf{a}(t)}{dt} = -\mathbf{a}(t)^{\mathsf{T}} \frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \mathbf{z}}$$

- We first define:

$$\mathbf{a}(t) = \frac{dL}{d\mathbf{z}(t)} \qquad \mathbf{z}(t+\varepsilon) = \int_t^{t+\varepsilon} f(\mathbf{z}(t), t, \theta)dt + \mathbf{z}(t) = T_\varepsilon(\mathbf{z}(t), t)$$

- From chain rule:

$$\frac{dL}{\partial \mathbf{z}(t)} = \frac{dL}{d\mathbf{z}(t+\varepsilon)} \frac{d\mathbf{z}(t+\varepsilon)}{d\mathbf{z}(t)} \quad \Rightarrow \quad \mathbf{a}(t) = \mathbf{a}(t+\varepsilon) \frac{\partial T_\varepsilon(\mathbf{z}(t), t)}{\partial \mathbf{z}(t)}$$

$$\frac{d\mathbf{a}(t)}{dt} = \lim_{\varepsilon \to 0^+} \frac{\mathbf{a}(t+\varepsilon) - \mathbf{a}(t)}{\varepsilon} \tag{39}$$

$$= \lim_{\varepsilon \to 0^+} \frac{\mathbf{a}(t+\varepsilon) - \mathbf{a}(t+\varepsilon)\frac{\partial}{\partial \mathbf{z}(t)} T_\varepsilon(\mathbf{z}(t))}{\varepsilon} \quad \text{(by Eq 38)} \tag{40}$$

$$= \lim_{\varepsilon \to 0^+} \frac{\mathbf{a}(t+\varepsilon) - \mathbf{a}(t+\varepsilon)\frac{\partial}{\partial \mathbf{z}(t)} \left(\mathbf{z}(t) + \varepsilon f(\mathbf{z}(t), t, \theta) + \mathcal{O}(\varepsilon^2)\right)}{\varepsilon} \quad \text{(Taylor series around } \mathbf{z}(t)\text{)} \tag{41}$$

$$= \lim_{\varepsilon \to 0^+} \frac{\mathbf{a}(t+\varepsilon) - \mathbf{a}(t+\varepsilon)\left(I + \varepsilon \frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \mathbf{z}(t)} + \mathcal{O}(\varepsilon^2)\right)}{\varepsilon} \tag{42}$$

$$= \lim_{\varepsilon \to 0^+} \frac{-\varepsilon \mathbf{a}(t+\varepsilon)\frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \mathbf{z}(t)} + \mathcal{O}(\varepsilon^2)}{\varepsilon} \tag{43}$$

$$= \lim_{\varepsilon \to 0^+} -\mathbf{a}(t+\varepsilon)\frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \mathbf{z}(t)} + \mathcal{O}(\varepsilon) \tag{44}$$

$$= -\mathbf{a}(t)\frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \mathbf{z}(t)} \tag{45}$$

Proof of Result 2

$$\frac{dL}{d\theta} = -\int_{t_1}^{t_0} \mathbf{a}(t)^\mathsf{T} \frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \theta} dt$$

- We have: $\frac{\partial \theta(t)}{\partial t} = \mathbf{0}$ $\frac{dt(t)}{dt} = 1$

- We define augmented ODE:

$$\frac{d}{dt}\begin{bmatrix}\mathbf{z}\\ \theta \\ t\end{bmatrix}(t) = f_{aug}([\mathbf{z}, \theta, t]) := \begin{bmatrix} f([\mathbf{z}, \theta, t]) \\ \mathbf{0} \\ 1 \end{bmatrix}, \ \mathbf{a}_{aug} := \begin{bmatrix}\mathbf{a}\\ \mathbf{a}_\theta \\ \mathbf{a}_t\end{bmatrix}, \ \mathbf{a}_\theta(t) := \frac{dL}{d\theta(t)}, \ \mathbf{a}_t(t) := \frac{dL}{dt(t)}$$

- We have

$$\frac{\partial f_{aug}}{\partial [\mathbf{z}, \theta, t]} = \begin{bmatrix} \frac{\partial f}{\partial \mathbf{z}} & \frac{\partial f}{\partial \theta} & \frac{\partial f}{\partial t} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix}(t)$$

Proof of Result 2

$$\frac{dL}{d\theta} = -\int_{t_1}^{t_0} \mathbf{a}(t)^\mathsf{T} \frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \theta} dt$$

- From key result 1, we have $\frac{d\mathbf{a}(t)}{dt} = -\mathbf{a}(t)^\mathsf{T} \frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \mathbf{z}}$

- Substitute it to $\mathbf{a_{aug}}$

$$\frac{d\mathbf{a}_{aug}(t)}{dt} = -\begin{bmatrix}\mathbf{a}(t) & \mathbf{a}_\theta(t) & \mathbf{a}_t(t)\end{bmatrix} \frac{\partial f_{aug}}{\partial[\mathbf{z}, \theta, t]}(t) = -\begin{bmatrix}\mathbf{a}\frac{\partial f}{\partial \mathbf{z}} & \mathbf{a}\frac{\partial f}{\partial \theta} & \mathbf{a}\frac{\partial f}{\partial t}\end{bmatrix}(t)$$

- As a result, we have:

$$\frac{dL}{d\theta} = \mathbf{a}_\theta(t_0) = -\int_{t_N}^{t_0} \mathbf{a}(t) \frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \theta} \, dt$$

Overall Algorithm

Algorithm 1 Reverse-mode derivative of an ODE initial value problem

Input: dynamics parameters θ, start time t_0, stop time t_1, final state $\mathbf{z}(t_1)$, loss gradient $\partial L / \partial \mathbf{z}(t_1)$

$s_0 = [\mathbf{z}(t_1), \frac{\partial L}{\partial \mathbf{z}(t_1)}, \mathbf{0}_{|\theta|}]$ ▷ Define initial augmented state

def aug_dynamics$([\mathbf{z}(t), \mathbf{a}(t), \cdot], t, \theta)$: ▷ Define dynamics on augmented state

  return $[f(\mathbf{z}(t), t, \theta), -\mathbf{a}(t)^\mathsf{T} \frac{\partial f}{\partial \mathbf{z}}, -\mathbf{a}(t)^\mathsf{T} \frac{\partial f}{\partial \theta}]$ ▷ Compute vector-Jacobian products

$[\mathbf{z}(t_0), \frac{\partial L}{\partial \mathbf{z}(t_0)}, \frac{\partial L}{\partial \theta}] = \text{ODESolve}(s_0, \text{aug\_dynamics}, t_1, t_0, \theta)$ ▷ Solve reverse-time ODE

return $\frac{\partial L}{\partial \mathbf{z}(t_0)}, \frac{\partial L}{\partial \theta}$ ▷ Return gradients

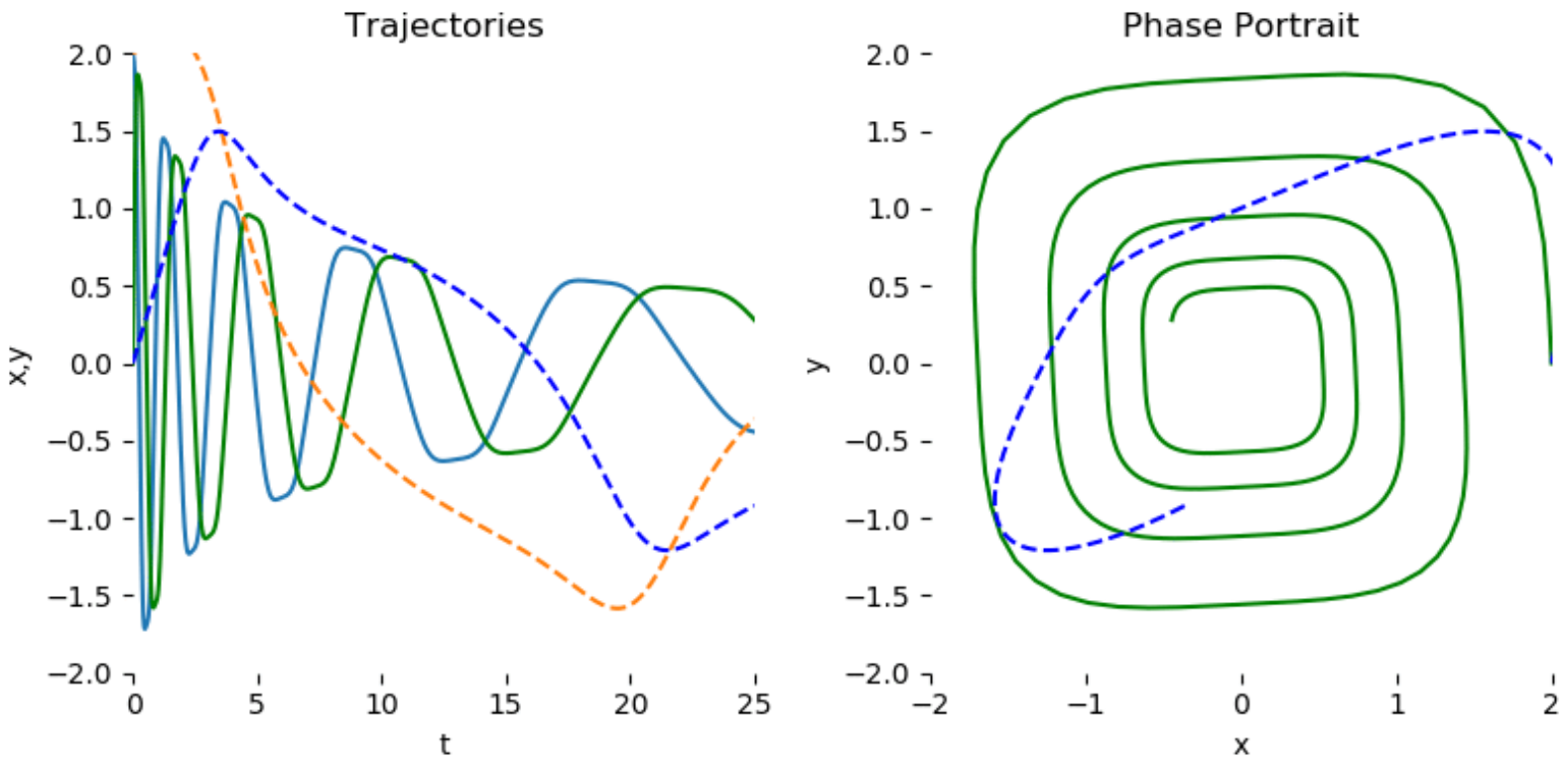
1. Obtain the dynamics of $\mathbf{a(t)}$ by reverse ODE

$$\frac{d\mathbf{a}(t)}{dt} = -\mathbf{a}(t)^\mathsf{T} \frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \mathbf{z}}$$

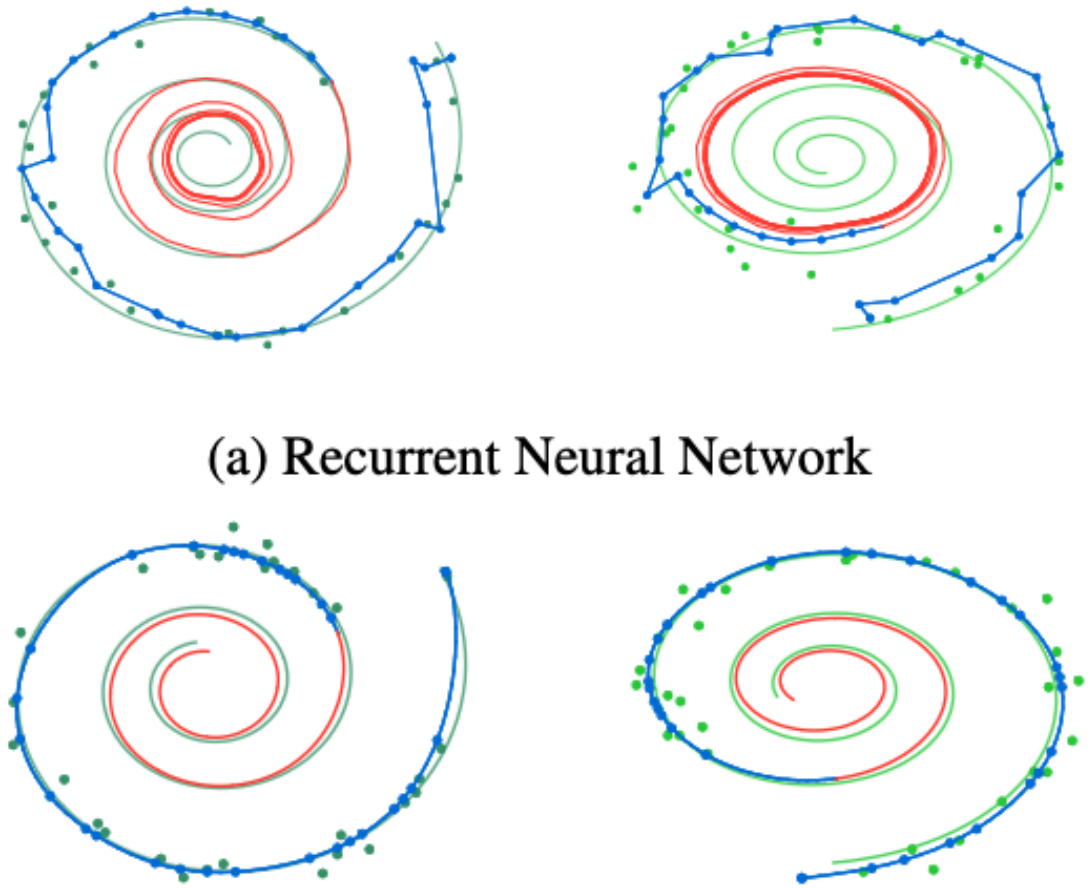
2. Compute the gradients with respect to parameter $\boldsymbol{\theta}$ with a third integral

$$\frac{dL}{d\theta} = -\int_{t_1}^{t_0} \mathbf{a}(t)^\mathsf{T} \frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \theta} dt$$

Results


Trajectories
x,y
t
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0
5
10
15
20
25
Phase Portrait
y
x
−2
−1
0
1
2

