Energy-consistent Neural Operators for Hamiltonian and Dissipative Partial Differential Equations

Yusuke Tanaka NTT
Takaharu Yaguchi Kobe University
Tomoharu Iwata NTT
Naonori Ueda RIKEN AIP

Abstract

The operator learning has received significant attention in recent years, with the aim of learning a mapping between function spaces. Prior works have proposed deep neural networks (DNNs) for learning such a mapping, enabling the learning of solution operators of partial differential equations (PDEs). However, these works still struggle to learn dynamics that obeys the laws of physics. This paper proposes *Energy-consistent Neural Operators (ENOs)*, a general framework for learning solution operators of PDEs that follows the energy conservation or dissipation law from observed solution trajectories. We introduce a novel penalty function inspired by *the energy-based theory of physics* for training, in which the functional derivative is calculated making full use of automatic differentiation, allowing one to bias the outputs of the DNN-based solution operators to obey appropriate energetic behavior without explicit PDEs. Experiments on multiple systems show that ENO outperforms existing DNN models in predicting solutions from data, especially in the low-data regime.

1 INTRODUCTION

Many physical systems are described by partial differential equations (PDEs). Obtaining their solutions is fundamental in such disciplines as weather forecasting (Lynch, 2008), molecular modeling (Lelièvre and Stoltz, 2016), astronomical simulations (Courant et al., 1967), and jet engine design (Athanasopoulos et al., 2009). Traditionally, one crafts a PDE by hand and obtains its solution via a numerical simulation. However, the PDE design requires domain knowledge and great effort; the numerical simulation depends on a spatio-temporal mesh, which is often high-resolution, yielding high computational costs.

In the machine learning community, interest in a novel approach called *operator learning* is growing, which predicts mesh-free solutions from data; PDE designs and numerical simulations are not required. Generally speaking, the goal of operator learning is to obtain a mapping (i.e., *an operator*) between function spaces. In a setting of PDEs, the solution operator is the mapping from the input function (e.g., initial and boundary conditions) to the solution function. There have been many deep neural networks (DNNs) for operator learning, such as deep operator networks (DeepONets) (Lu et al., 2021) and Fourier neural operators (FNOs) (Li et al., 2021), which can be used for approximating a solution operator from many input-output function pairs. However, these works still struggle to learn *physical laws* from data precisely, which is more critical when we cannot obtain sufficient data.

Recent studies attempt to use the prior knowledge of physics as an inductive bias for training DNNs, which has recently been addressed under the name of *physics-informed machine learning (PIML)* (Karniadakis et al., 2021; Meng et al., 2022). A few works have introduced physics-informed inductive bias into the operator learning framework, called physics-informed DeepONets (PI-DeepONets) (Wang et al., 2021) and physics-informed neural operators (PINOs) (Li et al., 2023; Wang et al., 2021), by adding PDE constraints to the loss function, as in physics-informed neural networks (Raissi et al., 2019). However, these works require explicit PDEs for training DNN-based operators and cannot be applied to systems for which their PDEs are unknown.

Hamiltonian neural networks (HNNs) (Greydanus et al., 2019) and its variants (e.g., (Chen et al., 2021; Zhong et al., 2020b; Matsubara et al., 2020)) have introduced an inductive bias based on the Hamiltonian mechanics; the primary idea is to parameterize

Proceedings of the 28th International Conference on Artificial Intelligence and Statistics (AISTATS) 2025, Mai Khao, Thailand. PMLR: Volume 258.

an energy function (called *Hamiltonian*) using a DNN and define a system's time-evolution using its gradient. HNN enables us to learn dynamics that follows the basic laws of physics, such as energy conservation or dissipation, without necessitating explicit differential equations. However, these works aim to obtain the time-evolution at a point of time, not solution operators, from data; hence they require computationally expensive numerical simulations to predict solutions.

This paper proposes *Energy-consistent Neural Operators (ENOs)*, a general data-driven framework for learning solution operators of *hidden* PDEs that adhere to the the energy conservation or dissipation law, without using explicit PDEs. Our proposed framework assumes that the solution operators are parameterized by a DNN (called *operator net*), where it does not depend on a particular choice of the DNN architecture as long as it is differentiable. The most significant contribution of ENO is a novel penalty function inspired by the energy-based theory (Furihata, 1999; Quispel and Capel, 1996), which is a generalization of Hamiltonian mechanics. This penalty function allows one to bias the time-derivative of the operator net to follow the energy conservation or dissipation law.

To obtain the penalty function, we model an *unknown* system's total energy defined by a *functional* using another DNN (called *energy net*); we then derive a gradient flow of the energy functional by calculating the functional derivatives (also called variational derivatives) via automatic differentiation. The use of automatic differentiation has the significant advantage of not requiring numerical approximation. According to the energy-based theory, the energetic consistency is guaranteed by employing this gradient flow as the time-derivative of the solution. Our penalty encourages the time-derivative of the operator net to be equal to the energy gradient flow. In training, the operator net and energy net are simultaneously optimized by minimizing the data loss to predict solutions and the penalty. Importantly, ENO estimates not only the solution operator (i.e., operator net) but also the energy functional (i.e., energy net); hence, it does not need the explicit PDEs or Hamiltonian and can obtain solution operators, unlike PI-DeepONets and PINOs. In testing, efficient and mesh-free simulation of physical systems is possible using the learned solution operator.

One advantage of ENO is that it can consider the penalty term at arbitrary points not included in the training data. Our training algorithm applies the penalty term evaluated at randomly sampled points, yielding a smoothing effect based on the laws of physics over the entire spatio-temporal domain. This is especially helpful in super-resolution settings: we predict higher-resolution data from only lower-resolution data.

The following are the main contributions of our work:

- Energy-consistent Neural Operator (ENO) is the first framework based on the energy-based theory for learning solution operators of Hamiltonian and dissipative PDEs from data without explicit PDEs. Our proposal is a penalty function designed to encourage the time-derivative of solutions to align with the energy gradient flow, which is calculated via automatic differentiation.
- We provide an algorithm to infer the solution operator and energy functional by simultaneously minimizing the data loss and our penalty term evaluated at randomly sampled points, yielding a smoothing effect that preserves the appropriate energetic behavior over the entire domain.
- Experiments on multiple PDE systems show that ENO more accurately predicts solutions while appropriately capturing systems' energetic behavior than baselines (e.g., DeepONet and FNO), especially when the training data is lower-resolution.

2 RELATED WORK

Roughly speaking, this work bridges two research topics in the machine learning community, that is, neural networks for operator learning and the energy-based theory of physics as an inductive bias. Below we describe prior works that represent two research lines. Table 1 compares the proposed model (ENO) with the existing representative models.

Neural networks for operator learning. Many deep neural networks (DNNs) have been proposed for learning solution operators of PDEs (Bhattacharya et al., 2021; Patel et al., 2021; Li et al., 2020b; De Ryck and Mishra, 2022; Gupta et al., 2021). In these studies, a mapping between input functions (e.g., initial conditions) and output functions (i.e., solutions) is parameterized by DNNs. Graph kernel networks (Li et al., 2020a) construct solution functions as a convolution of input function values on a graph-based discretization using DNN-based kernels. Fourier neural operators (FNOs) (Li et al., 2021) and its variants (e.g., (Bonev et al., 2023; Helwig et al., 2023; Rahman et al., 2023)), which are their extensions, allow for efficiently computing the convolution by Fourier approximations in the frequency domain. Deep operator networks (DeepONets) (Lu et al., 2021) and its variants (e.g., (Mao et al., 2021; Cai et al., 2021)) are general architectures designed based on the universal approximation theorem for operators (Chen and Chen, 1995). FNOs and DeepONets are the most popular models in this field.

Recently, there have been various kinds of DNN-based operators. Graph neural networks are used to model

Table 1: Comparison of ENO with existing models: we compared the models with six items. First and second items represent that they can handle systems defined by (A) ODEs and (B) PDEs. Third item (C) states that they can estimate solution operators; models without a check mark require numerical simulations to predict solutions. Fourth and fifth items represent that the models can use the physics priors, i.e., (D) energy conservation law and (E) energy dissipation law, respectively, as an inductive bias for training. Last item (F) shows that training can be done from only data; models without a check mark assume that explicit PDEs are known.

	DeepONet / FNO	PI-DeepONet / PINO	HNN	DGNet	ENO
(A) ODE systems	√	√	√	√	√
(B) PDE systems	√	√		√	√
(C) Solution operator	√	√			√
(D) Energy conservation law		√	√	√	√
(E) Energy dissipation law		√		√	√
(F) Without explicit equations	√		√	√	√

solution operators of PDEs by regarding spatial discretization as a graph (Horie and Mitsume, 2022; Brandstetter et al., 2022); the spatial dependencies are considered via a neural message passing algorithm. Several studies have proposed transformer-based architectures (Poli et al., 2022; Cao, 2021; Hao et al., 2023); one can regard the operator learning as a special case of a sequence-to-sequence problem, since input and output functions are often sampled at discrete grid points. A recent work has shown that convolutional neural networks (CNNs) are also effective in operator learning (Raonic et al., 2023); the modification of CNNs using U-Net architectures has been proposed, which allows one to train solution operators from data at multiple discrete resolutions while avoiding aliasing issues. DeepGreen (Gin et al., 2021) and its extension (Boullé et al., 2022) adopt the idea of Koopman operators (Brunton and Kutz, 2019) to learn Green's functions for elliptic PDEs, which can be regarded as a kind of solution operators.

However, they still suffer from learning *physical laws* from data and might fail to predict solutions accurately. It is incredibly challenging, especially in situations where there is an insufficient amount of training data with high spatial and temporal resolutions. Physics-informed DeepONets (PI-DeepONets) (Wang et al., 2021) and physics-informed neural operators (PINOs) (Li et al., 2023) have tackled this issue by introducing PDE constraints, in a manner similar to physics-informed neural networks(PINNs) (Raissi et al., 2019). Another related work (Mattheakis et al., 2022) has proposed a constraint based on the *known* energy function to encourage a DNN-based solution to satisfy energy conservation laws. These constraints are expected to infer the dynamics that follows the laws of physics even with limited data. These works, however, need explicit PDEs for training and are inapplicable to our problem setting, where we assume that explicit PDEs are unknown.

Several works have embedded the conservation law into DNN architecture as hard constraints (Hansen et al., 2023; Liu et al., 2024; Richter-Powell et al., 2022). In these works, the conservation law has been introduced using a *conservation form* (a kind of continuity equation). The important thing is that the states in the form contain some conserved quantities (e.g., mass or energy). This means that these works require observing conserved quantities; hence, they are not applicable to our problem setting of learning operators only from input-output function pairs, without using additional data (e.g., energy values).

Energy-based theory of physics as an inductive bias. Many machine learning models utilize the prior knowledge of Hamiltonian mechanics (Goldstein, 1980) as an inductive bias for inferring the dynamics that ensures the energetic consistency of physics (Chen et al., 2020, 2021; Finzi et al., 2020; Zhong et al., 2020b; Rath et al., 2021; Tanaka et al., 2022). Here, Hamiltonian mechanics can be regarded as a particular case of the energy-based theory (Celledoni et al., 2012). The pioneering work is Hamiltonian neural networks (HNNs) (Greydanus et al., 2019), the key idea of which is parameterizing the Hamiltonian (i.e., energy function) using DNNs; then, the time evolution of the ordinary differential equation (ODE) systems is given by the energy gradient (called symplectic gradient). The energy conservation law is guaranteed by employing the symplectic gradient as the time-derivative. By estimating the Hamiltonian from data, HNN makes it possible to embed energy conservation laws into the architectures without explicit differential equations.

