

Hamiltonian Neural PDE Solvers through Functional Approximation

Anthony Zhou

Carnegie Mellon University
ayz2@andrew.cmu.edu

Amir Barati Farimani

Carnegie Mellon University
barati@cmu.edu

Abstract

Designing neural networks within a Hamiltonian framework offers a principled way to ensure that conservation laws are respected in physical systems. While promising, these capabilities have been largely limited to discrete, analytically solvable systems. In contrast, many physical phenomena are governed by PDEs, which govern infinite-dimensional fields through Hamiltonian functionals and their functional derivatives. Building on prior work, we represent the Hamiltonian functional as a kernel integral parameterized by a neural field, enabling learnable function-to-scalar mappings and the use of automatic differentiation to calculate functional derivatives. This allows for an extension of Hamiltonian mechanics to neural PDE solvers by predicting a functional and learning in the gradient domain. We show that the resulting Hamiltonian Neural Solver (HNS) can be an effective surrogate model through improved stability and conserving energy-like quantities across 1D and 2D PDEs. This ability to respect conservation laws also allows HNS models to better generalize to longer time horizons or unseen initial conditions.

1 Introduction

Physics is remarkably complex; a set of governing laws can produce endlessly diverse and interesting phenomena. In an effort to unify governing laws, Hamiltonian mechanics describes physical phenomena under the Principle of Least Action, whereby the trajectory of a system evolves because it has locally the least action. Remarkably, the minimization of action can derive nearly all physical phenomena, from kinematics to quantum mechanics, suggesting an underlying, fundamental property of nature. Rather than describing physics through forces and masses, such as in Newtonian mechanics, using energy and symmetries allows for an elegant and unifying perspective of physical systems.

Despite the long history of Hamiltonian mechanics, most current numerical and neural PDE solvers operate within a Newtonian framework. This has been very successful; forces and masses are easy to intuitively understand, and both numerical and neural solvers have found important, practical uses. However, motivated by natural conservation laws and symmetries that arise from Hamiltonian mechanics, we seek to investigate neural PDE solvers that operate in this framework. We find that the Hamiltonian framework brings inductive biases that improve model performance, especially when conserving energy-like quantities.

Extending Hamiltonian mechanics to neural PDE solvers requires both theoretical and implementation insights. In particular, most PDEs are used to describe continuum systems (fluids, waves, elastic bodies, etc.); as such, infinite-dimensional Hamiltonian mechanics are needed. In this setup, the Hamiltonian is defined as a functional that maps from input fields to a scalar, often interpreted as the energy of the system. The variational or functional derivative of the Hamiltonian then contains dynamical information that evolves the system. This introduces interesting modeling challenges; in particular, the need to approximate infinite-dimensional to scalar mappings and for the approximation to have functional derivatives.

In this paper, we consider models based on a learnable kernel integral, a framework that can provably approximate linear functionals and implicitly learn functional derivatives. Through automatic differentiation, functional derivatives can be obtained and optimized in a Hamiltonian framework to predict future PDE states. This allows the development of Hamiltonian Neural Solvers (HNS), which we evaluate on three PDE systems and find both high accuracy and generalization capabilities, which we hypothesize arise from inductive biases to conserve energy. Code and datasets for this work are released at https://github.com/anthonyzhou-1/hamiltonian_pdes.

2 Background

Hamiltonian Mechanics Hamiltonian mechanics is usually introduced in the discrete setting, where there are a set of n bodies, each with a position and momentum. Therefore, the state of the system can be described by $2n$ quantities, usually assembled into a position and momentum vector $\mathbf{q}, \mathbf{p} \in \mathbb{R}^n$. In the discrete case, the Hamiltonian $\{\mathcal{H}(\mathbf{q}, \mathbf{p}) : \mathbb{R}^{2n} \rightarrow \mathbb{R}\}$ is a function that takes two vectors and returns a scalar \mathcal{H} , usually interpreted as the energy or another conserved quantity. Evolving the system in time is done through Hamilton’s equations, often expressed in terms of the symplectic matrix J and state vector \mathbf{u} :

$$\underbrace{\frac{d\mathbf{q}}{dt} = \frac{\partial \mathcal{H}}{\partial \mathbf{p}}, \quad \frac{d\mathbf{p}}{dt} = -\frac{\partial \mathcal{H}}{\partial \mathbf{q}}}_{\text{Hamilton's Equations}} \quad \underbrace{\frac{d\mathbf{u}}{dt} = \begin{bmatrix} d\mathbf{q}/dt \\ d\mathbf{p}/dt \end{bmatrix} = \begin{bmatrix} 0 & \mathbb{I} \\ -\mathbb{I} & 0 \end{bmatrix} \begin{bmatrix} \partial \mathcal{H} / \partial \mathbf{q} \\ \partial \mathcal{H} / \partial \mathbf{p} \end{bmatrix} = J \nabla_{\mathbf{u}} \mathcal{H}}_{\text{Symplectic Form}} \quad (1)$$

Derivatives can be evaluated from vector calculus identities; for clarity we consider the state vector \mathbf{u} with $2n$ components and write $\frac{d\mathbf{u}}{dt} = [\frac{du_1}{dt}, \dots, \frac{du_{2n}}{dt}]$, and $\nabla_{\mathbf{u}} \mathcal{H} = [\frac{\partial \mathcal{H}}{\partial u_1}, \dots, \frac{\partial \mathcal{H}}{\partial u_{2n}}]$.

Infinite-Dimensional Systems Many PDEs describe continuum systems, such as waves, fluids, and elastic bodies, where the positions and momenta of discrete bodies are not well defined. Despite this change, continuum systems can still conserve energy and be viewed in a Hamiltonian framework. Firstly, finite-dimensional vectors $\mathbf{q}, \mathbf{p} \in \mathbb{R}^n$ become generalized to functions $u(x, t)$ defined on continuously varying coordinates $x \in \Omega$ and at a time t . The function $u \in \mathcal{F}(\Omega)$ describes the state of the system by assigning a scalar or vector for each coordinate (such as velocity, pressure, etc.). Secondly, the Hamiltonian is generalized from a function $\{\mathcal{H}(\mathbf{q}, \mathbf{p}) : \mathbb{R}^{2n} \rightarrow \mathbb{R}\}$ to a functional $\{\mathcal{H}[u] : \mathcal{F}(\Omega) \rightarrow \mathbb{R}\}$ that returns a scalar given an input function. Lastly, the symplectic matrix J generalizes to a linear operator \mathcal{J} , and the gradient $\nabla_{\mathbf{u}}$ becomes the variational or functional derivative $\frac{\delta \mathcal{H}}{\delta u} \in \mathcal{F}(\Omega)$. This leads to Hamilton’s equations for infinite-dimensional systems, where $\epsilon \in \mathcal{F}(\Omega)$ is an arbitrary test function [1]:

$$\underbrace{\frac{\partial u}{\partial t} = \mathcal{J} \frac{\delta \mathcal{H}}{\delta u}}_{\text{Hamilton's Equations}} \quad \underbrace{\int_{\Omega} \frac{\delta \mathcal{H}}{\delta u}(x) \epsilon(x) dx = \lim_{h \rightarrow 0} \frac{\mathcal{H}[u + h\epsilon] - \mathcal{H}[u]}{h}}_{\text{Functional Derivative}} \quad (2)$$

From this background we can see why neural networks are a natural choice for modeling discrete Hamiltonian systems; in this setting, the Hamiltonian is a function and its gradient is naturally calculated with automatic differentiation. In the infinite-dimensional setting, learning the Hamiltonian functional from data becomes less obvious.

3 Functional Approximation

Theory The development of functional approximators [2–5] can be seen a result of the Riesz representation theorem, a fundamental result in functional analysis [6]. We give some intuition here, but provide a full proof of linear functional approximation in Appendix C.1. We start by observing that a functional can be equivalent to the inner product over suitable functions:

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:*

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

Immediately, we can see that the Hamiltonian $\mathcal{H}[u]$ can potentially be expressed as an inner product $\langle u, \kappa_\theta \rangle$ over the input function u and a learnable function κ_θ . This changes the problem of approximating a functional to the problem of approximating a function, which neural networks can readily achieve through universal approximation theorems [7, 8].

A common inner product for arbitrary functions u, v in a Hilbert space H involves an integral: $\langle u, v \rangle = \int_{\Omega} u(x)v(x)dx$. One can check that this satisfies the basic properties of inner products and is also valid for vector-valued functions u, v . Based on this formulation, functionals can be parameterized by an *Integral Kernel Functional* (IKF):

$$\mathcal{H}_\theta[u] = \int_{\Omega} \kappa_\theta(x)u(x)dx \quad (4)$$

In this form, we observe a connection between the integral kernel functional and the integral kernel operator that neural operators are built on [9]. Specifically, the integral kernel operator is given by:

$$\mathcal{K}_\theta(u)(x) = \int_{\Omega} \kappa_\theta(x, y)u(y)dy \quad (5)$$

where a learnable operator \mathcal{K}_θ acting on an input function $u(x) \in \mathcal{U}$ produces an output function $\mathcal{K}(v)(x) \in \mathcal{V}$ for Banach spaces \mathcal{U}, \mathcal{V} ; this expression is equivalent to the IKF when evaluated at a single point x . We can make a similar modification to neural operators where the kernel is allowed to be nonlinear by giving function values as input: $\kappa_\theta(x, y, u(x), u(y))$. In our setting, we can adopt a nonlinear kernel by allowing the kernel to change based on the input function: $\kappa_\theta(x, u(x))$.

One shortcoming of this framework is that the approximation of nonlinear functionals is not proven. Most Hamiltonians are nonlinear with respect to u , however, we empirically find that the proposed architecture can still approximate nonlinear functionals by using a nonlinear kernel $\kappa_\theta(x, u(x))$. There are some potential hypotheses for this. The Riesz representation theorem is a strong statement, implying that for linear functionals, there is a single, unique $\kappa_\theta \in H$ that can be used with any $u \in H$ to approximate $\mathcal{H}[u]$. In deep learning we are less constrained; we can potentially allow $\kappa_\theta(u)$ to depend on u and even relax the uniqueness of κ_θ to still be able to approximate $\mathcal{H}[u]$. Furthermore, under assumptions of regularity and compact support, for arbitrary functionals \mathcal{H} we can always show that a function $f \in H$ exists such that $\mathcal{H}[u] = \langle u, f \rangle$ by constructing $f = \frac{\mathcal{H}[u]}{\|u\|_2^2}u$, with the norm: $\|u\|_2 = \sqrt{\int_{\Omega} u(x)^2 dx}$.

Implementation There are two main implementation choices: approximating the integral and parameterizing the kernel. The integral can be approximated by a Riemann sum:

$$\mathcal{H}_\theta[u] = \int_{\Omega} \kappa_\theta(x, u(x))u(x)dx \approx \sum_i \kappa_\theta(x_i, u(x_i))u(x_i)\mu_i\Delta x \quad (6)$$

with quadrature weights μ_i . Interestingly, the full Riemann sum can be computed since the sum is only calculated once; in neural operator literature, approximating the integral with a full Riemann sum is usually too costly since the sum is evaluated for every query point. This results in approaches that truncate the sum [10–12] or represent it in Fourier space through the convolution theorem [13]. In our implementation, we evaluate different quadratures but find that the trapezoidal rule is enough to accurately estimate the integral.

To parameterize the kernel, we take the perspective from the Riesz representation theorem. In the linear case, the kernel is simply a function $\kappa_\theta(x) \in H$ that maps input coordinates to values in the codomain. This is exactly what a neural field is; therefore, we can inherit the substantial prior work on neural fields [14] to effectively parameterize κ_θ .

One useful concept is the conditional neural field. In particular, the nonlinear kernel $\kappa_\theta(x, u(x))$ can be seen as a conditional neural field whereby the kernel output changes based on the input function $u(x)$. This concept can also be further extended to distinguish between local and global conditioning [15]; in neural field literature, the field can be conditioned on local information $u_i = u(x_i)$ or global information $\mathbf{u} = [u(x_0), \dots, u(x_n)]$. Conditioning mechanisms can also be implemented in various ways. In the simplest case, the conditional information u_i or \mathbf{u} is concatenated to the input coordinate x [16–18], however, we adopt a approach based on Feature-wise Linear Modulation (FiLM) [19].

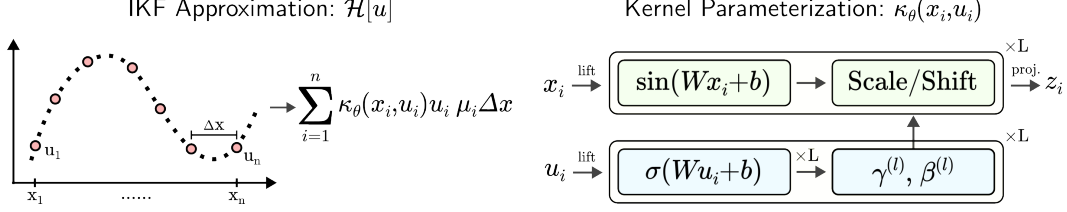


Figure 1: *Left:* The functional $\mathcal{H}[u]$ is approximated by an integral kernel functional (IKF). The integral is further approximated by a Riemann sum with a learnable kernel κ_θ . *Right:* The kernel can be parameterized with various architectures and conditioning mechanisms. A local, FiLM-conditioned SIREN kernel is shown, where each SIREN layer has a dedicated FiLM network to produce scale and shift parameters. The output z_i is for a single point x_i , which needs to be calculated $\forall i = \{1, \dots, n\}$

Beyond conditioning mechanisms, an important consideration is the architecture of the kernel itself. A powerful neural field architecture is the sinusoidal representation network or SIREN [20]. SIRENs have shown good performance not only in fitting a neural field κ_θ , but also in accurately representing gradients $\nabla \kappa_\theta$, which may help in fitting the functional derivative $\delta \mathcal{H} / \delta u$. With these architectural choices, a layer of the kernel $\kappa_\theta^{(l)}$ with local conditioning can be written as:

$$\kappa_\theta^{(l)}(x_i, u_i) = \gamma_\theta^{(l)}(u_i) \sin(Wx_i + b) + \beta_\theta^{(l)}(u_i) \quad (7)$$

where $\gamma_\theta^{(l)}$ and $\beta_\theta^{(l)}$ are the FiLM scale and shift networks at layer l . Each SIREN layer has its own FiLM network and each FiLM network can also have multiple layers. Inputs x_i, u_i can optionally be lifted and the output z_i can be projected back to the function dimension. Given the hierarchical nature of the proposed implementation, we also illustrate the architecture in Figure 1.

