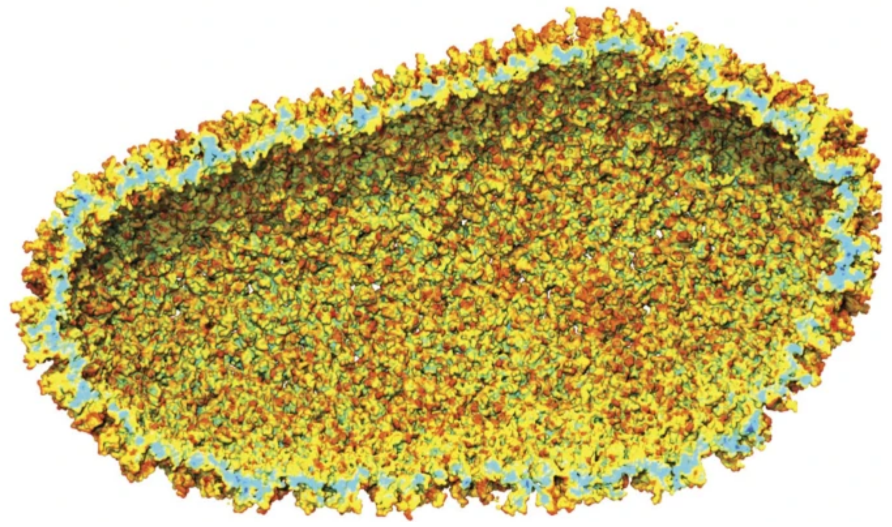
Neural Functional: Learning Function to Scalar Maps for Neural PDE Surrogates

Anthony Zhou, Amir Barati Farimani, 6/20/2025

Motivation - Accelerating Physics Simulation

HIV Capsid

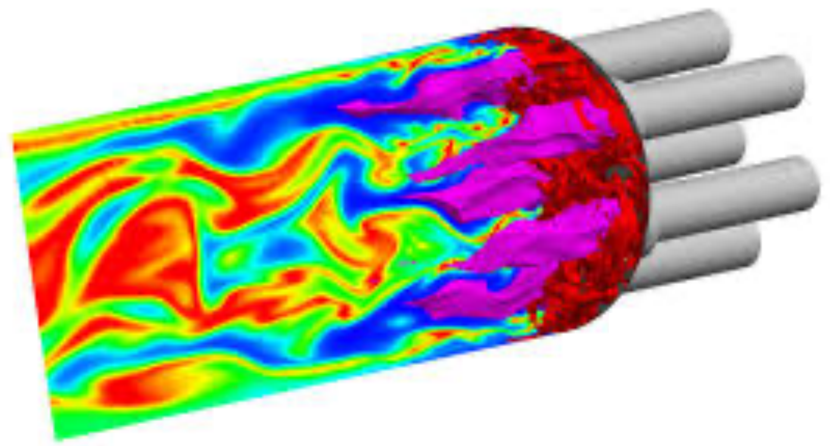


64 Million Atoms

3880 GPU Nodes

~130 days of simulation

Liquid Rocket Engine

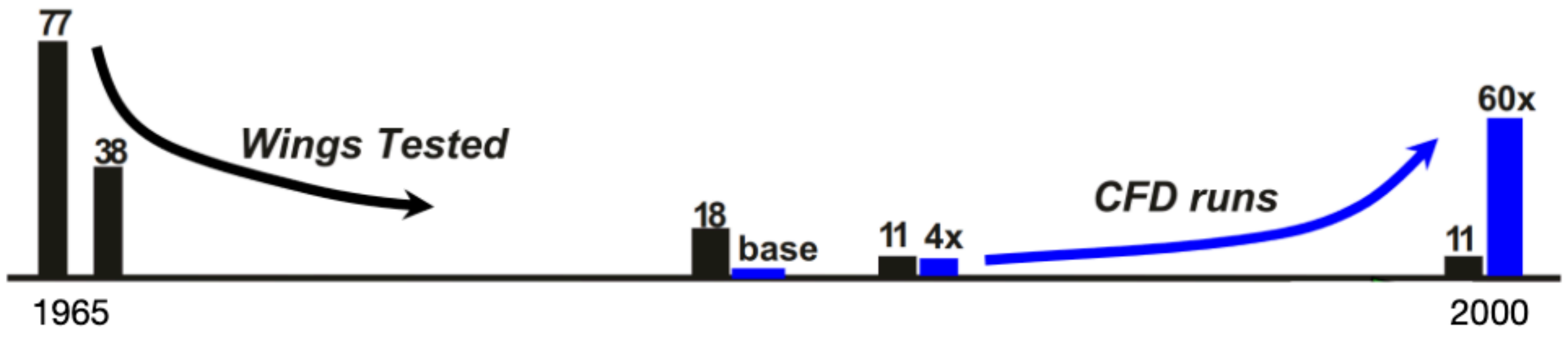


350 Million Cells

4000 CPU Nodes

~14 days of simulation

Physics simulation allows us to understand and design for the world



Analysis Timelines at Boeing

747: 4 Years

767: 3 Years

777: <1 Year

Faster simulations can accelerate scientific discovery and engineering progress

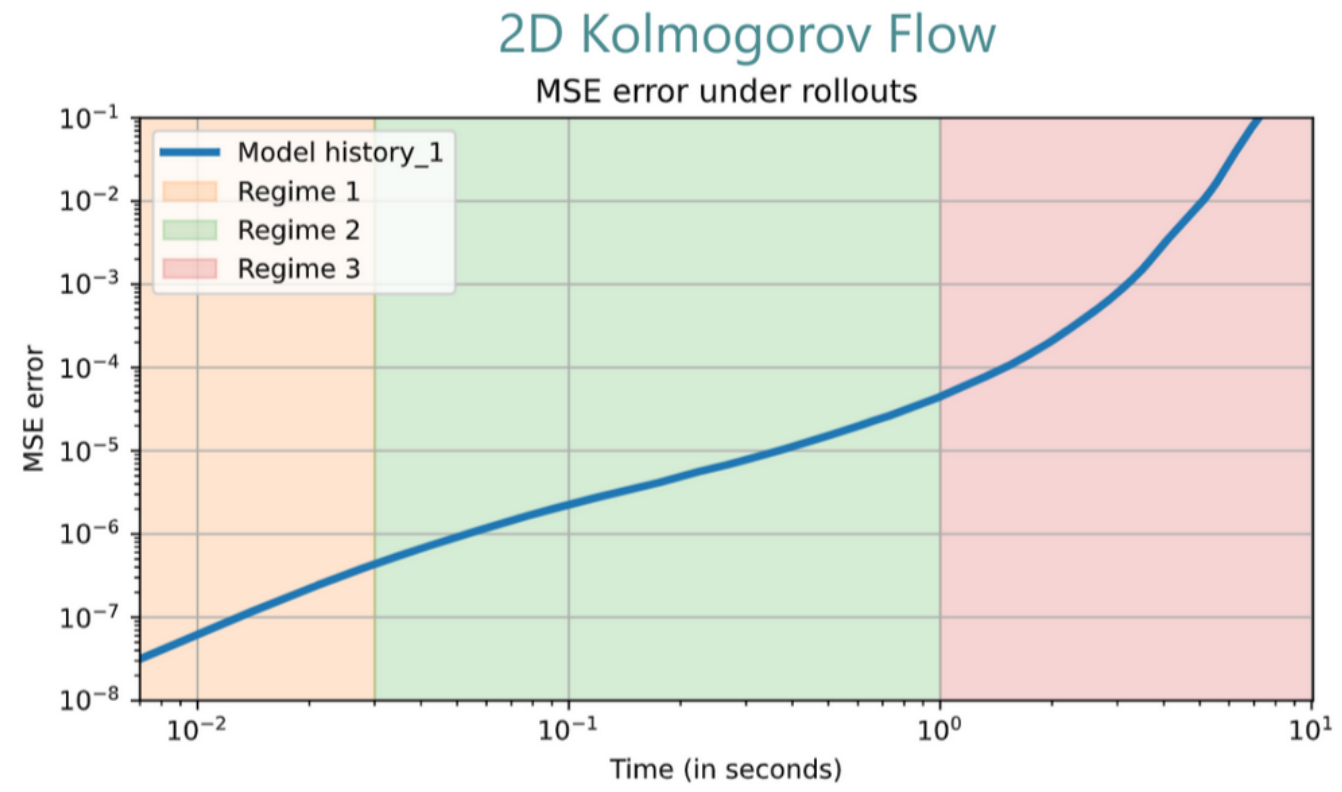
https://www.nas.nasa.gov/SC14/demos/demo20.html, Perilla, J., Schulten, K. Physical properties of the HIV-1 capsid from all-atom molecular dynamics simulations. *Nat Commun* **8**, 15959 (2017). https://doi.org/10.1038/ncomms15959

Forrester T. Johnson, Edward N. Tinoco, N. Jong Yu, Thirty years of development and application of CFD at Boeing Commercial Airplanes, Seattle, Computers & Fluids, Volume 34, Issue 10, 2005, Pages 1115-1151, ISSN 0045-7930, https://doi.org/10.1016/j.compfluid.2004.06.005

Technical Motivation - Physical Constraints in PDE Surrogates

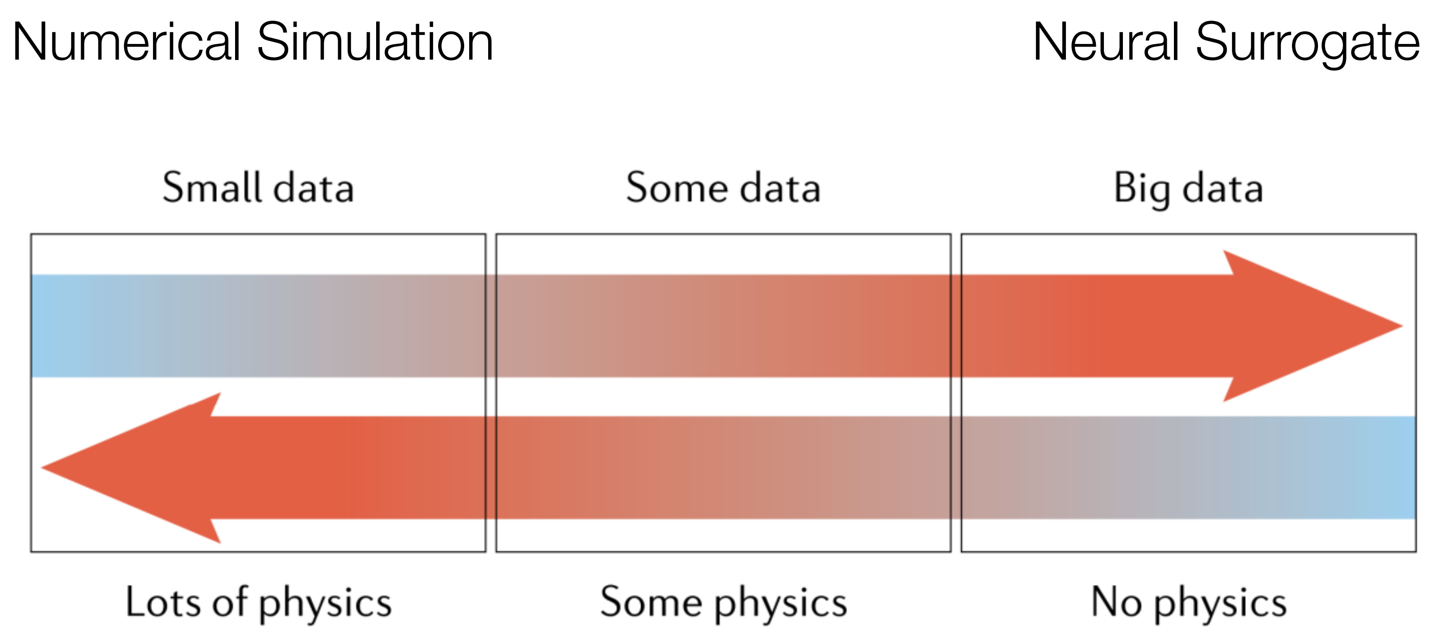
Problem

- Neural PDE surrogates make purely data-driven predictions
- Predictions usually become unstable due to lack of physical grounding