Most recent works have expanded the scope of application, such as Hamiltonian systems with energy dissipation (Zhong et al., 2020a; Sosanya and Greydanus, 2022), Hamiltonian systems with controllable inputs (Desai et al., 2021), stiff Hamiltonian systems (Liang et al., 2022), odd-dimensional chaotic sys-

tems (Course et al., 2020), and Poisson systems (Jin et al., 2022). DGNet (Matsubara et al., 2020) has been extended to handle Hamiltonian and dissipative PDEs by employing the energy-based theory (Furihata, 1999; Quispel and Capel, 1996). However, since all of the existing models infer the time-evolution at a point of time, not solution operators, from data, they require numerical simulations to predict solutions. This approach has a significant disadvantage in terms of computational costs, especially in the PDE setting. In addition, predictions can only be made for a predefined spatio-temporal discretization when applied to PDEs (Matsubara et al., 2020; Eidnes and Lye, 2024).

Our work. We integrate the energy-based theory into the operator learning framework, which allows one to utilize the physics prior (i.e., energy conservation and dissipation laws) to obtain solution operators without an explicit form of PDE and Hamiltonian, unlike the existing models (Wang et al., 2021; Li et al., 2023; Mattheakis et al., 2022). In addition, unlike HNN and its extensions that require computationally expensive numerical simulations, our solution operators can be used for efficient simulations without discretization.

3 PRELIMINARY

3.1 Energy-based Theory of Physics

This section presents an overview of the energy-based theory (Quispel and Capel, 1996; Furihata, 1999), a mathematical framework that generalizes Hamiltonian mechanics. The energy-based theory enables one to handle Hamiltonian and dissipative PDEs as well as Hamiltonian systems (defined by ordinary differential equations). See Appendix A for the relationship to the Hamiltonian system.

Let $\mathcal{T} = \mathbb{R}_{\geq 0}$ be the time domain, and let $\mathcal{X} \subset \mathbb{R}^D$ be the D-dimensional bounded spatial domain. We consider physical systems defined on the time-space $(t, \boldsymbol{x}) \in \mathcal{T} \times \mathcal{X}$. Let $\boldsymbol{u} : \mathcal{T} \times \mathcal{X} \to \mathbb{R}^M$ be a solution function that represents the system's state, where $\boldsymbol{u}$ belongs to a function space $\mathcal{U}$. In the energy-based formulation, the starting point is to define *energy functional* $\mathcal{H} : \mathcal{U} \to \mathbb{R}$, which denotes the system's total energy. The energy functional is given by

$$\mathcal{H}[\boldsymbol{u}] = \int_{\mathcal{X}} F(\boldsymbol{u}, \partial_{\boldsymbol{x}}\boldsymbol{u}, \partial_{\boldsymbol{xx}}\boldsymbol{u}, \dots)\, d\boldsymbol{x}, \tag{1}$$

where $F : \mathcal{U} \to \mathbb{R}$ is called an energy density. One can observe that energy functional $\mathcal{H}$ is obtained by integrating density F over spatial domain $\mathcal{X}$. Here, we adopt shorthand notation $\partial_{\boldsymbol{x}}\boldsymbol{u}, \partial_{\boldsymbol{xx}}\boldsymbol{u}, \dots$ for partial derivatives $\partial \boldsymbol{u}/\partial \boldsymbol{x}, \partial^2 \boldsymbol{u}/\partial \boldsymbol{x}^2$, and so on. Traditionally, the energy density F is manually designed to suit the system. Given the energy functional $\mathcal{H}$, PDE dynamics is given by

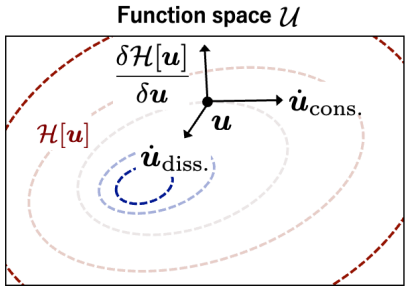


Figure 1: Intuitive view of energy gradient flows. Dashed lines represent contours of energy functional $\mathcal{H}$ on function space $\mathcal{U}$; blue dashed line represents low energy. Functional derivative $\delta\mathcal{H}/\delta\boldsymbol{u}$ is orthogonal to the contour at $\boldsymbol{u}$. Systems follow a flow $\dot{\boldsymbol{u}}_{\text{cons.}}$ conserving $\mathcal{H}$ if $\mathcal{G}$ is skew-symmetric and a flow $\dot{\boldsymbol{u}}_{\text{diss.}}$ dissipating $\mathcal{H}$ if $\mathcal{G}$ is negative-semidefinite.

$$\dot{\boldsymbol{u}} = \mathcal{G}\frac{\delta \mathcal{H}[\boldsymbol{u}]}{\delta \boldsymbol{u}}, \tag{2}$$

where $\dot{\boldsymbol{u}}$ denotes $\partial \boldsymbol{u}/\partial t$, and the right-hand side of (2) defines *the gradient flow of the energy functional*. Fig. 1 shows an intuitive view of the gradient flow (2). Here, $\mathcal{G}$ is typically a simple linear differential operator with respect to $\boldsymbol{x}$ and solely serves to define *the energetic behaviors* of systems. Details of $\mathcal{G}$ are described in the following paragraph. $\delta\mathcal{H}/\delta\boldsymbol{u}$ in (2) is a functional derivative (also called a variational derivative) of $\mathcal{H}$, defined by

$$\frac{d}{d\epsilon}\mathcal{H}[\boldsymbol{u} + \epsilon \boldsymbol{v}]\Big|_{\epsilon=0} = \left\langle \frac{\delta \mathcal{H}[\boldsymbol{u}]}{\delta \boldsymbol{u}}, \boldsymbol{v} \right\rangle, \tag{3}$$

for all $\boldsymbol{u}, \boldsymbol{v} \in \mathcal{U}$, where $\langle \cdot, \cdot \rangle$ is the inner product in $\mathcal{U}$. Intuitively, the functional derivative describes how a small perturbation $\epsilon\boldsymbol{v}$ affects the energy functional $\mathcal{H}$.

On the differential operator $\mathcal{G}$. The energetic behavior of (2) depends on the choice of $\mathcal{G}$, and the following properties are known to hold, which can be proved by considering conditions such that $\dot{\mathcal{H}} = 0$ (energy conservation) or $\dot{\mathcal{H}} \leq 0$ (energy dissipation). See (Celledoni et al., 2012) for details.

For energy-conserving systems (called *Hamiltonian PDEs*), $\mathcal{G}$ must be skew-symmetric in the following sense: $\langle \boldsymbol{u}, \mathcal{G}\boldsymbol{u} \rangle = \langle -\mathcal{G}\boldsymbol{u}, \boldsymbol{u} \rangle$, where $\langle \cdot, \cdot \rangle$ is the inner product in a certain function space. In addition to this, in many cases, physical systems should satisfy the mass conservation law: $\frac{d}{dt}\int \boldsymbol{u} d\boldsymbol{x} = 0$. There are only a limited number of $\mathcal{G}$'s that satisfy these conditions. The simplest one is $\mathcal{G} = \sum_{d=1}^{D} \partial/\partial x_d$, which appears in the Korteweg–de Vries (KdV) equation, the advection equation, the Burgers equation, and so on.

Similarly, for energy-dissipating systems (called *dissipative PDEs*), $\mathcal{G}$ must be negative-semidefinite: $\langle \boldsymbol{u}, \mathcal{G}\boldsymbol{u} \rangle \leq 0$. The simplest $\mathcal{G}$ that satisfies this is $\mathcal{G} = -1$ (e.g., the Allen–Cahn equation). If the mass conservation law is required, the simplest one is $\mathcal{G} = \sum_{d=1}^{D} \partial^2/\partial x_d^2$, (e.g., the Cahn-Hillard equation).

According to (Celledoni et al., 2012; Furihata, 1999), in most cases, $\mathcal{G}$ is either of the above a few operators $\left\{-1, \sum_{d=1}^{D} \partial/\partial x_d, \sum_{d=1}^{D} \partial^2/\partial x_d^2\right\}$ or a slightly modified version of them. Thus, determining $\mathcal{G}$ is much easier than determining the complete form of the PDEs. This fact enables us to select an appropriate operator via a validation procedure described in Section 4.

The energy-based theory covers many other physical systems. For example, the Schrödinger equation and the Ginzburg–Landau equation can be expressed by introducing complex state variables (Furihata, 1999).

3.2 Operator Learning

This section describes the operator learning framework. Generally speaking, the aim of operator learning is to obtain a mapping between two infinite-dimensional function spaces from a finite set of observed input-output function pairs. In the following, we elaborate on the problem of learning solution operators of PDE systems.

Learning solution operators. Let $\mathcal{A}$ and $\mathcal{U}$ be input and output function spaces. Input function $\boldsymbol{a} \in \mathcal{A}$ corresponds to the initial or boundary conditions, constant or variable coefficients, forcing terms, and so on; the input function can be chosen freely, depending on what we want to generalize to. Output function $\boldsymbol{u} \in \mathcal{U}$ corresponds to the solution, given input function $\boldsymbol{a}$. The goal is to approximate a *solution operator* $\mathcal{S} : \mathcal{A} \to \mathcal{U}$, which is a non-linear map between the input and output function spaces. Accordingly, the input and output functions satisfy the following relationship: $\boldsymbol{u} = \mathcal{S}[\boldsymbol{a}]$. In the training phase, we assume that we have I samples of input-output function pairs $\{(\bar{\boldsymbol{a}}_i, \bar{\boldsymbol{u}}_i) \mid i = 1, \ldots, I\}$, where the superscript bar indicates point-wise observations for functions; namely, $\bar{\boldsymbol{a}}_i$ and $\bar{\boldsymbol{u}}_i$ are the finite sets of the input and output function values evaluated on the discretization points. Let $\mathcal{Y} = \mathcal{T} \times \mathcal{X}$ denote the spatio-temporal domain, and let $\boldsymbol{y} \in \mathcal{Y}$ denote a *point*. Let $\bar{\mathcal{Y}}_i = \{\boldsymbol{y}_{i,j} \mid j = 1, \ldots, J_i\} \subset \mathcal{Y}$ be a J_i-point discretization for the solution $\boldsymbol{u}_i$. Also, the input function $\boldsymbol{a}_i$ may have a different discretization for each index i. Given the data, we wish to obtain solution operator $\boldsymbol{u}(\boldsymbol{y}) = \mathcal{S}_{\boldsymbol{\theta}}[\bar{\boldsymbol{a}}](\boldsymbol{y})$ approximated by deep neural networks (DNNs), where $\boldsymbol{\theta}$ is their parameters. The parameters can be estimated by minimizing the following mean squared error,

$$L(\boldsymbol{\theta}) = \frac{1}{I} \sum_{i=1}^{I} \left(\frac{1}{J_i} \sum_{j=1}^{J_i} \left\| \bar{\boldsymbol{u}}_i(\boldsymbol{y}_{i,j}) - \mathcal{S}_{\boldsymbol{\theta}}[\bar{\boldsymbol{a}}_i](\boldsymbol{y}_{i,j}) \right\|^2 \right), \tag{4}$$

where $\bar{\boldsymbol{u}}_i(\boldsymbol{y}_{i,j})$ and $\mathcal{S}_{\boldsymbol{\theta}}[\bar{\boldsymbol{a}}_i](\boldsymbol{y}_{i,j})$ denote the observed and predicted solution values at points $\boldsymbol{y}_{i,j}$ and $\|\cdot\|$ is the Euclidean norm. In the test phase, given *unseen* input function $\bar{\boldsymbol{a}}^*$, the solution $\boldsymbol{u}^*(\boldsymbol{y})$ can be predicted using the estimated operator, as $\boldsymbol{u}^*(\boldsymbol{y}) = \mathcal{S}_{\boldsymbol{\theta}}[\bar{\boldsymbol{a}}^*](\boldsymbol{y})$. Note that $\mathcal{S}_{\boldsymbol{\theta}}[\bar{\boldsymbol{a}}^*](\boldsymbol{y})$ can predict a solution value for arbitrary point $\boldsymbol{y} \in \mathcal{Y}$, potentially $\boldsymbol{y} \notin \bar{\mathcal{Y}}_1 \cup \cdots \cup \bar{\mathcal{Y}}_I$.