A final consideration during implementation is batching. The layer update in Equation 7 is written in pointwise form, but we can observe that the computation does not depend on other points $\{(x_j, u_j) : j \neq i\}$, and the weights are shared between points. Therefore, the forward and backward pass for each point can be done in parallel; in practice, the spatial dimension is reshaped into the batch dimension for efficient training and inference.

Hamiltonian Neural Solvers So far, integral kernel functionals are largely agnostic to the downstream task, only requiring that the input function u be discretized at a set of points (x_i, u_i) . To use functional approximators as a PDE solver, we introduce the Hamiltonian Neural Solver (HNS). The time evolution of PDEs can be described by Hamilton’s equations, which depend on the operator \mathcal{J} and the functional derivative $\delta \mathcal{H} / \delta u$. In Hamiltonian PDE systems, \mathcal{J} is analytically derived or can be looked up, is guaranteed to be linear, and is usually simple. A common example in 1D is $\mathcal{J} = \partial_x$. Therefore, we choose to approximate \mathcal{J} with a finite difference scheme, although it can also be learned with a neural operator. Lastly, we find that $\delta \mathcal{H} / \delta u$ can be computed directly with automatic differentiation. This is empirically shown in Section 4.1, but one way to see this is by considering the linear IKF. The gradient $\nabla_{\mathbf{u}} \sum_{i=1}^n \kappa_\theta(x_i) u_i \mu_i \Delta x = [\kappa_\theta(x_1), \dots, \kappa_\theta(x_n)]$, up to a constant factor, which is the discretization of the learned function $\kappa_\theta(x)$. In this case, the functional derivative is approximated by an arbitrary function, which is the desired behavior.

Within this framework, the training is described in Algorithm 1. For Hamiltonian systems, we assume knowledge of the analytical form of \mathcal{H} and $\delta \mathcal{H} / \delta u$, and labels for $\delta \mathcal{H} / \delta u$ or \mathbf{du} / dt can be calculated from training data using numerical schemes. The training loss can either be evaluated in the functional form $\delta \mathcal{H}_\theta / \delta u$ or the temporal form $\mathcal{J}(\delta \mathcal{H}_\theta / \delta u)$; we implement the former since it more directly optimizes κ_θ .

Algorithm 1 Training a HNS

- 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: $\mathcal{L} = \|\frac{\delta \mathcal{H}_\theta}{\delta u} - \frac{\delta \mathcal{H}}{\delta u}\|^2$ or $\|\mathcal{J}(\frac{\delta \mathcal{H}_\theta}{\delta u}) - \frac{\mathbf{du}}{dt}\|^2$
 - 5: $\theta \leftarrow \text{Update}(\theta, \nabla_\theta \mathcal{L})$
 - 6: **until** converged
-

During inference, the current state \mathbf{u}^t and coordinates \mathbf{x} are used in a forward pass to compute \mathcal{H}_θ . A backward pass is then used to calculate $\delta \mathcal{H}_\theta / \delta u$; using the operator \mathcal{J} , a prediction for $\frac{\mathbf{du}}{dt}|_{t=t} = \mathcal{J}(\frac{\delta \mathcal{H}_\theta}{\delta u})$ is calculated. The estimated derivative $\frac{\mathbf{du}}{dt}$ is used to update the state $\mathbf{u}^{t+1} = \text{ODEint}(\mathbf{u}^t, \frac{\mathbf{du}}{dt})$ using a numerical integrator. In our implementation, we use a 2nd-order Adams-Bashforth method.

Metric	MLP	Base			OOD ($c_i \in [1, 3]$)			Disc. ($x_i \in [-2, 2]$)		
		FNO	IKF		MLP	FNO	IKF	MLP	FNO	IKF
$\mathcal{F}_l[u]$	2.47e-5	2.76e-4	3.00e-16		0.046	0.268	2.13e-7	49.3	49.7	0.737
$\delta\mathcal{F}_l/\delta u$	0.083	0.066	1.15e-3		0.114	0.122	1.15e-3	2.64	2.70	0.077
$\mathcal{F}_{nl}[u]$	0.029	0.016	2.05e-3		6709	6493	2908	381	322	55.4
$\delta\mathcal{F}_{nl}/\delta u$	1.33	2.10	0.045		1730	1723	1079	92.1	88.1	35.5

Table 1: Validation MSE is reported across different experiments with linear and nonlinear functionals. Errors are calculated in the scalar $\mathcal{F}[u]$ and gradient domains $\delta\mathcal{F}/\delta u$ to evaluate performance in fitting functionals and implicitly learning functional derivatives. Additionally, generalization to OOD inputs and unseen discretizations are evaluated. Parameter counts are: MLP (7.5K), FNO (10.9K) and IKF (4.3K); each experiment is repeated 6 times and errors are reported as the average across seeds.

4 Experiments

4.1 Toy Examples

To better understand and situate integral kernel functionals, we examine their ability to model simple, analytically constructed functionals and compare their performance to other architectures. The experimental setup is as follows: given a functional $\mathcal{F}[u]$, random polynomials $u(x) = c_0x^p + c_1x^{p-1} + \dots + c_{p-1}x + c_p$ are generated by uniformly sampling $\{c_i \in [a, b] : i = 0, \dots, p\}$. Therefore, the dataset consists of N pairs of polynomials and evaluated functionals $(u^n(x), \mathcal{F}[u^n(x)])$ for $n = 1, \dots, N$. In practice, each function $u^n(x)$ is discretized at a set of points $\{x_i : i = 1, \dots, M\}$, such that it is represented as $\mathbf{u}^n = [u^n(x_1), \dots, u^n(x_M)]$, and $\mathcal{F}[\mathbf{u}^n]$ remains a scalar.

We consider two cases: a linear functional $\mathcal{F}_l[u] = \int_{x_1}^{x_M} u(x) * x^2 dx$ and a nonlinear functional $\mathcal{F}_{nl}[u] = \int_{x_1}^{x_M} (u(x))^3 dx$. In both cases, constructed polynomials $u^n(x)$ can be substituted and the definite integral can be analytically solved to use as training labels. Additionally, the functional derivatives are $\frac{\delta\mathcal{F}_l}{\delta u} = x^2$ and $\frac{\delta\mathcal{F}_{nl}}{\delta u} = 3u^2$, which are used to evaluate the gradient performance of models. We restrict the degree of $u^n(x)$ to $p = 2$ and sample $c_i \in [-1, 1]$; 100 training and 10 validation samples are generated for each case. Coordinates x_i are uniformly spaced on the domain $[-1, 1]$ with $M = 100$ points.

Two baselines are considered: (1) a **MLP** that takes concatenated inputs $[\mathbf{x}, \mathbf{u}] \in \mathbb{R}^{2M}$ and projects to an output $\mathcal{F}_\theta \in \mathbb{R}$ through hidden layers, and (2) a **FNO** with mean-pooling at the final layer to evaluate an operator baseline. The FNO baseline takes inputs with spatial and channel dimensions, or stacked inputs $[\mathbf{x}, \mathbf{u}] \in \mathbb{R}^{M \times 2}$ and projects to an output field $\mathbf{z} \in \mathbb{R}^M$, which is then pooled to $\mathcal{F}_\theta \in \mathbb{R}$. For simplicity, the integral kernel functional (**IKF**) is used with a local MLP kernel and is linear for \mathcal{F}_l and nonlinear for \mathcal{F}_{nl} . In the nonlinear case, FiLM is not used and $[x_i, u_i] \in \mathbb{R}^2$ is concatenated as input to κ_θ . Lastly, models are trained with the loss $\mathcal{L} = \sum_{n=1}^N \|\mathcal{F}[u^n(x)] - \mathcal{F}_\theta(\mathbf{u}^n, \mathbf{x})\|_2^2$; training is supervised with the functional to evaluate if models can implicitly learn accurate derivatives.

To further evaluate generalization, after training we consider two cases: (1) **OOD**, where validation inputs $u(x)$ are sampled with polynomial coefficients $c_i \in [1, 3]$ to model out-of-distribution functions, (2) **Disc.**, where validation inputs $u(x)$ are uniformly discretized on a larger domain $x_i \in [-2, 2]$ with a larger Δx , while keeping $M = 100$. In each experiment, validation errors are reported with the metric $\frac{1}{N} \sum_{n=1}^N \|\mathcal{F}[u^n] - \mathcal{F}_\theta(\mathbf{u}^n, \mathbf{x})\|_2^2$ or $\frac{1}{N} \sum_{n=1}^N \|\frac{\delta\mathcal{F}}{\delta u^n} - \text{autograd}(\mathcal{F}_\theta, \mathbf{u}^n)\|_2^2$ to measure performance in fitting functionals and learning accurate functional derivatives.

Results are given in Table 1. We observe that architectures based on an integral kernel can accurately represent functionals and their derivatives, outperforming other architectures. Despite only training on scalars $\mathcal{F}[u]$, IKFs can implicitly learn the correct functional derivatives, as plotted in Figure 2. This is new; even on toy problems, traditional architectures do not have well-behaved functional derivatives, as such, learning in the gradient domain for Hamiltonian systems would not be possible. Although the gradient performance is poor, conventional architectures can still fit functionals and the MSE for validation samples within the training distribution (Base) is low. When considering generalization performance, IKFs can readily generalize to unseen input functions as well as unseen discretizations in the linear case. This empirically observed generalization is theoretically motivated; the Riesz

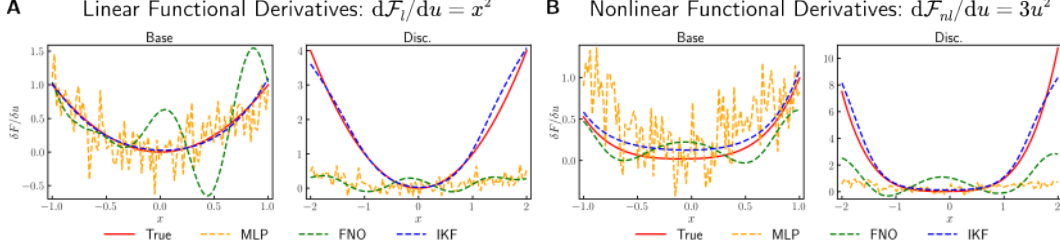


Figure 2: After training to regress $\mathcal{F}[u]$, the gradients of the network with respect to an input function u are plotted for different models. *Left, A:* Neural functionals are able to accurately model derivatives, even on unseen discretizations, whereas other architectures fail even on their training distribution. *Right, B:* Nonlinear functionals have more complex derivatives that change with input functions u . In this instance, a quartic function is learned and the model extrapolates to unseen discretizations.

representation theorem guarantees that a single kernel κ_θ can represent a linear functional \mathcal{F}_l for all possible input functions $u(x)$ (e.g., polynomial, exponential, etc.). Generalization performance with nonlinear functionals is more challenging, as larger polynomial coefficients c_i and larger coordinates x_i can quickly increase $F_{nl}[u(x)]$ due to large exponents. Despite this, IKFs still outperform other architectures in these domains, and learn accurate nonlinear functionals and their derivatives for samples within distribution.

4.2 1D Advection and KdV Equations

Formulation To evaluate the Hamiltonian Neural Solver (HNS) on PDE problems, we consider the 1D Advection and the 1D Korteweg–De Vries (KdV) equation. The Hamiltonians for each PDE are given as [21, 22]:

$$\mathcal{H}_{adv}[u] = \int_{\Omega} -\frac{1}{2}(u(x))^2 dx, \quad \mathcal{H}_{kdv}[u] = \int_{\Omega} -\frac{1}{6}(u(x))^3 - u(x) \frac{\partial^2 u}{\partial x^2}(x) dx, \quad (8)$$

Given an initial state u^0 , the Hamiltonian for each PDE is conserved over time (i.e., $\mathcal{H}[u^t] = \mathcal{H}[u^0]$). For both PDEs, the operator \mathcal{J} is defined as ∂_x . We can check that Hamilton’s equations hold for the 1D Advection equation:

$$\frac{\delta \mathcal{H}_{adv}}{\delta u} = -u(x), \quad \mathcal{J}\left(\frac{\delta \mathcal{H}_{adv}}{\delta u}\right) = -\frac{\partial u}{\partial x} = \frac{du}{dt} \quad (9)$$

which recovers the correct PDE for 1D Advection. A similar calculation can be done to recover the PDE for the 1D KdV equation ($u_t + uu_x + u_{xxx} = 0$) from Hamilton’s equations by using $\frac{\delta \mathcal{H}_{kdv}}{\delta u} = -\frac{1}{2}u^2 - u_{xx}$. Therefore, given a training dataset of spatiotemporal PDE data, the labels $\frac{\delta \mathcal{H}}{\delta u}$ can be computed using finite differences at every timestep t . The gradients of the HNS network $\nabla_{\mathbf{u}}(\mathcal{H}_\theta(\mathbf{x}, \mathbf{u}^t))$ are then fitted to these labels.