Results



(a) Recurrent Neural Network

(b) Latent Neural Ordinary Differential Equation

Content

- Pure Data-driven Approaches
- Hybrid Approaches
  - ➢Physics-Informed Diffusion Model
  - ➢Physics-Informed Neural Network
  - ➢Neural ODE
  - ➢Lagrangian Network / Hamiltonian Network
  - ➢Neural Operator
  - ➢Fourier Neural Operator

Lagrangian Mechanics

- The Lagrangian:

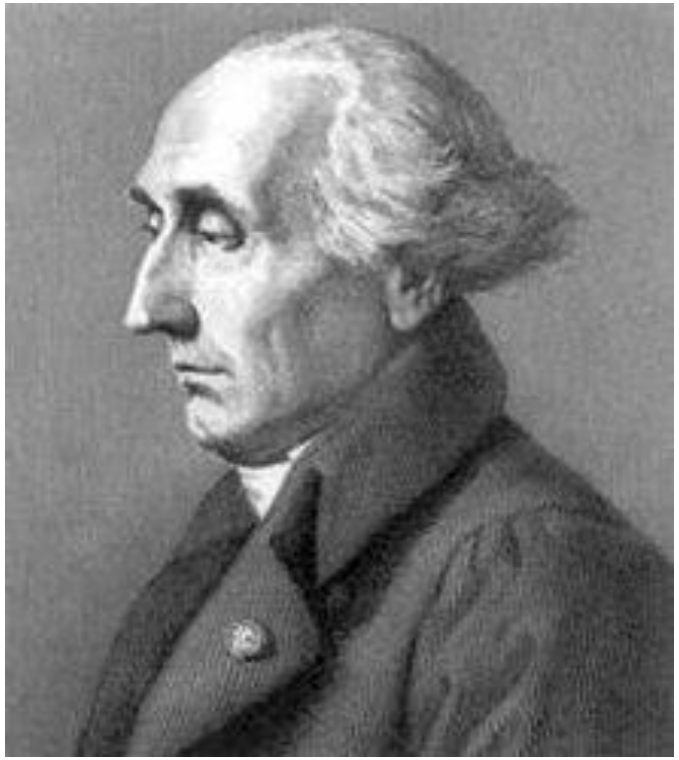
Potential energy

$$L = T - V$$

Kinetic energy

- Euler-Lagrange Equation:

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{\boldsymbol{q}}} = \frac{\partial L}{\partial \boldsymbol{q}} \Rightarrow \frac{d}{dt}\frac{\partial T}{\partial \dot{\boldsymbol{q}}} = -\frac{\partial V}{\partial \boldsymbol{q}}$$



Joseph-Louis Lagrange

Hamiltonian Mechanics

- We can derive Hamiltonian mechanics from Lagrangian:

$$p_i = \frac{\delta \mathrm{L}}{\delta \dot{\mathrm{q}}_i}; \quad \mathrm{H} = \sum_i \dot{\mathrm{q}}_i p_i - \mathrm{L}$$

- Hamiltonian Equation:

$$\frac{d\mathrm{q}}{dt} = \frac{\partial \mathrm{H}}{\partial \mathrm{p}}; \quad \frac{d\mathrm{p}}{dt} = -\frac{\partial \mathrm{H}}{\partial \mathrm{q}}$$

William Rowan Hamilton

Can We Directly Bake Lagrangian / Hamiltonian Mechanics into Neural Networks?

Hamiltonian Mechanics

- We can derive Hamiltonian mechanics from Lagrangian:

$$p_i = \frac{\delta \mathrm{L}}{\delta \dot{\mathrm{q}}_i}; \quad \mathrm{H} = \sum_i \dot{\mathrm{q}}_i p_i - \mathrm{L}$$

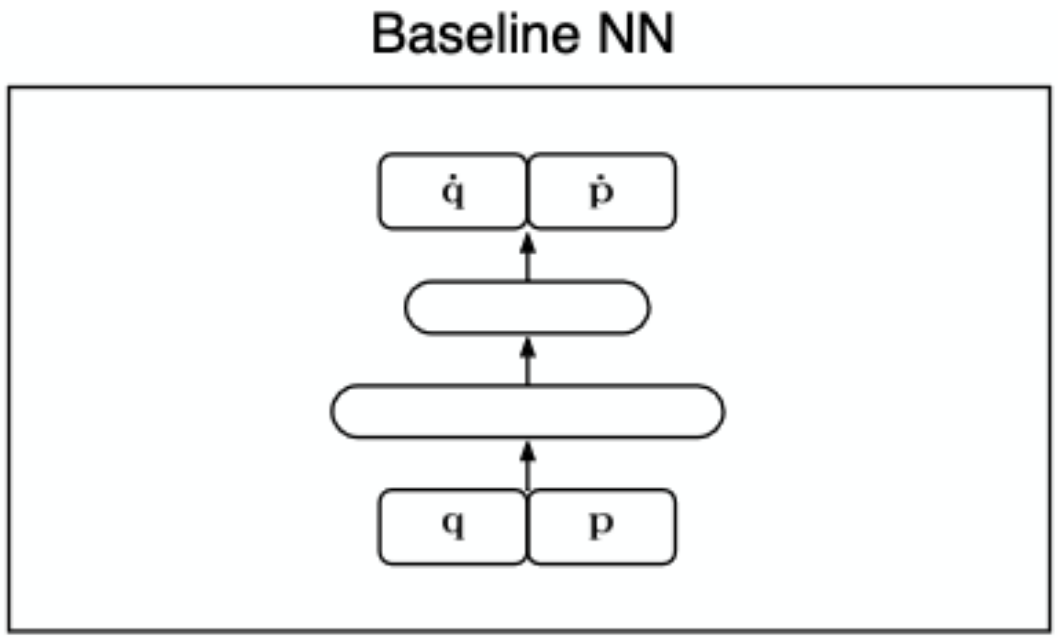
- Hamiltonian Equation:

$$\frac{d\mathrm{q}}{dt} = \frac{\partial \mathrm{H}}{\partial \mathrm{p}}; \quad \frac{d\mathrm{p}}{dt} = -\frac{\partial \mathrm{H}}{\partial \mathrm{q}}$$

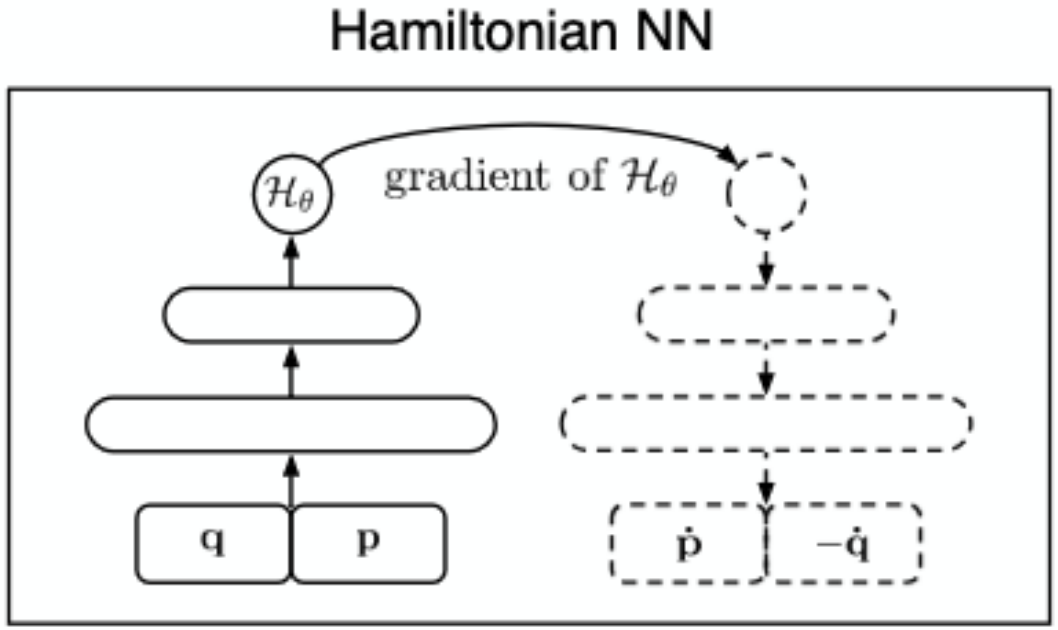
William Rowan Hamilton

Output Motions vs. Output Energy to Derive Motions

- Hamiltonian Equation: $\frac{d\mathrm{q}}{dt} = \frac{\partial \mathrm{H}}{\partial \mathrm{p}}; \quad \frac{d\mathrm{p}}{dt} = -\frac{\partial \mathrm{H}}{\partial \mathrm{q}}$



Output motions

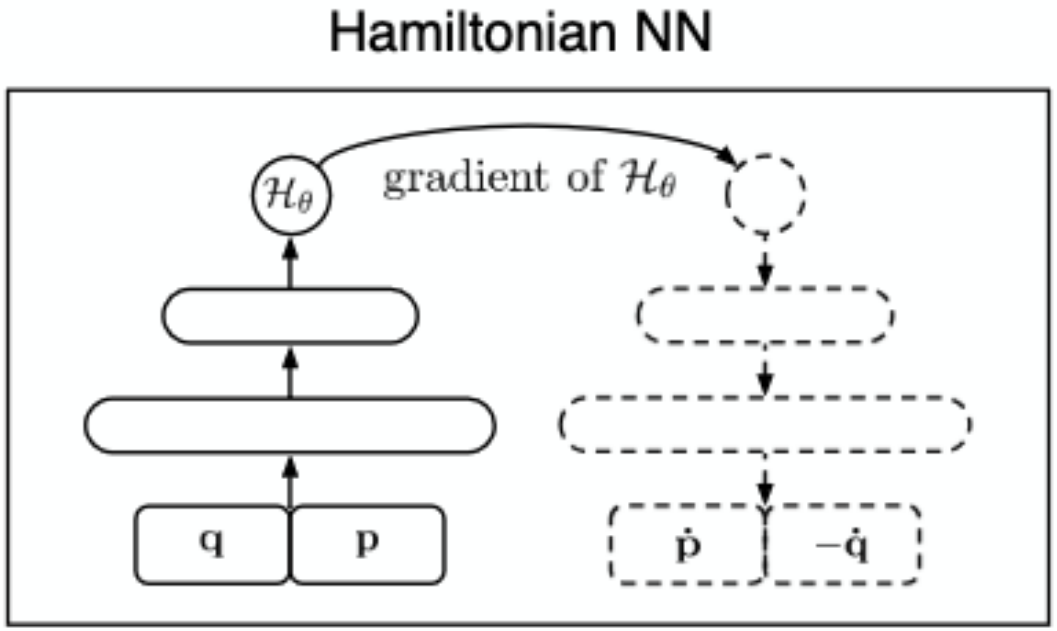


Output energy and derive motions with autograd

Output Motions vs. Output Energy to Derive Motions

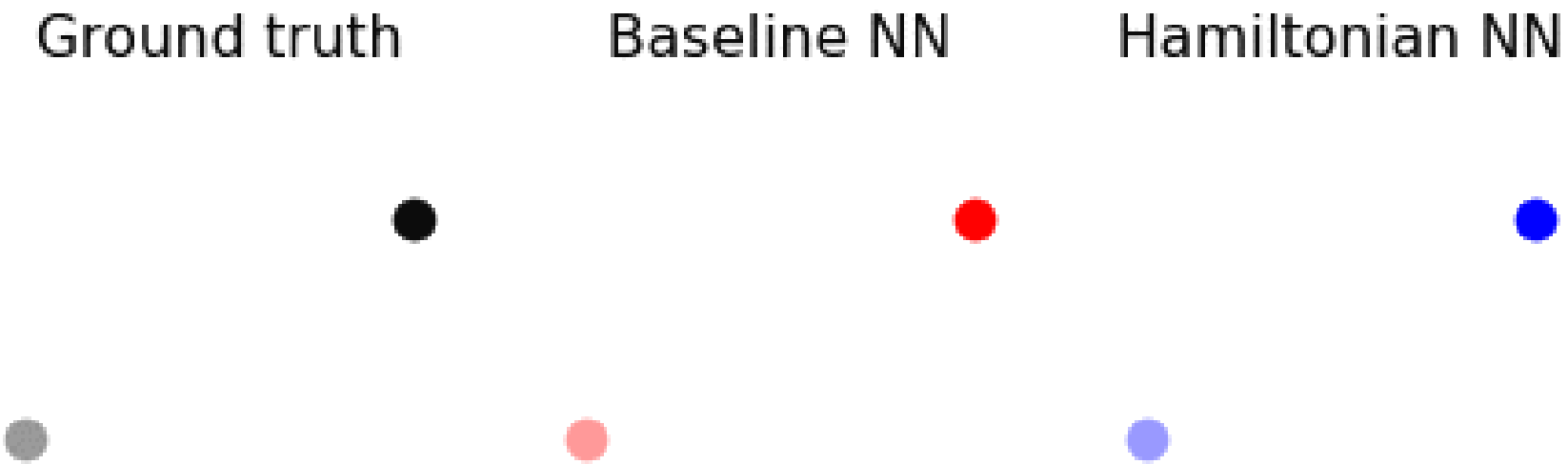
- Hamiltonian Equation: $\frac{d\mathrm{q}}{dt} = \frac{\partial \mathrm{H}}{\partial \mathrm{p}}; \quad \frac{d\mathrm{p}}{dt} = -\frac{\partial \mathrm{H}}{\partial \mathrm{q}}$

$$\mathcal{L}_{HNN} = \left\| \frac{\partial \mathcal{H}_\theta}{\partial \mathbf{p}} - \frac{\partial \mathbf{q}}{\partial t} \right\|_2 + \left\| \frac{\partial \mathcal{H}_\theta}{\partial \mathbf{q}} + \frac{\partial \mathbf{p}}{\partial t} \right\|_2$$