4 ENERGY-CONSISTENT NEURAL OPERATORS

We propose an Energy-consistent Neural Operator (ENO) as a general framework for learning solution operators of the systems that follow the energy conservation or dissipation law. Our basic idea is to introduce an inductive bias based on the theory of physics into the operator learning framework to bias the output of the DNN-based solution operator to satisfy the laws of physics. ENO is based on the energy-based theory and simultaneously estimate solution operators and gradient flows of energy functional from data. This allows ENO to appropriately capture the energetic behavior of the system without explicit PDEs, unlike existing approaches (e.g., PI-DeepONet and PINO).

Model. Fig. 2 schematically shows an architecture and a training loss of ENO. We model the solution operator and the gradient flow using DNNs. We employ the existing DNN models with parameters $\boldsymbol{\theta}$ for approximating the solution operator $\mathcal{S}$ as $\mathcal{S}_{\boldsymbol{\theta}}$, which we call *operator net* (see the left of Fig. 2). Note that any DNN architecture can be used as the operator net, such as the multi-layer perceptron (MLP), as long as it is differentiable; one can also use the advanced architectures (e.g., DeepONet and FNO) in our framework.

To model the right-hand side of (2), we first parameterize energy functional $\mathcal{H}$ (1) as follows:

$$\mathcal{H}_{\boldsymbol{\phi}}[\boldsymbol{u}^{\boldsymbol{\theta}}] = \int_{\mathcal{X}} F_{\boldsymbol{\phi}} \left(\boldsymbol{u}^{\boldsymbol{\theta}}, \partial_{\boldsymbol{x}} \boldsymbol{u}^{\boldsymbol{\theta}}, \partial_{\boldsymbol{xx}} \boldsymbol{u}^{\boldsymbol{\theta}}, \ldots \right) d\boldsymbol{x}, \tag{5}$$

where we model the energy density F as $F_{\boldsymbol{\phi}}$ using another DNN with parameters $\boldsymbol{\phi}$, which we call *energy net* (see the right of Fig. 2) [1]. Note that $\boldsymbol{u}^{\boldsymbol{\theta}}$

[1] We need to determine the types of inputs the energy net takes. Since the energy density only needs at most up to second derivatives in most cases, one practical solution is that the up to second derivatives are taken into the energy net. It can be expected that the energy net automatically selects the inputs in the learning process.

in (5) represents that $\boldsymbol{u}$ is parameterized by $\boldsymbol{\theta}$, as $\boldsymbol{u}^{\boldsymbol{\theta}}(\boldsymbol{y}) = \mathcal{S}_{\boldsymbol{\theta}}[\bar{\boldsymbol{a}}](\boldsymbol{y})$. Also, one can use any DNN architecture as the energy net, such as the MLP, as long as it is differentiable.

The way to calculate the functional derivatives $\frac{\delta \mathcal{H}[u]}{\delta u}$ efficiently and accurately is not trivial. A naive approach is to approximate the energy functional $\mathcal{H}[u]$ by discretizing a space domain $\mathcal{X}$, which allows one to evaluate it using the gradients of the finite-dimensional approximated solution, as in (Celledoni et al., 2012); however, the accurate computation of $\mathcal{H}[u]$ requires a fine-scale discretization, which leads to a prohibitive computational cost. Thus, we propose to utilize the result of functional calculus based on the Taylor expansion (Olver, 1993), represented by

$$\frac{\delta \mathcal{H}_{\boldsymbol{\phi}}[\boldsymbol{u}^{\boldsymbol{\theta}}]}{\delta u_m^{\boldsymbol{\theta}}} = \frac{\partial F_{\boldsymbol{\phi}}}{\partial u_m^{\boldsymbol{\theta}}} - \sum_{d=1}^{D} \left\{ \frac{\partial}{\partial x_d} \left(\frac{\partial F_{\boldsymbol{\phi}}}{\partial u_{m,d}^{\boldsymbol{\theta}}} \right) \right\} + \cdots, \quad (6)$$

for $m = 1, \ldots, M$. Here, $u_m^{\boldsymbol{\theta}}$ is the mth element in $\boldsymbol{u}^{\boldsymbol{\theta}}$, x_d is the dth element in $\boldsymbol{x}$, and $u_{m,d}^{\boldsymbol{\theta}}$ denotes $\partial u_m^{\theta} / \partial x_d$. The use of (6) enables us to evaluate the functional derivative using automatic differentiation. This approach is advantageous because it does not rely on numerical approximation to obtain the functional derivative; moreover, the use of automatic differentiation is more accurate and efficient compared to the naive approximation via finite differences.

Training. The loss function of ENO is defined by

$$L_{\mathrm{ENO}}(\boldsymbol{\theta}, \boldsymbol{\phi}) = L(\boldsymbol{\theta}) + \lambda\, \Omega(\boldsymbol{\theta}, \boldsymbol{\phi}), \quad (7)$$

where $L(\boldsymbol{\theta})$ is the data loss (4) of the standard operator learning, $\Omega(\boldsymbol{\theta}, \boldsymbol{\phi})$ is the penalty, and $\lambda \in \mathbb{R}_{\geq 0}$ is a hyperparameter. The penalty $\Omega(\boldsymbol{\theta}, \boldsymbol{\phi})$ is given by

$$\Omega(\boldsymbol{\theta}, \boldsymbol{\phi}) = \frac{1}{I} \sum_{i=1}^{I} \left(\frac{1}{K} \sum_{k=1}^{K} \left\| \dot{\boldsymbol{u}}_i^{\boldsymbol{\theta}}(\boldsymbol{y}_k) - \mathcal{G} \frac{\delta \mathcal{H}_{\boldsymbol{\phi}}[\boldsymbol{u}_i^{\boldsymbol{\theta}}]}{\delta \boldsymbol{u}_i^{\boldsymbol{\theta}}}(\boldsymbol{y}_k) \right\|^2 \right), \quad (8)$$

where $\boldsymbol{u}_i^{\boldsymbol{\theta}} = \mathcal{S}_{\boldsymbol{\theta}}[\bar{\boldsymbol{a}}_i]$ and $\dot{\boldsymbol{u}}_i^{\boldsymbol{\theta}}$ denotes its time-derivative that can be obtained using automatic differentiation. Our penalty function (8) can be considered at $K \in \mathbb{N}$ arbitrary points $\{\boldsymbol{y}_k \mid k = 1, \ldots, K\} \subset \mathcal{Y}$, potentially not included in training data points $\bar{\mathcal{Y}}_1 \cup \cdots \cup \bar{\mathcal{Y}}_I$. By considering the penalty, we can introduce the inductive bias to the time-derivative $\dot{\boldsymbol{u}}_i^{\boldsymbol{\theta}}$ of the solution to obey the energy conservation or dissipation law. See the last paragraph for how to determine λ and $\mathcal{G}$.

Parameters $\boldsymbol{\theta}$ (for operator net) and $\boldsymbol{\phi}$ (for energy net) are estimated by minimizing the loss (7); ENO can infer not only the solution operator but also the energy functional from data. The training procedure for ENO is shown in Algorithm 1. In line 6 of Algorithm 1, we uniformly sample K points $\{\boldsymbol{y}_k\}$ and add up the penalty terms at the sampled points. This scheme resembles the concept of *collocation points* in physics-informed neural networks (Raissi et al., 2019). This scheme yields the smoothing effect based on the laws of physics over the entire spatio-temporal domain, allowing the appropriate training of solution operators, even when training data resolution is lower, such as super-resolution settings.

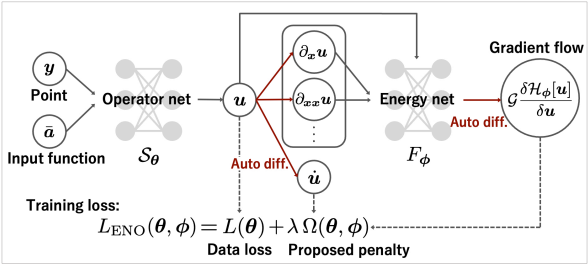


Figure 2: Schematic sketch of the architecture and training loss of ENO. Red arrows indicate automatic differentiation. ENO contains two networks: *operator net* and *energy net* parameterize solution operator and energy functional, respectively. By simultaneously minimizing data loss $L(\boldsymbol{\theta})$ and penalty $\Omega(\boldsymbol{\theta}, \boldsymbol{\phi})$ inspired by energy-based theory, we can obtain a solution operator (i.e., operator net) to predict a solution that follows energy conservation or dissipation law without using explicit PDEs and Hamiltonian.

Hyperparameters. ENO has two hyperparameters to be determined. The first is λ in (7) that controls the penalty for violating physical constraints. The second is the differential operator $\mathcal{G}$ in (8) that governs systems' energetic behavior. They can be determined based on the validation errors. As described in Section 3.1, in most cases, $\mathcal{G}$ is a few simple operators; hence, we can adopt a validation procedure to select an appropriate operator from predefined candidates.

Computational costs. Although there may be concerns regarding the computational costs of using automatic differentiation, these costs scale linearly with the dimension D of the space, as the differential operator $\mathcal{G}$ is typically represented by a sum of spatial derivatives (see Section 3.1). Moreover, the number K of sampled evaluation points can be controlled. As a result, the computational costs in multi-dimensional cases do not increase significantly.

5 EXPERIMENTS

We demonstrate the effectiveness of our proposed model, ENO, using simulation data of Hamiltonian and dissipative PDE systems. We focus on tasks that predict solution functions when we are given initial conditions as input functions.