Evaluation Data for the 1D Advection and KdV equations are generated from a numerical solver using random initial conditions [23, 24]:

$$u^0(x) = \sum_{i=1}^J A_i * \sin\left(\frac{2\pi l_i * x}{L} + \phi_i\right) \quad (10)$$

where L is the length of the domain, $A_i \in [-0.5, 0.5]$, $l_i \in \{1, 2, 3\}$, and $\phi_i \in [0, 2\pi]$; $J = 5$ for Advection and $J = 10$ for KdV. The Advection equation is solved on a uniform domain $x_i \in [0, 16]$ with $n_x = 128$ and the KdV equation is solved on a domain $x_i \in [0, 100]$ with $n_x = 256$. Both equations have periodic boundary conditions. For the Advection equation, 1024/256 samples are generated for train/validation, and for the KdV equation, 2048/256 samples are generated. To evaluate temporal extrapolation, the training dataset is truncated to a shorter time horizon. For the Advection equation, it is solved from $t = 0s$ to $t = 4s$ during training ($n_t = 200$) and evaluated from $t = 0s$ to $t = 20s$ during validation ($n_t = 1000$); a similar scenario is constructed for the KdV equation with $t = 0s$ to $t = 25s$ for training ($n_t = 50$) and $t = 0s$ to $t = 100s$ for validation ($n_t = 200$).

	Adv	KdV
Metric:	Roll. Err. ↓	Corr. Time ↑
FNO	0.83±0.17	68.0±10.5
Unet	0.40±0.29	134.0±10.6
FNO($\frac{du}{dt}$)	0.048±0.007	77.6±3.6
Unet($\frac{du}{dt}$)	0.057±0.024	113.3±15.0
HNS	0.0039±0.0008	151.1±3.0

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

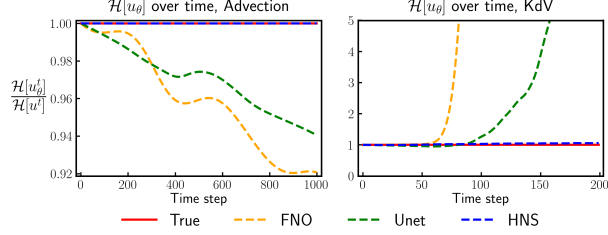


Figure 3: Hamiltonians for Adv and KdV for predicted trajectories, plotted over time. FNO and Unet models are shown with their du/dt variants. HNS can better conserve Hamiltonians and remain stable over time.

Two baselines are considered, a FNO [13] and a Unet [25] model. Additionally, the HNS model is used with a nonlinear, FiLM-conditioned SIREN kernel. Global conditioning is used in the KdV experiments by calculating the scale/shift parameters as $\gamma_i^{(l)} = [\gamma_\theta^{(l)}(\mathbf{u}^t)]_i$ and $\beta_i^{(l)} = [\beta_\theta^{(l)}(\mathbf{u}^t)]_i$; the global FiLM networks are each parameterized by a shallow, 1D CNN. We find that global conditioning is needed in more complex PDEs since the kernel output $z_i = k_\theta(\cdot)$ can depend on more than just u_i at a given point x_i . In particular, the Hamiltonian for the KdV equation has a u_{xx} term that requires non-local knowledge of \mathbf{u}^t . Lastly, to ensure a fair comparison, we also train variants of the FNO and Unet baselines that predict temporal derivatives $d\mathbf{u}/dt$ [26], as opposed to the usual variants that predict \mathbf{u}^{t+1} ; these use the same numerical integrator during inference as the HNS model.

The metrics that are reported are the rollout error $= \frac{1}{T} \sum_{t=1}^T \mathcal{L}(\mathbf{u}^t, \mathbf{u}_\theta^t)$, and the correlation time $= \max(t) \text{ s.t. } \mathcal{C}(\mathbf{u}^t, \mathbf{u}_\theta^t) > 0.8$. The loss used is a relative L2 loss, or $\mathcal{L}(\mathbf{u}^t, \mathbf{u}_\theta^t) = \frac{\|\mathbf{u}^t - \mathbf{u}_\theta^t\|_2^2}{\|\mathbf{u}^t\|_2^2}$ and the correlation criterion \mathcal{C} is the Pearson correlation. Rollout error is used to measure the accuracy of the predicted trajectory; for chaotic dynamics, rollout error is often skewed after autoregressive drift, therefore correlation time measures the portion of the trajectory that is accurate. Each metric is averaged over the validation set and the mean and standard deviation over six seeds is reported in Table 2. Furthermore, given a predicted trajectory $[\mathbf{u}_\theta^1, \dots, \mathbf{u}_\theta^T]$ from a model, we can examine its Hamiltonian at each timestep and plot the scalar value $\mathcal{H}[\mathbf{u}_\theta^t]$ over time, shown in Figure 3.

We observe that the proposed HNS model is able to outperform current baselines, both on quantitative metrics and when conserving Hamiltonians over time. HNS is also efficient, capable of achieving better performance with nearly half the parameters of baseline models due to sharing weights between input points x_i, u_i . Furthermore, despite training on a shorter time horizon, HNS can extrapolate and remain stable in longer prediction horizons. This is more pronounced in the KdV equation, where initial states are smoother than later, more chaotic states. An example trajectory is shown in Figure 5, where the baselines struggle to predict later states since these are not seen during training.

4.3 2D Shallow Water Equations

Formulation To evaluate HNS on a more complex system, we consider the 2D Shallow Water Equations (SWE). These equations are an approximation of the Navier-Stokes equations when the horizontal length scale is much larger than the vertical length scale. We consider its conservative form, which is often used as a simple model for atmosphere or ocean dynamics.

For vector-valued functions $\mathbf{u}(x, y) \in \mathbb{R}^d$, the operator \mathcal{J} becomes a $d \times d$ matrix of operators. For the 2D Shallow Water Equations, the operator matrix \mathcal{J} is given by [27–29]:

$$\mathcal{J} = \begin{bmatrix} 0 & -q & \partial_x \\ q & 0 & \partial_y \\ \partial_x & \partial_y & 0 \end{bmatrix}, \quad q = \frac{\partial v_x}{\partial y} - \frac{\partial v_y}{\partial x} \quad (11)$$

where q is the vorticity. The Hamiltonian for this system is:

$$\mathcal{H}[\mathbf{u}] = \mathcal{H}[v_x, v_y, h] = \int_{\Omega} \frac{1}{2} h(v_x^2 + v_y^2) + \frac{1}{2} g h^2 dA \quad (12)$$

where g is the gravitational constant and the velocities and height v_x, v_y, h are usually defined on a grid (i.e., $v_x = v_x(x_i, y_j)$). One can verify Hamilton’s equations and that $\mathcal{J} \frac{\delta \mathcal{H}}{\delta \mathbf{u}}$ recovers the 2D

Model	Sines	Pulse
Transolver	0.084 ± 0.0081	0.122 ± 0.0074
FNO	0.057 ± 0.002	0.117 ± 0.0009
PINO	0.053 ± 0.005	0.114 ± 0.0003
Unet	0.010 ± 0.0014	0.042 ± 0.0006
HNS	0.026 ± 0.0003	0.021 ± 0.0015

Table 3: Rollout errors for 2D SWE. Parameter counts are: FNO/PINO (7M), Transolver (4M), Unet (3M), HNS (3M). Models are evaluated on in-distribution ICs (Sines) or out-of-distribution ICs (Pulse).

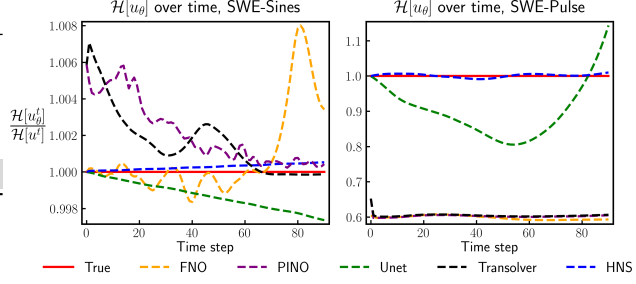


Figure 4: Hamiltonians for Sines and Pulse ICs, plotted over time. Despite potentially larger rollout errors, HNS can better conserve the Hamiltonian of predicted solutions.

Shallow Water Equations:

$$\partial_t h + \nabla \cdot (\mathbf{v}h) = 0, \quad \partial_t \mathbf{v} + \mathbf{v} \cdot \nabla \mathbf{v} = -g \nabla h \quad (13)$$

Evaluation Data for the 2D SWE system is generated from PyClaw [30–32] using random sinusoidal initial conditions (Sines) for the height:

$$h^0(x, y) = H + \sum_{i=1}^J A_i \sin\left(\frac{2\pi m_i * x}{L_x} + \frac{2\pi n_i * y}{L_y} + \phi_i\right) \quad (14)$$

with initial velocities set to zero. Constants are randomly sampled from $A_i \in [-0.1, 0.1]$, $m_i, n_i \in \{1, 2, 3\}$, and $\phi_i \in [0, 2\pi]$ with $J = 5$, $H = 1$, and $L_x = L_y = 2$. The domain is $(x_i, y_i) \in [-1, 1]^2$ with $n_x = n_y = 128$; each sample has $n_t = 100$ timesteps from $t = 0s$ to $t = 2s$. The gravitational constant is set to $g = 1$; 256 samples are generated for training and 64 samples are generated for validation. Periodic boundary conditions are used. To evaluate model generalization, we generate an additional test set of 64 samples with a random Gaussian pulse as initial conditions (Pulse). The initial height is set to an isotropic Gaussian centered at the origin with random covariance $\sigma \in [0.1, 0.5]$ and the height is scaled to be between $[1, 1.5]$; initial velocities are zero. The domain, BCs, and discretizations remain the same.

We consider benchmarking against Unet and FNO models, as well as PINO [33] and Transolver [34] models to compare against a physics-informed method and a newer attention-based surrogate. We instantiate the HNS model with a global, FiLM-conditioned SIREN kernel. Validation rollout errors for in-distribution ICs (Sines) and out-of-distribution ICs (Pulse) are given in Table 3, where means and standard deviations are calculated over three seeds. Additionally, the Hamiltonian over time for model rollouts are plotted in Figure 4. The derivative variants of baselines did not train stably, possibly due to sharp gradients that develop in the absence of viscosity, and are not reported. Additionally, the baselines are trained with the benefit of normalization, as the height channel has a different scale than the velocity channels; however, since the Hamiltonian is not invariant under scaling and shifting, normalization was not performed for the HNS model.

For test samples within the training distribution (Sines), conventional neural surrogates are able to perform well and achieve low loss, and HNS is on-par with these models. Adding a physics-informed loss in PINO only marginally increases the performance of FNO, which is consistent with its interpretation as a soft regularizer. When looking at the Hamiltonian of predicted solutions, we find that HNS can predict solutions that better conserve physical invariances. We validate this in Appendix B.1 by calculating the error of the Hamiltonian of predicted trajectories across different experiments. This ability to conserve the Hamiltonian helps HNS to generalize to unseen initial conditions, and it achieves better performance on the Gaussian Pulse test set. Interestingly, it also conserves the Hamiltonian on out-of-distribution inputs, suggesting a strong inductive bias in the model. To understand what mechanisms enable energy conservation, we also ablate different methods within HNS to understand their contributions, shown in Appendix B.1. Lastly, to visualize model performance, predicted trajectories of Sines and Pulse ICs are shown in Figures 6 and 7.

5 Related Works

Hamiltonian Neural Networks Original work on Hamiltonian Neural Networks (HNNs) [35, 36] established them as a promising architecture to respect physical laws, while later work improved their performance [37–41], investigated the mechanisms behind HNNs [42], or used HNNs in different applications [43–45]. A closely related set of works develop neural networks in a Lagrangian framework [46, 47], which also preserve learned energies. More broadly, learning an energy and optimizing its gradient is an established framework in ML-based molecular dynamics simulation [48–52]. Despite research in this area, no prior works have considered learning with infinite-dimensional Hamiltonian systems and most HNNs have only been demonstrated in simple, particle-based systems.

Functional Approximation While uncommon, there are prior works that investigate learning infinite-dimensional to scalar/vector maps. Most methods are based on a kernel integral and are used to fit functionals [2], construct functional autoencoders [4], model parameter-to-observable maps [3], or build infinite-dimensional discriminators for generative modeling [5]. The use of a kernel integral to model a functional is also well studied in machine learning for DFT, since the Hohenberg-Kohn Theorems guarantee that the total energy is the integral of an unknown function of electron density [53–55]. Lastly, a separate approach considers modeling functionals through the cylindrical approximation, rather than a kernel integral [56].

Neural PDE Solvers Neural PDE solvers are a growing field, with many works proposing architectures to improve model accuracy [57, 34, 58], generalizability [59–61], or stability [62–65]. Specific to our work, many PDE solvers have also employed physics-based prior knowledge to improve model performance and adherence to governing laws. The most straightforward approach is to use physics-informed losses [66, 33, 67], but another, more fundamental approach is to enforce physical symmetries and equivariances in the network itself [68–72]. Interestingly, by Noether’s theorem, every symmetry has an associated conservation law, which HNS relies on as an inductive bias; through this lens the current work is another way of softly enforcing symmetries in predictions.