Output energy and derive motions with autograd

Results

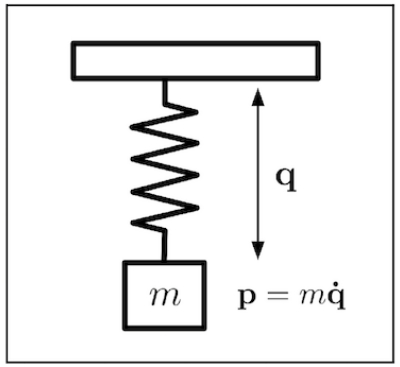


Task 1: Ideal Mass-Spring System

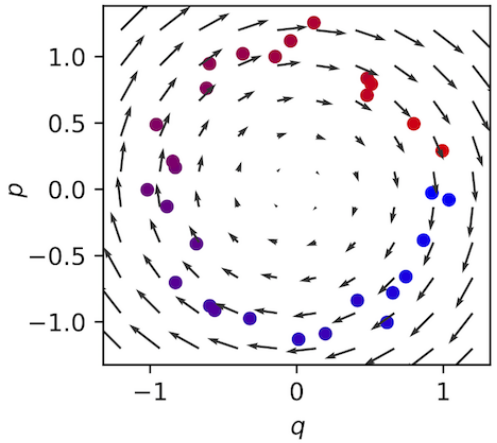
Hamiltonian

$$\mathcal{H} = \frac{1}{2}kq^2 + \frac{p^2}{2m}$$

Ideal mass-spring system



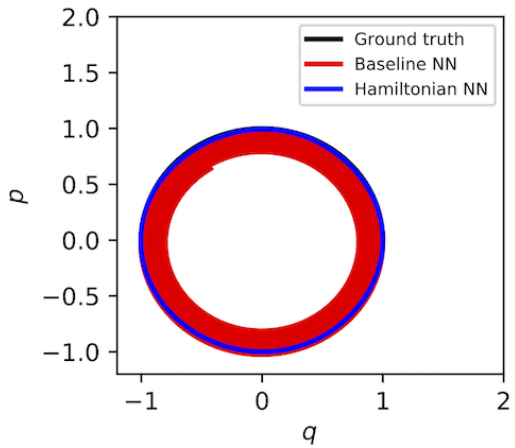
Noisy observations



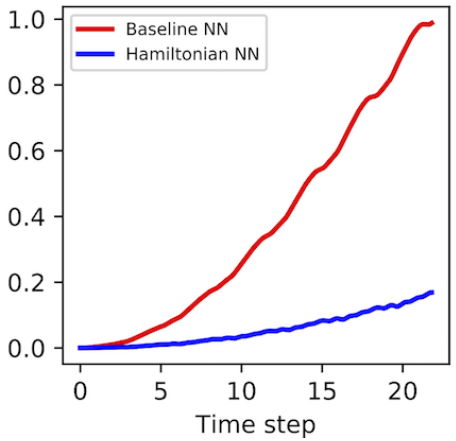
Training settings

- 25 trajectories of 30 observations each
- Adam with 10^{-3} learn rate
- 3 layers of 200 units
- 2000 gradient steps
- `tanh` activations

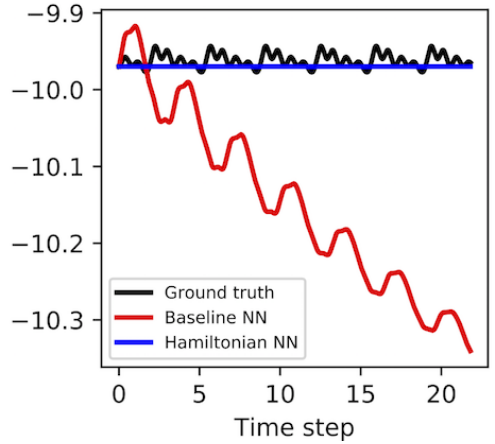
Predictions



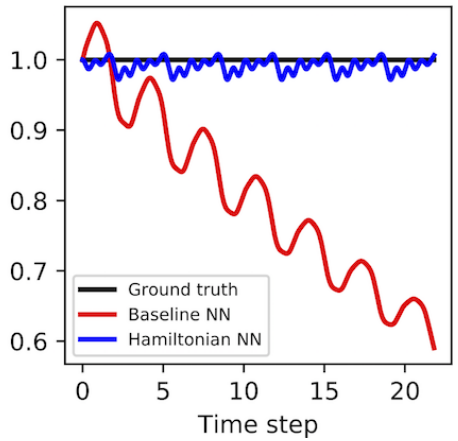
MSE between coordinates



Total HNN-conserved quantity



Total energy



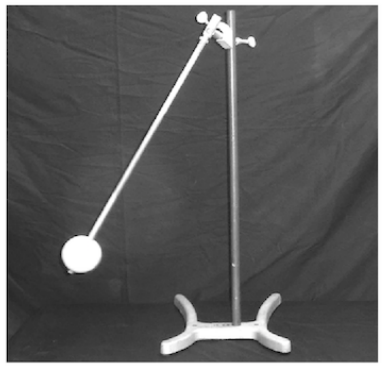
Energy is conserved!

Task 3: Real Pendulum

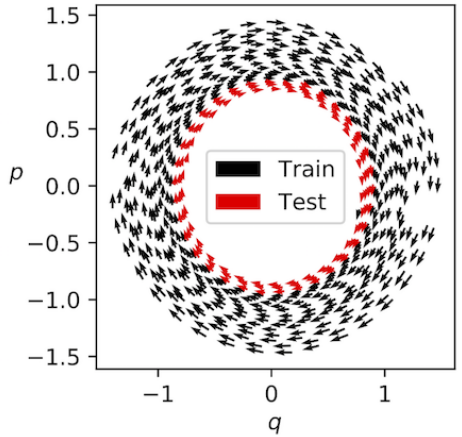
Hamiltonian

$$\mathcal{H} = 2mgl(1 - \cos q) + \frac{l^2 p^2}{2m}$$

Real pendulum



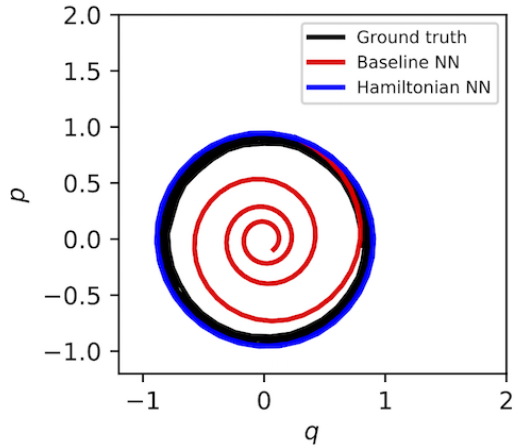
Noisy observations



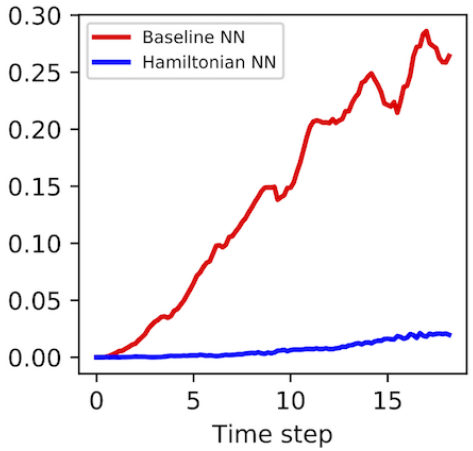
Training settings

- 1 trajectory of 555 observations
- Adam with 10^{-3} learn rate
- 3 layers of 200 units
- 2000 gradient steps
- `tanh` activations

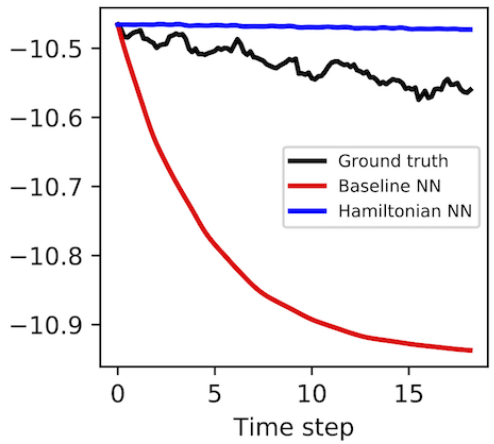
Predictions



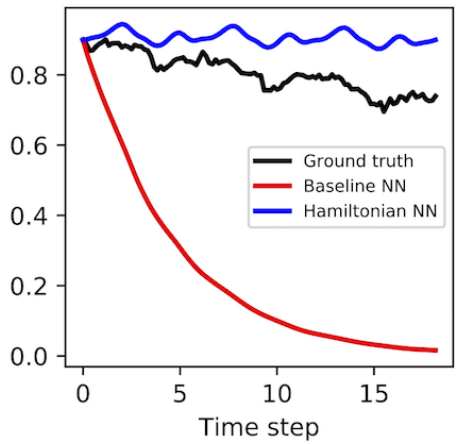
MSE between coordinates



Total HNN-conserved quantity



Total energy



Energy is conserved!

Lagrangian Mechanics

- The Lagrangian:

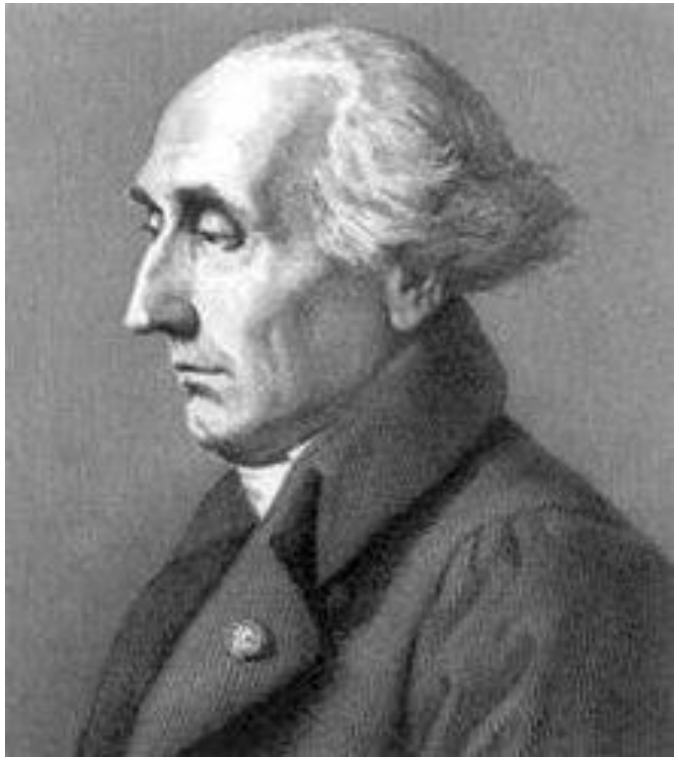
Potential energy

$$L = T - V$$

Kinetic energy

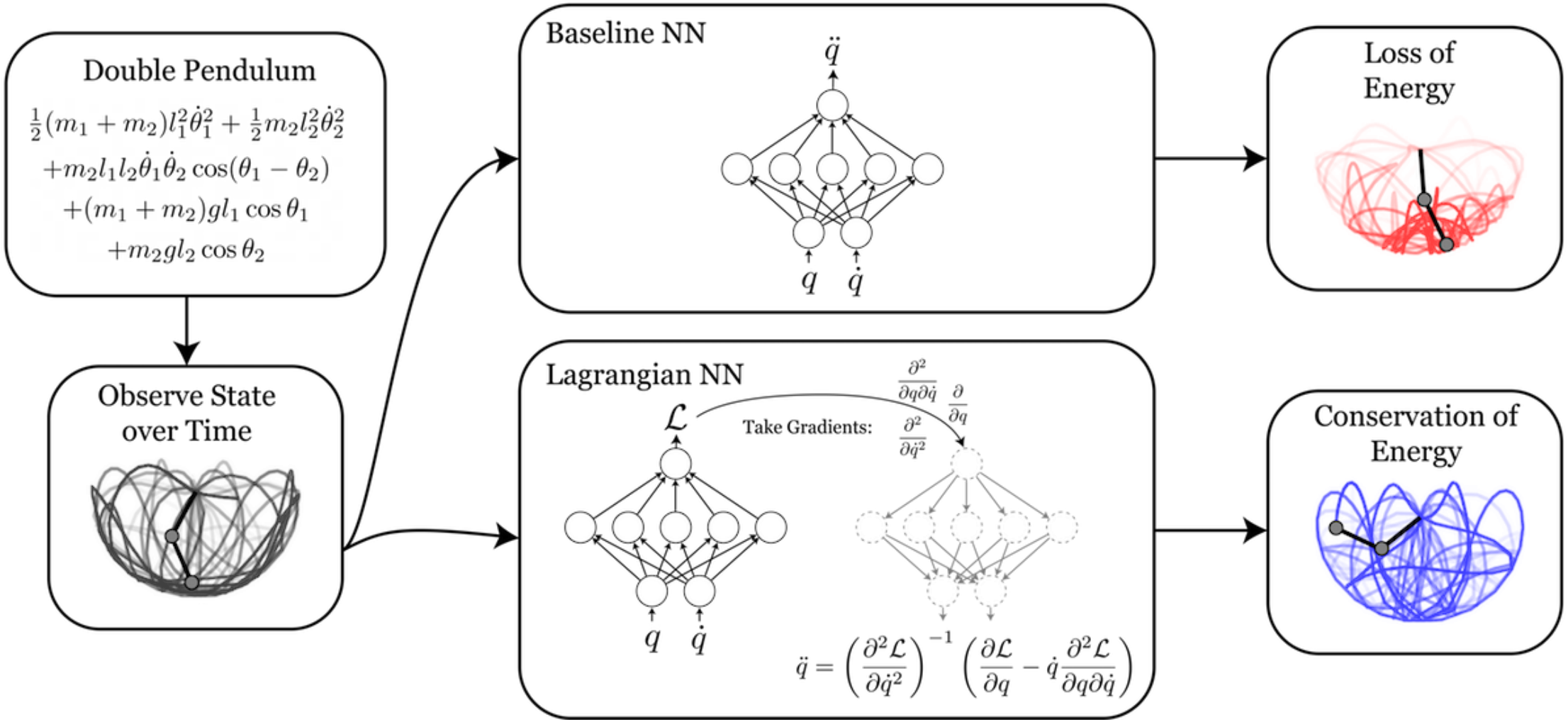
- Euler-Lagrange Equation:

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{\boldsymbol{q}}} = \frac{\partial L}{\partial \boldsymbol{q}} \Rightarrow \frac{d}{dt}\frac{\partial T}{\partial \dot{\boldsymbol{q}}} = -\frac{\partial V}{\partial \boldsymbol{q}}$$