Algorithm 1 Training procedure for ENO

Input: Data $\{(\bar{\boldsymbol{a}}_i, \bar{\boldsymbol{u}}_i)\}_{i=1}^I$, data points $\bar{\mathcal{Y}} = \bar{\mathcal{Y}}_1 \cup \cdots \cup \bar{\mathcal{Y}}_I$, mini-batch size I_b, number of points K for penalty, hyperparameters λ, $\mathcal{G}$
Output: Trained DNN parameters $\boldsymbol{\theta}$, $\boldsymbol{\phi}$
Initialize DNN parameters $\boldsymbol{\theta}$, $\boldsymbol{\phi}$.
repeat
  Randomly sample I_b indices from $\{1, \ldots, I\}$.
  Uniformly sample K points $\{\boldsymbol{y}_k\}$ for penalty, all of which are contained in domain $\mathcal{Y}$.
  /* Predict solutions */
  Predict $\boldsymbol{u}$ at all points $\bar{\mathcal{Y}} \cup \{\boldsymbol{y}_k\}$ for respective input functions $\{\bar{\boldsymbol{a}}_i\}$ via operator net $\mathcal{S}_{\boldsymbol{\theta}}$.
  /* Estimate gradient flows */
  Obtain partial derivatives $\dot{\boldsymbol{u}}, \partial_{\boldsymbol{x}}\boldsymbol{u}, \partial_{\boldsymbol{xx}}\boldsymbol{u}, \ldots$ at points $\{\boldsymbol{y}_k\}$ via automatic differentiation.
  Compute the energy by inputting the estimated solution and partial derivatives into the energy net $F_{\boldsymbol{\phi}}$, and obtain the gradient flow (6) through automatic differentiation.
  /* Update parameters */
  Update parameters $\boldsymbol{\theta}$, $\boldsymbol{\phi}$ using the gradient of ENO loss (7) via a stochastic gradient method.
until End condition is satisfied.

Data. We generated simulation data of PDE systems whose energy functional $\mathcal{H}$ and differential operator $\mathcal{G}$ are known. Notice that the explicit forms of $\mathcal{H}$ and $\mathcal{G}$ were used only for data generation and were assumed to be *unknown in the training phase*. We evaluated the proposed ENO on a Hamiltonian PDE, the Korteweg–de Vries (KdV) equation (Korteweg and de Vries, 1895), and a dissipative PDE, the Cahn–Hilliard equation (Cahn and Hilliard, 1958). We generated trajectories of u from randomly sampled initial conditions by employing a numerical solver. Space was uniformly discretized to $N_\mathrm{x} = 100$ cells in $\mathcal{X}$, and time was uniformly discretized to $N_\mathrm{t} = 1000$ points in $\mathcal{T}$. The details of data are described in Appendix B.1.

Task. In our experiments, we considered a super-resolution setting; we predicted the high-resolution test data from the low-resolution training data. We first generated 1000 trajectories from different initial conditions with high-resolution: $(N_\mathrm{x}, N_\mathrm{t}) = (100, 1000)$, of which 90% were used for training and 10% for validation. We then created three different resolutions $(N_\mathrm{x}, N_\mathrm{t}) \in \{(10, 10), (15, 15), (25, 25)\}$ of data by downsampling the original high-resolution data. We generated 100 test trajectories with high-resolution, whose initial conditions are different from the training and validation data. The experiments were conducted five times by resampling the training, validation, and test sets.

ENO setup. We adopted two multi-layer perceptrons (MLPs) to implement the operator net $\mathcal{S}_{\boldsymbol{\theta}}[a](\boldsymbol{y}) = \mathrm{MLP}(\bar{a} \oplus \boldsymbol{y})$ and the energy net $F_{\boldsymbol{\phi}}(u, \partial_x u) = \mathrm{MLP}(u \oplus \partial_x u)$, where $\oplus$ denotes the concatenation operator, and the types of inputs the energy net takes were assumed to be known. The respective nets had three layers, 200 hidden units, and tanh activations. We trained them by minimizing the ENO loss (7), where the validation data were used for early stopping, and the maximum number of epochs was 10000. We used the Adam optimizer (Kingma and Ba, 2015) implemented in PyTorch (Paszke et al., 2019), and set the learning rates for $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ to 10^{-3} and 10^{-4}, respectively. Mini-batch size I_b in Algorithm 1 was 30. Points $\{\boldsymbol{y}_k\}_{k=1}^K$ for the penalty (8) were uniformly sampled in the spatio-temporal domain, where K was set to 200. Hyperparameters λ and $\mathcal{G}$ were chosen from $\{10^{-8}, 10^{-7}, \cdots, 10^{-1}\}$ and $\{-1, \partial/\partial x, \partial^2/\partial x^2\}$ based on the loss (4) for the validation data.

One benefit of ENO is that it can consider the penalty term (8) at uniformly sampled points in Algorithm 1. To verify its effectiveness, we prepared a variant (called ENO (FIXED)) considering the penalty term evaluated only at fixed data points in training data. If fewer than 200 points were available in training data, all points were used; otherwise, 200 points were randomly selected from fixed training data points at each epoch.

Baselines. To evaluate the effectiveness of the proposed penalty (8), we compared ENO with a vanilla neural operator (called VANILLA NO) implemented by the MLP, which corresponds to the model that excludes the penalty term from our ENO. We also adopted the two most popular models that can learn solution operators from data without requiring explicit PDEs: the deep operator network (DEEPONET) (Lu et al., 2021) and the Fourier neural operator (FNO) (Li et al., 2021), with further details provided in Appendix B.2.

In addition, we compared ENO with a case (called Energy Regularizer (ENERREG)) where the explicit forms of $\mathcal{G}$ and $\mathcal{H}$ are available, which corresponds to an extension of (Mattheakis et al., 2022) that regularizes neural networks using a *known* energy functional. Also, ENERREG can essentially be regarded as equivalent to PI-DeepONet and PINO, which utilize the explicit forms of PDEs. Indeed, this comparison is unfair because ENO assumes that $\mathcal{G}$ and $\mathcal{H}$ are unavailable; nevertheless, it is still helpful in showing the gap from an *upper-bound performance*.

Results. Table 2 shows the mean squared error (MSE) between the true and predicted solution trajectories for ENO and the baselines (see the column TRAJ.). It also shows the MSE for energy and mass

Table 2: Average MSEs when using three different resolutions of training data. The best results (excluding EnerReg) are emphasized by bold font. EnerReg addresses the problem different from ours; it assumes the availability of explicit forms of $\mathcal{G}$ and $\mathcal{H}$, which corresponds to an upper-bound performance for our ENO. Standard deviations are in Appendix B.4.

(a) KdV equation: Multiplied by 10^4 for Traj. and by 10^2 for Mass.

	$(N_\mathrm{x}, N_\mathrm{t}) = (10, 10)$			$(N_\mathrm{x}, N_\mathrm{t}) = (15, 15)$			$(N_\mathrm{x}, N_\mathrm{t}) = (25, 25)$		
	Traj.	Energy	Mass	Traj.	Energy	Mass	Traj.	Energy	Mass
ENO	**2.14**	**3.38**	**3.88**	**0.82**	**0.34**	**1.49**	**0.53**	**0.12**	0.75
ENO (fixed)	31.07	36.16	83.16	1.05	0.45	2.42	0.54	0.14	**0.62**
Vanilla NO	77.23	132.13	187.59	26.55	25.85	18.85	0.69	0.15	0.63
DeepONet	91.82	168.18	359.81	47.54	42.70	80.35	5.22	1.45	7.86
FNO	62.88	168.26	168.85	34.89	40.65	96.61	1.13	0.58	2.35
EnerReg	1.39	1.95	2.50	0.81	0.36	1.51	0.51	0.15	0.57

(b) Cahn–Hilliard equation: Multiplied by 10^3 for Traj., by 10^6 for Energy, and by 10^1 for Mass.

	$(N_\mathrm{x}, N_\mathrm{t}) = (10, 10)$			$(N_\mathrm{x}, N_\mathrm{t}) = (15, 15)$			$(N_\mathrm{x}, N_\mathrm{t}) = (25, 25)$		
	Traj.	Energy	Mass	Traj.	Energy	Mass	Traj.	Energy	Mass
ENO	**70.88**	**62.43**	**9.98**	**2.71**	**16.42**	**2.76**	**0.45**	**0.71**	**0.22**
ENO (fixed)	154.13	1461.30	137.83	12.49	127.55	5.44	0.77	1.59	0.29
Vanilla NO	193.32	1918.18	436.69	39.41	1783.07	29.95	0.87	1.86	0.30
DeepONet	127.76	895.19	301.72	18.54	425.36	11.81	3.39	13.91	0.94
FNO	176.40	405.09	245.25	88.38	702.60	50.41	2.14	8.37	1.86
EnerReg	38.42	62.42	6.81	1.48	7.84	2.29	0.30	0.54	0.15

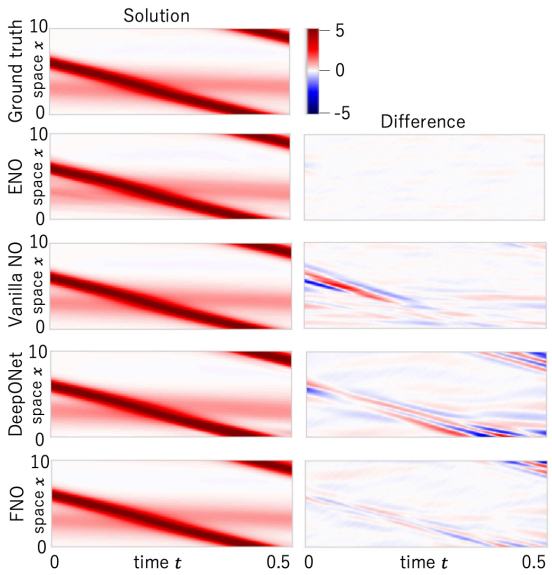
calculated using the predicted trajectories (see the columns Energy and Mass). These MSE metrics are detailed in Appendix B.3. In Table 2, we provided an average of MSEs over five trials; we omitted the standard deviations for readability (see Appendix B.4 for the full results). In all cases, ENO achieved comparable or better performance than the baselines (excluding EnerReg) regarding the trajectory, energy, and mass; the performance improvements were significant in the settings where the training data resolution was lower. Also, in all cases, ENO was able to select the true differential operator $\mathcal{G}$ via the validation procedure (see Table 4 in Appendix B.4). The errors of ENO were lower than those of ENO (Fixed), especially in these settings. This indicates the effectiveness of considering the penalty inspired by the energy-based theory at uniformly sampled points to regularize the operator net on the entire spatio-temporal domain (not only at data points). In addition, ENO achieved performance comparable [2] to EnerReg (corresponding to the upper bound performance) when the training data resolution is relatively high. This is reasonable because the energy functional is better estimated by the energy net of ENO in such situations.

[2] Although EnerReg slightly underperforms ENO in a few cases in Table 2, this is due to the effect of random seeds, and there were no statistically significant differences between them (Student's t-test, $P < 0.05$).

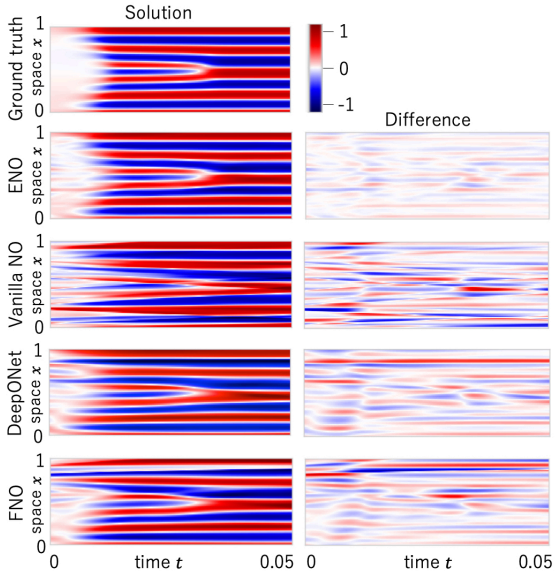
Fig. 3 visualizes the solutions predicted by ENO, Vanilla NO, DeepONet, and FNO. Also, the energetic behavior estimated by each model is shown in Fig. 4. Visualization results with other models are shown in Appendix B.4. These figures show that our ENO more appropriately captured the physical behavior than the baselines.