Physical Grounding

- More efficient: Inductive biases can lead to faster training and fewer parameters
- More stable: Ensure predictions are physically consistent and obey known principles
- More generalizable: Different systems can have the same physical priors



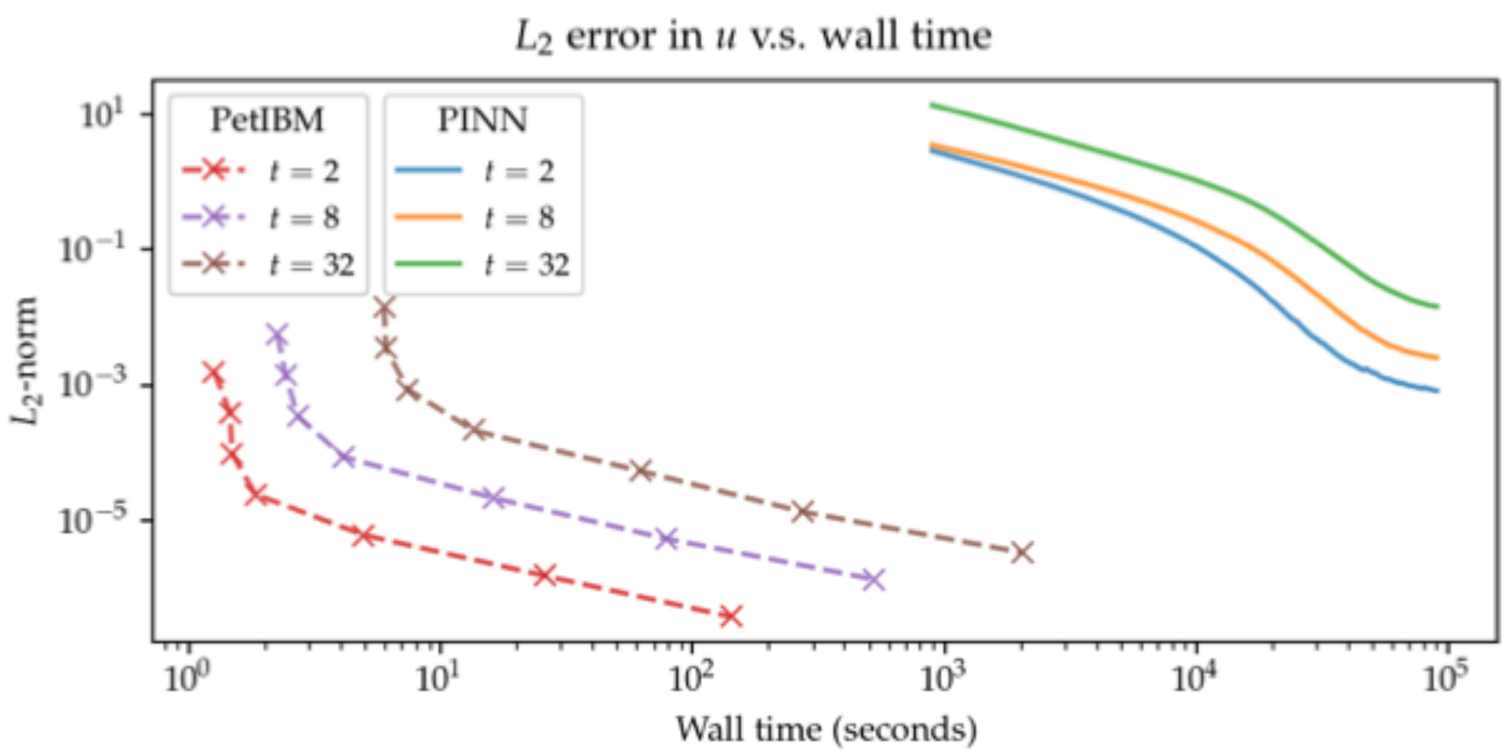
Phillip Lippe, Microsoft AI4Science, https://drive.google.com/file/d/1Qk7hX1lnUmFwr9wk0HuWmB-OJ9llbAMG/view, Karniadakis, G.E., Kevrekidis, I.G., Lu, L. *et al.* Physics-informed machine learning. *Nat Rev Phys* **3**, 422–440 (2021). https://doi.org/10.1038/s42254-021-00314-5

What are prior methods for embedding physical priors?

Physics-Informed Losses

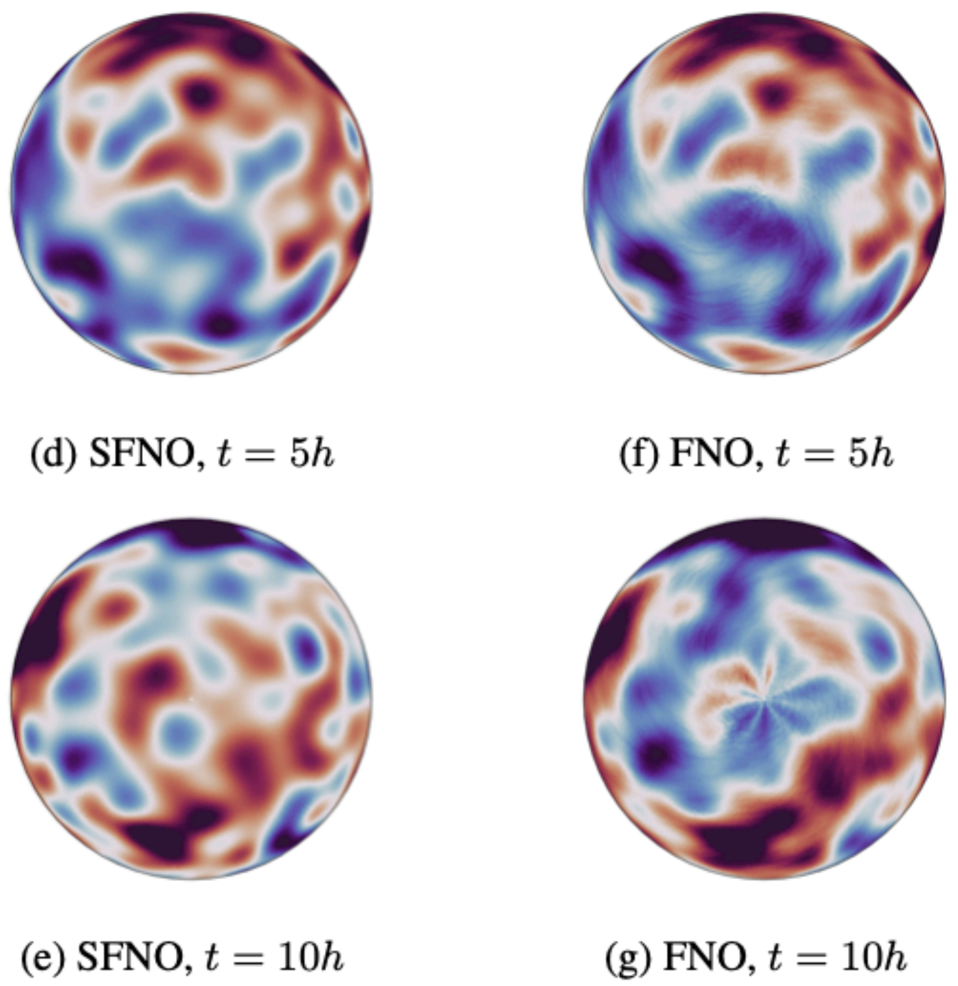
- Degenerate for complex systems
- Challenging to optimize in practice

$$\mathcal{P}(u,a)=0, \qquad \text{in } D\subset\mathbb{R}^d$$
$$u=g, \qquad \text{in } \partial D$$
$$\mathcal{L}_{\text{pde}}(a,u_\theta)=\left\|\mathcal{P}(a,u_\theta)\right\|^2_{L^2(D)}+\alpha\left\|u_\theta|_{\partial D}-g\right\|^2_{L^2(\partial D)}$$



Equivariant/Invariant NNs

- Challenging to design/implement
- Work well for targeted systems



Subset

Hamiltonian NNs

- Only shown for simple, particle systems

How do we extend these to PDE systems?

Chuang, P.-Y., Barba, L.A.: Experience report of physics-informed neural networks in fluid simulations: pitfalls and frustration. arXiv preprint arXiv:2205.14249 (2022); https://greydanus.github.io/2019/05/15/hamiltonian-nns/
Boris Bonev, Thorsten Kurth, Christian Hundt, Jaideep Pathak, Maximilian Baust, Karthik Kashinath, Anima Anandkumar, Spherical Fourier Neural Operators: Learning Stable Dynamics on the Sphere, https://arxiv.org/abs/2306.03838

Background - What is Hamiltonian Mechanics?

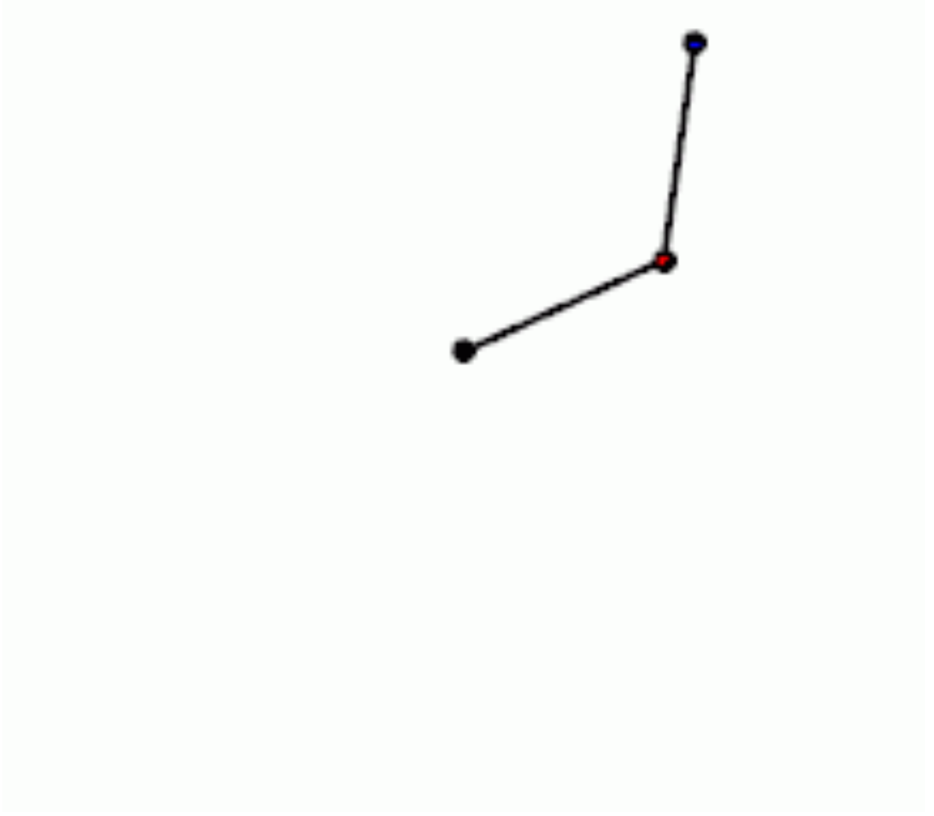
Hamiltonian Mechanics is a way of interpreting/deriving physics

Why does the world evolve in the way it does?