6 Conclusion

Limitations Despite an interesting theoretical basis and its novelty, Hamiltonian Neural Solvers have limitations that we hope future work can address. Firstly, the theory for nonlinear functionals is underdeveloped. There is also additional overhead during inference to backpropagate through the network and evaluate the linear operator \mathcal{J} . Scalability is also a concern due to the reliance on numerical quadratures. This increase in runtime is quantified in Appendix B.2

Additionally, implementing \mathcal{J} without a neural operator may require care to avoid numerical errors; this is further discussed in Appendix D.1. Furthermore, HNS is only applicable to Hamiltonian systems; this would exclude systems with dissipation, although modifications exist to extend Hamiltonian mechanics to non-conservative systems [73–76]. Even for conservative PDEs, implementing a Hamiltonian structure is not as straightforward as training a neural solver with next-step prediction. Lastly, while not a limitation, an interesting observation is that introducing a loss on $\mathcal{H}[\mathbf{u}_0^t] - \mathcal{H}[\mathbf{u}_0^t]$ harms HNS performance; we hypothesize that since the Hamiltonian is conserved over time, learning the identity mapping is the easiest way to minimize this error and thus degrades model training.

Outlook In this work, we have proposed a novel method for designing neural PDE solvers that respect conservation laws. We verify that kernel integrals are able to implicitly learn functional derivatives, as well as propose parametrization using neural fields. Using the resulting architecture in a Hamiltonian framework allows stable and energy-conserving predictions of 1D and 2D PDEs. These capabilities enable Hamiltonian Neural Solvers to generalize to certain out-of-distribution inputs, such as a longer time horizon or unseen initial conditions.

While learning functions and operators are dominant paradigms, approximating functionals may also have interesting uses. Functionals provide concise descriptions of physical systems by integrating physical variables, such as describing the energy of a molecule or the drag of an airfoil. Although concise, functional derivatives are often more relevant and can contain dynamical information or perhaps be used for optimization. We hope that future work can continue to improve functional approximators as well as take advantage of these models in new and insightful applications.

Acknowledgments and Disclosure of Funding

We would like to thank Dr. Amit Acharya for his insightful discussions and help in conceptualizing this work.

References

- [1] Jerrold E. Marsden and Tudor S. Ratiu. *Introduction to Mechanics and Symmetry*. Springer New York, NY, 1999.
- [2] Zheyuan Hu, Tianbo Li, Zekun Shi, Kunhao Zheng, Giovanni Vignale, Kenji Kawaguchi, Shuicheng YAN, and Min Lin. Neural integral functionals. In *ICLR 2023 Workshop on Physics for Machine Learning*, 2023. URL <https://openreview.net/forum?id=aQuqw6eVKP>.
- [3] Daniel Zhengyu Huang, Nicholas H. Nelsen, and Margaret Trautner. An operator learning perspective on parameter-to-observable maps. *Foundations of Data Science*, 7(1):163–225, 2025. ISSN 2639-8001. doi: 10.3934/fods.2024037. URL <http://dx.doi.org/10.3934/fods.2024037>.
- [4] Justin Bunker, Mark Girolami, Hefin Lambley, Andrew M. Stuart, and T. J. Sullivan. Autoencoders in function space, 2025. URL <https://arxiv.org/abs/2408.01362>.
- [5] Md Ashiqur Rahman, Manuel A Florez, Anima Anandkumar, Zachary E Ross, and Kamyar Azizzadenesheli. Generative adversarial neural operators. *Transactions on Machine Learning Research*, 2022. ISSN 2835-8856. URL <https://openreview.net/forum?id=X1VzbBU6xZ>.
- [6] E. Kreyszig. *Introductory Functional Analysis with Applications*. Wiley Classics Library. Wiley, 1991. ISBN 9780471504597. URL <https://books.google.com/books?id=AQtMEAAQBAJ>.
- [7] Allan Pinkus. Approximation theory of the mlp model in neural networks. *Acta Numerica*, 8: 143–195, 1999. doi: 10.1017/S0962492900002919.
- [8] Moshe Leshno, Vladimir Ya. Lin, Allan Pinkus, and Shimon Schocken. Multilayer feed-forward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6(6):861–867, 1993. ISSN 0893-6080. doi: [https://doi.org/10.1016/S0893-6080\(05\)80131-5](https://doi.org/10.1016/S0893-6080(05)80131-5). URL <https://www.sciencedirect.com/science/article/pii/S0893608005801315>.
- [9] Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces with applications to pdes. *Journal of Machine Learning Research*, 24(89):1–97, 2023. URL <http://jmlr.org/papers/v24/21-1524.html>.
- [10] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Graph kernel network for partial differential equations, 2020. URL <https://arxiv.org/abs/2003.03485>.
- [11] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Andrew Stuart, Kaushik Bhattacharya, and Anima Anandkumar. Multipole graph neural operator for parametric partial differential equations. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 6755–6766. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/4b21cf96d4cf612f239a6c322b10c8fe-Paper.pdf.
- [12] Zongyi Li, Nikola Borislavov Kovachki, Chris Choy, Boyi Li, Jean Kossaifi, Shourya Prakash Otta, Mohammad Amin Nabian, Maximilian Stadler, Christian Hundt, Kamyar Azizzadenesheli, and Anima Anandkumar. Geometry-informed neural operator for large-scale 3d pdes, 2023. URL <https://arxiv.org/abs/2309.00583>.
- [13] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations, 2021. URL <https://arxiv.org/abs/2010.08895>.

- [14] Yiheng Xie, Towaki Takikawa, Shunsuke Saito, Or Litany, Shiqin Yan, Numair Khan, Federico Tombari, James Tompkin, Vincent Sitzmann, and Srinath Sridhar. Neural fields in visual computing and beyond, 2022. URL <https://arxiv.org/abs/2111.11426>.
- [15] Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys, and Andreas Geiger. Convolutional occupancy networks, 2020. URL <https://arxiv.org/abs/2003.04618>.
- [16] Ishit Mehta, Michaël Gharbi, Connelly Barnes, Eli Shechtman, Ravi Ramamoorthi, and Manmohan Chandraker. Modulated periodic activations for generalizable local functional representations, 2021. URL <https://arxiv.org/abs/2104.03960>.
- [17] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. DeepSDF: Learning continuous signed distance functions for shape representation, 2019. URL <https://arxiv.org/abs/1901.05103>.
- [18] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space, 2019. URL <https://arxiv.org/abs/1812.03828>.
- [19] Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer, 2017. URL <https://arxiv.org/abs/1709.07871>.
- [20] Vincent Sitzmann, Julien N. P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions, 2020. URL <https://arxiv.org/abs/2006.09661>.
- [21] Robert M. Miura. The korteweg–devries equation: A survey of results. *SIAM Review*, 18(3): 412–459, 1976. doi: 10.1137/1018076. URL <https://doi.org/10.1137/1018076>.
- [22] Peter D. Miller. The korteweg-de vries equation, 2018. URL https://dept.math.lsa.umich.edu/~millerpd/docs/651_Winter18/Topic02-651-W18.pdf.
- [23] Anthony Zhou and Amir Barati Farimani. Masked autoencoders are pde learners, 2024. URL <https://arxiv.org/abs/2403.17728>.
- [24] Johannes Brandstetter, Max Welling, and Daniel E. Worrall. Lie point symmetry data augmentation for neural pde solvers, 2022. URL <https://arxiv.org/abs/2202.07643>.
- [25] Jayesh K. Gupta and Johannes Brandstetter. Towards multi-spatiotemporal-scale generalized pde modeling, 2022. URL <https://arxiv.org/abs/2209.15616>.
- [26] Anthony Zhou and Amir Barati Farimani. Predicting change, not states: An alternate framework for neural pde surrogates. *Computer Methods in Applied Mechanics and Engineering*, 441: 117990, June 2025. ISSN 0045-7825. doi: 10.1016/j.cma.2025.117990. URL <http://dx.doi.org/10.1016/j.cma.2025.117990>.
- [27] J.S. Allen and Darryl D. Holm. Extended-geostrophic hamiltonian models for rotating shallow water motion. *Physica D: Nonlinear Phenomena*, 98(2):229–248, 1996. ISSN 0167-2789. doi: [https://doi.org/10.1016/0167-2789\(96\)00120-0](https://doi.org/10.1016/0167-2789(96)00120-0). URL <https://www.sciencedirect.com/science/article/pii/0167278996001200>. Nonlinear Phenomena in Ocean Dynamics.
- [28] P. Ripa. Conservation laws for primitive equations models with inhomogeneous layers. *Geophysical & Astrophysical Fluid Dynamics*, 70:85–111, 6 1993. ISSN 10290419. doi: 10.1080/03091929308203588.
- [29] Paul J. Dellar. Common hamiltonian structure of the shallow water equations with horizontal temperature gradients and magnetic fields. *Physics of Fluids*, 15:292–297, 2003. ISSN 10706631. doi: 10.1063/1.1530576.
- [30] Clawpack Development Team. Clawpack software, 2024. URL <http://www.clawpack.org>. Version 5.11.0.

- [31] David I. Ketcheson, Kyle T. Mandli, Aron J. Ahmadi, Amal Alghamdi, Manuel Quezada de Luna, Matteo Parsani, Matthew G. Knepley, and Matthew Emmett. PyClaw: Accessible, Extensible, Scalable Tools for Wave Propagation Problems. *SIAM Journal on Scientific Computing*, 34(4):C210–C231, November 2012.
- [32] Kyle T Mandli, Aron J Ahmadi, Marsha Berger, Donna Calhoun, David L George, Yiannis Hadjimichael, David I Ketcheson, Grady I Lemoine, and Randall J LeVeque. Clawpack: building an open source ecosystem for solving hyperbolic pdes. *PeerJ Computer Science*, 2: e68, 2016. doi: 10.7717/peerj-cs.68.
- [33] Zongyi Li, Hongkai Zheng, Nikola Kovachki, David Jin, Haoxuan Chen, Burigede Liu, Kamyar Azizzadenesheli, and Anima Anandkumar. Physics-informed neural operator for learning partial differential equations, 2023. URL <https://arxiv.org/abs/2111.03794>.
- [34] Haixu Wu, Huakun Luo, Haowen Wang, Jianmin Wang, and Mingsheng Long. Transolver: A fast transformer solver for pdes on general geometries, 2024. URL <https://arxiv.org/abs/2402.02366>.
- [35] Sam Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian neural networks, 2019. URL <https://arxiv.org/abs/1906.01563>.
- [36] Tom Bertalan, Felix Dietrich, Igor Mezić, and Ioannis G. Kevrekidis. On learning hamiltonian systems from data. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 29(12):121107, 12 2019. ISSN 1054-1500. doi: 10.1063/1.5128231. URL <https://doi.org/10.1063/1.5128231>.
- [37] Marco David and Florian Méhats. Symplectic learning for hamiltonian neural networks. *Journal of Computational Physics*, 494:112495, 2023. ISSN 0021-9991. doi: <https://doi.org/10.1016/j.jcp.2023.112495>. URL <https://www.sciencedirect.com/science/article/pii/S0021999123005909>.
- [38] Philipp Horn, Veronica Saz Ulibarrena, Barry Koren, and Simon Portegies Zwart. A generalized framework of neural networks for hamiltonian systems. *Journal of Computational Physics*, 521:113536, 2025. ISSN 0021-9991. doi: <https://doi.org/10.1016/j.jcp.2024.113536>. URL <https://www.sciencedirect.com/science/article/pii/S0021999124007848>.
- [39] Yaofeng Desmond Zhong, Biswadip Dey, and Amit Chakraborty. Symplectic ode-net: Learning hamiltonian dynamics with control. In *International Conference on Learning Representations*, 2020. URL <https://openreview.net/forum?id=ryxmb1rKDS>.
- [40] Zhengdao Chen, Jianyu Zhang, Martin Arjovsky, and Léon Bottou. Symplectic recurrent neural networks. In *International Conference on Learning Representations*, 2020. URL <https://openreview.net/forum?id=BkgYPREtPr>.
- [41] Alvaro Sanchez-Gonzalez, Victor Bapst, Kyle Cranmer, and Peter Battaglia. Hamiltonian graph networks with ode integrators, 2019. URL <https://arxiv.org/abs/1909.12790>.
- [42] Nate Gruver, Marc Finzi, Samuel Stanton, and Andrew Gordon Wilson. Deconstructing the inductive biases of hamiltonian neural networks, 2022. URL <https://arxiv.org/abs/2202.04836>.
- [43] Marios Mattheakis, David Sondak, Akshunna S. Dogra, and Pavlos Protopapas. Hamiltonian neural networks for solving equations of motion. *Physical Review E*, 105(6), June 2022. ISSN 2470-0053. doi: 10.1103/physreve.105.065305. URL <http://dx.doi.org/10.1103/PhysRevE.105.065305>.
- [44] Peter Toth, Danilo J. Rezende, Andrew Jaegle, Sébastien Racanière, Aleksandar Botev, and Irina Higgins. Hamiltonian generative networks. In *International Conference on Learning Representations*, 2020. URL <https://openreview.net/forum?id=HJenn6VFvB>.
- [45] Yaofeng Desmond Zhong, Biswadip Dey, and Amit Chakraborty. Extending lagrangian and hamiltonian neural networks with differentiable contact models, 2021. URL <https://arxiv.org/abs/2102.06794>.