Joseph-Louis Lagrange

Lagrangian Neural Network Follows the Same Idea



The Derivation of Motions from Lagrangian

$$\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{q}_j} = \frac{\partial \mathcal{L}}{\partial q_j} \qquad \text{the Euler-Lagrange equation} \quad (5)$$

$$\frac{d}{dt}\nabla_{\dot{q}}\mathcal{L} = \nabla_q \mathcal{L} \qquad \text{switch to vector notation} \quad (6)$$

$$(\nabla_{\dot{q}}\nabla_{\dot{q}}^\top \mathcal{L})\ddot{q} + (\nabla_q \nabla_{\dot{q}}^\top \mathcal{L})\dot{q} = \nabla_q \mathcal{L} \qquad \text{expand time derivative } \frac{d}{dt} \quad (7)$$

$$\ddot{q} = (\nabla_{\dot{q}}\nabla_{\dot{q}}^\top \mathcal{L})^{-1}[\nabla_q \mathcal{L} - (\nabla_q \nabla_{\dot{q}}^\top \mathcal{L})\dot{q}] \qquad \text{matrix inverse to solve for } \ddot{q} \quad (8)$$

https://greydanus.github.io/2020/03/10/lagrangian-nns/

Results

Baseline NN

Lagrangian NN

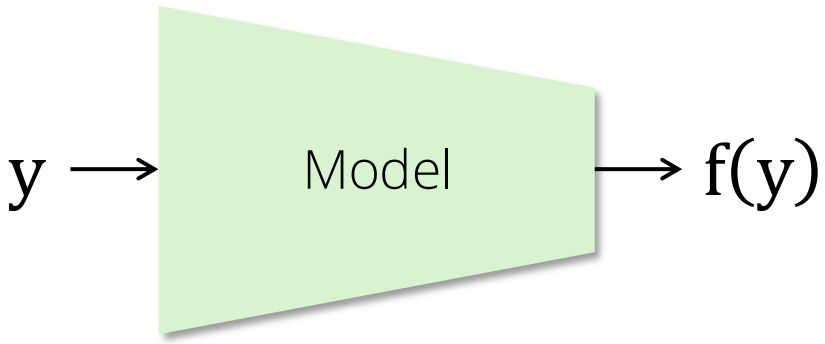
https://www.youtube.com/watch?v=ulQKNtTEuJI&t=14s

Content

- Pure Data-driven Approaches
- Hybrid Approaches
  - ➢Physics-Informed Diffusion Model
  - ➢Physics-Informed Neural Network
  - ➢Neural ODE
  - ➢Lagrangian Network / Hamiltonian Network
  - ➢Neural Operator
  - ➢Fourier Neural Operator

Function Learning

- Function learning: given an input point $\mathbf{y}$, learn to predict $\mathbf{f}(\mathbf{y})$
- For example, the function $\mathbf{f}(\cdot)$ describes the fluid motion

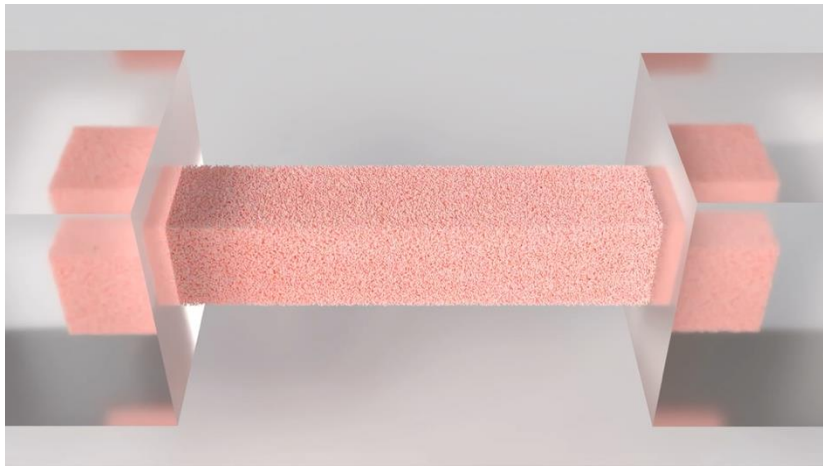


The model only learns to simulate one function

Video source: Genesis simulator

The Idea of Functional

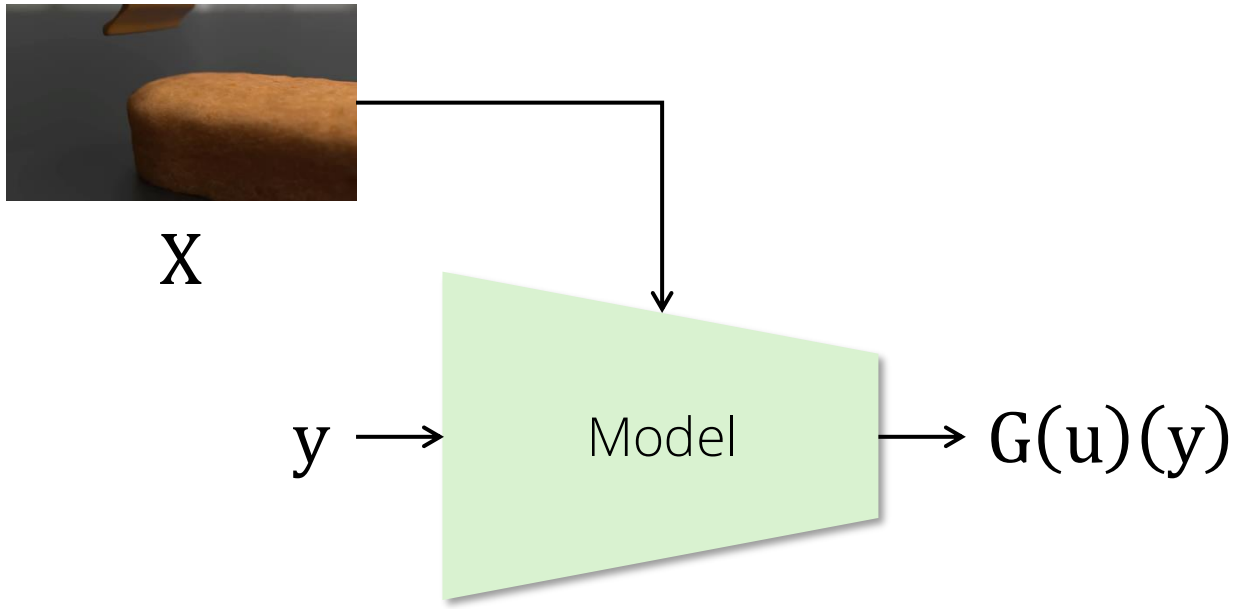
- Let $\mathbf{G}$ denote an operator taking an input function $\mathbf{u}$, then $\mathbf{G(u)}$ is the corresponding output function and $\mathbf{G(u)(y)}$ evaluates the function at point $\mathbf{y}$
- For example, the function $\mathbf{G(fluid), G(sponge), G(fruit)}$ describes the fluid, sponge and fruit motion


Video source: Genesis simulator

Extend Function Learning to Operator Learning

- Operator learning: given an input function $\mathbf{u}$ and an input point $\mathbf{y}$, learn to predict $\mathbf{G(u)(y)}$
- The input function $\mathbf{u}$ is obtained from a sparse set of observations $\mathbf{X}$.
- For example, $\mathbf{X}$ may be the video of the fluid motion and $\mathbf{y}$ corresponds to an internal point of the fluid.



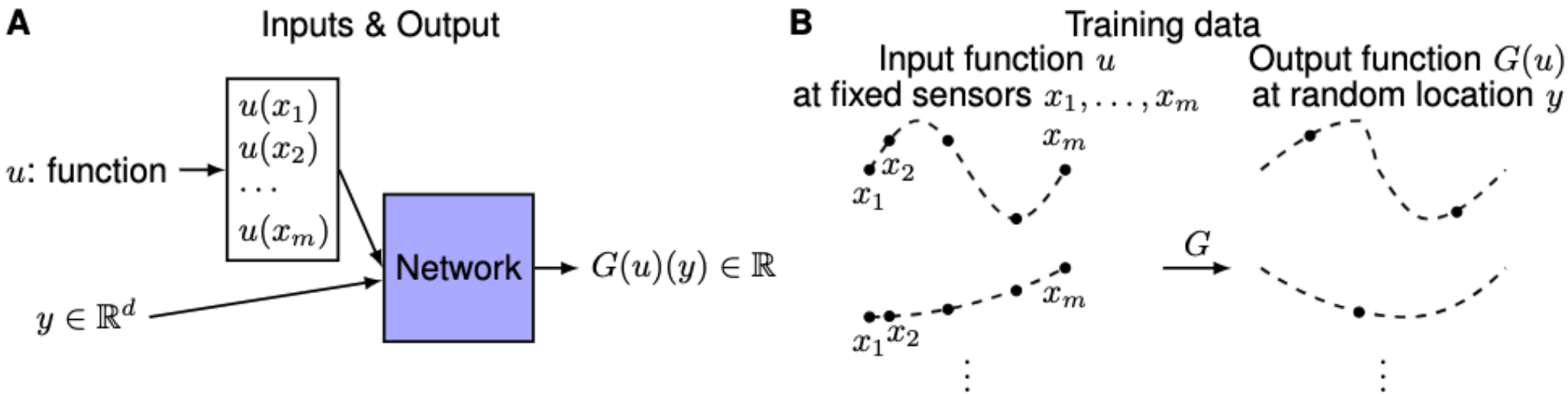
Why Operator Learning is powerful?

- Share structures among different dynamics
- Decouple conditional inputs from outputs
  - Predict high-res outputs while conditioning on low-res inputs
  - Conditions inputs from different modalities (e.g. lidar, radar, video ...)

DeepONet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators

Lu Lu[1], Pengzhan Jin[2], and George Em Karniadakis[1]

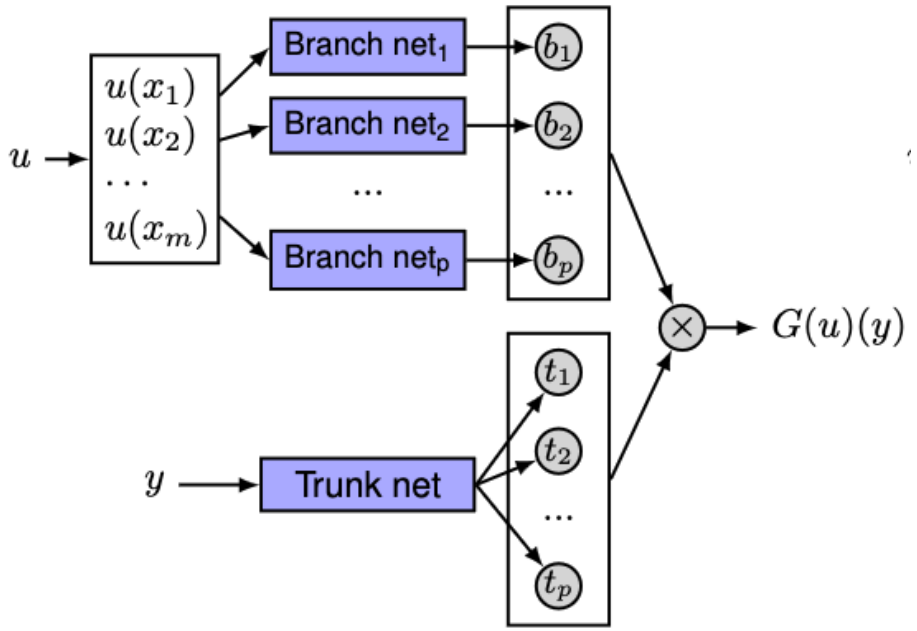
[1]Division of Applied Mathematics, Brown University, Providence, RI 02912, USA
[2]LSEC, ICMSEC, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing 100190, China



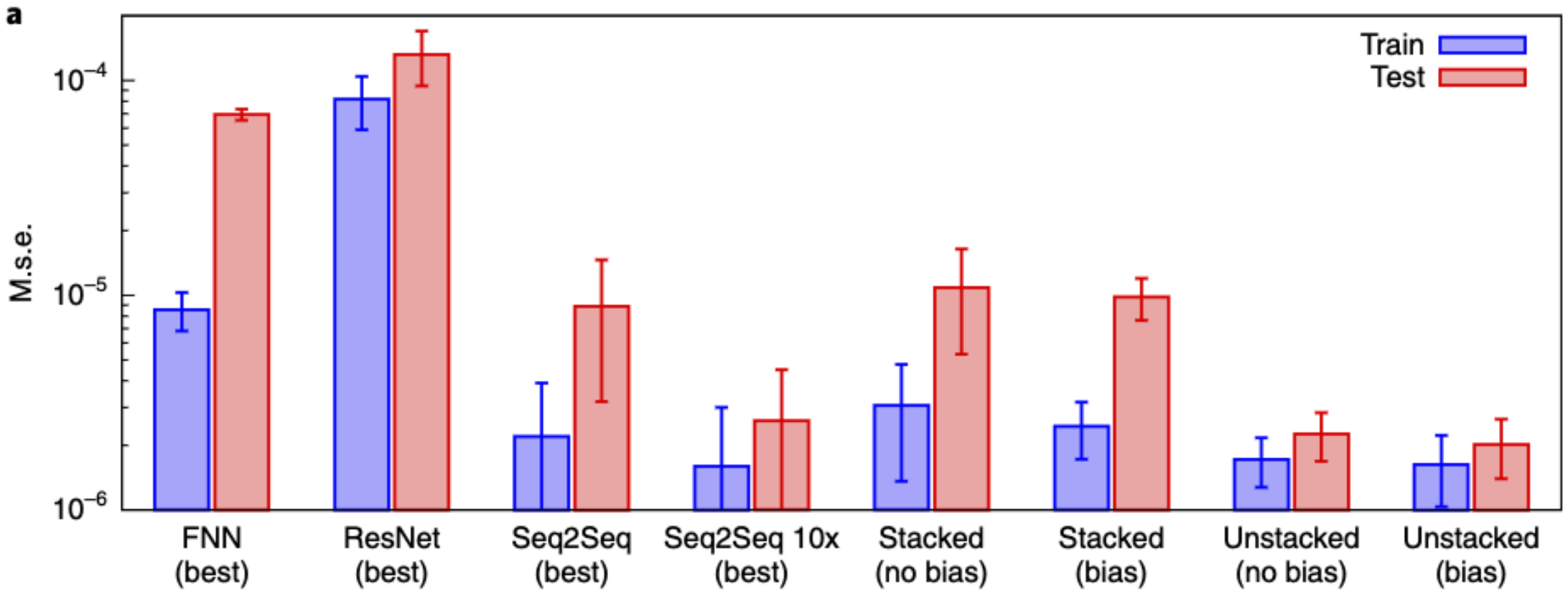
A Tensor-Product Factorization

- Idea: predict $\mathbf{G(u)(y)}$ with the factorization of $\mathbf{h(X)\otimes h'(y)}$ since tensor product $\mathbf{V\otimes W}$ forms the most general vector space that receives a bilinear map from $\mathbf{V\times W}$