Newton: Physics occurs as a result of forces (Newton's Laws)

Lagrange/Hamilton: Physics occurs because it minimizes the universe's action

Double Pendulum



Newtonian Mechanics: Write force balance and simplify

$$\begin{aligned} m_1 l_1\left(\ddot{\theta}_1 \cos\theta_1 - \dot{\theta}_1^2 \sin\theta_1\right) &= -T_1 \sin\theta_1 + T_2 \sin\theta_2 \\ -m_1 l_1\left(\ddot{\theta}_1 \sin\theta_1 + \dot{\theta}_1^2 \cos\theta_1\right) &= -T_1 \cos\theta_1 + T_2 \cos\theta_2 + m_1 g \end{aligned}$$

$$\begin{aligned} m_2\left(l_1\ddot{\theta}_1 \cos\theta_1 - l_1\dot{\theta}_1^2 \sin\theta_1 + l_2\ddot{\theta}_2 \cos\theta_1 - l_2\dot{\theta}_2^2 \sin\theta_2\right) &= -T_2 \sin\theta_2 \\ -m_2\left(l_1\ddot{\theta}_1 \sin\theta_1 + l_1\dot{\theta}_1^2 \cos\theta_1 + l_2\ddot{\theta}_2 \sin\theta_2 + l_2\dot{\theta}_2^2 \cos\theta_2\right) &= -T_2 \cos\theta_2 + m_2 g. \end{aligned}$$

… plug and chug

Hamiltonian Mechanics: Write total energy and minimize

$$H = \frac{m_2 l_2^2 p_{\theta_1}^2 + (m_1+m_2) l_1^2 p_{\theta_2}^2 - 2 m_2 l_1 l_2 p_{\theta_1} p_{\theta_2} \cos(\theta_1 - \theta_2)}{2 m_2 l_1^2 l_2^2 \left[m_1 + m_2 \sin^2(\theta_1 - \theta_2)\right]} - (m_1+m_2) g l_1 \cos\theta_1 - m_2 g l_2 \cos\theta_2$$

$$\dot{\theta}_i = \frac{\partial H}{\partial p_{\theta_i}} \qquad \dot{p}_{\theta_i} = -\frac{\partial H}{\partial \theta_i}$$

Minima found with Hamilton's Equations of Motion

A simpler and more universal framework for physics

https://www.phys.lsu.edu/faculty/gonzalez/Teaching/Phys7221/DoublePendulum.pdf

How is Hamiltonian Mechanics useful for learning physics?

Original problem: Neural networks lack physical grounding.

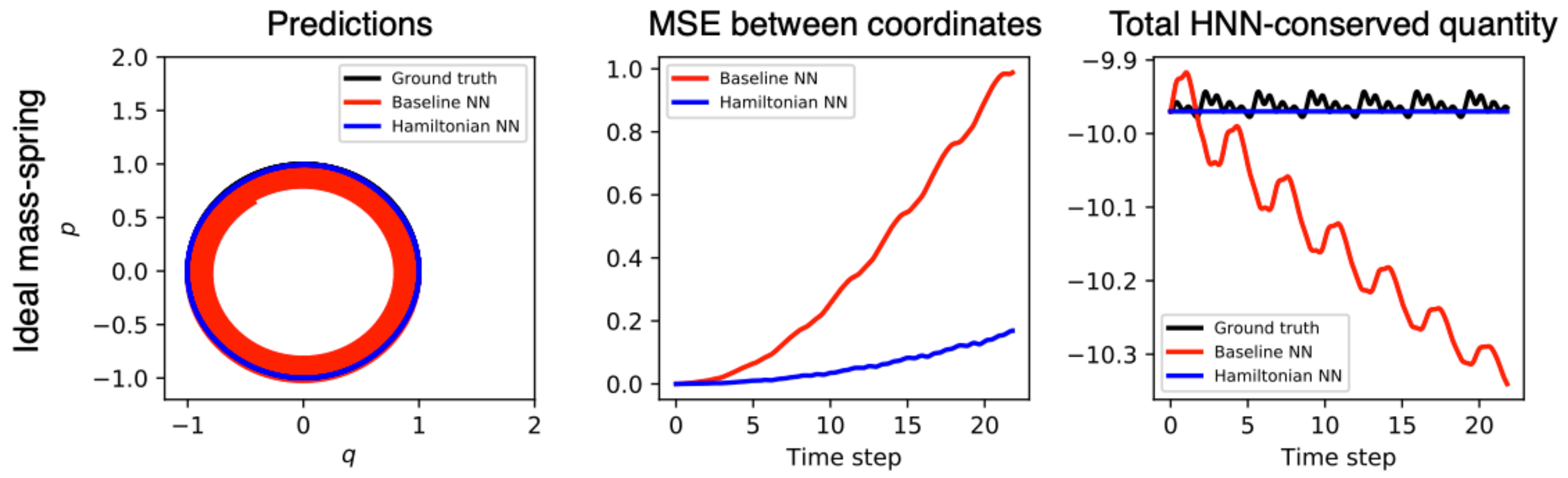
Solution (HNNs): Apply neural networks in a Hamiltonian framework.

Recall: Hamiltonian mechanics relies on an energy and its derivatives

$$\frac{d\mathbf{q}}{dt} = \frac{\partial \mathcal{H}}{\partial \mathbf{p}}, \quad \frac{d\mathbf{p}}{dt} = -\frac{\partial \mathcal{H}}{\partial \mathbf{q}}.$$

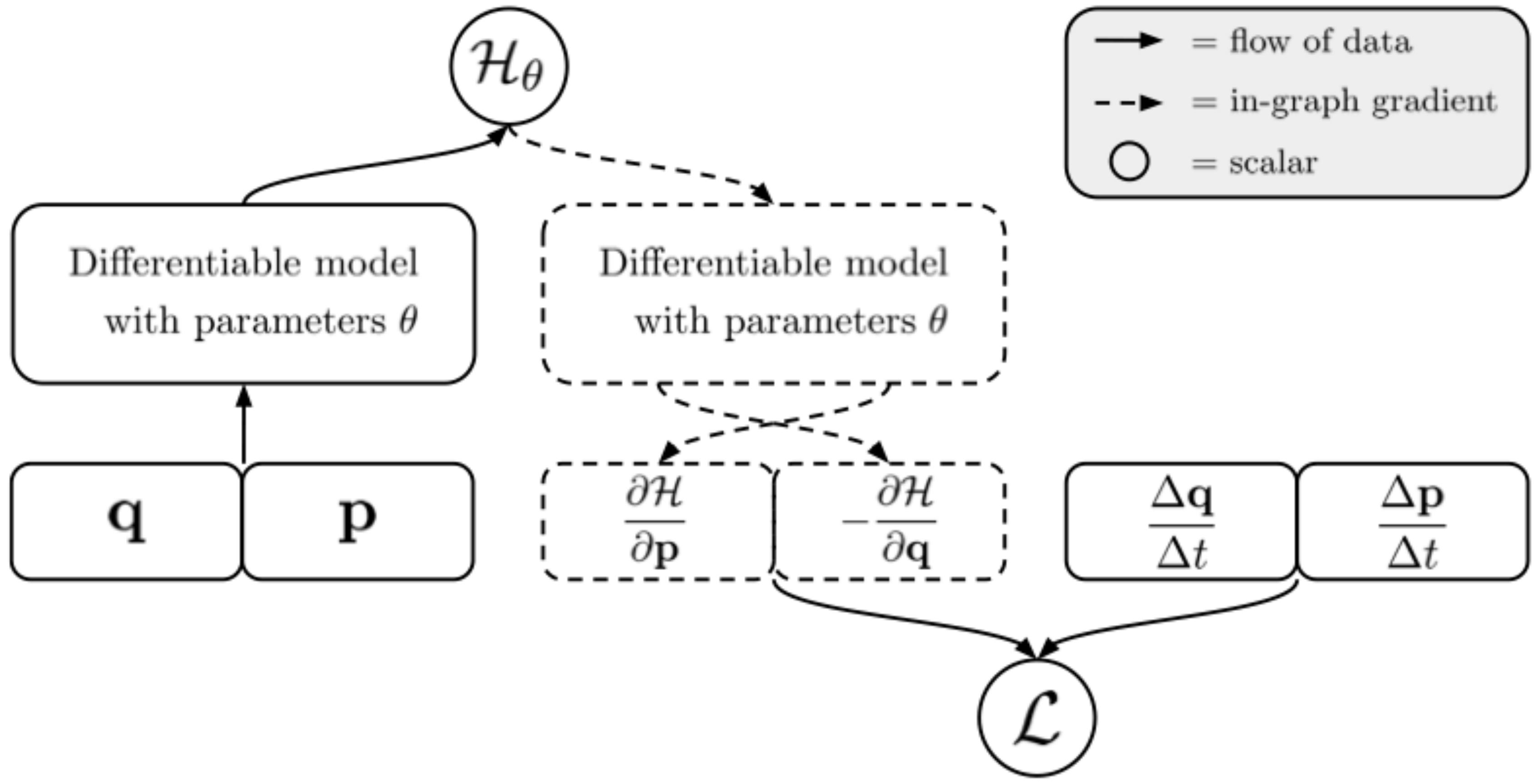
Define Hamiltonian (H); Update position and momentum by differentiating the Hamiltonian

Result: Models trained in this manner tend to conserve energy*



*Reasons for why this happens are still not well-understood

In Machine Learning: Learn H with neural network, find derivatives w/ autograd

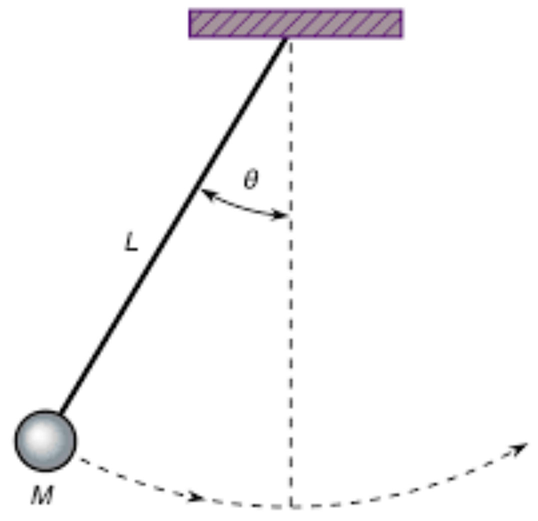


- Model inputs are position/momentum. Model output is a scalar.
- Model output is differentiated w.r.t. inputs to get dynamical information.
- During training this is optimized, during inference this is used for prediction.

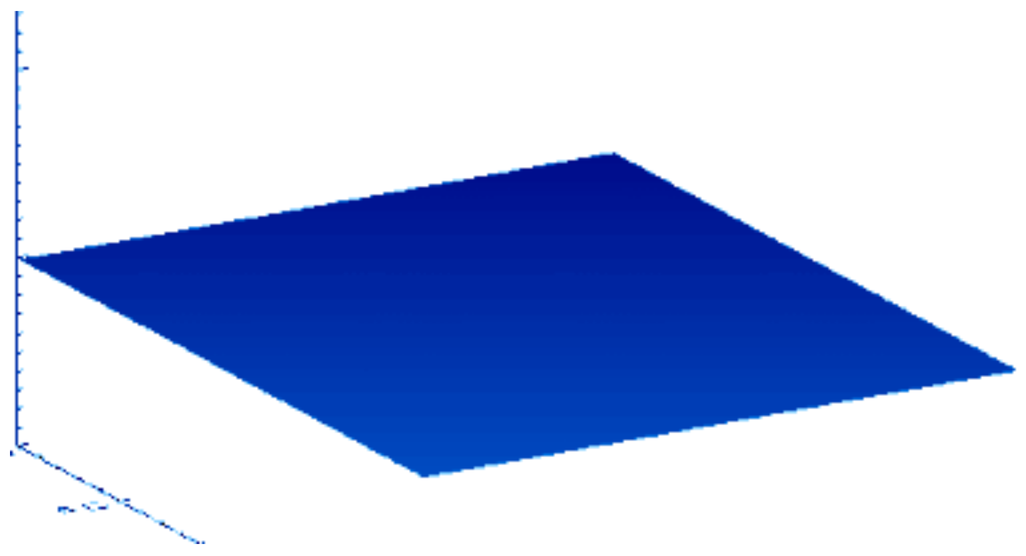
Sam Greydanus, Misko Dzamba, Jason Yosinski, Hamiltonian Neural Networks, https://arxiv.org/abs/1906.01563

How can we extend this to more complex, PDE systems?