- [46] Miles Cranmer, Sam Greydanus, Stephan Hoyer, Peter Battaglia, David Spergel, and Shirley Ho. Lagrangian neural networks. In *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations*, 2019. URL <https://openreview.net/forum?id=iE8tFa4Nq>.
- [47] Michael Lutter, Christian Ritter, and Jan Peters. Deep lagrangian networks: Using physics as model prior for deep learning, 2019. URL <https://arxiv.org/abs/1907.04490>.
- [48] Stefan Chmiela, Alexandre Tkatchenko, Huziel E. Sauceda, Igor Poltavsky, Kristof T. Schütt, and Klaus-Robert Müller. Machine learning of accurate energy-conserving molecular force fields. *Science Advances*, 3(5):e1603015, 2017. doi: 10.1126/sciadv.1603015. URL <https://www.science.org/doi/abs/10.1126/sciadv.1603015>.
- [49] Johannes Gasteiger, Janek Groß, and Stephan Günnemann. Directional message passing for molecular graphs, 2022. URL <https://arxiv.org/abs/2003.03123>.
- [50] Kristof T. Schütt, Pieter-Jan Kindermans, Huziel E. Sauceda, Stefan Chmiela, Alexandre Tkatchenko, and Klaus-Robert Müller. Schnet: A continuous-filter convolutional neural network for modeling quantum interactions, 2017. URL <https://arxiv.org/abs/1706.08566>.
- [51] Linfeng Zhang, Jiequn Han, Han Wang, Wissam A. Saidi, Roberto Car, and Weinan E. End-to-end symmetry preserving inter-atomic potential energy model for finite and extended systems, 2018. URL <https://arxiv.org/abs/1805.09003>.
- [52] Simon Batzner, Albert Musaelian, Lixin Sun, Mario Geiger, Jonathan P. Mailoa, Mordechai Kornbluth, Nicola Molinari, Tess E. Smidt, and Boris Kozinsky. E(3)-equivariant graph neural networks for data-efficient and accurate interatomic potentials. *Nature Communications*, 13(1), May 2022. ISSN 2041-1723. doi: 10.1038/s41467-022-29939-5. URL <http://dx.doi.org/10.1038/s41467-022-29939-5>.
- [53] Ryan Pederson, Bhupalee Kalita, and Kieron Burke. Machine learning and density functional theory. *Nature Reviews Physics*, 4(6):357–358, May 2022. ISSN 2522-5820. doi: 10.1038/s42254-022-00470-2. URL <http://dx.doi.org/10.1038/s42254-022-00470-2>.
- [54] Tianbo Li, Min Lin, Zheyuan Hu, Kunhao Zheng, Giovanni Vignale, Kenji Kawaguchi, A.H. Castro Neto, Kostya S. Novoselov, and Shuicheng YAN. D4FT: A deep learning approach to kohn-sham density functional theory. In *The Eleventh International Conference on Learning Representations*, 2023. URL <https://openreview.net/forum?id=aBwnqqsuot7>.
- [55] L. Fiedler, K. Shah, M. Bussmann, and A. Cangi. Deep dive into machine learning density functional theory for materials science and chemistry. *Phys. Rev. Mater.*, 6:040301, Apr 2022. doi: 10.1103/PhysRevMaterials.6.040301. URL <https://link.aps.org/doi/10.1103/PhysRevMaterials.6.040301>.
- [56] Taiki Miyagawa and Takeru Yokota. Physics-informed neural networks for functional differential equations: Cylindrical approximation and its convergence guarantees. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL <https://openreview.net/forum?id=H5z0XqEX57>.
- [57] Phillip Lippe, Bastiaan S. Veeling, Paris Perdikaris, Richard E. Turner, and Johannes Brandstetter. Pde-refiner: Achieving accurate long rollouts with neural pde solvers, 2023. URL <https://arxiv.org/abs/2308.05732>.
- [58] Benedikt Alkin, Andreas Fürst, Simon Schmid, Lukas Gruber, Markus Holzleitner, and Johannes Brandstetter. Universal physics transformers: A framework for efficiently scaling neural operators, 2025. URL <https://arxiv.org/abs/2402.12365>.
- [59] Anthony Zhou, Cooper Lorsung, AmirPouya Hemmasian, and Amir Barati Farimani. Strategies for pretraining neural operators, 2024. URL <https://arxiv.org/abs/2406.08473>.
- [60] Zhongkai Hao, Chang Su, Songming Liu, Julius Berner, Chengyang Ying, Hang Su, Anima Anandkumar, Jian Song, and Jun Zhu. Dpot: Auto-regressive denoising operator transformer for large-scale pde pre-training, 2024. URL <https://arxiv.org/abs/2403.03542>.

- [61] Maximilian Herde, Bogdan Raonić, Tobias Rohner, Roger Käppeli, Roberto Molinaro, Emmanuel de Bézenac, and Siddhartha Mishra. Poseidon: Efficient foundation models for pdes, 2024. URL <https://arxiv.org/abs/2405.19101>.
- [62] Anthony Zhou, Zijie Li, Michael Schneier, John R Buchanan Jr, and Amir Barati Farimani. Text2pde: Latent diffusion models for accessible physics simulation, 2025. URL <https://arxiv.org/abs/2410.01153>.
- [63] Zongyi Li, Miguel Liu-Schiaffini, Nikola Kovachki, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Learning dissipative dynamics in chaotic systems, 2022. URL <https://arxiv.org/abs/2106.06898>.
- [64] Bjoern List, Li-Wei Chen, Kartik Bali, and Nils Thuerey. Differentiability in unrolled training of neural physics simulators on transient dynamics. *Computer Methods in Applied Mechanics and Engineering*, 433:117441, 2025. ISSN 0045-7825. doi: <https://doi.org/10.1016/j.cma.2024.117441>. URL <https://www.sciencedirect.com/science/article/pii/S0045782524006960>.
- [65] Zijie Li, Anthony Zhou, and Amir Barati Farimani. Generative latent neural pde solver using flow matching, 2025. URL <https://arxiv.org/abs/2503.22600>.
- [66] M. Raissi, P. Perdikaris, and G.E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019. ISSN 0021-9991. doi: <https://doi.org/10.1016/j.jcp.2018.10.045>. URL <https://www.sciencedirect.com/science/article/pii/S0021999118307125>.
- [67] Sifan Wang, Hanwen Wang, and Paris Perdikaris. Learning the solution operator of parametric partial differential equations with physics-informed deepnets, 2021. URL <https://arxiv.org/abs/2103.10974>.
- [68] Zakhar Shumaylov, Peter Zaika, James Rowbottom, Ferdia Sherry, Melanie Weber, and Carola-Bibiane Schönlieb. Lie algebra canonicalization: Equivariant neural operators under arbitrary lie groups, 2025. URL <https://arxiv.org/abs/2410.02698>.
- [69] Rui Wang, Robin Walters, and Rose Yu. Incorporating symmetry into deep dynamics models for improved generalization. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=wta_8Hx2KD.
- [70] Johannes Brandstetter, Rianne van den Berg, Max Welling, and Jayesh K. Gupta. Clifford neural layers for pde modeling, 2023. URL <https://arxiv.org/abs/2209.04934>.
- [71] Tara Akhound-Sadegh, Laurence Perreault-Levasseur, Johannes Brandstetter, Max Welling, and Siamak Ravanbakhsh. Lie point symmetry and physics informed networks, 2023. URL <https://arxiv.org/abs/2311.04293>.
- [72] Masanobu Horie and Naoto Mitsume. Physics-embedded neural networks: Graph neural pde solvers with mixed boundary conditions, 2023. URL <https://arxiv.org/abs/2205.11912>.
- [73] Andrew Sosanya and Sam Greydanus. Dissipative hamiltonian neural networks: Learning dissipative and conservative dynamics separately, 2022. URL <https://arxiv.org/abs/2201.10085>.
- [74] Harsh Bhatia, Gregory Norgard, Valerio Pascucci, and Peer-Timo Bremer. The helmholtz-hodge decomposition—a survey. *IEEE Transactions on Visualization and Computer Graphics*, 19(8): 1386–1404, 2013. doi: 10.1109/TVCG.2012.316.
- [75] John W. Sanders, A.C. DeVoria, Nathan J. Washuta, Gafar A. Elamin, Kevin L. Skenes, and Joel C. Berlinghieri. A canonical hamiltonian formulation of the navier–stokes problem. *Journal of Fluid Mechanics*, 984:A27, 2024. doi: 10.1017/jfm.2024.229.
- [76] Alexander Figotin and Jeffrey H. Schenker. Hamiltonian structure for dispersive and dissipative dynamical systems. *Journal of Statistical Physics*, 128(4):969–1056, June 2007. ISSN 1572-9613. doi: 10.1007/s10955-007-9321-1. URL <http://dx.doi.org/10.1007/s10955-007-9321-1>.

- [77] Kaushik Bhattacharya, Bamdad Hosseini, Nikola B. Kovachki, and Andrew M. Stuart. Model reduction and neural networks for parametric pdes, 2021. URL <https://arxiv.org/abs/2005.03180>.
- [78] Samuel Lanthaler, Siddhartha Mishra, and George Em Karniadakis. Error estimates for deep-onets: A deep learning framework in infinite dimensions, 2022. URL <https://arxiv.org/abs/2102.09618>.
- [79] Nikola Kovachki, Samuel Lanthaler, and Siddhartha Mishra. On universal approximation and error bounds for fourier neural operators. *Journal of Machine Learning Research*, 22(290): 1–76, 2021. URL <http://jmlr.org/papers/v22/21-0806.html>.
- [80] Taiki Miyagawa and Takeru Yokota. Physics-informed neural networks for functional differential equations: Cylindrical approximation and its convergence guarantees. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 72274–72409. Curran Associates, Inc., 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/84c6c76526783f4afd82687829d3aa97-Paper-Conference.pdf.
- [81] Daniele Venturi and Alec Dektor. Spectral methods for nonlinear functionals and functional differential equations. *Research in the Mathematical Sciences*, 8(2):27, Apr 2021. ISSN 2197-9847. doi: 10.1007/s40687-021-00265-4. URL <https://doi.org/10.1007/s40687-021-00265-4>.
- [82] K. O. Friedrichs and H. N. Shapiro. Integration over hilbert space and outer extensions. *Proceedings of the National Academy of Sciences*, 43(4):336–338, 1957. doi: 10.1073/pnas.43.4.336. URL <https://www.pnas.org/doi/abs/10.1073/pnas.43.4.336>.
- [83] Randall J. LeVeque and David L. George. *High-resolution Finite Volume Methods for the Shallow Water Equations with Bathymetry and Dry States*, pages 43–73. doi: 10.1142/9789812790910_0002. URL https://www.worldscientific.com/doi/abs/10.1142/9789812790910_0002.
- [84] Nguyen Ba Hoai Linh and Dao Huy Cuong. A well-balanced finite volume scheme based on planar riemann solutions for 2d shallow water equations with bathymetry. *Applied Mathematics and Computation*, 457:128167, 2023. ISSN 0096-3003. doi: <https://doi.org/10.1016/j.amc.2023.128167>. URL <https://www.sciencedirect.com/science/article/pii/S0096300323003363>.
- [85] Pavel Holoborodko. Smooth noise robust differentiators, 2008.
- [86] Ami Harten, Bjorn Engquist, Stanley Osher, and Sukumar R. Chakravarthy. Uniformly high order accurate essentially non-oscillatory schemes, iii. *Journal of Computational Physics*, 131(1):3–47, 1997. ISSN 0021-9991. doi: <https://doi.org/10.1006/jcph.1996.5632>. URL <https://www.sciencedirect.com/science/article/pii/S0021999196956326>.
- [87] Xu-Dong Liu, Stanley Osher, and Tony Chan. Weighted essentially non-oscillatory schemes. *Journal of Computational Physics*, 115(1):200–212, 1994. ISSN 0021-9991. doi: <https://doi.org/10.1006/jcph.1994.1187>. URL <https://www.sciencedirect.com/science/article/pii/S0021999184711879>.
- [88] Kumar Rahul and S.N. Bhattacharyya. One-sided finite-difference approximations suitable for use with richardson extrapolation. *Journal of Computational Physics*, 219(1): 13–20, 2006. ISSN 0021-9991. doi: <https://doi.org/10.1016/j.jcp.2006.05.035>. URL <https://www.sciencedirect.com/science/article/pii/S002199910600266X>.

A Additional Visualizations

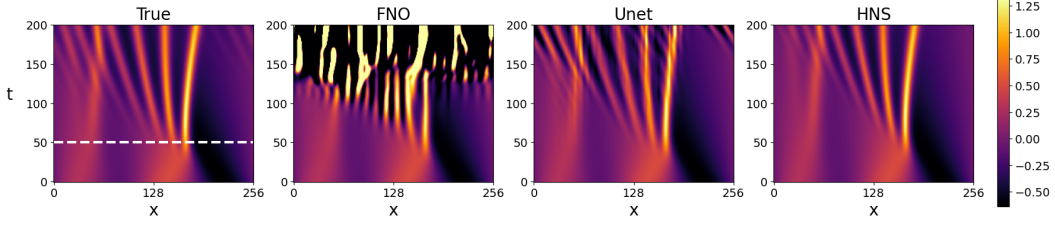


Figure 5: Predicted trajectories of the KdV equation. The white dashed line denotes the training horizon, where only states from $t = 0$ to $t = 50$ are seen during training. Despite not seeing later, more chaotic states, HNS still extrapolates beyond the training horizon by conserving the Hamiltonian, while other models struggle. FNO and Unet models are shown with their du/dt variants.