These results indicate that our ENO can accurately predict solutions while better capturing the energy conservation or dissipation law than naive neural network-based operators. Also, ENO can significantly improve the performance, especially when training data resolution is lower.

Computational time. The average training time (per hyperparameter set) of ENO was 6111.40 seconds and 6120.17 seconds for the KdV equation and Cahn-Hilliard equation, respectively, when the data resolution was $(N_\mathrm{x}, N_\mathrm{t}) = (15, 15)$. Comparisons with other models are shown in Appendix B.4. In testing, ENO (and other models) took only about 0.15 seconds to obtain one solution. The experiments were conducted on a single NVIDIA A100 GPU. Although ENO training often requires more time than the baseline methods, it can lead to significant improvements in prediction accuracy, as demonstrated in Table 2 and Figs. 3 and 4. One can observe that ENO took up to 20 times longer than Vanilla NO, but the prediction error of the solu-

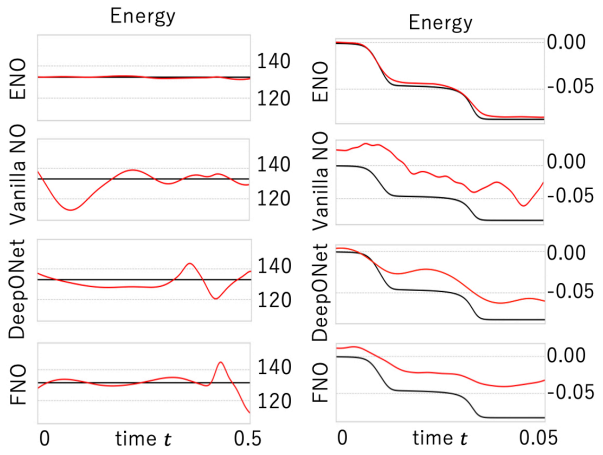


(a) KdV equation ($(N_\mathrm{x}, N_\mathrm{t}) = (10, 10)$).



(b) Cahn-Hilliard equation ($(N_\mathrm{x}, N_\mathrm{t}) = (15, 15)$).

Figure 3: Predicted solutions. Right column for each system is the difference between ground truth and its prediction, where the difference values for KdV equation were multiplied by 4. Visualization results with other models are in Appendix B.4.



(a) KdV equation. (b) Cahn-Hilliard equation.

Figure 4: Estimated energetic behavior. Black and red lines are the true energy and its estimate, respectively.

tion was improved by a factor of 30 (see TRAJ. column when $(N_x, N_t) = (10, 10)$ in Table 2(a)). It is also important to note that the key aspect of operator learning is the speed of inference.

Accuracy in energy estimation. We can compare the estimated energy functional $\mathcal{H}$ with its actual one through an appropriate calibration process. In Appendix B.5, we report the accuracy of estimating the energy functional $\mathcal{H}$ via the proposed model.

Additional experiments. We also conducted additional experiments on the Allen-Cahn equation and the two-dimensional Cahn-Hilliard equation, which we describe in Appendix B.6.

6 DISCUSSION

Conclusion. We proposed an Energy-consistent Neural Operator (ENO) to train neural operators capable of predicting physical behaviors that adhere to energy conservation or dissipation laws without the need for explicit PDEs. Our key contribution is a penalty function derived from energy-based theory, where functional derivatives are computed using automatic differentiation, eliminating the need for numerical approximation. ENO is a general framework that can be applied to various operator learning problems. In our experiments, we demonstrated ENO's effectiveness across multiple PDEs, outperforming multiple baselines.

The scope of ENO. Our ENO can be applied to many time-dependent PDEs that are modeled as Hamiltonian formulations contained in (Celledoni et al., 2012; Sergei, 2000). For example, the Navier-Stokes equations for an *inviscid and incompressible* fluid (also called the Euler equation) can actually be modeled as a Hamiltonian formulation (Sanders et al., 2024; Olver, 1982); thus, our proposed ENO is applicable to this equation. Meanwhile, *time-independent* steady-state phenomena (e.g., the Darcy flow equation) are out of the focus of our research because Hamiltonian and dissipative PDEs are for *time-dependent* PDE systems.

Acknowledgments

This work was supported by JST ASPIRE Grant Number JPMJAP2329 and JST ACT-X Grant Number JPMJAX210D, Japan.

References

Athanasopoulos, M., Ugail, H., and Castro, G. G. (2009). Parametric design of aircraft geometry using partial differential equations. *Advances in Engineering Software*, 40(7):479–486.

Bhattacharya, K., Hosseini, B., Kovachki, N. B., and Stuart, A. M. (2021). Model reduction and neural networks for parametric PDEs. *The SMAI Journal of computational mathematics*, 7:121–157.

Bonev, B., Kurth, T., Hundt, C., Pathak, J., Baust, M., Kashinath, K., and Anandkumar, A. (2023). Spherical Fourier neural operators: Learning stable dynamics on the sphere. In *International Conference on Machine Learning*, volume 202, pages 2806–2823.

Boullé, N., Earls, C. J., and Townsend, A. (2022). Data-driven discovery of Green's functions with human-understandable deep learning. *Scientific Reports*, 12(1):4824.

Brandstetter, J., Worrall, D. E., and Welling, M. (2022). Message passing neural PDE solvers. In *International Conference on Learning Representations*.

Brunton, S. L. and Kutz, J. N. (2019). *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press.

Cahn, J. W. and Hilliard, J. E. (1958). Free energy of a nonuniform system. I. Interfacial free energy. *The Journal of Chemical Physics*, 28(2):258–267.

Cai, S., Wang, Z., Lu, L., Zaki, T. A., and Karniadakis, G. E. (2021). DeepM&Mnet: Inferring the electroconvection multiphysics fields based on operator approximation by neural networks. *Journal of Computational Physics*, 436:110296.

Cao, S. (2021). Choose a transformer: Fourier or Galerkin. In *Advances in Neural Information Processing Systems*, volume 34, pages 24924–24940.

Celledoni, E., Grimm, V., McLachlan, R., McLaren, D., O'Neale, D., Owren, B., and Quispel, G. (2012). Preserving energy resp. dissipation in numerical PDEs using the "average vector field" method. *Journal of Computational Physics*, 231(20):6770–6789.

Chen, T. and Chen, H. (1995). Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. *IEEE Transactions on Neural Networks*, 6(4):911–917.

Chen, Y., Matsubara, T., and Yaguchi, T. (2021). Neural symplectic form: Learning Hamiltonian equations on general coordinate systems. In *Advances in Neural Information Processing Systems*.

Chen, Z., Zhang, J., Arjovsky, M., and Bottou, L. (2020). Symplectic recurrent neural networks. In *International Conference on Learning Representations*.

Courant, R., Friedrichs, K. O., and Lewy, H. (1967). On the partial difference equations of mathematical physics. *IBM Journal of Research and Development*, 11(2):215–234.

Course, K., Evans, T., and Nair, P. (2020). Weak form generalized Hamiltonian learning. In *Advances in Neural Information Processing Systems*, volume 33, pages 18716–18726.

De Ryck, T. and Mishra, S. (2022). Generic bounds on the approximation error for physics-informed (and) operator learning. In *Advances in Neural Information Processing Systems*, volume 35, pages 10945–10958.

Desai, S. A., Mattheakis, M., Sondak, D., Protopapas, P., and Roberts, S. J. (2021). Port-Hamiltonian neural networks for learning explicit time-dependent dynamical systems. *Physical Review E*, 104:034312.

Eidnes, S. and Lye, K. O. (2024). Pseudo-Hamiltonian neural networks for learning partial differential equations. *Journal of Computational Physics*, 500:112738.

Finzi, M., Wang, K. A., and Wilson, A. G. (2020). Simplifying Hamiltonian and Lagrangian neural networks via explicit constraints. In *Advances in Neural Information Processing Systems*, volume 33, pages 13880–13889.

Furihata, D. (1999). Finite difference schemes for $\partial u/\partial t = (\partial/\partial x)^{\alpha}\delta g/\delta u$ that inherit energy conservation or dissipation property. *Journal of Computational Physics*, 156(1):181–205.

Gin, C. R., Shea, D. E., Brunton, S. L., and Kutz, J. N. (2021). DeepGreen: Deep learning of Green's functions for nonlinear boundary value problems. *Scientific Reports*, 11(1):21614.

Goldstein, H. (1980). *Classical Mechanics*. Addison-Wesley.

Greydanus, S., Dzamba, M., and Yosinski, J. (2019). Hamiltonian neural networks. In *Advances in Neural Information Processing Systems*, volume 32.

Gupta, G., Xiao, X., and Bogdan, P. (2021). Multiwavelet-based operator learning for differential equations. In *Advances in Neural Information Processing Systems*, volume 34, pages 24048–24062.

Hansen, D., Maddix, D. C., Alizadeh, S., Gupta, G., and Mahoney, M. W. (2023). Learning physical models that can respect conservation laws. In *International Conference on Machine Learning*, volume 202, pages 12469–12510.

Hao, Z., Wang, Z., Su, H., Ying, C., Dong, Y., Liu, S., Cheng, Z., Song, J., and Zhu, J. (2023). Gnot: A general neural operator transformer for operator learning. In *International Conference on Machine Learning*.

Helwig, J., Zhang, X., Fu, C., Kurtin, J., Wojtowytsch, S., and Ji, S. (2023). Group equivariant Fourier neural operators for partial differential equations. In *International Conference on Machine Learning*, volume 202, pages 12907–12930.

Horie, M. and Mitsume, N. (2022). Physics-embedded neural networks: Graph neural PDE solvers with mixed boundary conditions. In *Advances in Neural Information Processing Systems*, volume 35, pages 23218–23229.

Jin, P., Zhang, Z., Kevrekidis, I. G., and Karniadakis, G. E. (2022). Learning Poisson systems and trajectories of autonomous systems via Poisson neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–13.

Karniadakis, G. E., Kevrekidis, I. G., Lu, L., Perdikaris, P., Wang, S., and Yang, L. (2021). Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–400.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *International Conference on Learning Representations*.

Korteweg, D. J. and de Vries, G. (1895). On the change of form of long waves advancing in a rectangular canal, and on a new type of long stationary waves. *Philosophical Magazine Series 1*, 39:422–443.

Lelièvre, T. and Stoltz, G. (2016). Partial differential equations and stochastic methods in molecular dynamics. *Acta Numerica*, 25:681–880.

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., and Anandkumar, A. (2020a). Neural operator: Graph kernel network for partial differential equations. In *arXiv*.

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Stuart, A., Bhattacharya, K., and Anandkumar, A. (2020b). Multipole graph neural operator for parametric partial differential equations. In *Advances in Neural Information Processing Systems*, volume 33, pages 6755–6766.

Li, Z., Kovachki, N. B., Azizzadenesheli, K., liu, B., Bhattacharya, K., Stuart, A., and Anandkumar, A. (2021). Fourier neural operator for parametric partial differential equations. In *International Conference on Learning Representations*.

Li, Z., Zheng, H., Kovachki, N., Jin, D., Chen, H., Liu, B., Azizzadenesheli, K., and Anandkumar, A. (2023). Physics-informed neural operator for learning partial differential equations. In *arXiv*.