Current limitations: Only applicable to particle systems. Most studies work with analytically-solved systems.



How to extend?



Discrete particles, simple physics

Infinite particles, complex physics

To extend to PDE systems, we make two key observations:

1. PDE systems can have Hamiltonian formulations and conserve energy (Infinite-Dimensional Hamiltonian Mechanics).

2. New architectures are needed to model infinite-dimensional Hamiltonians.

Infinite-Dimensional Hamiltonian Mechanics

Finite	Vectors	Functions	Gradient	Hamilton's Equations
	$\mathbf{q}, \mathbf{p} \in \mathbb{R}^n$	$\mathcal{H}(\mathbf{q}, \mathbf{p}) : \mathbb{R}^{2n} \rightarrow \mathbb{R}$	$\nabla_{\mathbf{u}} \mathcal{H} = [\frac{\partial \mathcal{H}}{\partial u_1}, \ldots, \frac{\partial \mathcal{H}}{\partial u_{2n}}]$	$\frac{\mathbf{du}}{\mathbf{d}t} = \begin{bmatrix} 0 & \mathbb{I} \\ -\mathbb{I} & 0 \end{bmatrix} \begin{bmatrix} \partial \mathcal{H} / \partial \mathbf{q} \\ \partial \mathcal{H} / \partial \mathbf{p} \end{bmatrix} = J \nabla_{\mathbf{u}} \mathcal{H}$
	↓	↓	↓	↓
	$u \in \mathcal{F}(\Omega)$	$\mathcal{H}[u] : \mathcal{F}(\Omega) \rightarrow \mathbb{R}$	$\int_\Omega \frac{\delta \mathcal{H}}{\delta u}(x) \epsilon(x) \mathrm{d}x = \lim_{h \rightarrow 0} \frac{\mathcal{H}[u + h\epsilon] - \mathcal{H}[u]}{h}$	$\frac{\partial u}{\partial t} = \mathcal{J} \frac{\delta \mathcal{H}}{\delta u}$
Infinite	Functions	Functionals	Functional Derivative	Hamilton's Equations

To move from finite to infinite dimensions, there are 4 changes:

1. Inputs change from vectors to functions
2. Hamiltonian changes from a function to a functional
3. Gradients become functional derivatives
4. Hamilton's equations of motion are modified

This gives us tools to analyze PDE systems but…

1. No current architectures can theoretically approximate functionals (function to scalar mappings)
2. Derivatives of neural networks are gradients; how do you use autograd to calculate a functional derivative? Is this even well-defined?

How do we learn functionals?

Key observation: Learning functionals can be recast as learning a function through the Riesz Representation Theorem.

Theorem 3.1 (Riesz representation theorem). *Let H be a Hilbert space whose inner product $\langle x, y\rangle$ is defined. For every continuous linear functional $\varphi \in H^*$ there exists a unique function $f_\varphi \in H$, called the Riesz representation of φ, such that:*

$$\varphi[x] = \langle x, f_\varphi\rangle \quad \textit{for all } x \in H. \tag{3}$$

A linear functional can be represented as an inner product between the input function and its primal function.

↓

Can we define an inner product between functions?

$$\langle u, v\rangle = \int_\Omega u(x)v(x)\mathrm{d}x$$

Yes!

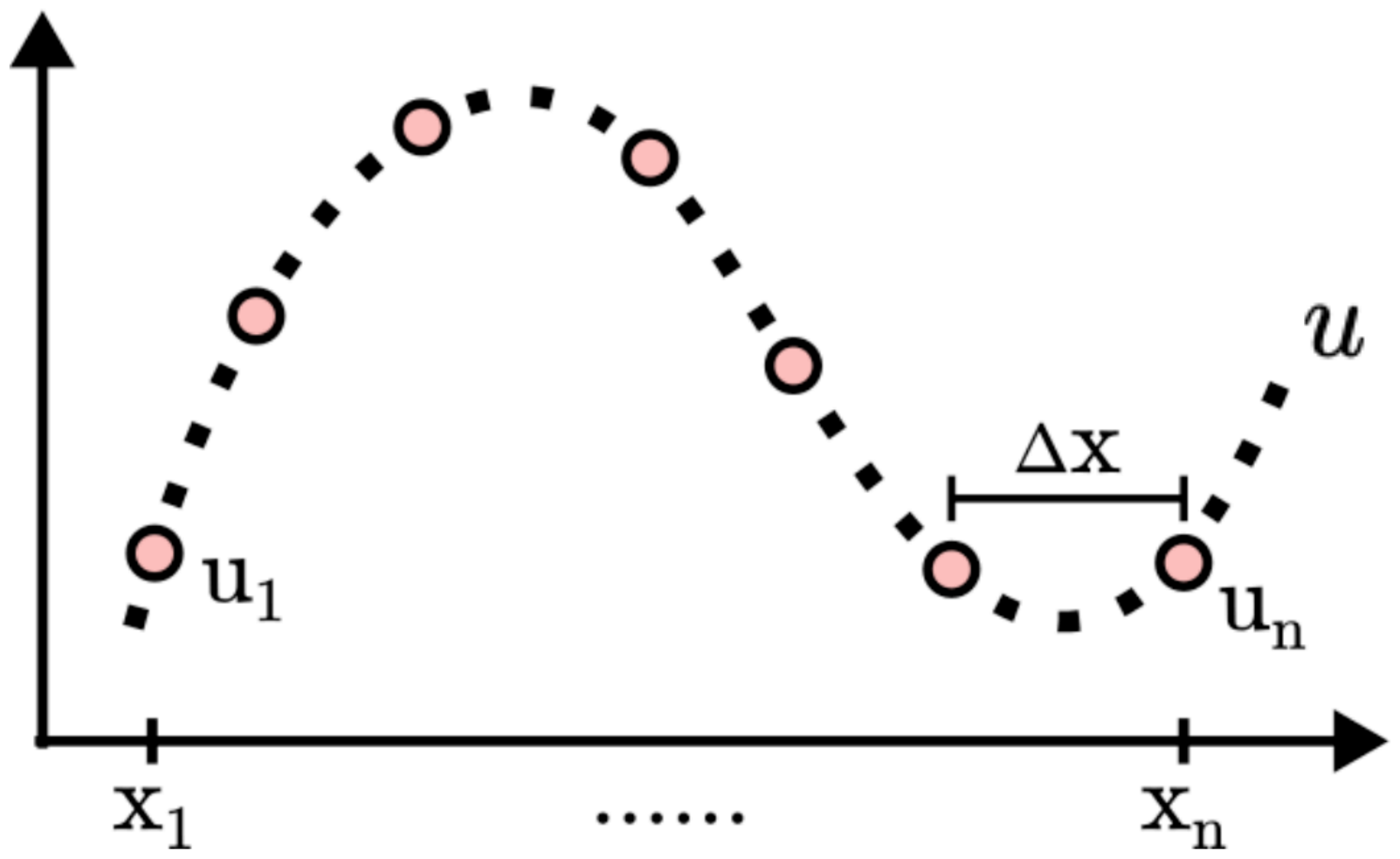
↓

Propose a representation for the Hamiltonian: Integral Kernel Functional

$$\mathcal{H}_\theta[u] = \int_\Omega \kappa_\theta(x)u(x)\mathrm{d}x$$

Recast learning H into learning a kernel function

The Integral Kernel Functional: How do we implement this?



Consider a 1D function, defined on a set of points.

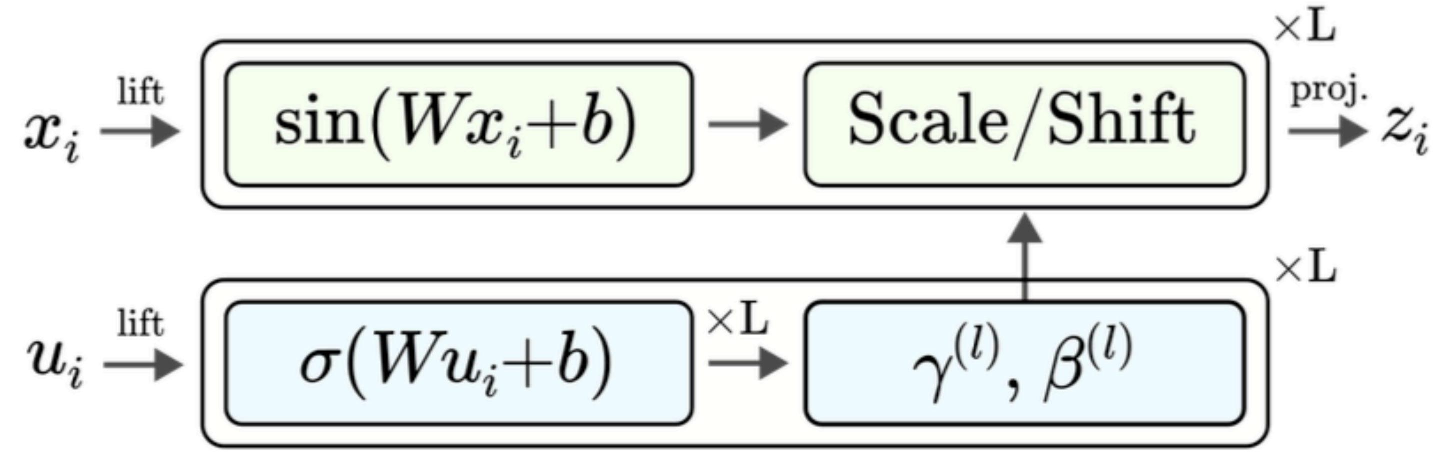
- We want to approximate functionals of u, a continuous function (dotted line).
- We have access to its discretized representation (red points)

$$\mathcal{H}_\theta[u] = \int_\Omega \kappa_\theta(x)u(x)\mathrm{d}x$$

1. Write the IKF. Optionally use nonlinear kernel $\kappa_\theta(x, u(x))$

$$\int_\Omega \kappa_\theta(x, u(x))u(x)dx \approx \sum_i \kappa_\theta(x_i, u(x_i))u(x_i)\mu_i \Delta x$$

2. Approximate the IKF with a Riemann Sum



3. Parameterize kernel w/ a neural network. (In this case a SIREN)

Note: This can provably approximate linear functionals up to arbitrary accuracy. Also closely related to neural operators.