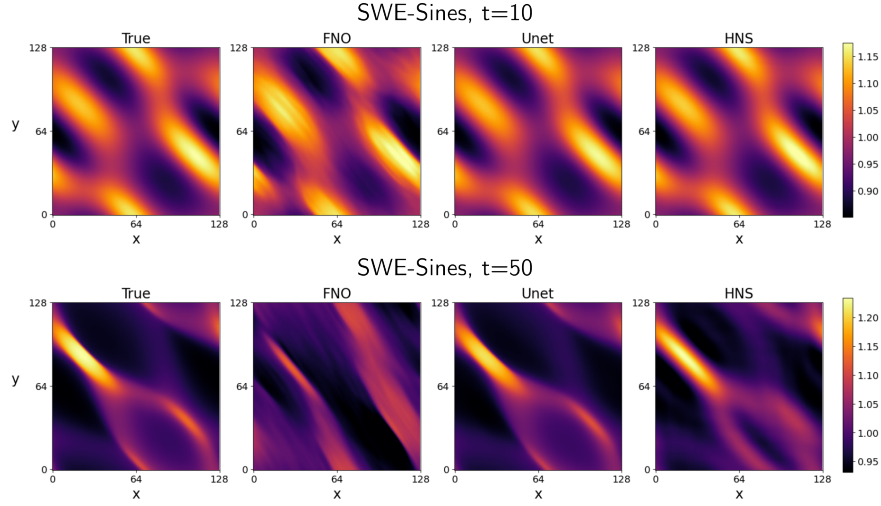


Figure 6: Predicted trajectories of the Shallow Water Equations (SWE) with sinusoidal ICs, displayed at $t = 10$ and $t = 50$. All models are able to achieve good performance on in-distribution samples.

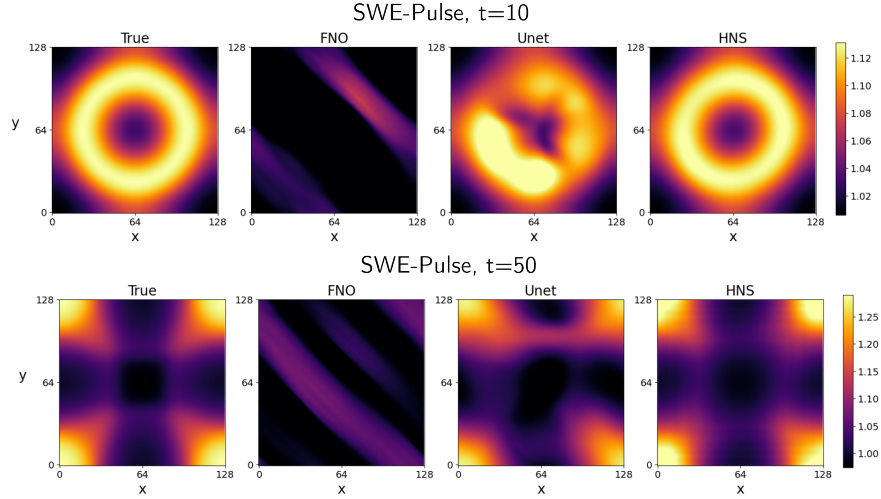


Figure 7: Predicted SWE trajectories with Gaussian pulse ICs, displayed at $t = 10$ and $t = 50$. Despite showing different behavior, HNS can better generalize to OOD samples.

B Additional Results

B.1 Hamiltonian Errors and Inductive Bias

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	-	-
HNS	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, HNS is exceptional at predicting solutions that conserve the Hamiltonian.

In the main body, we presented qualitative evidence that HNS is able to better conserve the Hamiltonian of predicted trajectories. To quantify this claim, we calculate the Relative L2 Error of the Hamiltonian of predicted trajectories across all PDE experiments. Specifically, the error of a sample is given by: $\frac{1}{T} \sum_{t=1}^T \frac{\|\mathcal{H}[\mathbf{u}^t] - \mathcal{H}[\mathbf{u}_\theta^t]\|_2^2}{\|\mathcal{H}[\mathbf{u}^t]\|_2^2}$, where \mathbf{u}^t is the true solution at time t , \mathbf{u}_θ^t is the predicted solution at time t , and $\mathcal{H}[\cdot]$ is the Hamiltonian given by the integrals in Equations 8 and 12. The Hamiltonian errors are averaged across the validation sets and reported in Table 4. We note that in the KdV experiments, the dynamics are chaotic and the Hamiltonian is highly nonlinear, both of which contribute to large errors once predicted trajectories diverge. Overall, we find that HNS has exceptional capabilities in implicitly conserving energy-like quantities in their prediction.

Interestingly, this capability exists even when not trained with a loss on the Hamiltonian (i.e., $\|\mathcal{H}[\mathbf{u}^t] - \mathcal{H}[\mathbf{u}_\theta^t]\|_2^2$) and when predicting out-of-distribution trajectories. Where does this inductive bias to conserve the Hamiltonian come from? Based on Gruver et al. [42], we hypothesize four inductive biases in HNS that may contribute to this:

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

We note that these biases are listed from simplest to most complex and are cumulative; for example, it is not possible to implement Hamiltonian structure without also implementing an ODE integrator. We modify a Unet to incorporate each inductive bias and evaluate its performance on the KdV equation, which is the most chaotic setting. The ODE bias was considered in the main work by training $d\mathbf{u}/dt$ model variants, while incorporating Hamiltonian bias is done by changing the training label to $\delta \mathcal{H}/\delta \mathbf{u}$ and using \mathcal{J} during inference. To extend Unets to gradient learning, we add a linear head to project the output field to a scalar and use automatic differentiation to obtain $\delta \mathcal{H}/\delta \mathbf{u}$. After training, we evaluate model correlation time and Hamiltonian error on the validation set and the results are reported in Table 5.

We find that the main contributors to HNS performance and its ability to conserve energies over time is using Hamiltonian structure as well as the neural functional architecture. Using an ODE integrator alone degrades model performance and implementing gradient learning with Unets also harms performance. Interestingly, using gradient learning with conventional architectures disproportionately harms energy conservation; this implies that in PDEs, gradient domain learning improves energy conservation only when architectures have meaningful functional derivatives. This is not readily apparent, as observations from molecular dynamics or HNNs suggest that gradient domain learning generally improves energy conservation without considering the underlying architectures [35, 46, 48]. Based on these findings, we hypothesize the following mechanism underlying HNS: integral kernel functionals enable the learning of functional derivatives, combining this with infinite-dimensional Hamiltonian mechanics leads to its performance and conservation properties.

Metric:	Correlation Time (\uparrow)	Hamiltonian Error (\downarrow)
Unet (Base)	125.25	2.37
Unet (ODE)	120.5	5.76e6
Unet (Ham.)	143.75	2.06
Unet (Grad.)	141	4.71
HNS	151.75	1.32

Table 5: Correlation time and Hamiltonian errors for Unet models with increasingly more inductive biases, compared to HNS 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.

B.2 Timing Experiments

Model (#Params)	Adv	KdV	SWE-Sines
FNO (65K/135K/7M)	0.083	0.091	0.967
Unet (65K/146K/3M)	0.138	0.146	1.345
HNS (32K/87K/3M)	0.126	0.228	4.547
Numerical Solver	0.000	110.9	33.26

Table 6: Computational cost per inference step for each model across different PDEs, given in milliseconds (ms). Model sizes are also reported for the (Adv/Kdv/SWE) experiments. Each run uses a single NVIDIA RTX 6000 Ada GPU. Numerical solvers are run on a AMD Ryzen Threadripper PRO 5975WX 32-Core CPU due to lack of GPU compatibility.

The additional overhead of using automatic differentiation, evaluating the operator \mathcal{J} , and applying an ODE integrator during inference increases the computational cost of HNS. We examine this in Table 6 by performing a model rollout and averaging the time per inference step across a batch. Additionally, we report the runtimes of the numerical solvers used. In the case of 1D Advection, the analytical solution is used. Randomly sampled initial conditions can cause variability in the stiffness of the solution, therefore, solver runtimes are averaged across 10 samples. We note that the 2D SWE solver (PyClaw) [31] uses a Fortran compiler and is optimized for hyperbolic PDEs, while the KdV solver is purely Python-based. We note that for a fairer comparison, the numerical solvers should be coarsened to match the accuracy of the neural solvers.

Consistent with prior works, FNO models remain a fast neural PDE solver. While HNS may have a lower parameter count, this may be a misleading measure of inference speed. Automatic differentiation calculates the gradient with respect to each input point, therefore the computational cost of this step can scale with the size of the input grid. This can be seen when comparing the Adv and KdV experiments, where the grid size doubles from $n_x = 128$ to $n_x = 256$, and the computational cost roughly doubles as well. In 2D experiments, the cost of HNS is appreciably more than other baselines, although it is still modestly faster than an optimized numerical solver. We do not anticipate that the finite-difference (FD) schemes used in \mathcal{J} or ODE integrators contribute meaningfully to computational cost, since FD schemes use $\mathcal{O}(M)$ operations, where M is the number of grid points, and numerical integration occurs in constant time.

B.3 Ablation Studies

There are many choices for parameterizing a kernel integral, such as from different kernels (linear, nonlinear), different architectures (MLP, SIREN), different conditioning mechanisms (concat/FiLM) and different receptive fields (local, global). Since this design space is large, we consider the effects of incrementally improving each aspect and report the performance of different HNS models in Table 7. We find that a nonlinear kernel is needed in all cases, since all the Hamiltonians tested are nonlinear. In addition, the use of FiLM conditioning and SIRENs tend to improve performance. Lastly, global conditioning is necessary for the KdV equation, which has nonlocal terms ($\frac{\partial^2 u}{\partial x^2}$) in the Hamiltonian.

Model	Rollout Error	Model	Correlation Time	Model	Rollout Error
Base	1.248	Base	51.75	Base	0.113
+ Nonlinear	0.028	+ Nonlinear	73.00	+ Nonlinear	0.034
+ FiLM	0.016	+ FiLM	65.25	+ FiLM	0.030
+ SIREN	0.003	+ SIREN	73.25	+ SIREN	0.026
+ Global	0.057	+ Global	151.75	+ Global	0.024

(a) Ablations for 1D Advection (b) Ablations for 1D KdV (c) Ablations for 2D SWE-Sines

Table 7: Ablation studies for the HNS architecture. The base architecture is specified by (Linear, Concat, SIREN, Local). Each successive row is cumulative, adding an additional feature to the model. Parameter sizes are approximately constant across ablations.

B.4 Varying Dataset Sizes

We study the effect of varying the number of samples in the training dataset on HNS performance in Table 8. The training dataset is truncated by a factor of 8, 4, or 2, and evaluated on the same validation dataset. This is repeated across all experiments/PDEs. In general, increasing the dataset size consistently improves model performance. This is more acute for 1D equations. For 2D SWE, the extra spatial dimension may allow for more data per sample relative to the complexity of the dynamics.

Dataset Size	Adv	KdV	SWE-Sines	SWE-Pulse
$f=1/8$	1.320	71.00	0.0293	0.0243
$f=1/4$	0.302	71.25	0.0274	0.0230
$f=1/2$	0.075	95.25	0.0254	0.0227
$f=1$	0.003	151.75	0.0249	0.0216

Table 8: Rollout error (\downarrow) or correlation times (\uparrow) of HNS models trained on datasets downsampled by a factor of f (e.g., $f = 1/2$ uses half the number of training samples in the training set).

B.5 Discretization Invariance

We experiment with querying models at different discretizations than its training resolution. The results are in Table 9, with the training resolution in bold. Rollout errors are averaged over the validation set, where validation labels are either downsampled or re-solved at a higher resolution using the same initial conditions. Additionally, for 1D Advection, we conducted an experiment in which the model is queried on an unseen grid $x = [0, 32]$, $n_x = 256$, after being trained on a grid $x = [0, 16]$, $n_x = 128$, denoted as 256*.

(a) Rollout Error at Different Resolutions, 1D Adv				(b) Rollout Error at Different Resolutions, 2D SWE (Sines)				(c) Rollout Error at Different Resolutions, 2D SWE (Pulse)			
Resolution	HNF	FNO	Unet	Resolution	HNF	FNO	Unet	Resolution	HNF	FNO	Unet
64	0.0029	0.064	1.19	(64, 64)	0.028	0.066	0.098	(64, 64)	0.035	0.173	0.173
128	0.0033	0.064	0.075	(128, 128)	0.029	0.066	0.008	(128, 128)	0.024	0.174	0.117
256	0.0055	0.050	1.32	(256, 256)	0.026	0.066	0.110	(256, 256)	0.019	0.174	0.225
256*	0.0021	1.34	0.072								

Table 9: Rollout error of models at different resolutions for 1D Adv and 2D SWE datasets.

In all cases, HNFs are capable of zero-shot super-resolution, as is possible with Neural Operators. Additionally, the error is approximately constant across discretizations and on a new, unseen grid for 1D Advection. Predictably, FNO models have nearly constant error across discretizations. Interestingly, for an extrapolated grid in 1D Advection, FNO struggles to make predictions. For Unet

models, the error increases when the grid spacing changes, however for a constant spacing, it is able to extrapolate to unseen grids (i.e. $[0, 16] \rightarrow [0, 32]$ with dx unchanged).