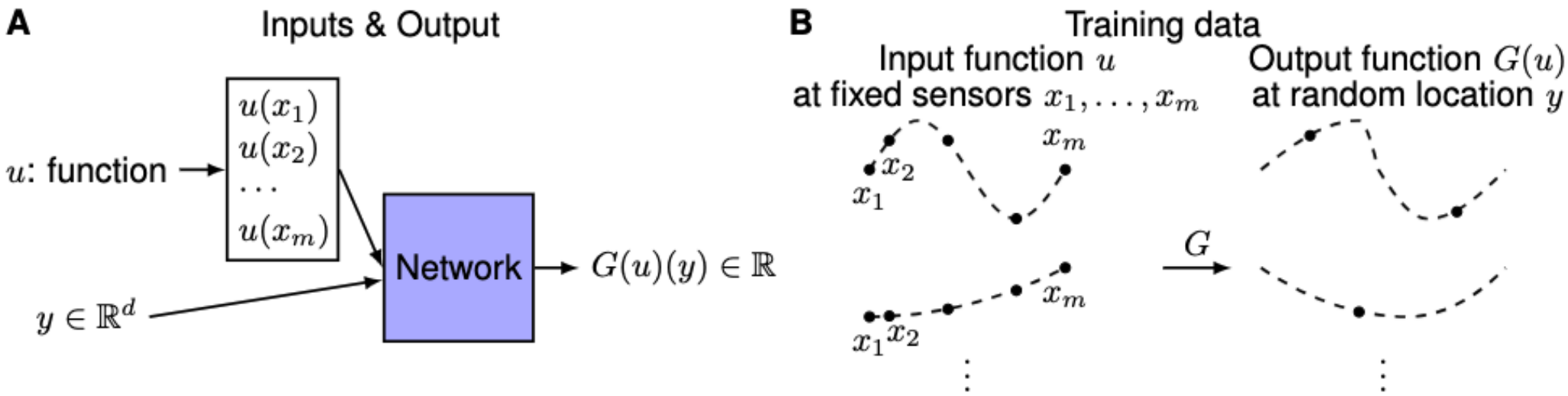
$$\left| G(u)(y) - \sum_{k=1}^{p} \underbrace{\sum_{i=1}^{n} c_i^k \sigma \left(\sum_{j=1}^{m} \xi_{ij}^k u(x_j) + \theta_i^k \right)}_{branch} \underbrace{\sigma(w_k \cdot y + \zeta_k)}_{trunk} \right| < \epsilon$$



Results



Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. Lu et al.



- Pros: query the output at any point
- Cons: the model doesn't generalize to different resolutions of observations of input functions

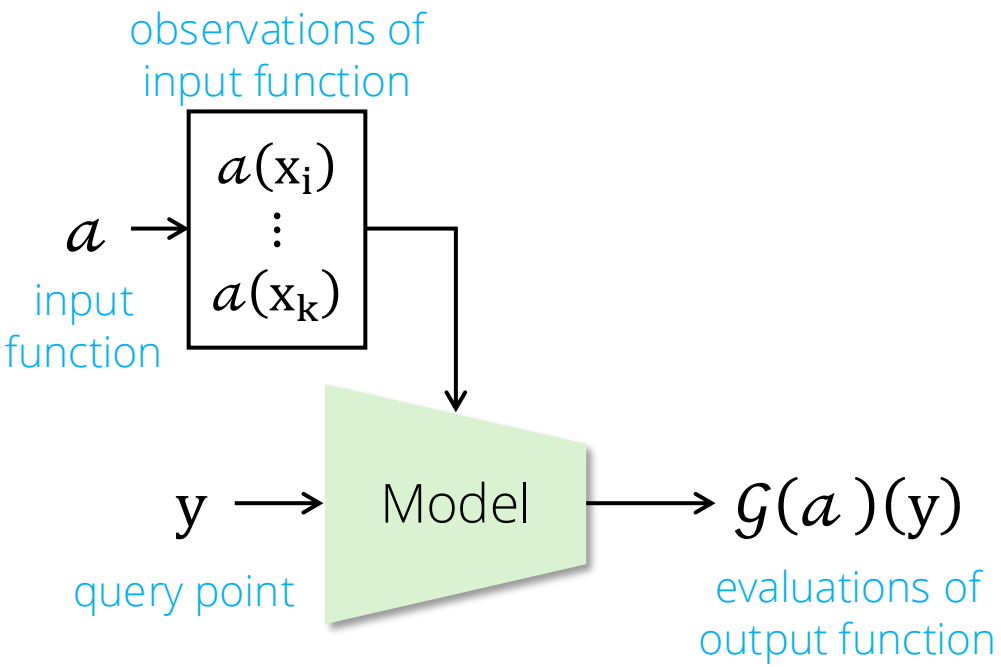
Can we build a neural operator that is Discretization Invariant?

Content

- Pure Data-driven Approaches
- Hybrid Approaches
  - Physics-Informed Diffusion Model
  - Physics-Informed Neural Network
  - Neural ODE
  - Lagrangian Network / Hamiltonian Network
  - Neural Operator
  - Fourier Neural Operator

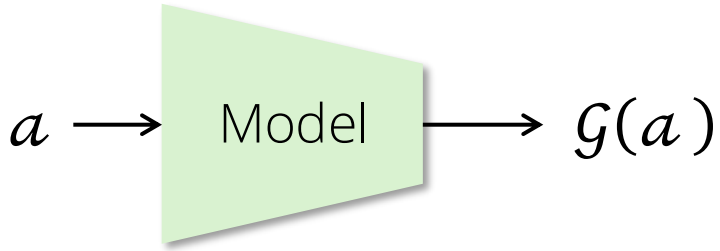
Conceptual Comparison

DeepONet



Neural Operator:

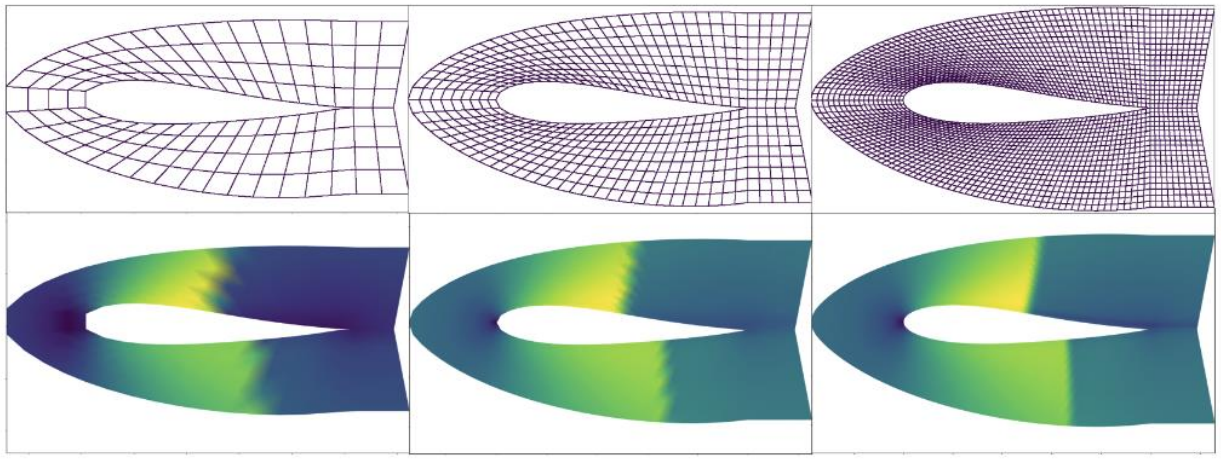
Convert an input function a to an output function $\mathcal{G}(a)$



Neural Operator in Reality

Sampled points at different levels of discretization

x_1 ... x_L



$\mathcal{G}(x_1)$ ... $\mathcal{G}(x_L)$

Evaluations of output functions at different levels of discretization

Neural Operator:

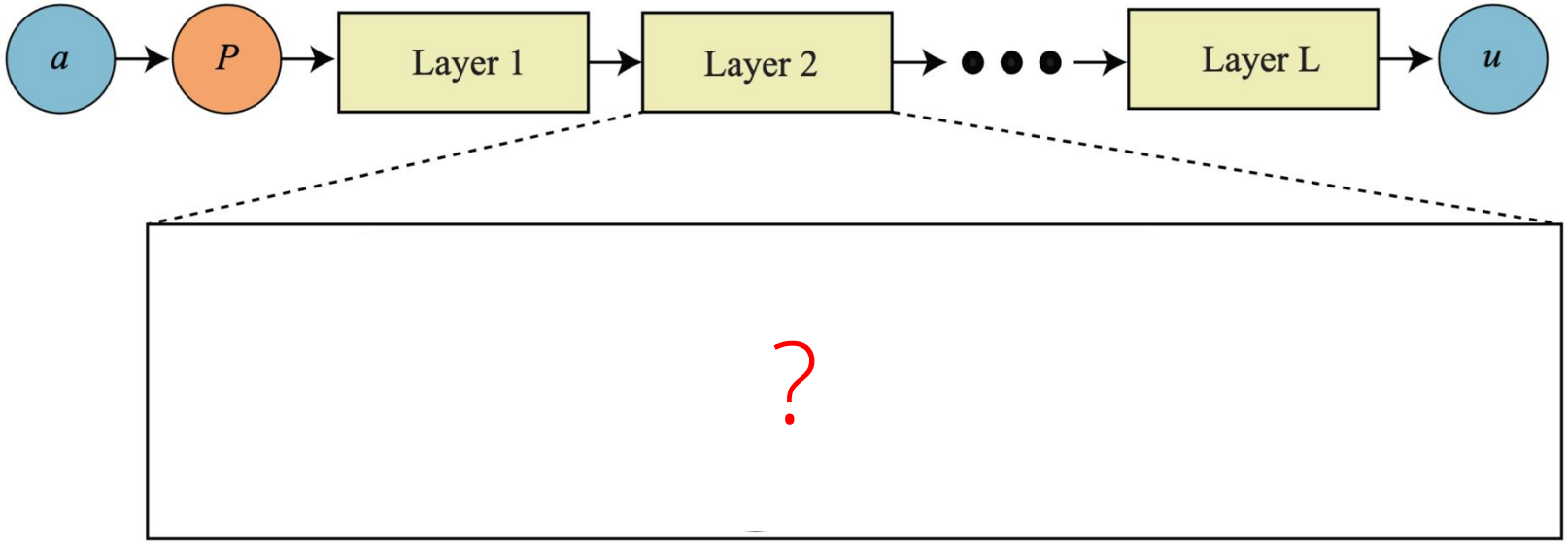
Convert an input function a to an output function $\mathcal{G}(a)$

$a(x_l) \longrightarrow$ Model $\longrightarrow \mathcal{G}(a)(x_l)$

Can we train on one level of discretization while generalizing to different levels of discretization?

Neural Operator: Learning Maps Between Function Spaces With Applications to PDEs. Nikola et al.

Map One Function to Another while Maintaining Discretization Invariant



Train

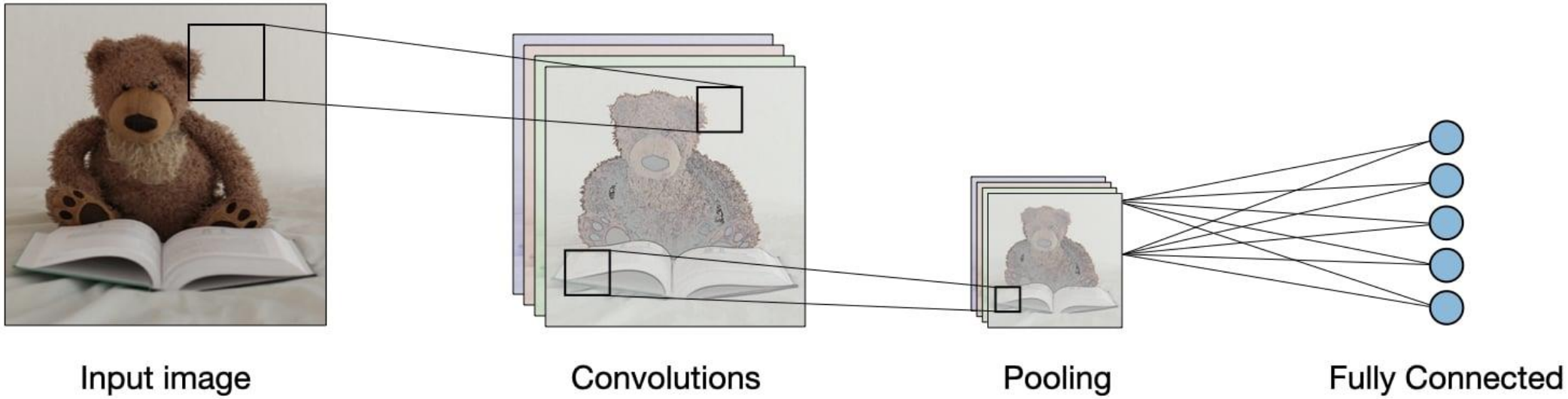
$a(\mathrm{x}_1) \longrightarrow$ Model $\longrightarrow \mathrm{u}(\mathrm{x}_1)$

Test

$a(\mathrm{x}_\mathrm{L}) \longrightarrow$ Model $\longrightarrow \mathrm{u}(\mathrm{x}_\mathrm{L})$

Neural Operator: Learning Maps Between Function Spaces With Applications to PDEs. Nikola et al.