Putting everything together: The Hamiltonian Neural Functional

At this point, we are ready to introduce a PDE surrogate model (Neural Functionals + Hamiltonian Mechanics)

Algorithm 1 Training a HNF

1: **repeat**
2: $\mathcal{H}_\theta \leftarrow \sum_{i=1}^{n} \kappa_\theta(x_i, u_i) u_i \; \mu_i \Delta x$
3: $\frac{\delta \mathcal{H}_\theta}{\delta u} \leftarrow \text{autograd}(\mathcal{H}_\theta, \mathbf{u})$
4: $\mathcal{L} = || \frac{\delta \mathcal{H}_\theta}{\delta u} - \frac{\delta H}{\delta u} ||^2$ **or** $|| \mathcal{J}(\frac{\delta \mathcal{H}_\theta}{\delta u}) - \frac{\mathrm{d}\mathbf{u}}{\mathrm{d}t} ||^2$
5: $\theta \leftarrow \text{Update}(\theta, \nabla_\theta \mathcal{L})$
6: **until** converged

Forward pass to get a scalar H

Backward pass to get functional derivative

Evaluate loss on training data. Recall:

$$\frac{\partial u}{\partial t} = \mathcal{J} \frac{\delta \mathcal{H}}{\delta u}$$

Algorithm 2 Inference with a HNF

1: **repeat**
2: $\mathcal{H}_\theta \leftarrow \sum_{i=1}^{n} \kappa_\theta(x_i, u_i^t) u_i^t \; \mu_i \Delta x$
3: $\frac{\delta \mathcal{H}_\theta}{\delta u} \leftarrow \text{autograd}(\mathcal{H}_\theta, \mathbf{u}^t)$
4: $\frac{\mathrm{d}\mathbf{u}}{\mathrm{d}t} \leftarrow \mathcal{J}(\frac{\delta \mathcal{H}_\theta}{\delta u})$
5: $\mathbf{u}^{t+1} \leftarrow \text{ODEint}(\mathbf{u}^t, \frac{\mathrm{d}\mathbf{u}}{\mathrm{d}t})$
6: **until** done

Forward/Backward Pass

Evaluate operator J

Forward solution with ODE integrator

Experimental Setup: Toy Examples

Sample random p-order polynomials: $u(x) = c_0 x^p + c_1 x^{p-1} + \ldots + c_{p-1} x + c_p$

We can construct analytically-known functionals: $\mathcal{F}_l[u] = \int_{x_1}^{x_M} u(x) * x^2 \mathrm{d}x \qquad \mathcal{F}_{nl}[u] = \int_{x_1}^{x_M} (u(x))^3 \mathrm{d}x$

Generate N samples of train/val data: $(\underbrace{u^n(x)}_{\text{vector}}, \underbrace{\mathcal{F}[u^n(x)]}_{\text{scalar}}) \textbf{ for } n = 1, \ldots, N$

Instantiate network, train on the loss: $\mathcal{L} = \sum_{n=1}^{N} ||\mathcal{F}[u^n(x)] - \mathcal{F}_\theta(\mathbf{u}^n, \mathbf{x})||_2^2$

How well can current neural architectures learn functionals? After training on functionals, are their derivatives also learned?

Models evaluated:

- Neural Network (MLP): Vector to vector mappings
- Neural Operator (FNO): Function to function mappings
- Neural Functional (NF): Function to vector mappings

Modifications:

- OOD: Coefficients during testing are sampled outside the training distribution
- Disc: Data are discretized on an unseen domain and spacing

Results: Toy Examples

Metric	Base MLP	Base FNO	Base NF	OOD ($c_i \in [1,3]$) MLP	OOD ($c_i \in [1,3]$) FNO	OOD ($c_i \in [1,3]$) NF	Disc. ($x_i \in [-2,2]$) MLP	Disc. ($x_i \in [-2,2]$) FNO	Disc. ($x_i \in [-2,2]$) NF
$\mathcal{F}_l[u]$	1.0e-5	5.4e-4	9.8e-16	0.043	0.097	2.7e-14	31.21	31.53	0.21
$\delta\mathcal{F}_l/\delta u$	0.081	0.24	7.0e-4	0.14	0.29	7.0e-4	2.64	2.68	0.033
$\mathcal{F}_{nl}[u]$	0.12	0.026	0.0023	6131	5864	2126	315.9	281.0	62.3
$\delta\mathcal{F}_{nl}/\delta u$	1.99	1.84	0.089	1684	1659	998	74.6	69.9	29.5

Consider two metrics:

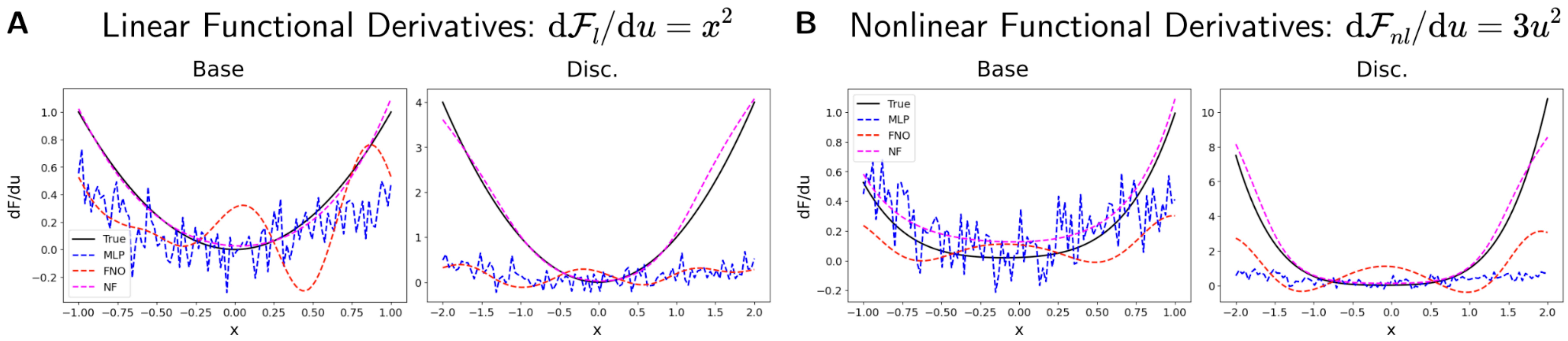
$$\frac{1}{N}\sum_{n=1}^{N}||\mathcal{F}[u^n] - \mathcal{F}_\theta(\mathbf{u}^n, \mathbf{x})||_2^2 \text{ or } \frac{1}{N}\sum_{n=1}^{N}||\frac{\delta\mathcal{F}}{\delta u^n} - \mathbf{autograd}(F_\theta, \mathbf{u}^n)||_2^2$$

Fitting functionals

Fitting functional derivatives

- Conventional architectures can approximate functionals in-distribution, although NFs are extremely good.
- In OOD/Disc regimes, conventional architectures suffer and NFs retain good performance for linear functionals.
- Conventional architectures cannot approximate functional derivatives well, and this is worse in OOD/Disc regimes.
- No architectures can extrapolate for nonlinear functionals.

Visualizations: Toy Examples



- When examining functional derivatives, conventional architectures have trouble even on in-distribution samples.
  - On unseen discretizations, the error is even worse.
- NFs are able to implicitly learn smooth, accurate functional derivatives, even when only trained on scalar functional data and with unseen inputs.

Experimental Setup: 1D Advection and KdV Equations

Define Hamiltonians for Adv/KdV:

$$\mathcal{H}_{adv}[u] = \int_\Omega -\frac{1}{2}(u(x))^2 \mathrm{d}x, \qquad \mathcal{H}_{kdv}[u] = \int_\Omega -\frac{1}{6}(u(x))^3 - u(x)\frac{\partial^2 u}{\partial x^2}(x)\mathrm{d}x,$$

Lookup and check Hamiltonian structure (Adv):

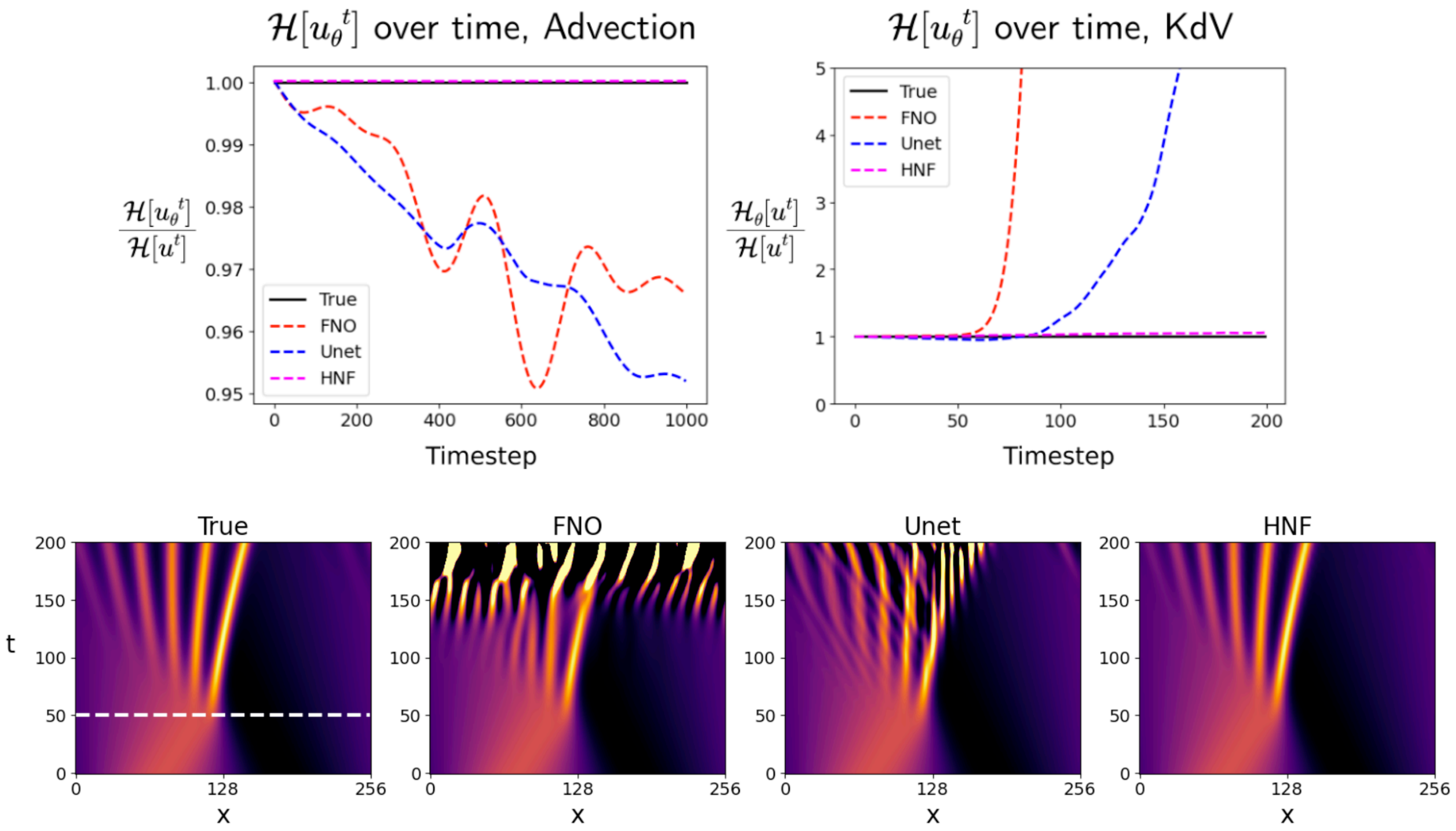
$$\mathcal{J} \text{ is defined as } \partial_x \qquad \frac{\delta \mathcal{H}_{adv}}{\delta u} = -u(x), \qquad \mathcal{J}(\frac{\delta \mathcal{H}_{adv}}{\delta u}) = -\frac{\partial u}{\partial x} = \frac{\mathrm{d}u}{\mathrm{d}t}$$

- Generate train/val samples using numerical methods. Calculate necessary Hamiltonian quantities from data.
- To evaluate generalization, training data is solved only to 20% or 25% of the validation time horizon.
- Compare to Unet/FNO baselines. Train additional baselines that predict du/dt + use ODE integrator.