C Additional Theory

C.1 Proof of Linear Functional Approximation

Interestingly, while development of neural functionals was arrived upon independently, a proof of linear functional approximation exists. In fact, approximating a linear functional with an integral kernel is an intermediate step in operator approximation theorems [77–79] and its proof comes as a series of lemmas in Kovachki et al. [9]. For the sake of completeness, we provide relevant definitions and cite the necessary lemmas here.

Definitions (Appendix A, Kovachki et al. [9]) Let $D \subset \mathbb{R}^d$ be a domain. Let $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$ denote the natural numbers that include zero. Let \mathcal{X} be a Banach space, and \mathcal{X}^* be its continuous dual space. Specifically, the dual space contains all continuous, linear functionals $f : \mathcal{X} \rightarrow \mathbb{R}$ with the norm: $\|f\|_{\mathcal{X}^*} = \sup_{x \in \mathcal{X}, \|x\|_{\mathcal{X}}=1} |f(x)| < \infty$.

For any multi-index $\alpha \in \mathbb{N}_0^d$, ∂^α is the α -th partial derivative of f . The following spaces are defined for $m \in \mathbb{N}_0$:

- $C(D) = \{f : D \rightarrow \mathbb{R} : f \text{ is continuous}\}$
- $C^m(D) = \{f : D \rightarrow \mathbb{R} : \partial^\alpha f \in C^{m-|\alpha|_1}(D) \forall 0 \leq |\alpha|_1 \leq m\}$
- $C_c^\infty(D) = \{f \in C^\infty(D) : \text{supp}(f) \subset D \text{ is compact}\}$
- $C_b^m(D) = \{f \in C^m(D) : \max_{0 \leq |\alpha|_1 \leq m} \sup_{x \in D} |\partial^\alpha f(x)| < \infty\}$
- $C^m(\bar{D}) = \{f \in C_b^m(D) : \partial^\alpha f \text{ is uniformly continuous } \forall 0 \leq |\alpha|_1 \leq m\}$

Additionally, we cite definitions of neural networks from Section 8.1, Kovachki et al. [9]. For any $n \in \mathbb{N}$ and $\sigma : \mathbb{R} \rightarrow \mathbb{R}$, a neural network with n layers is given by:

$$\begin{aligned} N_n(\sigma; \mathbb{R}^d, \mathbb{R}^{d'}) &:= \{f : \mathbb{R}^d \rightarrow \mathbb{R}^{d'} : f(x) = W_n \sigma(\dots W_1 \sigma(W_0 x + b_0) + b_1 \dots) + b_n, \\ &\quad W_0 \in \mathbb{R}^{d_0 \times d}, W_1 \in \mathbb{R}^{d_1 \times d_0}, \dots, W_n \in \mathbb{R}^{d' \times d_{n-1}}, \\ &\quad b_0 \in \mathbb{R}^{d_0}, b_1 \in \mathbb{R}^{d_1}, \dots, b_n \in \mathbb{R}^{d'}, d_0, d_1, \dots, d' \in \mathbb{N}\} \end{aligned}$$

The activation functions σ are restricted to the set:

$$\sigma \in A_m := \{\sigma \in C(\mathbb{R}) : \exists n \in \mathbb{N} \text{ s.t. } N_n \text{ is dense in } C^m(K) \forall K \subset \mathbb{R}^d \text{ compact}\}$$

to allow universal approximation. Lastly, functions are restricted to be real-valued, although extensions to vector-valued functions are possible (Section 8.3, Kovachki et al. [9]).

Theorem C.1 (Lemma 30, Kovachki et al. [9]). *Let $D \subset \mathbb{R}^d$ be a domain and $L \in (C(\bar{D}))^*$ be a linear functional. For any compact set $K \subset C(\bar{D})$ and $\epsilon > 0$, there exists a function $\kappa \in C_c^\infty(D)$ such that:*

$$\sup_{u \in K} |L(u) - \int_D \kappa u dx| < \epsilon$$

This result establishes that a linear functional acting on functions in $C(\bar{D})$ can be approximated by a smooth integral kernel with arbitrary error ϵ . To show functional approximation with a neural network, we slightly modify an existing lemma:

Theorem C.2 (Lemma 32, Kovachki et al. [9]). *Let $D \subset \mathbb{R}^d$ be a domain and $\mathcal{A} = C(\bar{D})$. Let $L \in \mathcal{A}^*$ be a linear functional. For any compact set $K \subset \mathcal{A}$, $\sigma \in A_0$, and $\epsilon > 0$, there exists a number $L \in \mathbb{N}$ and neural network $\kappa_\theta \in N_L(\sigma; \mathbb{R}^d, \mathbb{R})$ such that:*

$$\sup_{u \in K} |L(u) - \int_D \kappa_\theta(x) u(x) dx|_1 < \epsilon$$

Proof. Since K is bounded, there exists a number $M > 0$ such that:

$$\sup_{u \in K} \|u\|_{\mathcal{A}} \leq M$$

By Lemma 30, there exists a function $\kappa \in C_c^\infty(D)$ such that:

$$\sup_{u \in K} |L(u) - \int_D \kappa u dx| < \frac{\epsilon}{2}$$

Since $\sigma \in A_0$, there exists some $L \in \mathbb{N}$ and a neural network $\kappa_\theta \in N_L(\sigma; \mathbb{R}^d, \mathbb{R})$ such that:

$$\|\kappa_\theta - \kappa\|_C \leq \frac{\epsilon}{2M|D|}$$

Then for any $u \in K$:

$$\begin{aligned} |L(u) - \int_D \kappa_\theta(x) u(x) dx|_1 &\leq |L(u) - \int_D \kappa u dx|_1 + \left| \int_D (\kappa - \kappa_\theta) u dx \right|_1 \\ &\leq \frac{\epsilon}{2} + M|D| * \|\kappa - \kappa_\theta\|_C \\ &\leq \epsilon \end{aligned}$$

□

Therefore, linear functionals for continuously differentiable function classes can be approximated to arbitrary accuracy by a neural network using an integral kernel. Extensions to Sobolev and Hilbert spaces can also be proved in a similar manner.

Interestingly, this overlap between neural functionals and neural operators suggests a deeper connection between the two. The integral kernel functional bears many similarities to the integral kernel operator and, in practice, is implemented as an integral kernel operator evaluated at a single output. From an implementation perspective, the two architectures are equivalent in this manner; examining the theoretical basis also suggests that neural functionals are a subset of neural operators when approximating linear functionals. This can be seen by looking at operator approximation theorems (Lemma 22, Kovachki et al. [9]). We refrain from rewriting the theorems here, but focus on the necessary modifications. Specifically, to use a neural operator to approximate a linear functional, the output Banach space \mathcal{Y} is set to \mathbb{R} and intermediate dimensions J, J' are set to 1. The linear map $F_J : \mathcal{X} \rightarrow \mathbb{R}^J$ from the input field $x \in \mathcal{X}$ becomes $F_J = w_1(x)$, where $w_1(x) \in \mathcal{X}^*$. Lastly, the finite-dimensional map is defined as $\varphi \in C(\mathbb{R}, \mathbb{R})$ and the output map is defined as $G_{J'} : \mathbb{R} \rightarrow \mathbb{R}$.

C.2 Connection to the Cylindrical Approximation

Miyagawa and Yokota [56] propose an alternate mechanism to approximate functionals based on the cylindrical approximation. We explore a possible connection with the current method of using an integral kernel.

We consider functionals $F[u] : H \rightarrow \mathbb{R}$, where we restrict functions $u(x) \in H$ to be in a Hilbert space H with inner product $\langle \cdot, \cdot \rangle_H$. Since functions can be represented as a sum of basis functions, $u(x)$ can be written as $u(x) = \sum_{k=1}^{\infty} \langle u, \phi_k \rangle \phi_k(x)$, with an orthonormal basis $\{\phi_1, \phi_2, \dots\}$. Truncating the sum to a finite number of terms m allows the cylindrical approximation of functionals [80–82]. This is obtained by substituting the truncated series into the functional:

$$f(a_1, a_2, \dots, a_m) := F\left[\sum_{k=1}^m a_k \phi_k(x)\right] \quad (15)$$

where a_k are basis coefficients $\langle u, \phi_k \rangle$. Using the cylindrical approximation f , functional derivatives can be approximated by:

$$\frac{\delta F}{\delta u} \approx \sum_{k=1}^m \frac{\partial f}{\partial a_k} \phi_k(x) \quad (16)$$

which converges to the true functional derivative as $m \rightarrow \infty$ [81]. When discretizing the functional derivative, we define a set of n points $\mathbf{x} = [x_1, x_2, \dots, x_n]$ at which the basis functions are evaluated, which results in the discrete approximation of the functional derivative:

$$\sum_{k=1}^m \frac{\partial f}{\partial a_k} \phi_k(x) \approx \sum_{k=1}^m \frac{\partial f}{\partial a_k} \phi_k(\mathbf{x}) = \begin{bmatrix} \sum \frac{\partial f}{\partial a_k} \phi_k(x_1) \\ \sum \frac{\partial f}{\partial a_k} \phi_k(x_2) \\ \vdots \\ \sum \frac{\partial f}{\partial a_k} \phi_k(x_n) \end{bmatrix} \quad (17)$$

We are interested if the gradient of a neural functional that is parameterized by a kernel integral can represent this discrete object, and what conditions are needed to do so. Furthermore, there are two levels of approximation (a truncated basis and a discretized domain) and we are interested if the gradient of a neural functional can recover a functional derivative in its limit. We will work through three cases of increasing complexity, using a linear kernel, a nonlinear kernel, and a nonlinear and global kernel.

Linear Kernel Integrals A functional parameterized by a linear kernel integral can be written as:

$$F_\theta[u] = \int_{\Omega} \kappa_\theta(x) u(x) dx \approx \sum_{i=1}^n \kappa_\theta(x_i) u_i \mu_i \quad (18)$$

using a Riemann sum approximation and quadrature weights μ_i . Without loss of generality, we can assume some constant quadrature weight μ and write the gradient of the Riemann sum as:

$$\mu \nabla_u \left(\sum_{i=1}^n \kappa_\theta(x_i) u_i \right) = \mu \begin{bmatrix} \frac{\partial}{\partial u_1} (\kappa_\theta(x_1) u_1) \\ \frac{\partial}{\partial u_2} (\kappa_\theta(x_2) u_2) \\ \vdots \\ \frac{\partial}{\partial u_n} (\kappa_\theta(x_n) u_n) \end{bmatrix} = \mu \begin{bmatrix} \kappa_\theta(x_1) \\ \kappa_\theta(x_2) \\ \vdots \\ \kappa_\theta(x_n) \end{bmatrix} \quad (19)$$

We can interpret this gradient as the discretization of a single basis function $\kappa_\theta = \phi_1$, and without the necessary coefficients $\frac{\partial f}{\partial a_k}$. Evidently, this is not enough expressivity to approach a sum of m coefficients and basis functions, even as $n \rightarrow \infty$. Intuitively, the output $F_\theta[u]$ does not depend on u enough, causing its gradient ∇_u to be too simple. A potential remedy is to modify the kernel to be nonlinear by using $\kappa_\theta(x, u(x))$.

Nonlinear Kernel Integrals Following the same approximation, the gradient of a nonlinear kernel integral can be written as:

$$\mu \nabla_u \left(\sum_{i=1}^n \kappa_\theta(x_i, u_i) u_i \right) = \mu \begin{bmatrix} \frac{\partial}{\partial u_1} (\kappa_\theta(x_1, u_1) u_1) \\ \frac{\partial}{\partial u_2} (\kappa_\theta(x_2, u_2) u_2) \\ \vdots \\ \frac{\partial}{\partial u_n} (\kappa_\theta(x_n, u_n) u_n) \end{bmatrix} = \mu \begin{bmatrix} u_1 \frac{\partial}{\partial u_1} \kappa_\theta(x_1, u_1) + \kappa_\theta(x_1, u_1) \\ u_2 \frac{\partial}{\partial u_2} \kappa_\theta(x_2, u_2) + \kappa_\theta(x_2, u_2) \\ \vdots \\ u_n \frac{\partial}{\partial u_n} \kappa_\theta(x_n, u_n) + \kappa_\theta(x_n, u_n) \end{bmatrix} \quad (20)$$

using the product rule. In general, $\kappa_\theta(x_i, u_i)$ can be quite complex and nonlinear, however, by construction it only depends on (x_i, u_i) . We define a function $d_i(x_i) = \frac{\partial}{\partial u_i} \kappa_\theta(x_i, u_i)|_{u_i=u_i}$, which evaluates the partial derivative at x_i . This lets us construct at most n basis functions $\{d_1, d_2, \dots, d_n\}$, however, this is still not enough expressivity, as each entry of the discretized functional derivative is still restricted to one basis function, albeit a different one at each row. Despite this shortcoming, writing this expansion allows us to see that producing a sum over multiple basis functions requires the kernel to depend globally on $\mathbf{u} = [u_1, u_2, \dots, u_n]$, since the gradient of the Riemann sum no longer degenerates to a single term at each row.