Methods Function on Grid is not Discretization Invariant



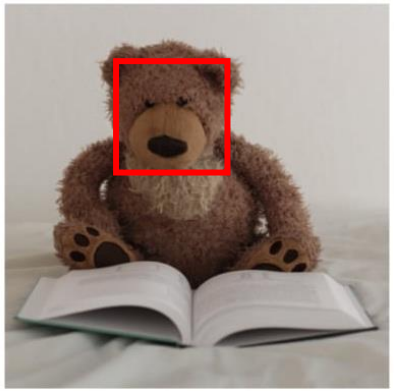


Image source: https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks/

Key Idea: Learn a Point-wise Kernel

$$\int \kappa_\phi(x, y)\nu(y)dy$$

$\kappa_\phi(x, y)$ denotes a kernel function that takes as input a pair of points (x, y) and outputs a scalar

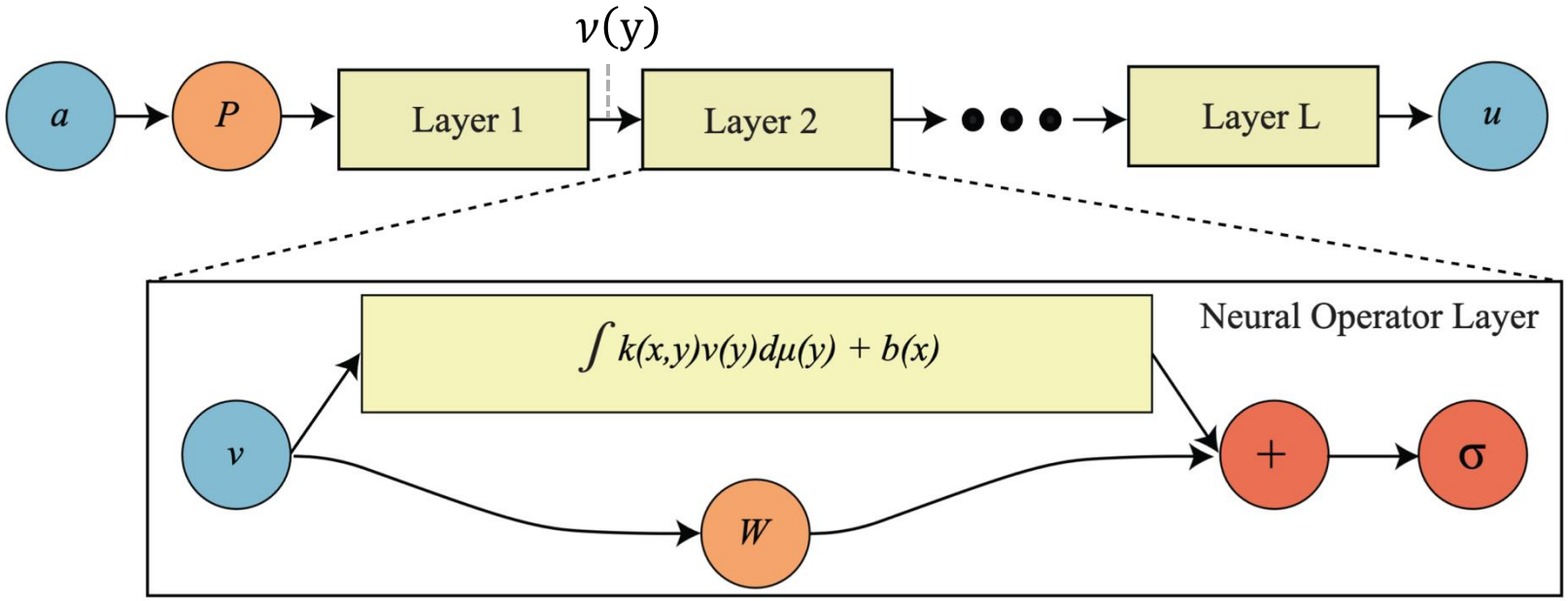
How to ensure discretization invariance?

If κ_ϕ is a continuous kernel, optimizing it with subsampled x_l fits κ_ϕ on the entire "continuous" domain of x

$\kappa_\phi(x, y)\nu(y)$ aggregates features from every point y for the query point x

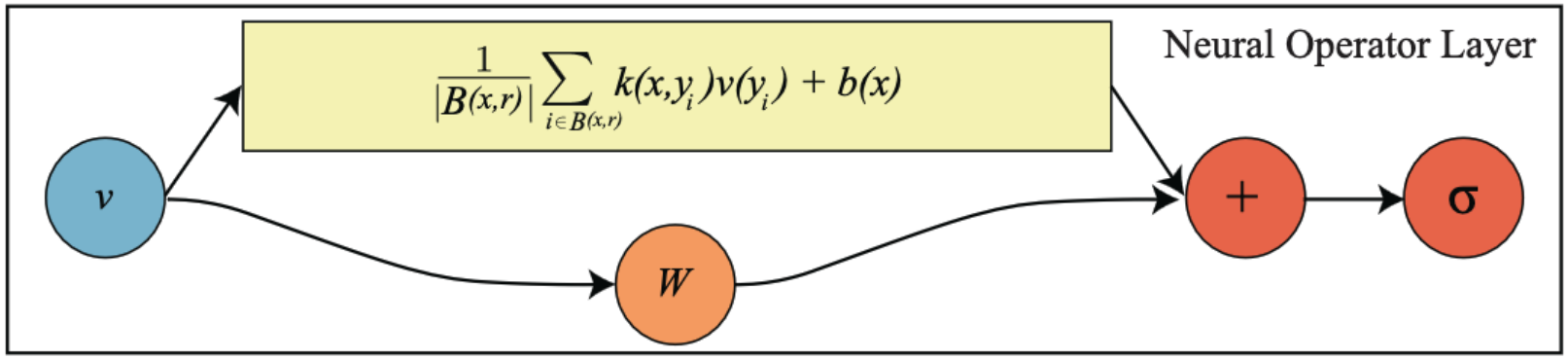
Neural Operator: Learning Maps Between Function Spaces With Applications to PDEs. Nikola et al.

The Diagram of Neural Operator



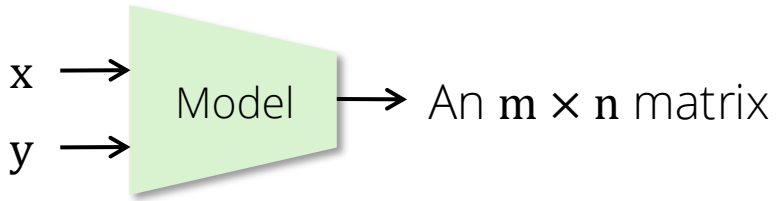
Neural Operator: Learning Maps Between Function Spaces With Applications to PDEs. Nikola et al.

Choice 1: A Shallow Neural Layer



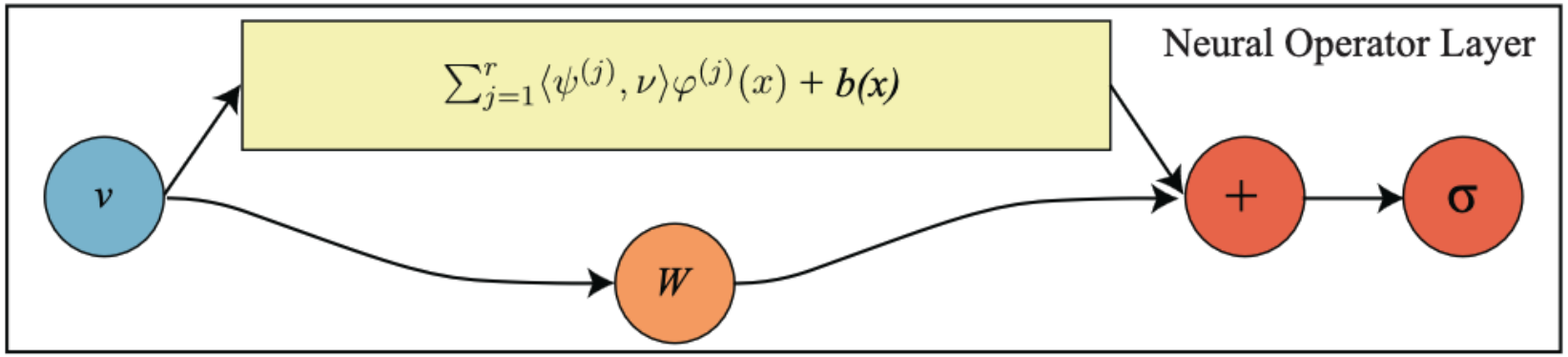
- Let $\mathbf{n}$ be the input channel dimension, $\mathbf{m}$ be the output channel dimensions, and $\mathbf{d}$ be the coordinate dimension of point $\mathbf{x}, \mathbf{y}$
- Define $\boldsymbol{\kappa_\phi(x, y)}$ as a kernel matrix $\mathbf{K} \in \mathbb{R}^{\mathbf{md \times nd}}$

$$\boldsymbol{\kappa_\phi(x_i, x_j) = K_{ij} \in \mathbb{R}^{m \times n}}$$



Neural Operator: Learning Maps Between Function Spaces With Applications to PDEs. Nikola et al.

Choice 2: Tensor Product Factorization

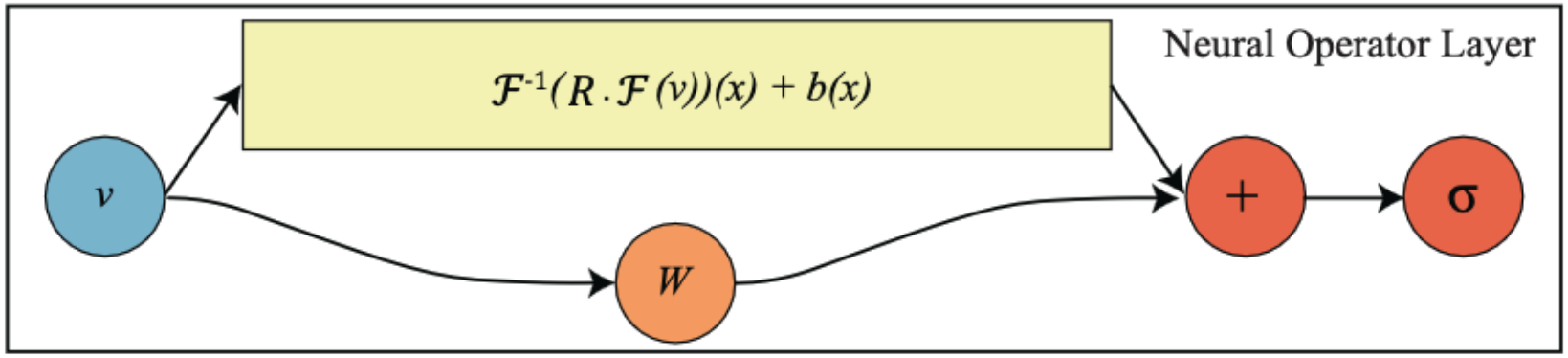


- Let $\varphi^{(j)}: \mathbb{R}^d \to \mathbb{R}^m$ and $\psi^{(j)}: \mathbb{R}^d \to \mathbb{R}^n$
- Define $\kappa_\phi(x, y)$ as a summation of tensor products

$$\kappa_\phi(x, y) = \sum_j \varphi^{(j)}(x) \otimes \psi^{(j)}(y)$$

Neural Operator: Learning Maps Between Function Spaces With Applications to PDEs. Nikola et al.

Choice 3: Spectral Factorization



- Define $\kappa_\phi(x, y)$ in the Fourier domain R_ϕ

Neural Operator: Learning Maps Between Function Spaces With Applications to PDEs. Nikola et al.

Choice 3: Spectral Factorization

- Let's first assume $\kappa_\phi(x, y)$ is translation invariant:

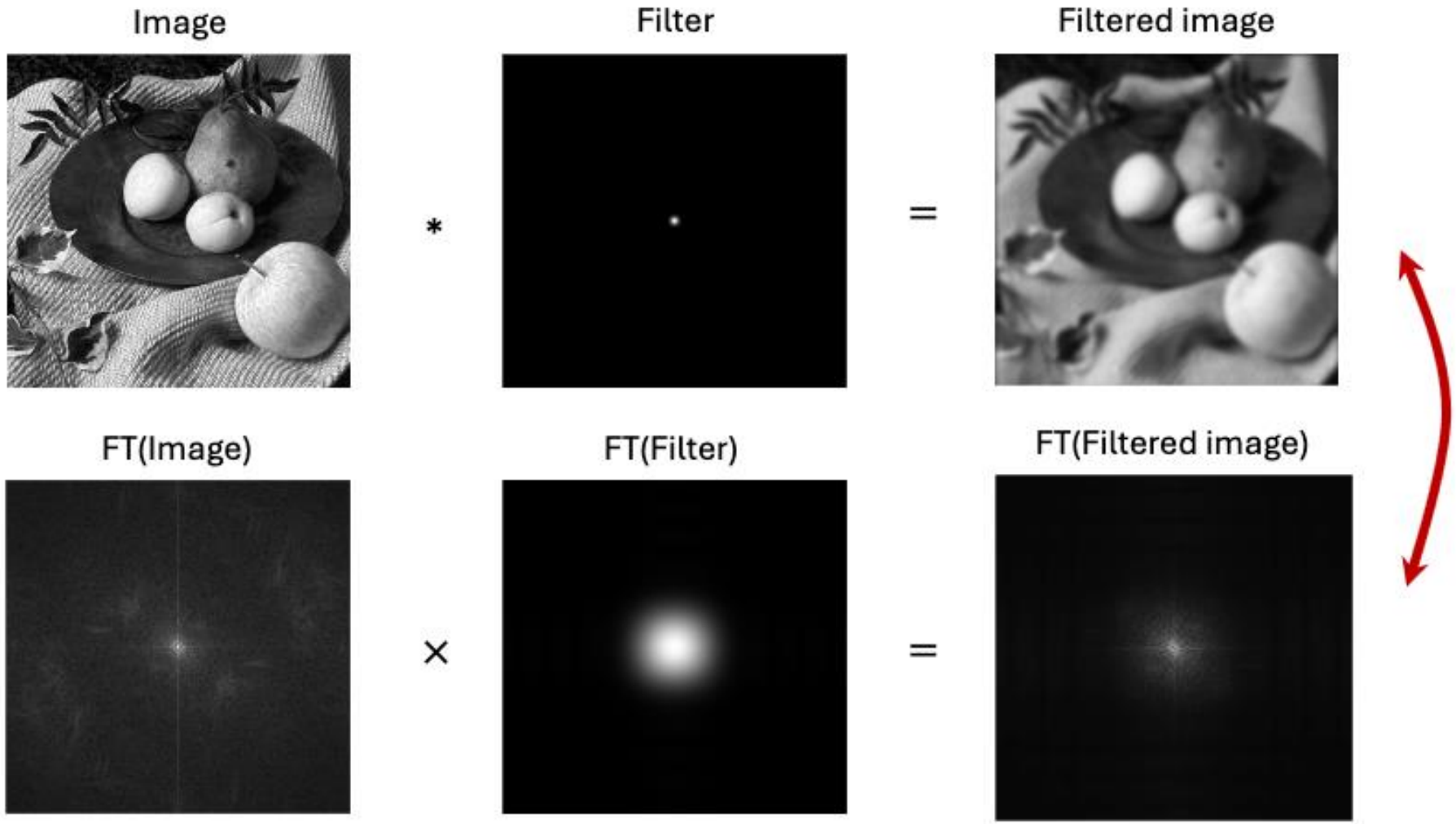
$$\kappa_\phi(x, y) = \kappa_\phi(x - y)$$