Liang, S., Huang, Z., and Zhang, H. (2022). Stiffness-aware neural network for learning Hamiltonian systems. In *International Conference on Learning Representations*.

Liu, N., Fan, Y., Zeng, X., Klöwer, M., Zhang, L., and Yu, Y. (2024). Harnessing the power of neural operators with automatically encoded conservation laws. In *International Conference on Machine Learning*.

Lu, L., Jin, P., Pang, G., Zhang, Z., and Karniadakis, G. E. (2021). Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229.

Lynch, P. (2008). The origins of computer weather prediction and climate modeling. *Journal of Computational Physics*, 227(7):3431–3444.

Mao, Z., Lu, L., Marxen, O., Zaki, T. A., and Karniadakis, G. E. (2021). DeepM&Mnet for hypersonics: Predicting the coupled flow and finite-rate chemistry behind a normal shock using neural-network approximation of operators. *Journal of Computational Physics*, 447:110698.

Matsubara, T., Ishikawa, A., and Yaguchi, T. (2020). Deep energy-based modeling of discrete-time physics. In *Advances in Neural Information Processing Systems*, volume 33, pages 13100–13111.

Mattheakis, M., Sondak, D., Dogra, A. S., and Protopapas, P. (2022). Hamiltonian neural networks for solving equations of motion. *Physical Review E*, 105:065305.

Meng, C., Seo, S., Cao, D., Griesemer, S., and Liu, Y. (2022). When physics meets machine learning: A survey of physics-informed machine learning. In *arXiv*.

Olver, P. J. (1982). A nonlinear Hamiltonian structure for the Euler equations. *Journal of Mathematical Analysis and Applications*, 89(1):233–250.

Olver, P. J. (1993). *Applications of Lie Groups to Differential Equations, Second Edition*. American Mathematical Soc.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala,

S. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024–8035.

Patel, R. G., Trask, N. A., Wood, M. A., and Cyr, E. C. (2021). A physics-informed operator regression framework for extracting data-driven continuum models. *Computer Methods in Applied Mechanics and Engineering*, 373:113500.

Poli, M., Massaroli, S., Berto, F., Park, J., Dao, T., Ré, C., and Ermon, S. (2022). Transform once: Efficient operator learning in frequency domain. In *Advances in Neural Information Processing Systems*, volume 35, pages 7947–7959.

Quispel, G. and Capel, H. (1996). Solving ODEs numerically while preserving a first integral. *Physics Letters A*, 218(3):223–228.

Rahman, M. A., Ross, Z. E., and Azizzadenesheli, K. (2023). U-NO: U-shaped neural operators. *Transactions on Machine Learning Research*.

Raissi, M., Perdikaris, P., and Karniadakis, G. (2019). 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.

Raonic, B., Molinaro, R., De Ryck, T., Rohner, T., Bartolucci, F., Alaifari, R., Mishra, S., and de Bézenac, E. (2023). Convolutional neural operators for robust and accurate learning of PDEs. In *Advances in Neural Information Processing Systems*, volume 36, pages 77187–77200.

Rath, K., Albert, C. G., Bischl, B., and von Toussaint, U. (2021). Symplectic Gaussian process regression of maps in Hamiltonian systems. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 31(5):053121.

Richter-Powell, J., Lipman, Y., and Chen, R. T. Q. (2022). Neural conservation laws: A divergence-free perspective. In *Advances in Neural Information Processing Systems*.

Sanders, J. W., DeVoria, A., Washuta, N. J., Elamin, G. A., Skenes, K. L., and Berlinghieri, J. C. (2024). A canonical Hamiltonian formulation of the Navier–Stokes problem. *Journal of Fluid Mechanics*, 984:A27.

Sergei, B. K. (2000). *Analysis of Hamiltonian PDEs*. Oxford.

Sosanya, A. and Greydanus, S. (2022). Dissipative Hamiltonian neural networks: Learning dissipative and conservative dynamics separately. In *arXiv*.

Tanaka, Y., Iwata, T., and Ueda, N. (2022). Symplectic spectrum Gaussian processes: Learning Hamiltonians from noisy and sparse data. In *Advances in Neural Information Processing Systems*, volume 35, pages 20795–20808.

Wang, S., Wang, H., and Perdikaris, P. (2021). Learning the solution operator of parametric partial differential equations with physics-informed DeepONets. *Science Advances*, 7(40):eabi8605.

Zhong, Y. D., Dey, B., and Chakraborty, A. (2020a). Dissipative SymODEN: Encoding Hamiltonian dynamics with dissipation and control into deep learning. In *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations*.

Zhong, Y. D., Dey, B., and Chakraborty, A. (2020b). Symplectic ODE-Net: Learning Hamiltonian dynamics with control. In *International Conference on Learning Representations*.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Yes] The proposed model has been clearly formulated in Section 4; the training algorithm has been described in Algorithm 1.
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Yes] We have reported the performance and computational time in multiple experimental settings (see Tables 2, 3, 5).
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [Not Applicable]

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. [Not Applicable]
   (b) Complete proofs of all theoretical results. [Not Applicable]
   (c) Clear explanations of any assumptions. [Not Applicable]

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [Yes] We have attached the code to the supplemental material. The details of data generation were described in Appendix B.1.

(b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Yes] The training details have been specified in the ENO setup paragraph in Section 5.

(c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [Yes] The evaluation metrics have been detailed in Appendix B.3. Standard deviations were in Appendix B.4.

(d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Yes] See the Computational time paragraph in Section 5.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:

(a) Citations of the creator If your work uses existing assets. [Yes] We have cited the existing assets of PyTorch (Section 5) and FNO (Appendix B.2).

(b) The license information of the assets, if applicable. [Yes] Same as above.

(c) New assets either in the supplemental material or as a URL, if applicable. [Yes] We have attached the code and documentation to the supplemental material.

(d) Information about consent from data providers/curators. [Not Applicable]

(e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Not Applicable]

5. If you used crowdsourcing or conducted research with human subjects, check if you include:

(a) The full text of instructions given to participants and screenshots. [Not Applicable]

(b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Not Applicable]

(c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Not Applicable]

Energy-consistent Neural Operators for Hamiltonian and Dissipative Partial Differential Equations: Supplementary Materials

A RELATION TO HAMILTONIAN SYSTEMS

In this appendix, we mention the relationship between the energy-based theory (described in Section 3.1) and Hamiltonian systems. Hamiltonian systems are well known as ordinary differential equation (ODE) systems with energy conservation laws and can be regarded as a special case of (2). In Hamiltonian mechanics (Goldstein, 1980), the system's state $\boldsymbol{u} : \mathcal{T} \to \mathbb{R}^M$ is defined on the product space of generalized coordinates and generalized momenta. The functional $\mathcal{H}$ and its functional derivative in (2) are replaced with the energy *function* (called *Hamiltonian*) and its gradient, respectively. The differential operator $\mathcal{G}$ is reduced to the skew-symmetric matrix.

B EXPERIMENTAL RESULTS

B.1 Data

This appendix contains the details of data used in Section 5. We evaluated the proposed ENO on a Hamiltonian PDE, namely the one-dimensional Korteweg–de Vries (KdV) equation (Korteweg and de Vries, 1895) under the periodic boundary condition, which is a shallow water wave equation defined on the time-space $\mathcal{T} \times \mathcal{X} = [0, 0.5] \times [0, 10]$. The energy functional $\mathcal{H}$ for the function $u : \mathcal{T} \times \mathcal{X} \to \mathbb{R}$ is given by $\mathcal{H}[u] = \int_{\mathcal{X}} F(u, \partial_x u)\, dx$, where

$$F(u, \partial_x u) = u^3 - \frac{1}{2}(\partial_x u)^2. \tag{9}$$

An input function (i.e., an initial condition) $a : \mathcal{X} \to \mathbb{R}$ was set to a sum of two solitons, represented by

$$a(x) = \sum_{i=1}^{2} 2\kappa_i^2 \mathrm{sech}^2\left(\kappa_i (x - d_i)\right), \tag{10}$$

where we set $d_1 = 3$ and $d_2 = 6$, and κ_1 and κ_2 were uniformly sampled across ranges $[0.5, 1.0]$ and $[1.5, 2.0]$, respectively.

We also evaluated ENO on a dissipative PDE, namely the one-dimensional Cahn–Hilliard equation (Cahn and Hilliard, 1958) under the periodic boundary condition, which is often used for modeling a phase separation of copolymer melts, defined on the time-space $\mathcal{T} \times \mathcal{X} = [0, 0.05] \times [0, 1]$. The energy functional $\mathcal{H}$ for the function $u : \mathcal{T} \times \mathcal{X} \to \mathbb{R}$ is given by $\mathcal{H}[u] = \int_{\mathcal{X}} F(u, \partial_x u)\, dx$, where

$$F(u, \partial_x u) = \frac{1}{4}u^4 - \frac{1}{2}u^2 + \frac{\gamma}{2}(\partial_x u)^2. \tag{11}$$

Here, the coefficient $\gamma \in \mathbb{R}_{>0}$ denotes the mobility of the monomers, which we set to 0.0005. We used the orthogonal polynomials of degree five as an initial condition $a : \mathcal{X} \to \mathbb{R}$, represented by

$$a(x) = \sum_{i=1}^{5} \beta_i C_i(x), \tag{12}$$

where $C_i(x)$ are Chebyshev polynomials of the first kind, and β_i were uniformly sampled across a range $[0, 0.05]$.

To generate trajectory data of u from respective initial conditions, we first discretize PDEs (2) in the spatial domain, where we used an appropriate discretization introduced in (Celledoni et al., 2012) to ensure the energy conservation or dissipation law. Then, we obtained trajectories by applying a numerical solver, i.e., the Dormand–Prince method with adaptive time-stepping (implemented in SciPy), to the discretized PDEs. Here, the relative and absolute tolerances were set to 10^{-12} and 10^{-14}, respectively. Space was uniformly discretized to $N_{\mathrm{x}} = 100$ cells in $\mathcal{X}$, and time was uniformly discretized to $N_{\mathrm{t}} = 1000$ points in $\mathcal{T}$.

B.2 Baseline

In DeepONet, the solution operator is modeled by an inner product of two latent variables $\boldsymbol{z}(\bar{\boldsymbol{a}}), \boldsymbol{z}(\boldsymbol{y}) \in \mathbb{R}^Q$, as $\mathcal{S}_{\boldsymbol{\theta}}[\bar{\boldsymbol{a}}](\boldsymbol{y}) = \boldsymbol{z}(\bar{\boldsymbol{a}})^\top \boldsymbol{z}(\boldsymbol{y})$, where $\boldsymbol{z}(\bar{\boldsymbol{a}})$ (called branch net) and $\boldsymbol{z}(\boldsymbol{y})$ (called trunk net) are modeled by any neural network with $\bar{\boldsymbol{a}}$ and $\boldsymbol{y}$ as inputs, respectively. In our experiments, the branch and trunk nets were modeled by an MLP with three layers, 200 hidden units, and tanh activations. Dimension Q of the latent variables was set to 30. In FNO, we used the original implementation available online [3]. FNO had four Fourier layers, the dimension of latent space of 32, the Fourier mode of six, and tanh activations. The above hyperparameters were determined based on the validation errors.