Results: 1D Advection and KdV Equations

Metric:	Adv Roll. Err. ↓	KdV Corr. Time ↑
FNO	0.81 ± 0.17	69.5 ± 8.2
Unet	0.52 ± 0.35	125.7 ± 8.5
FNO($\frac{d\mathbf{u}}{dt}$)	0.044 ± 0.002	75.5 ± 1.9
Unet($\frac{d\mathbf{u}}{dt}$)	0.068 ± 0.029	127.4 ± 4.9
HNF	0.0039 ± 0.0002	150.9 ± 3.3

Table 2: Results for 1D PDEs. Parameter counts are: FNO (65K), Unet (65K), HNF (32K) for Adv, and FNO (135K), Unet (146K), HNF (87K) for KdV.



- HNFs can be more **efficient**, with around half the parameters of other models.
- HNFs can be more **stable**, predicting solutions that conserve energies better than baselines.
- HNFs can be more **generalizable**, predicting solutions at timesteps unseen in the training horizon/distribution.

Experimental Setup: 2D Shallow Water Equations (SWE)

Define Hamiltonian for SWE:

$$\mathcal{H}[\mathbf{u}] = \mathcal{H}[v_x, v_y, h] = \int_\Omega \frac{1}{2}h(v_x^2 + v_y^2) + \frac{1}{2}gh^2 \mathbf{d}A$$

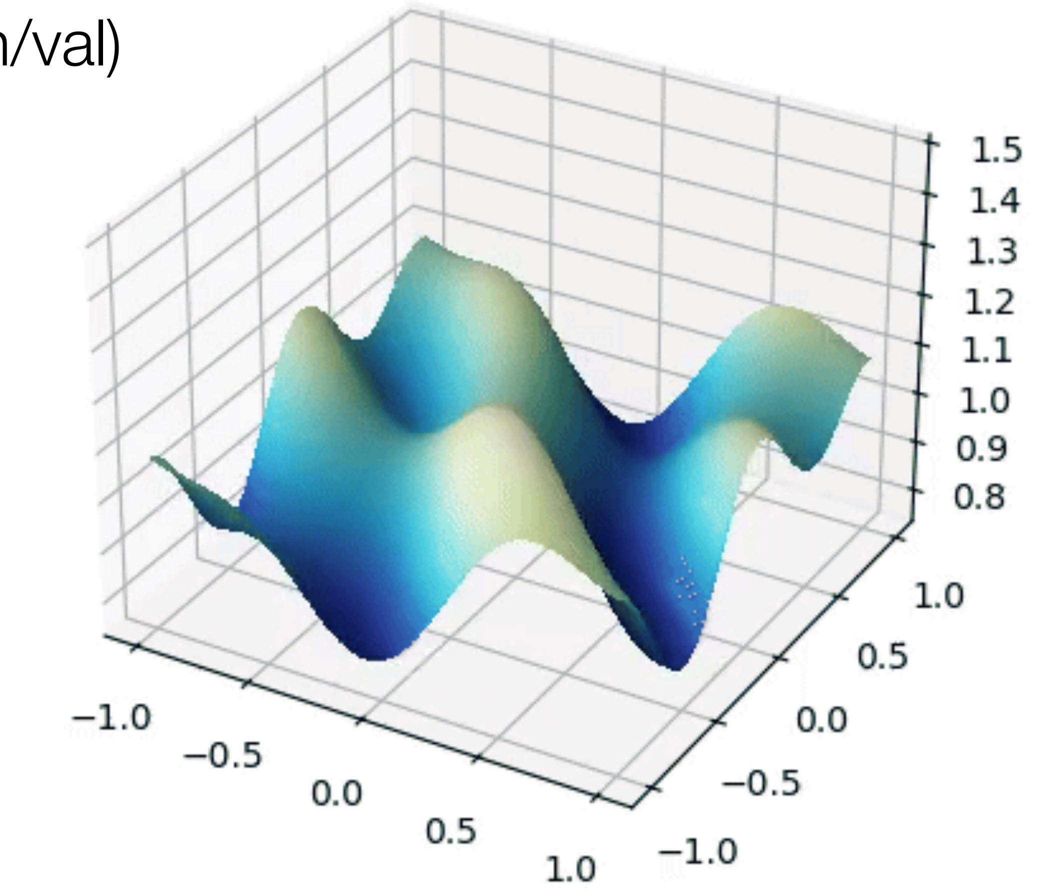
Lookup Hamiltonian Structure:

$$\mathcal{J} = \begin{bmatrix} 0 & -q & \partial_x \\ q & 0 & \partial_y \\ \partial_x & \partial_y & 0 \end{bmatrix}, \qquad q = \frac{\partial v_x}{\partial y} - \frac{\partial v_y}{\partial x}$$

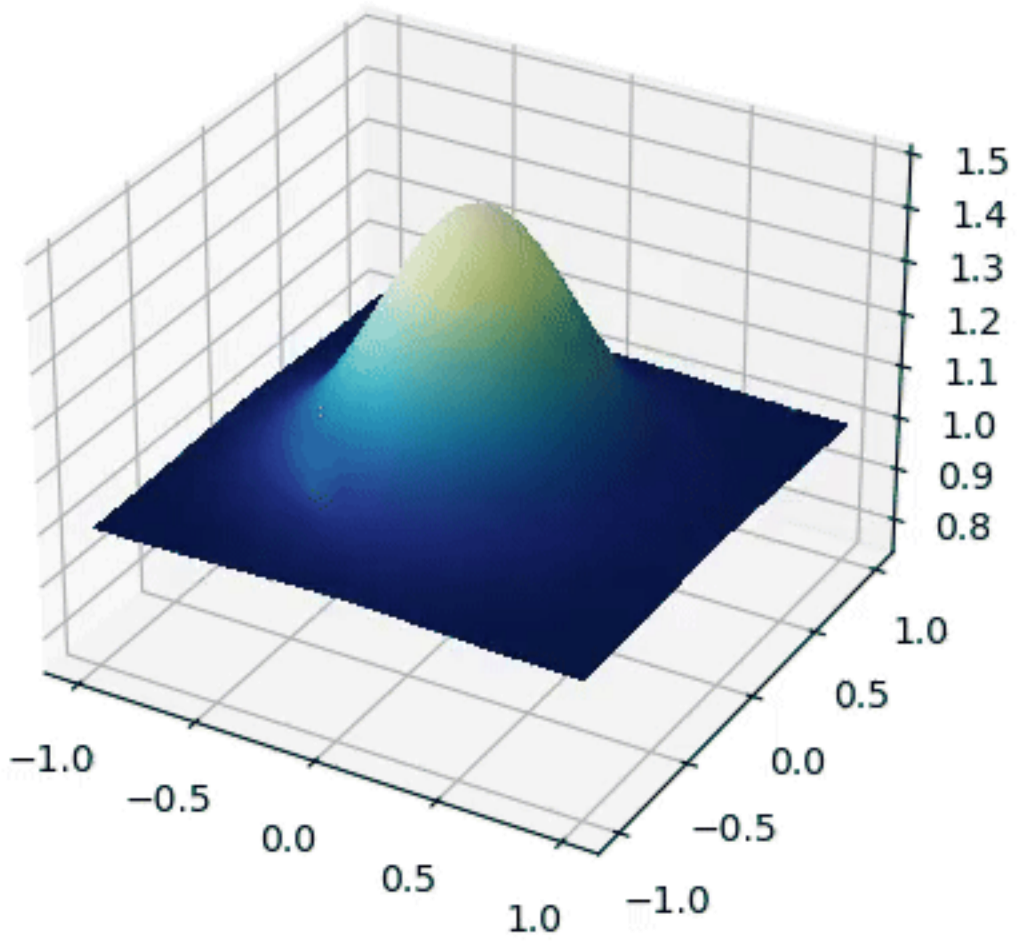
$$\mathcal{J}\frac{\delta \mathcal{H}}{\delta \mathbf{u}} \longrightarrow \partial_t h + \nabla \cdot (\mathbf{v}h) = 0, \qquad \partial_t \mathbf{v} + \mathbf{v} \cdot \nabla \mathbf{v} = -g\nabla h$$

To evaluate generalization, generate an additional testing set with unseen initial conditions:

Sines (train/val)



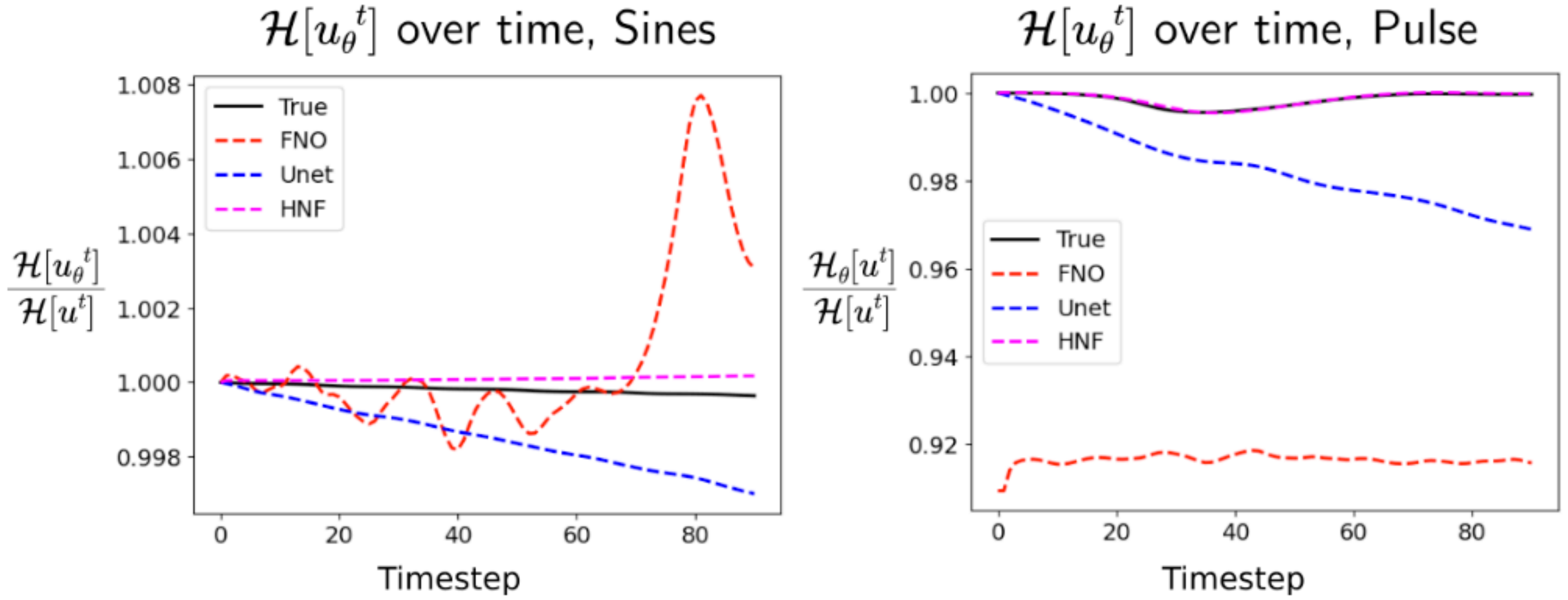
Pulse (val)

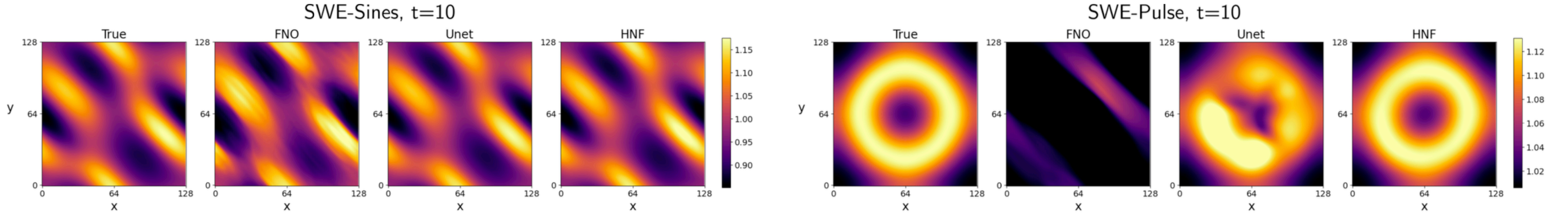


Results: 2D Shallow Water Equations (SWE)

Model	Sines	Pulse
FNO	0.057±0.002	0.117±0.0009
Unet	0.010±0.0014	0.042±0.0006
HNF	0.026±0.0003	0.021±0.0015

Table 3: Rollout errors for 2D SWE.