- Therefore, the kernel operation is equivalent to convolutional operation

$$u(x) = \int \kappa_\phi(x, y)\nu(y)dy = \int \kappa_\phi(x - y)\nu(y)dy = (\kappa * \nu)(x)$$

- By convolution theorem, the Fourier transform of a convolution of two functions is the product of their Fourier transforms

2D convolution theorem example



Slide credit D.A. Forsyth

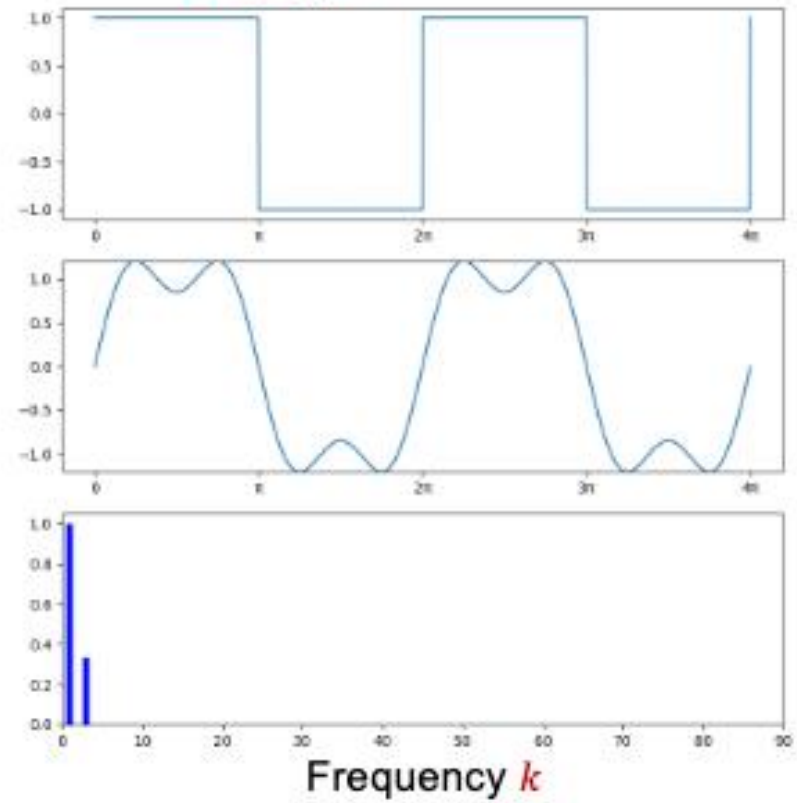
Fourier analysis

- Any “reasonable” univariate function can be expressed as a weighted sum of sinusoids of different frequencies (1807)

Jean-Baptiste Joseph Fourier (1768-1830)

Example: series for a square wave

$$\sum_{k=1,3,5,\ldots}^{\infty} \frac{1}{k}\sin(kt)$$



1D Fourier transform

- Let's define basis functions parameterized by $u \in (-\infty, \infty)$:

$$\psi_u(t) = e^{i2\pi ut}$$

- Inner product for complex functions is given by:

$$\langle f, g \rangle = \int_{-\infty}^{\infty} f(t) g^*(t) dt$$

- Orthonormality:

$$\langle \psi_{u_1}, \psi_{u_2} \rangle = \begin{cases} 1 & \text{if } u_1 = u_2 \\ 0 & \text{otherwise} \end{cases}$$

Slide credit S. Lazebnik

1D Fourier transform

- Given a signal $f(t)$, we want to represent it as a weighted combination of the basis functions $\psi_u(t) = e^{i2\pi ut}$ with weights $F(u)$:

$$f(t) = \int_{-\infty}^{\infty} F(u) e^{i2\pi ut} du$$

- Each weight $F(u)$ is given by the inner product of f and ψ_u:

$$F(u) = \langle f, \psi_u \rangle = \int_{-\infty}^{\infty} f(t) e^{-i2\pi ut} dt$$

Slide credit S. Lazebnik

1D Fourier transform

- Forward transform: $f(t) \xrightarrow{\mathcal{F}} F(u)$

$$F(u) = \int_{-\infty}^{\infty} f(t) e^{-i2\pi ut} dt$$

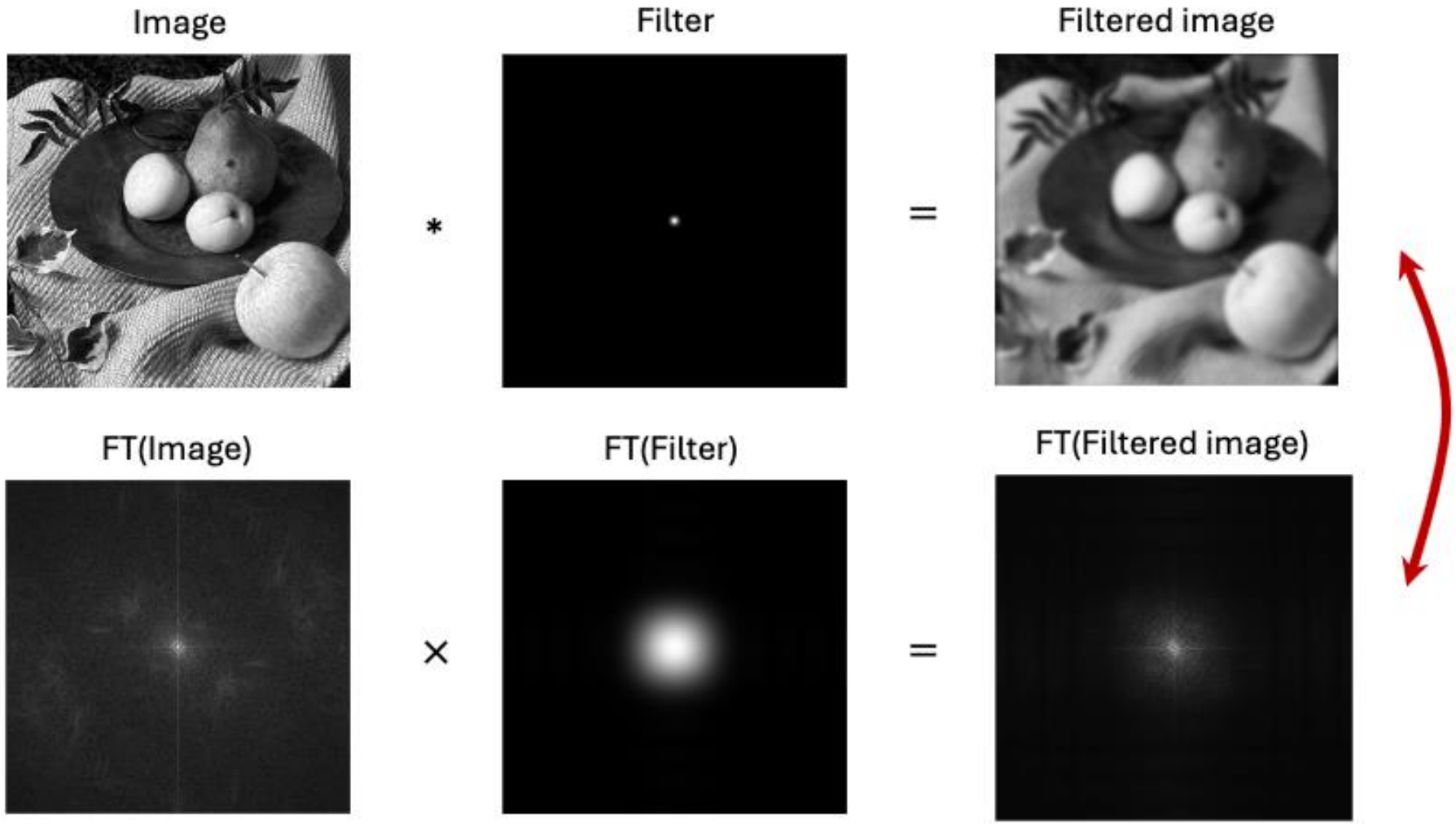
- Inverse transform: $F(u) \xrightarrow{\mathcal{F}^{-1}} f(t)$

$$f(t) = \int_{-\infty}^{\infty} F(u) e^{i2\pi ut} du$$

- Duality: if $f(t) \xrightarrow{\mathcal{F}} F(u)$, then $F(t) \xrightarrow{\mathcal{F}} f(-u)$
  - Thus, we can talk about *Fourier transform pairs* $f(t) \leftrightarrow F(u)$

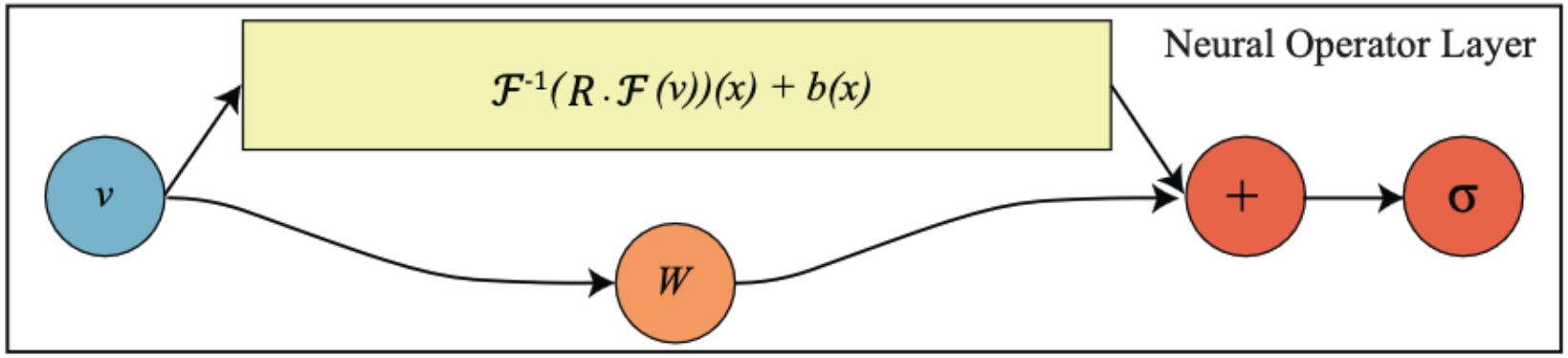
Slide credit S. Lazebnik

2D convolution theorem example



Slide credit D.A. Forsyth

Choice 3: Spectral Factorization



- Define $\kappa_\phi(x, y)$ in the Fourier domain R_ϕ
- Therefore

$$u(x) = (\kappa * v)(x) = \mathcal{F}^{-1}\left(R_\phi \cdot \mathcal{F}(v)\right)(x)$$

Why spectral factorization? We can make the computation very efficient by truncating to few lowest-frequency modes

Neural Operator: Learning Maps Between Function Spaces With Applications to PDEs. Nikola et al.