B.3 Evaluation Metrics

In this appendix, we describe the evaluation metrics used in Section 5. In Table 2, we verified the effectiveness of ENO using the mean squared errors (MSEs) for the trajectory, energy, and mass predicted by each model. Let $\{(\bar{a}_i^*, \bar{u}_i^*) \mid i = 1, \dots, I^*\}$ be the discretized input-output functions for testing, where I^* is the number of function pairs. Let N_{t}^* and N_{x}^* be the numbers of points in the time and space, respectively.

Traj. in Table 2 is the MSE between the true and predicted solution trajectories, represented by

$$\frac{1}{I^*}\sum_{i=1}^{I^*}\left[\frac{1}{N_{\mathrm{t}}^*}\sum_{j=1}^{N_{\mathrm{t}}^*}\left(\frac{1}{N_{\mathrm{x}}^*}\sum_{k=1}^{N_{\mathrm{x}}^*}||\bar{u}_i^*(t_j,x_k) - u_i^\theta(t_j,x_k)||^2\right)\right], \tag{13}$$

where $\bar{u}_i^*(t_j, x_k)$ and $u_i^\theta(t_j, x_k) = \mathcal{S}_\theta[\bar{a}_i^*](t_j, x_k)$ are the true trajectory and its prediction, respectively.

Energy in Table 2 is the MSE between the true energy $\mathcal{H}[\bar{u}_i^*]$ and the energy $\mathcal{H}[u_i^\theta]$ evaluated using the predicted trajectories. The MSE of the energy is given by

$$\frac{1}{I^*}\sum_{i=1}^{I^*}\left(\frac{1}{N_{\mathrm{t}}^*}\sum_{j=1}^{N_{\mathrm{t}}^*}||\mathcal{H}[\bar{u}_i^*](t_j) - \mathcal{H}[u_i^\theta](t_j)||^2\right), \tag{14}$$

where

$$\mathcal{H}[\bar{u}_i^*](t_j) = \sum_{k=1}^{N_{\mathrm{x}}^*} F\left(\bar{u}_i^*(t_j,x_k), \partial_x \bar{u}_i^*(t_j,x_k)\right)\Delta x \tag{15}$$

and

$$\mathcal{H}[u_i^\theta](t_j) = \sum_{k=1}^{N_{\mathrm{x}}^*} F\left(u_i^\theta(t_j,x_k), \partial_x u_i^\theta(t_j,x_k)\right)\Delta x. \tag{16}$$

Here, Δx is the grid size used to discretize the spatial domain in the test data. We used the appropriate approximation provided in (Celledoni et al., 2012) to calculate $\partial_x \bar{u}_i^*(t_j, x_k)$.

It should be noted that we used the true energy density F (i.e., Eqs. (9) and (11)), not the energy net F_ϕ, for evaluation. This evaluation metric has been widely used to verify whether the predicted trajectories satisfy the energy conservation or dissipation law (e.g., (Greydanus et al., 2019; Matsubara et al., 2020)). Again, the explicit form of F was used only for evaluation and not for training.

Mass in Table 2 is the MSE between the true mass M^{true} and the mass $\mathcal{M}[u_i^\theta]$ evaluated using the predicted trajectories, represented by

$$\frac{1}{I^*}\sum_{i=1}^{I^*}\left(\frac{1}{N_{\mathrm{t}}^*}\sum_{j=1}^{N_{\mathrm{t}}^*}||M^{\mathrm{true}} - \mathcal{M}[u_i^\theta](t_j)||^2\right), \tag{17}$$

where

$$\mathcal{M}[u_i^\theta](t_j) = \sum_{k=1}^{N_{\mathrm{x}}^*} u_i^\theta(t_j,x_k)\Delta x. \tag{18}$$

[3] https://github.com/khassibi/fourier-neural-operator

B.4 Results

This appendix contains the full results in Section 5. Table 3 shows the mean squared errors (MSEs) and standard deviations between the true and predicted solution trajectories for ENO and the baselines (see the column TRAJ.). It also shows the MSEs and standard deviations for the energy and mass calculated using the predicted trajectories (see the columns ENERGY and MASS). The factor of variability that the standard deviations are capturing is the training/validation/test split. In all cases, ENO achieved comparable or better performance than the baselines (excluding ENERREG) regarding the trajectory, energy, and mass; the performance improvements were significant in the settings where the training data resolution was lower.

Table 4 shows the hyperparameters λ and $\mathcal{G}$ estimated by the validation procedures. In each system, ENO was able to select the true differential operator $\mathcal{G}$. This result indicates that ENO can appropriately estimate the systems' energetic behavior without using prior knowledge. ENO(FIXED), the degraded version of the proposed model, sometimes failed to estimate $\mathcal{G}$ when the resolution of the training data was low.

Figs. 5 and 6 show the visualization results for the KdV equation and the Cahn-Hilliard equation, respectively. As shown in the first and second columns of these figures, ENO can more accurately predict solutions than the other models (excluding ENERREG). Moreover, the third column of these figures shows that ENO can appropriately capture the energy conservation (Fig. 5) or dissipation law (Fig. 6) from data without an explicit form of PDE and energy functional.

Table 5 shows the average training time for each model. Although ENO training often requires more time than the baseline methods, it can lead to significant improvements in prediction accuracy, as demonstrated in Table 3 and Figs. 5, 6, and 4. It is also important to note that the key aspect of operator learning is the speed of inference. Actually, in the testing, ENO (and other models) took only about 0.15 seconds to obtain one solution.

B.5 Accuracy in energy estimation

As shown in Eq. (5), the energy functional $\mathcal{H}$ is given by the integral of the energy density F. We have confirmed whether the energy net F_ϕ accurately approximates the actual energy density F^*. Table 6 shows the MSEs between F^* and F_ϕ evaluated using the validation data (sample size: 100) for the KdV equation when $(N_x, N_t) = (10, 10)$. Here, we used the hyperparameters $\lambda, \mathcal{G}$ automatically determined by the validation procedures, and the actual energy density F^* is shown in Eq. (9) in Appendix B.1. It can be seen that the energy net F_ϕ approaches the actual energy F^* as the training progresses. Note that the above evaluation requires an appropriate *calibration process*, which we describe below.

F_ϕ is trained so that the right-hand side $\mathcal{G}\frac{\delta \mathcal{H}}{\delta u}$ of Eq. (2) matches the time derivative $\dot{u}$ of the solutions, as in Eq. (8). The right-hand side of the KdV equation is given by

$$\frac{\partial}{\partial x}\left(\frac{\partial F}{\partial u} - \frac{\partial}{\partial x}\frac{\partial F}{\partial u_x}\right). \tag{19}$$

If F_ϕ is represented as $F_\phi = a + bu + cu_x + G(u, u_x)$, where a, b, c are scalar coefficients and $G(u, u_x)$ are nonlinear terms, then the constant and linear terms a, bu, cu_x are eliminated by the partial derivatives in the right-hand side of the KdV equation above. Thus, the constant and linear terms are not uniquely determined due to the nature of Hamiltonian PDEs of this type. Since these terms do not affect the resulting Hamiltonian PDEs, they can only be determined up to arbitrary constant and linear terms.

This is not a problem when using our penalty term to improve the solution operator. However, if one wants to compare F_ϕ with its actual F^*, the coefficients a, b, c must be calibrated, which is easily done. Suppose $F^*(u, u_x) = F_\phi(u, u_x) + au + bu_x + c$; then we can determine three coefficients using three samples $\{(u^{(i)}, u_x^{(i)})\}_{i=1}^3$; we simply solve the corresponding linear equation, where $u_x^{(i)}$ was obtained by automatic differentiation. In this way, we can compare the calibrated energy $F_\phi(u, u_x) + \hat{a}u + \hat{b}u_x + \hat{c}$ with the actual F^*.

B.6 Additional experiments

In this appendix, we provide additional experiments on the Allen-Cahn equation and the two-dimensional Cahn-Hilliard equation. The energy functional $\mathcal{H}$ of the Allen-Cahn equation for the function $u : \mathcal{T} \times \mathcal{X} \to \mathbb{R}$ is given by $\mathcal{H}[u] = \int_{\mathcal{X}} F(u, \partial_x u)\, dx$, where

$$F(u, \partial_x u) = \frac{\eta}{2}(\partial_x u)^2 - \frac{1}{2}u^2 + \frac{1}{4}u^4. \tag{20}$$

Here, the coefficient $\eta \in \mathbb{R}_{>0}$ was set to 0.001. The experimental setup is the same as for the one-dimensional Cahn-Hilliard equation: three kinds of training resolutions, randomly sampled initial conditions, etc. Table 7 shows the results, where we did not report the errors for mass in the Allen-Cahn equation because this equation do not follow the mass conservation law due to its nature. One can observe that the proposed ENO achieved lower prediction errors for solution trajectories, energy, and mass than the baseline models.

Table 3: Average MSEs and standard deviations when using three different resolutions of training data. The best results (excluding EnerReg) are emphasized by bold font. EnerReg addresses the problem different from ours; it assumes the availability of explicit forms of $\mathcal{G}$ and $\mathcal{H}$, which corresponds to an upper-bound performance for our ENO.

(a) KdV equation

	$(N_x, N_t) = (10, 10)$			$(N_x, N_t) = (15, 15)$			$(N_x, N_t) = (25, 25)$		
	Traj.	Energy	Mass	Traj.	Energy	Mass	Traj.	Energy	Mass
ENO	**2.14**±0.03 ($\times 10^{-4}$)	**3.38**±0.30 ($\times 10^{0}$)	**3.88**±0.48 ($\times 10^{-2}$)	**8.23**±0.35 ($\times 10^{-5}$)	**3.38**±0.45 ($\times 10^{-1}$)	**1.49**±0.32 ($\times 10^{-2}$)	**5.29**±0.08 ($\times 10^{-5}$)	**1.20**±0.04 ($\times 10^{-1}$)	7.48±2.53 ($\times 10^{-3}$)
ENO (fixed)	3.11±0.44 ($\times 10^{-3}$)	3.62±0.54 ($\times 10^{1}$)	8.32±3.14 ($\times 10^{-1}$)	1.05±0.07 ($\times 10^{-4}$)	4.51±0.26 ($\times 10^{-1}$)	2.42±0.53 ($\times 10^{-2}$)	5.42±0.14 ($\times 10^{-5}$)	1.42±0.07 ($\times 10^{-1}$)	**6.20**±0.05 ($\times 10^{-3}$)
Vanilla NO	7.72±0.77 ($\times 10^{-3}$)	1.32±0.13 ($\times 10^{2}$)	1.88±0.47 ($\times 10^{0}$)	2.65±0.39 ($\times 10^{-3}$)	2.59±0.35 ($\times 10^{1}$)	1.88±0.31 ($\times 10^{-1}$)	6.88±0.16 ($\times 10^{-5}$)	1.45±0.03 ($\times 10^{-1}$)	6.33±0.10 ($\times 10^{-3}$)
DeepONet	9.18±0.29 ($\times 10^{-3}$)	1.68±0.07 ($\times 10^{2}$)	3.60±0.08 ($\times 10^{0}$)	4.75±0.59 ($\times 10^{-3}$)	4.27±1.63 ($\times 10^{1}$)	8.04±2.73 ($\times 10^{-1}$)	5.22±0.59 ($\times 10^{-4}$)	1.45±0.28 ($\times 10^{0}$)	7.86±2.71 ($\times 10^{-2}$)
FNO	6.29±0.16 ($\times 10^{-3}$)	1.68±0.06 ($\times 10^{2}$)	1.69±0.06 ($\times 10^{0}$)	3.49±0.51 ($\times 10^{-3}$)	4.07±1.01 ($\times 10^{1}$)	9.66±2.57 ($\times 10^{-1}$)	1.13±0.20 ($\times 10^{-4}$)	5.81±0.62 ($\times 10^{-1}$)	2.35±1.13 ($\times 10^{-2}$)
EnerReg	1.39±0.06 ($\times 10^{-4}$)	1.95±0.09 ($\times 10^{0}$)	2.50±0.23 ($\times 10^{-2}$)	8.09±0.30 ($\times 10^{-5}$)	3.58±0.03 ($\times 10^{-1}$)	1.55±0.43 ($\times 10^{-2}$)	5.09±0.09 ($\times 10^{-5}$)	1.48±0.36 ($\times 10^{-1}$)	5.69±0.36 ($\times 10^{-3}$)