- All modes work well for equations in-distribution. HNFs work well on unseen initial conditions.
- In both test cases, HNFs have exceptional capabilities in conserving energy.

Limitations

1. Many PDEs do not have a convenient Hamiltonian structure or conserve energy

"It is impossible to derive the equations of steady motion of a viscous, incompressible fluid from a variation principle involving as Lagrangian function"
- Robert Millikan, 1923 Nobel Prize in Physics

2. Theory for nonlinear functional approximation is underdeveloped.

?

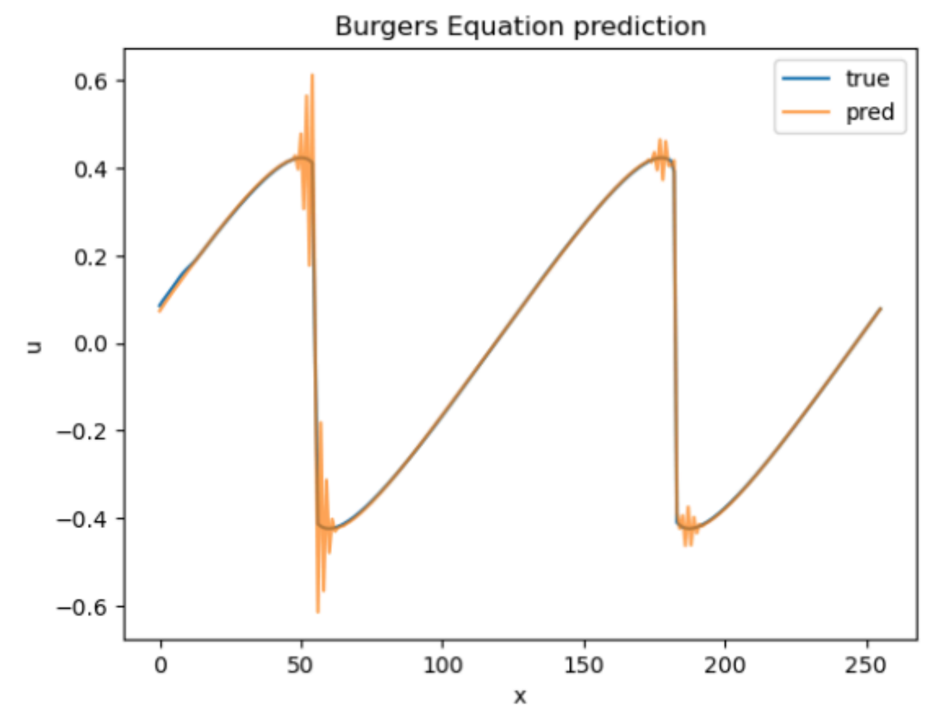
Way out of my depth…

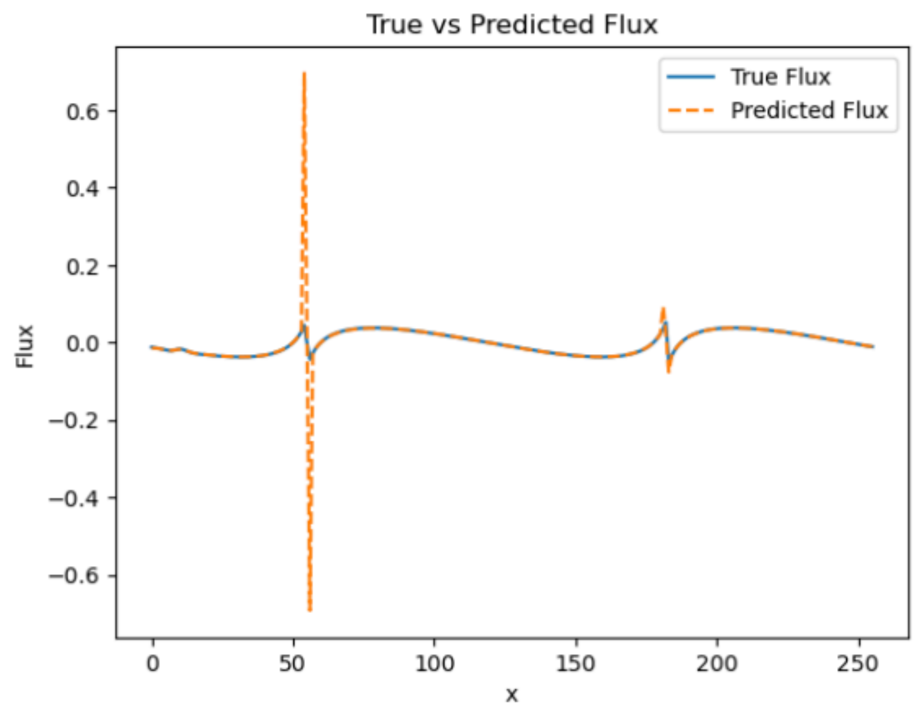
3. Additional overhead during training/inference to backpropagate through network

Model (#Params)	Adv	KdV	SWE-Sines
FNO (65K/135K/7M)	0.0829	0.091	0.967
Unet (65K/146K/3M)	0.138	0.146	1.345
HNF (32K/87K/3M)	0.126	0.228	4.547

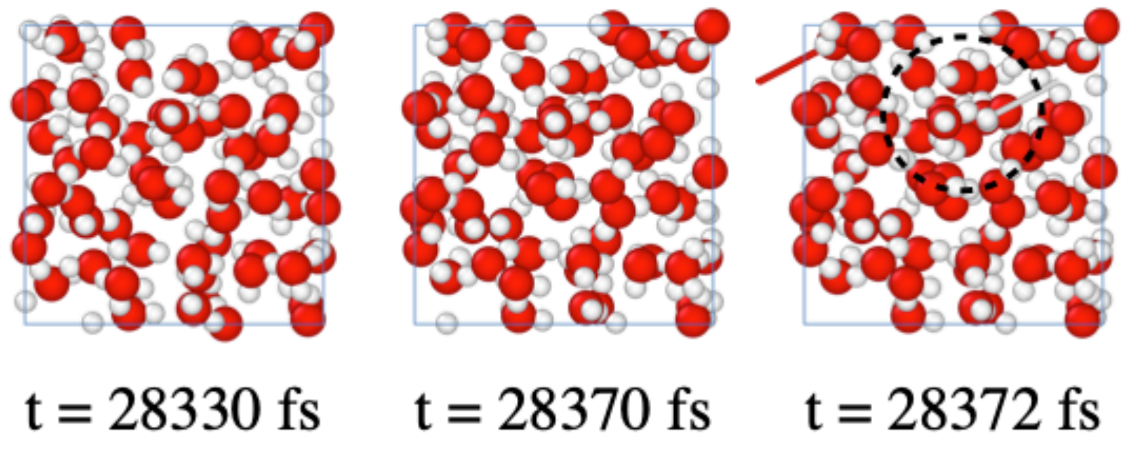
Times are given in ms

4. Evaluating J in Hamilton's equations needs to be done carefully.



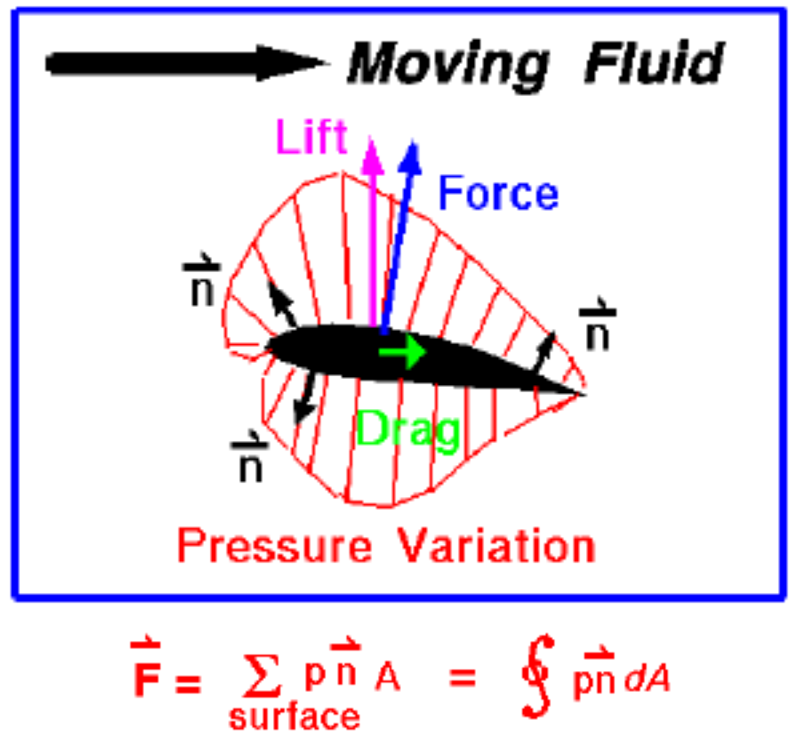


Outlook and Conclusion



Outlook

- Neural Functionals may have more applications:
  - Molecular dynamics: Functionals are system energies; energy is conserved, and derivatives are forces.
  - Design: Functionals are system properties (weight/drag/etc.); derivatives can be used for optimization.
- My personal view: An interesting model/framework, but requires a lot of work to make it successful in PDEs.
  - Pure data-driven methods work well and removes the burden of carefully discretizing, developing solvers, etc.



Conclusions

- Maybe the first to extend Hamiltonian mechanics to learning complex PDEs.
- Proposing a new architecture for learning functionals (there are prior works learning function to vector mappings)[1]
- HNFs require some tuning, but work well for conservative PDEs.
  - Excellent energy conservation: can allow more generalizable, stable, and efficient models.

Xiang Fu, Zhenghao Wu, Wujie Wang, Tian Xie, Sinan Keten, Rafael Gomez-Bombarelli, Tommi Jaakkola, Forces are not Enough: Benchmark and Critical Evaluation for Machine Learning Force Fields with Molecular Simulations, https://arxiv.org/abs/2210.07237, https://www.grc.nasa.gov/WWW/K-12/VirtualAero/BottleRocket/airplane/presar.html
1. Daniel Zhengyu Huang, Nicholas H. Nelsen, Margaret Trautner, An operator learning perspective on parameter-to-observable maps, https://arxiv.org/abs/2402.06031

Thank You!

Questions?