1D Fourier transform

- Forward transform: $f(t) \xrightarrow{\mathcal{F}} F(u)$

$$F(u) = \int_{-\infty}^{\infty} f(t) e^{-i2\pi ut} dt$$

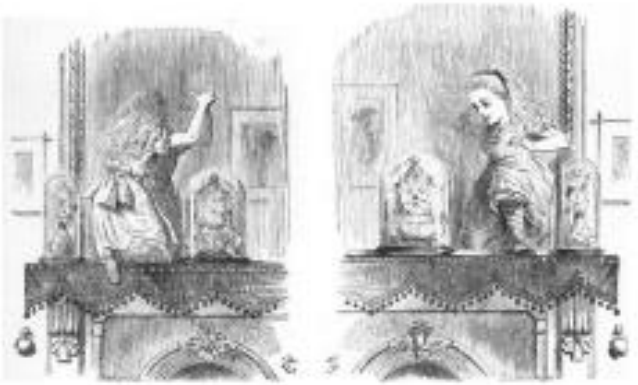
$-k_{max} \leq u \leq k_{max}$

- Inverse transform: $F(u) \xrightarrow{\mathcal{F}^{-1}} f(t)$

$$f(t) = \int_{-\infty}^{\infty} F(u) e^{i2\pi ut} du$$

k_{max}

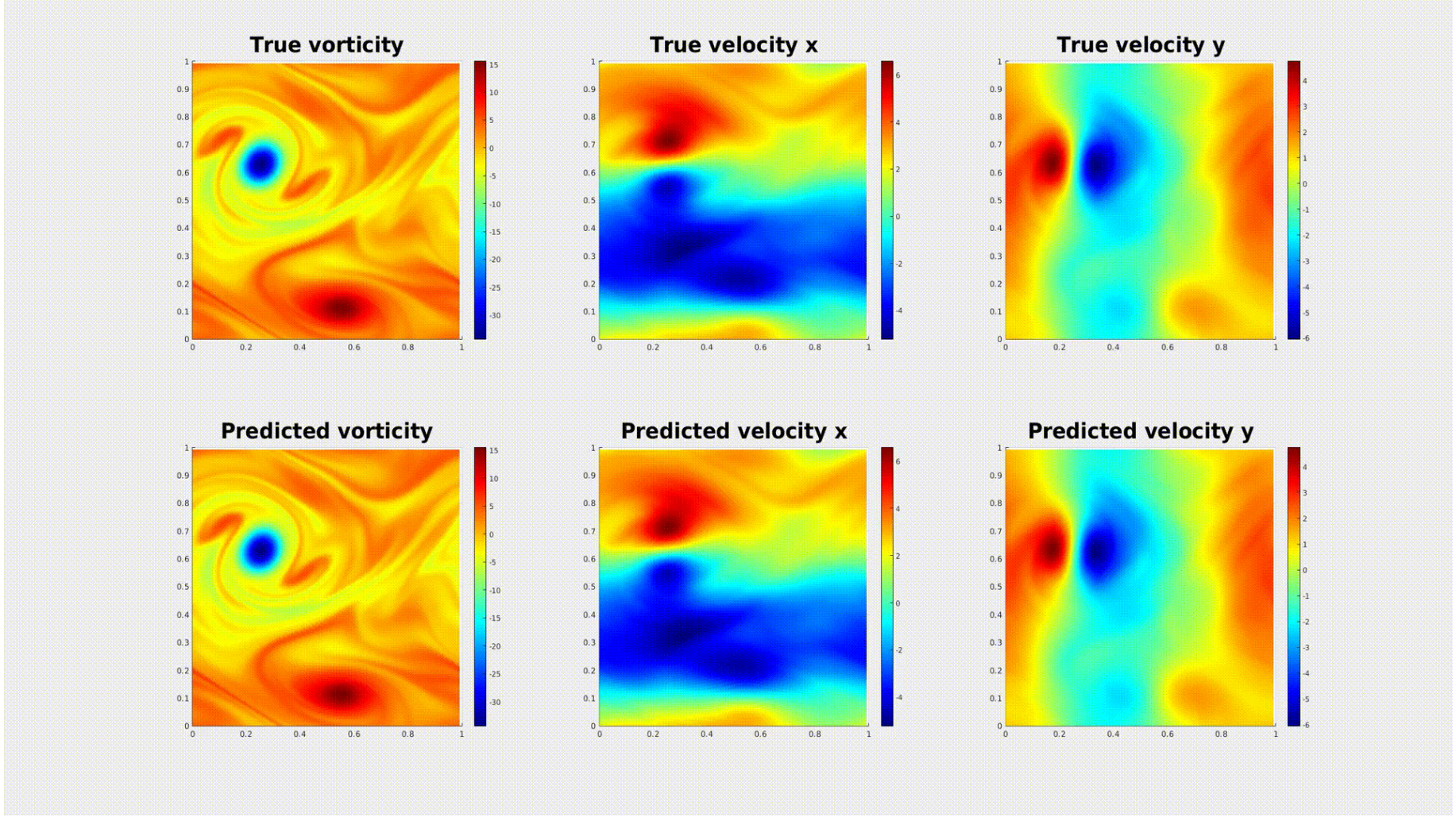
$-k_{max}$



- Duality: if $f(t) \xrightarrow{\mathcal{F}} F(u)$, then $F(t) \xrightarrow{\mathcal{F}} f(-u)$
  - Thus, we can talk about *Fourier transform pairs* $f(t) \leftrightarrow F(u)$

Slide credit S. Lazebnik

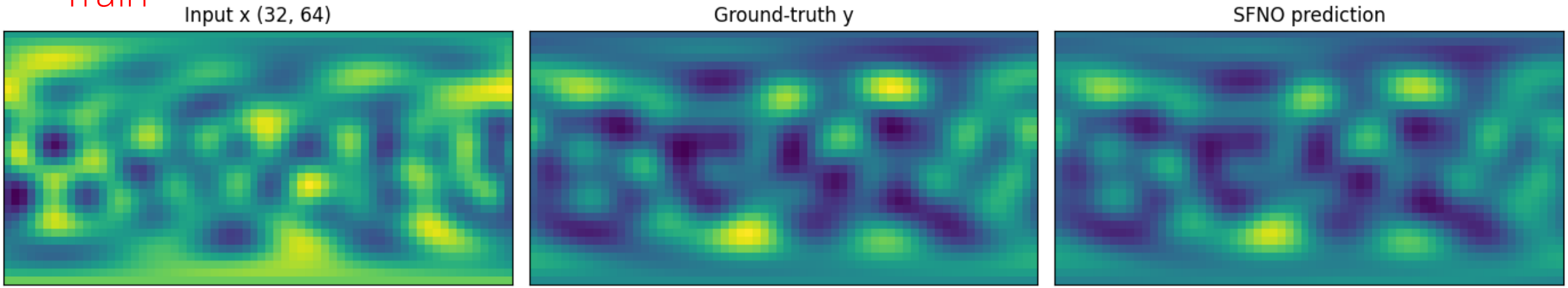
Results



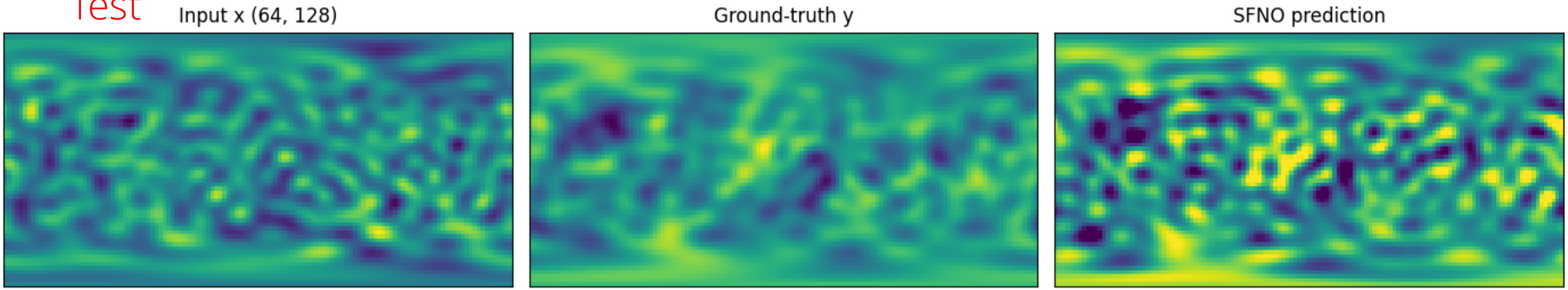
https://zongyi-li.github.io/neural-operator/

SFNO predictions on spherical shallow water equations

Train

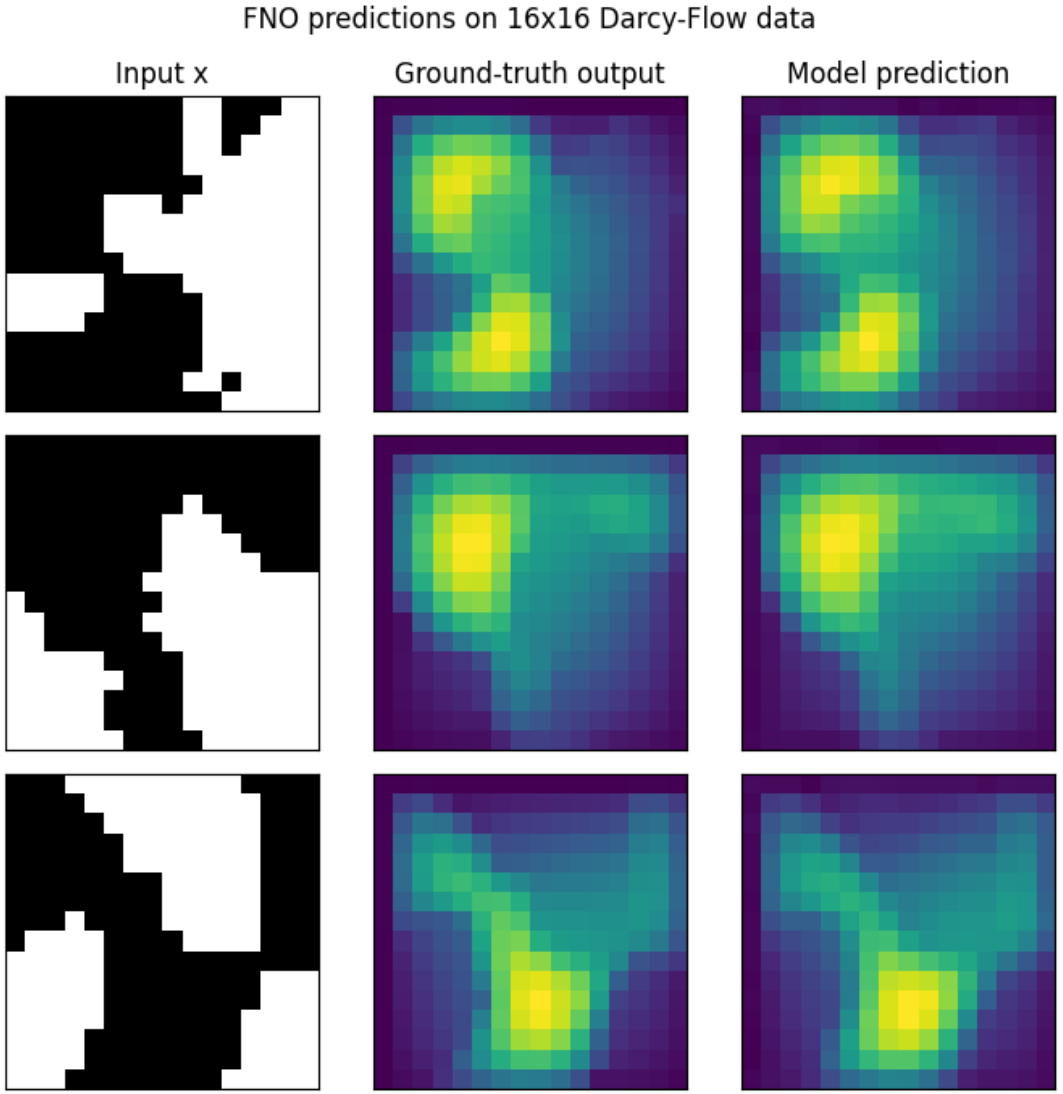


Test

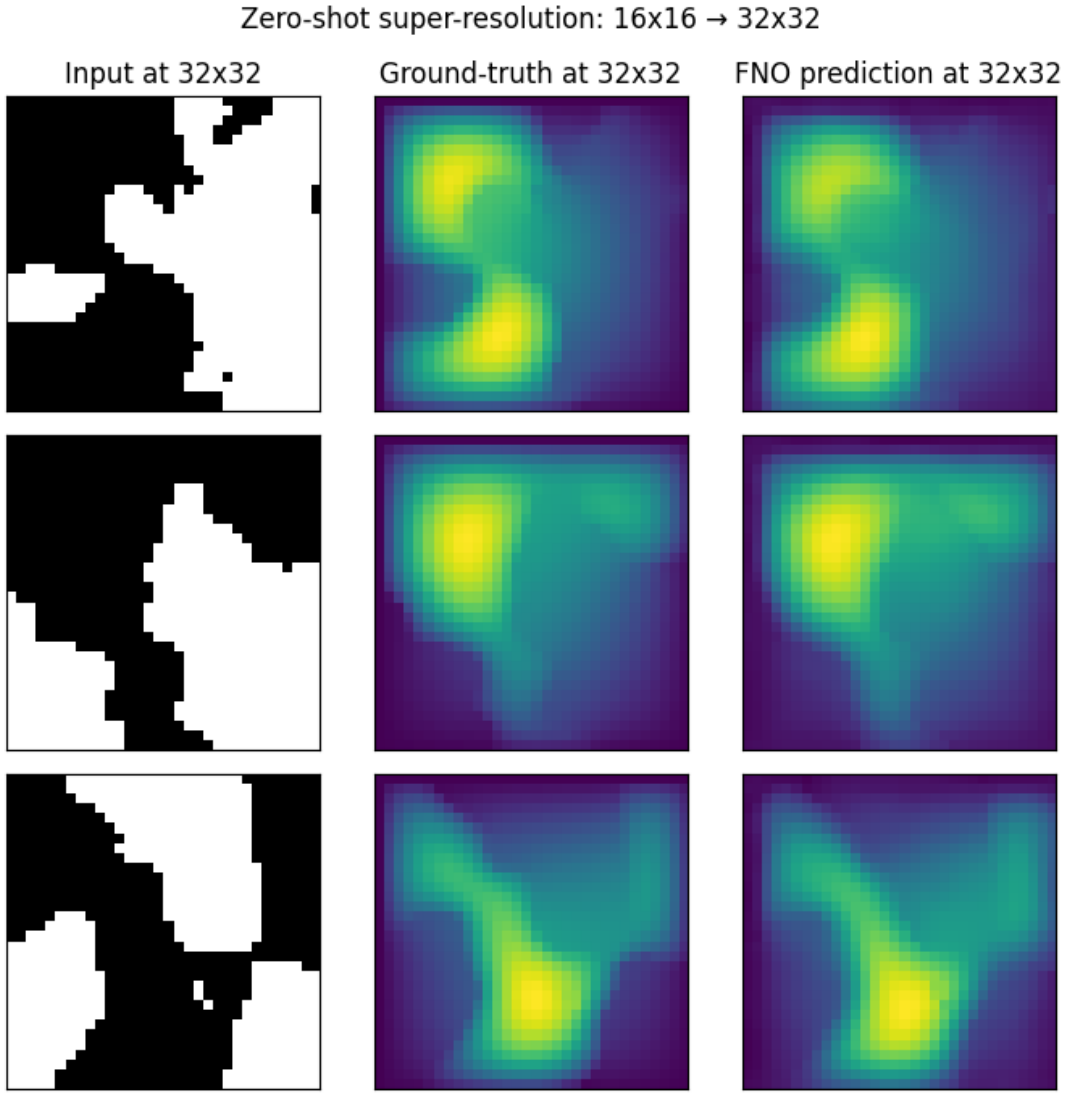


https://neuraloperator.github.io/dev/auto_examples/models/plot_SFNO_swe.html

Train



Test



https://neuraloperator.github.io/dev/auto_examples/models/plot_FNO_darcy.html

Quick Summary

- To model the physical world, we've cover
  - 3D/4D modeling
  - Generative modeling
  - Rigid-body motion
  - Particle dynamics and mass-spring systems
  - Lagrangian dynamics and Incremental Potential Contact
  - Continuum dynamics and Material Point Method
  - Learning-based dynamics

What We Will Cover the Next Week

- Embodied perception
  - Perceptive Action Decision
  - Multi-sensory Perception
  - Visual language model for embodied perception