Nonlinear Global Kernel Integrals Using the same approximation, the gradient of a nonlinear, global kernel integral can be written as:

$$\mu \nabla_{\mathbf{u}} \left(\sum_{i=1}^n \kappa_{\theta}(x_i, \mathbf{u}) u_i \right) = \mu \begin{bmatrix} \frac{\partial}{\partial u_1} \left(\sum \kappa_{\theta}(x_i, \mathbf{u}) u_i \right) \\ \frac{\partial}{\partial u_2} \left(\sum \kappa_{\theta}(x_i, \mathbf{u}) u_i \right) \\ \vdots \\ \frac{\partial}{\partial u_n} \left(\sum \kappa_{\theta}(x_i, \mathbf{u}) u_i \right) \end{bmatrix} \quad (21)$$

The expression $\frac{\partial}{\partial u_j} \left(\sum \kappa_{\theta}(x_i, \mathbf{u}) u_i \right)$ can be expanded into:

$$\begin{aligned} \frac{\partial}{\partial u_j} \left(\sum_{i=1}^n \kappa_{\theta}(x_i, \mathbf{u}) u_i \right) &= u_1 \frac{\partial}{\partial u_j} \kappa_{\theta}(x_1, \mathbf{u}) + \dots \\ &+ u_j \frac{\partial}{\partial u_j} \kappa_{\theta}(x_j, \mathbf{u}) + \kappa_{\theta}(x_j, \mathbf{u}) + \dots \\ &+ u_n \frac{\partial}{\partial u_j} \kappa_{\theta}(x_n, \mathbf{u}) \end{aligned} \quad (22)$$

In general, we can construct a kernel κ_{θ} such that $\frac{\partial}{\partial u_j} \kappa_{\theta}(x_i, \mathbf{u})$ is different for $i = \{1, \dots, n\}$. A simple example is provided based on FiLM modulation:

$$\kappa_{\theta}(x_i, \mathbf{u}) = [W^u \mathbf{u}]_i (W^x x_i + b^x) + [b^u]_i \quad (23)$$

where $W^u \in \mathbb{R}^{n \times n}$, $W^x \in \mathbb{R}^{1 \times 1}$, $b^u \in \mathbb{R}^n$, $b^x \in \mathbb{R}^1$ and $[\cdot]_i$ is an operation that takes the i -th row of a column vector. In this case, $\frac{\partial}{\partial u_j} \kappa_{\theta}(x_i, \mathbf{u})$ is equivalent to a function $d_{ij}(x_i) = W_{ij}^q W^x x_i$. With even deeper and more complex kernels, functions d_{ij} can become more expressive. Without loss of generality, we can also absorb the basis coefficients into d_{ij} .

With this formulation, we can see that the j -th row of the gradient vector can be written as a sum of n functions $\sum_{i=1}^n d_{ij}(x_i)$, which reduces to the cylindrical approximation when letting d_{ij} be arbitrary. Furthermore, letting $n \rightarrow \infty$ increases the number of functions d_{ij} and coordinates x_i , and is consistent with the interpretation of simultaneously increasing the number of bases to approach the true functional derivative as well as converting from a discrete to continuous representation.

The cylindrical approximation makes no assumption on the linearity of F , therefore, when using a global, nonlinear kernel, a kernel integral can represent a functional derivative. This theoretical insight is also supported by empirical observations. Through ablation studies in Section B.3, we find that modeling complex functional derivatives (such as in the KdV equation) is only possible with a global, nonlinear kernel, and simplifications of the kernel degrade the performance.

C.3 Derivations with Varying Coefficients and Source Terms

Using HNS with PDEs with varying coefficients and source terms is possible on a case-by-case basis, requiring additional effort to derive the Hamiltonian structure. We provide examples below by introducing coefficients to 1D Advection and 1D KdV as well as varying bathymetry (source terms) to 2D SWE.

Varying Coefficients In the presence of coefficients, the Hamiltonians for 1D Advection and 1D KdV become:

$$\mathcal{H}_{adv}[u] = \int_{\Omega} -\frac{c}{2} (u(x))^2 dx, \quad \mathcal{H}_{kdv}[u] = \int_{\Omega} -\frac{\alpha}{6} (u(x))^3 - \beta u(x) \frac{\partial^2 u}{\partial x^2}(x) dx, \quad (24)$$

For 1D Advection, one can verify the Hamiltonian structure:

$$\frac{\delta \mathcal{H}_{adv}}{\delta u} = -cu(x), \quad \mathcal{J} \left(\frac{\delta \mathcal{H}_{adv}}{\delta u} \right) = -c \frac{\partial u}{\partial x} = \frac{du}{dt} \quad (25)$$

A similar calculation recovers the PDE for the 1D KdV equation ($u_t + \alpha u u_x + \beta u_{xxx} = 0$) from Hamilton's equations by using $\frac{\delta \mathcal{H}_{kdv}}{\delta u} = -\frac{\alpha}{2} u^2 - \beta u_{xx}$. To accommodate varying coefficients, this suggests training HNS with coefficient information using the loss: $\mathcal{L}(\nabla_u \mathcal{H}_{\theta}(\mathbf{u}, \mathbf{x}, \mathbf{c}), \frac{\delta \mathcal{H}}{\delta u})$. In special cases where the coefficients can be factored out (i.e., $\frac{\delta \mathcal{H}[\mathbf{u}, \mathbf{c}]}{\delta u} = c \frac{\delta \mathcal{H}[\mathbf{u}]}{\delta u}$), one can scale the base model output $\nabla_u \mathcal{H}_{\theta}(\mathbf{u}, \mathbf{x})$ by c to achieve the same effect.

Varying Source Terms The 2D shallow-water equations describe free surfaces where the horizontal length scale is significantly larger than the vertical length scale. Within this setup, the free surface can be above a spatially-varying bathymetry $b(x, y)$, or the topography of the bottom surface. Changes in the elevation of the bathymetry can accelerate or decelerate flows, which acts as a source term on the momentum balance [83, 84]. In particular, the shallow-water equations are modified to become:

$$\partial_t h + \nabla \cdot (\mathbf{v}h) = 0, \quad \partial_t \mathbf{v} + \mathbf{v} \cdot \nabla \mathbf{v} = -g\nabla h - gh\nabla b \quad (26)$$

Furthermore, $h(x, y, t)$ now represents the height above the bathymetry $b(x, y, t)$. This source term is represented in the Hamiltonian as:

$$\mathcal{H}^b[\mathbf{u}] = \mathcal{H}^b[v_x, v_y, h] = \int_{\Omega} \frac{1}{2}h(v_x^2 + v_y^2) + \frac{1}{2}gh^2(1 + b) \, dA \quad (27)$$

and the operator matrix \mathcal{J} is left unchanged (Equation 11). One can check that applying Hamilton's equations recovers the modified shallow-water equations. When comparing this derivation to one with constant bathymetry, one can show that:

$$\frac{\delta \mathcal{H}^b[u]}{\delta u} = \frac{\delta \mathcal{H}[u]}{\delta u} + \begin{bmatrix} 0 \\ 0 \\ ghb \end{bmatrix} \quad (28)$$

This suggests that instead of scaling model outputs $\nabla_u \mathcal{H}_{\theta}(\mathbf{u}, \mathbf{x})$ (as in the case of coefficients), source terms can be potentially captured by adding terms to model outputs. In PDEs where a decomposition like this is not possible, the source term will need to be added to the model input to produce the output $\nabla_u \mathcal{H}_{\theta}(\mathbf{u}, \mathbf{x}, \mathbf{b})$, where bold terms indicate discretized fields/coordinates.

D Implementation Details

D.1 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 $\frac{d\mathbf{u}}{dt}|_{t=t} = f(\mathbf{u}^t)$:

$$\mathbf{u}^{t+1} = \mathbf{u}^t + \Delta t \left(\frac{3}{2}f(\mathbf{u}^t) - \frac{1}{2}f(\mathbf{u}^{t-1}) \right) \quad (29)$$

This can be implemented in constant time by caching prior derivative estimates $\{f(\mathbf{u}^i) : i < t\}$, allowing the ODE integrator to have a 2nd-order truncation error ($\mathcal{O}(\Delta t^2)$) while remaining fast. At the first timestep, there are no cached derivatives, therefore a first-order Euler integrator is used: $\mathbf{u}^{t+1} = \mathbf{u}^t + \Delta t f(\mathbf{u}^t)$. For the given experiments, these integrators are accurate since Δt is small enough; to be used with larger timesteps Δt , higher-order integrators may be needed or smaller, intermediate steps will need to be taken [26]. This adds additional consideration to the training and deployment of derivative-based neural PDE solvers.

Finite Difference Schemes Implementing infinite-dimensional Hamiltonian structure involves approximating the operator \mathcal{J} , either through numerical methods or neural operators. We opt for numerical methods, as the operators are simple and in all our tests only involve first-order spatial derivatives ∂_x . For the 1D Advection equation, we implement a 2nd-order central difference scheme. For the 1D KdV equation, the spatial derivatives exhibit larger gradients, therefore we use an 8th-order central difference scheme with a smoothing stencil to approximate ∂_x [85]. A similar scheme is implemented in 2D to approximate ∂_x and ∂_y for the 2D SWE experiments, and additionally, a Savitzky-Golay filter is used to further damp numerical oscillations. These arise since the lack of viscosity creates discontinuities in the shallow-water equations, alternative methods may include flux-limiting or non-oscillatory schemes [86, 87]. While these implementations work, they still required tuning, as such, the use of a neural operator to approximate \mathcal{J} to make HNS fully learnable would be an interesting future direction.

Additionally, to calculate temporal derivatives $d\mathbf{u}/dt$ during training, a 4-th order Richardson extrapolation is used to provide accurate derivatives. At the boundaries $t = 0$, $t = T$ a one-sided Richardson extrapolation is used [88].

Numerical Solvers The analytical solution to the Advection equation is used to generate its datasets. The numerical solver for the KdV equation is from Brandstetter et al. [24]; due to the stiffness of the PDE, the solutions do not always conserve the Hamiltonian. The WENO schemes used allow for more stable derivatives, but at the cost of numerical viscosity, which can affect the Hamiltonian since it is highly nonlinear. Therefore, we filter the generated KdV solutions and extract 50% of the trajectories with the smallest variation in the Hamiltonian. Similar fluctuations in the Hamiltonian are seen in SWE trajectories generated from PyClaw [31], but the variations are small enough to be ignored.

D.2 Experimental Details

Hyperparameters Each model has different hyperparameters for different PDEs (Adv/KdV/SWE). We provide key hyperparameters for these experiments in Tables 10, 11, 12.

	Adv	KdV	SWE
Modes	10	12	12
Width	24	32	48
Layers	5	5	5

Table 10: FNO Parameters

	Adv	KdV	SWE
Width	8	12	32
Bottleneck Dim	32	48	128
Layers	6	6	6

Table 11: Unet Parameters

	Adv	KdV	SWE
Width	32	32	64
Kernel	local	global	global
Cond.	FiLM	FiLM	FiLM

Table 12: HNS Parameters

Computational Resources All experiments were run on a single NVIDIA RTX 6000 Ada GPU; 1D experiments were usually run within 10 minutes, while 2D experiments were usually run within an hour. The largest dataset is around 20 GB, for the 2D Shallow-Water equations.

NeurIPS Paper Checklist

1. Claims

Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

Answer: [\[Yes\]](#)

Justification: The authors were careful to not claim too much in the abstract and introduction, focusing on summarizing the paper and providing an introduction to the field and how the paper fits in.

Guidelines:

- The answer NA means that the abstract and introduction do not include the claims made in the paper.
- The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
- The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
- It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. Limitations

Question: Does the paper discuss the limitations of the work performed by the authors?

Answer: [\[Yes\]](#)

Justification: Limitations are discussed in Section ?? (Discussion).

Guidelines:

- The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. Theory assumptions and proofs

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [\[Yes\]](#)

Justification: An informal proof is provided in the main body in Section 3, but a formal proof is given in Appendix C.1.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. Experimental result reproducibility

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [\[Yes\]](#)

Justification: We have tried to provide the necessary detail for reproducing results, and all code and datasets will be also be released.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general, releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
 - (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
 - (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
 - (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
 - (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. Open access to data and code

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: All code and datasets will be made publicly available.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (<https://nips.cc/public/guides/CodeSubmissionPolicy>) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so “No” is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (<https://nips.cc/public/guides/CodeSubmissionPolicy>) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. Experimental setting/details

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

Answer: [Yes]

Justification: We have provided information on the experimental setup and specific details can be found in the config files of the provided code.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. Experiment statistical significance

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: Error bars are given for the main quantitative, PDE results in Tables 2 and 3.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. Experiments compute resources

Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [Yes]

Justification: Details on the compute resources are provided in the Appendix D.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. Code of ethics

Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics <https://neurips.cc/public/EthicsGuidelines>?

Answer: [Yes]

Justification: The paper does not violate the code of ethics.

Guidelines:

- The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. Broader impacts

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [NA]

Justification: The work is largely foundational and in the field of physics; at this moment there are no likely negative impacts of the work.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. Safeguards

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

Answer: [NA]

Justification: There are no risks for misuse.

Guidelines:

- The answer NA means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. Licenses for existing assets

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: The authors have created all assets used in the paper.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, paperswithcode.com/datasets has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.

- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset’s creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [\[Yes\]](#)

Justification: New code and datasets will have accompanying documentation.

Guidelines:

- The answer NA means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [\[NA\]](#)

Justification: The research does not involve crowdsourcing or human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [\[NA\]](#)

Justification: The paper does not involve crowdsourcing or human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. Declaration of LLM usage

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

Answer: [NA]

Justification: LLMs were not used in the core method development of the research.

Guidelines:

- The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy (<https://neurips.cc/Conferences/2025/LLM>) for what should or should not be described.