Appendix

Double Pendulum

Newtonian Mechanics

Force Balance (Newton's 2nd, 3rd Laws)

$$\begin{aligned} m_1 l_1\left(\ddot{\theta}_1\cos\theta_1 - \dot{\theta}_1^2\sin\theta_1\right) &= -T_1\sin\theta_1 + T_2\sin\theta_2 \\ -m_1 l_1\left(\ddot{\theta}_1\sin\theta_1 + \dot{\theta}_1^2\cos\theta_1\right) &= -T_1\cos\theta_1 + T_2\cos\theta_2 + m_1 g \end{aligned}$$

$$\begin{aligned} m_2\left(l_1\ddot{\theta}_1\cos\theta_1 - l_1\dot{\theta}_1^2\sin\theta_1 + l_2\ddot{\theta}_2\cos\theta_1 - l_2\dot{\theta}_2^2\sin\theta_2\right) &= -T_2\sin\theta_2 \\ -m_2\left(l_1\ddot{\theta}_1\sin\theta_1 + l_1\dot{\theta}_1^2\cos\theta_1 + l_2\ddot{\theta}_2\sin\theta_2 + l_2\dot{\theta}_2^2\cos\theta_2\right) &= -T_2\cos\theta_2 + m_2 g. \end{aligned}$$

"Simplify"

$$\begin{aligned} l_1\ddot{\theta}_1 &= (T_2/m_1)\sin(\theta_2-\theta_1) - g\sin\theta_1 \\ l_1\dot{\theta}_1^2 &= (T_1/m_1) - (T_2/m_1)\cos(\theta_2 - t_1) - g\cos\theta_1 \end{aligned}$$

$$\begin{aligned} l_1\ddot{\theta}_1\cos(\theta_2-\theta_1) + l_1\dot{\theta}_1^2\sin(\theta_2-\theta_1) + l_2\ddot{\theta}_2 &= -g\sin\theta_2 \\ -l_1\ddot{\theta}_1\sin(\theta_2-\theta_1) + l_1\dot{\theta}_1^2\cos(\theta_2-\theta_1) + l_2\dot{\theta}_2^2 &= (T_2/m_2) - g\cos\theta_2. \end{aligned}$$

$$\begin{aligned} l_2\ddot{\theta}_2 &= -g\sin\theta_2 - ((T_2/m_1)\sin(\theta_2-\theta_1) - g\sin\theta_1)\cos(\theta_2-\theta_1) \\ &\quad - ((T_1/m_1) - (T_2/m_1)\cos(\theta_2-t_1) - g\cos\theta_1)\sin(\theta_2-\theta_1) \\ &= -(T_1/m_1)\sin(\theta_2-\theta_1) \end{aligned}$$

$$\begin{aligned} l_2\dot{\theta}_2^2 &= (T_2/m_2) - g\cos\theta_2 + ((T_2/m_1)\sin(\theta_2-\theta_1) - g\sin\theta_1)\sin(\theta_2-\theta_1) \\ &\quad - ((T_1/m_1) - (T_2/m_1)\cos(\theta_2-t_1) - g\cos\theta_1)\cos(\theta_2-\theta_1) \\ &= (T_2/m_2) + (T_2/m_1) - (T_1/m_1)\cos(\theta_2-\theta_1) \end{aligned}$$

Solution

$$\begin{aligned} l_1\ddot{\theta}_1 &= (T_2/m_1)\sin(\theta_2-\theta_1) - g\sin\theta_1 \\ l_1\dot{\theta}_1^2 &= (T_1/m_1) - (T_2/m_1)\cos(\theta_2-\theta_1) - g\cos\theta_1 \\ l_2\ddot{\theta}_2 &= -(T_1/m_1)\sin(\theta_2-\theta_1) \\ l_2\dot{\theta}_2^2 &= (T_2/m_2) + (T_2/m_1) - (T_1/m_1)\cos(\theta_2-\theta_1) \end{aligned}$$

$$\begin{aligned} T_1 &= -m_1\frac{l_2\ddot{\theta}_2}{\sin(\theta_2-\theta_1)} \\ T_2 &= m_1\frac{l_1\ddot{\theta}_1 + g\sin\theta_1}{\sin(\theta_2-\theta_1)} \end{aligned}$$

Hamiltonian Mechanics

Define Hamiltonian (Kinetic+Potential Energy)

$$H = \frac{m_2 l_2^2 p_{\theta_1}^2 + (m_1+m_2) l_1^2 p_{\theta_2}^2 - 2m_2 l_1 l_2 p_{\theta_1} p_{\theta_2}\cos(\theta_1-\theta_2)}{2m_2 l_1^2 l_2^2\left[m_1 + m_2\sin^2(\theta_1-\theta_2)\right]} - (m_1+m_2) g l_1\cos\theta_1 - m_2 g l_2\cos\theta_2$$

Apply Hamilton's Equations of Motion

$$\dot{\theta}_i = \frac{\partial H}{\partial p_{\theta_i}}$$

$$\dot{p}_{\theta_i} = -\frac{\partial H}{\partial \theta_i}$$

Solution

$$\begin{aligned} \dot{\theta}_1 &= \frac{\partial H}{\partial p_{\theta_1}} = \frac{l_2 p_{\theta_1} - l_1 p_{\theta_2}\cos(\theta_1-\theta_2)}{l_1^2 l_2\left[m_1 + m_2\sin^2(\theta_1-\theta_2)\right]} \\ \dot{\theta}_2 &= \frac{\partial H}{\partial p_{\theta_2}} = \frac{-m_2 l_2 p_{\theta_1}\cos(\theta_1-\theta_2) + (m_1+m_2) l_1 p_{\theta_2}}{m_2 l_1 l_2^2\left[m_1 + m_2\sin^2(\theta_1-\theta_2)\right]} \\ \dot{p}_{\theta_1} &= -\frac{\partial H}{\partial \theta_1} = -(m_1+m_2) g l_1\sin\theta_1 - h_1 + h_2\sin\left[2(\theta_1-\theta_2)\right] \\ \dot{p}_{\theta_2} &= -\frac{\partial H}{\partial \theta_2} = -m_2 g l_2\sin\theta_2 + h_1 - h_2\sin\left[2(\theta_1-\theta_2)\right] \end{aligned}$$

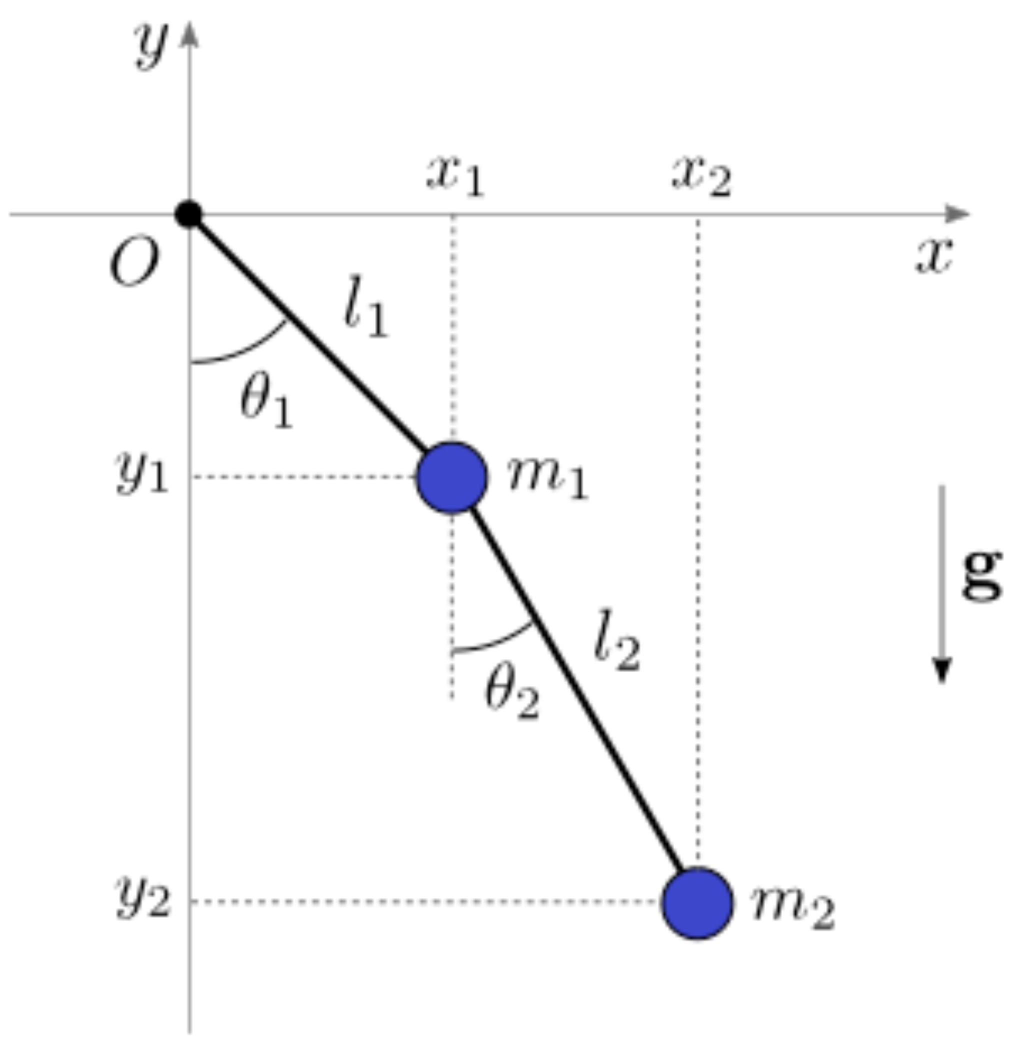


Fig. 1: A double pendulum.

Derivation is simpler

Change of perspective

Newtonian: Objects move in response to forces

Hamiltonian: Objects move to minimize energy

https://www.phys.lsu.edu/faculty/gonzalez/Teaching/Phys7221/DoublePendulum.pdf, https://dassencio.org/46

Hamiltonian Errors

Model	Adv	KdV	SWE-Sines	SWE-Pulse
FNO	2.18	NaN	0.0091	0.1326
Unet	0.22	2.37	0.0053	0.0158
FNO($\frac{d\mathbf{u}}{dt}$)	0.033	6.75e8	-	-
Unet($\frac{d\mathbf{u}}{dt}$)	0.043	5.76e6	-	-
HNF	0.0002	1.32	0.0015	0.0003

Table 4: Relative L2 Error for each experiment, evaluated on the Hamiltonian of predicted trajectories. Despite not including the Hamiltonian in the training loss and testing on OOD samples, HNFs are exceptional at predicting solutions that conserve the Hamiltonian.

$$\frac{1}{T} \sum_{t=1}^{T} \frac{||\mathcal{H}[\mathbf{u}^t] - \mathcal{H}[\mathbf{u}_\theta^t]||_2^2}{||\mathcal{H}[\mathbf{u}^t]||_2^2}$$

Inductive Biases

1. ODE Bias: HNFs predict $\frac{d\mathbf{u}}{dt}$ and use an ODE integrator to evolve PDE dynamics.
2. Hamiltonian Bias: HNFs rely on $\mathcal{J}\frac{\delta\mathcal{H}}{\delta\mathbf{u}}$ to calculate $\frac{d\mathbf{u}}{dt}$.
3. Gradient Learning Bias: HNFs rely on $\text{autograd}(\mathcal{H}_\theta[\mathbf{u}], \mathbf{u})$ to calculate $\frac{\delta\mathcal{H}}{\delta\mathbf{u}}$.
4. Neural Functional Bias: HNFs rely on neural functionals to calculate $\text{autograd}(\mathcal{H}_\theta[\mathbf{u}], \mathbf{u})$.

Metric:	Correlation Time (↑)	Hamiltonian Error (↓)
Unet (Base)	125.25	2.37
Unet (ODE)	120.5	5.76e6
Unet (Ham.)	143.75	2.06
Unet (Grad.)	141	4.71
HNF	151.75	1.32

Table 5: Correlation time and Hamiltonian errors for Unet models with increasingly more inductive biases, compared to HNFs on the KdV equation. Using ODE integrators or gradient domain learning both degrade performance, while using Hamiltonian structure or neural functionals both increase performance and energy conservation.

Numerical Methods

Numerical Integration ODE integration is performed using a 2nd-order Adams-Bashforth scheme. The solution at the next timestep $\mathbf{u}^{t+1}$ is calculated using the current timestep $\mathbf{u}^t$ and an estimate of the current derivative $\frac{d\mathbf{u}}{dt}|_{t=t} = f(\mathbf{u}^t)$:

$$\mathbf{u}^{t+1} = \mathbf{u}^t + \Delta t \left(\frac{3}{2} f(\mathbf{u}^t) - \frac{1}{2} f(\mathbf{u}^{t-1}) \right) \tag{15}$$