(b) Cahn–Hilliard equation

	$(N_x, N_t) = (10, 10)$			$(N_x, N_t) = (15, 15)$			$(N_x, N_t) = (25, 25)$		
	Traj.	Energy	Mass	Traj.	Energy	Mass	Traj.	Energy	Mass
ENO	**7.09**±0.45 ($\times 10^{-2}$)	**6.24**±0.57 ($\times 10^{-5}$)	**9.98**±3.93 ($\times 10^{-1}$)	**2.71**±0.15 ($\times 10^{-3}$)	**1.64**±0.28 ($\times 10^{-5}$)	**2.76**±0.39 ($\times 10^{-1}$)	**4.51**±0.46 ($\times 10^{-4}$)	**7.06**±1.43 ($\times 10^{-7}$)	**2.21**±0.47 ($\times 10^{-2}$)
ENO (fixed)	1.54±0.54 ($\times 10^{-1}$)	1.46±1.13 ($\times 10^{-3}$)	1.38±0.67 ($\times 10^{1}$)	1.25±0.30 ($\times 10^{-2}$)	1.28±0.48 ($\times 10^{-4}$)	5.44±2.22 ($\times 10^{-1}$)	7.70±1.31 ($\times 10^{-4}$)	1.59±0.60 ($\times 10^{-6}$)	2.90±0.67 ($\times 10^{-2}$)
Vanilla NO	1.93±0.17 ($\times 10^{-1}$)	1.92±0.18 ($\times 10^{-3}$)	4.37±2.15 ($\times 10^{1}$)	3.94±0.67 ($\times 10^{-2}$)	1.78±0.40 ($\times 10^{-3}$)	3.00±1.62 ($\times 10^{0}$)	8.71±0.95 ($\times 10^{-4}$)	1.86±0.34 ($\times 10^{-6}$)	3.03±1.49 ($\times 10^{-2}$)
DeepONet	1.28±0.18 ($\times 10^{-1}$)	8.95±2.43 ($\times 10^{-4}$)	3.02±1.36 ($\times 10^{1}$)	1.85±0.34 ($\times 10^{-2}$)	4.25±2.79 ($\times 10^{-4}$)	1.18±0.66 ($\times 10^{0}$)	3.39±0.66 ($\times 10^{-3}$)	1.39±0.39 ($\times 10^{-5}$)	9.43±1.80 ($\times 10^{-2}$)
FNO	1.76±0.16 ($\times 10^{-1}$)	4.05±2.21 ($\times 10^{-4}$)	2.45±2.02 ($\times 10^{1}$)	8.84±0.55 ($\times 10^{-2}$)	7.03±3.05 ($\times 10^{-4}$)	5.04±1.56 ($\times 10^{0}$)	2.14±0.35 ($\times 10^{-3}$)	8.37±0.93 ($\times 10^{-6}$)	1.86±0.77 ($\times 10^{-1}$)
EnerReg	3.84±0.19 ($\times 10^{-2}$)	6.24±1.69 ($\times 10^{-5}$)	6.81±1.87 ($\times 10^{-1}$)	1.48±0.31 ($\times 10^{-3}$)	7.84±4.52 ($\times 10^{-6}$)	2.29±0.17 ($\times 10^{-1}$)	2.95±0.54 ($\times 10^{-4}$)	5.38±1.49 ($\times 10^{-7}$)	1.53±0.27 ($\times 10^{-2}$)

Table 4: Estimated hyperparameters. λ shows the median; $\mathcal{G}$ shows the mode.

(a) KdV equation

	$(N_\mathrm{x}, N_\mathrm{t}) = (10, 10)$		$(N_\mathrm{x}, N_\mathrm{t}) = (15, 15)$		$(N_\mathrm{x}, N_\mathrm{t}) = (25, 25)$	
	λ	$\mathcal{G}$	λ	$\mathcal{G}$	λ	$\mathcal{G}$
True	N/A	$\partial/\partial x$	N/A	$\partial/\partial x$	N/A	$\partial/\partial x$
ENO	10^{-4}	$\partial/\partial x$	10^{-4}	$\partial/\partial x$	10^{-4}	$\partial/\partial x$
ENO (fixed)	10^{-5}	$\partial/\partial x$	10^{-4}	$\partial/\partial x$	10^{-5}	$\partial/\partial x$

(b) Cahn–Hilliard

	$(N_\mathrm{x}, N_\mathrm{t}) = (10, 10)$		$(N_\mathrm{x}, N_\mathrm{t}) = (15, 15)$		$(N_\mathrm{x}, N_\mathrm{t}) = (25, 25)$	
	λ	$\mathcal{G}$	λ	$\mathcal{G}$	λ	$\mathcal{G}$
True	N/A	$\partial^2/\partial x^2$	N/A	$\partial^2/\partial x^2$	N/A	$\partial^2/\partial x^2$
ENO	10^{-7}	$\partial^2/\partial x^2$	10^{-6}	$\partial^2/\partial x^2$	10^{-7}	$\partial^2/\partial x^2$
ENO (fixed)	10^{-7}	$\partial/\partial x$	10^{-6}	-1	10^{-7}	$\partial^2/\partial x^2$

Table 5: Average computational time in seconds for training. In ENO and ENO (fixed), we show the time averaged over hyperparameter sets; it does not contain the time of validation procedures.

(a) KdV equation

	$(N_\mathrm{x}, N_\mathrm{t}) = (10, 10)$	$(N_\mathrm{x}, N_\mathrm{t}) = (15, 15)$	$(N_\mathrm{x}, N_\mathrm{t}) = (25, 25)$
ENO	6052.40	6111.40	6259.20
ENO (fixed)	4034.54	6122.38	6350.58
Vanilla NO	298.30	319.09	531.04
DeepONet	463.62	529.01	955.08
FNO	1974.10	3081.41	7321.45
EnerReg	2798.14	2846.54	3000.87

(b) Cahn–Hilliard

	$(N_\mathrm{x}, N_\mathrm{t}) = (10, 10)$	$(N_\mathrm{x}, N_\mathrm{t}) = (15, 15)$	$(N_\mathrm{x}, N_\mathrm{t}) = (25, 25)$
ENO	6017.09	6120.17	6299.84
ENO (fixed)	4007.82	6109.25	6350.79
Vanilla NO	284.64	314.21	530.20
DeepONet	459.35	515.52	957.95
FNO	2002.27	3048.83	7331.25
EnerReg	7327.72	7354.38	7539.02

Table 6: MSEs between the actual energy density F^* and its estimate F_ϕ.

Epoch	100	1000	2000	5000	10000
	3.69×10^2	3.88×10^{-1}	2.05×10^{-1}	1.25×10^{-1}	1.18×10^{-1}

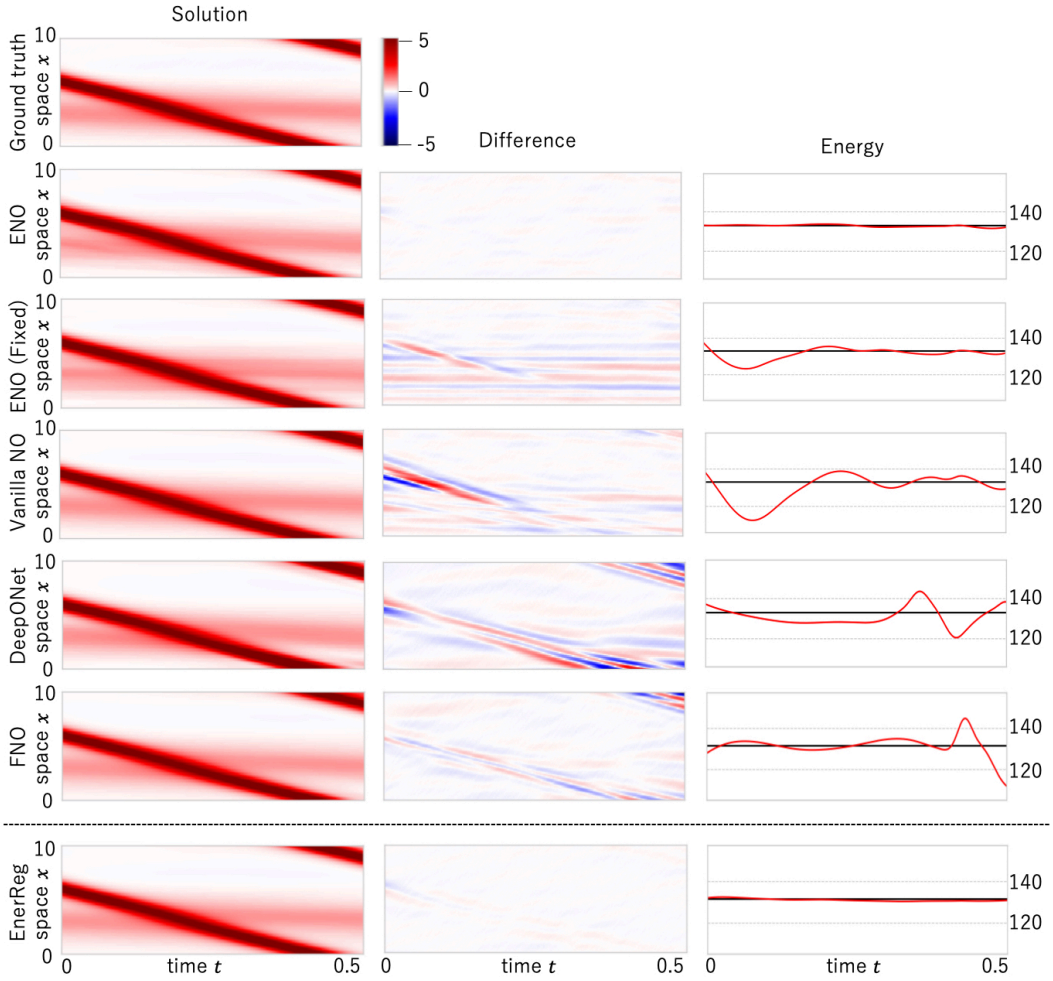


Figure 5: Results for KdV equation ($(N_x, N_t) = (10, 10)$). First column is a visualization of predicted solutions. Second column is a difference between ground truth and its prediction, where the difference values were multiplied by 4. Third column provides a comparison between the true energy (black line) and its estimate (red line).

Table 7: Average MSEs when using three different resolutions of training data.

(a) Allen-Cahn equation: Multiplied by 10^3 for TRAJ. and by 10^5 for ENERGY.

	$(N_x, N_t) = (10, 10)$			$(N_x, N_t) = (15, 15)$			$(N_x, N_t) = (25, 25)$		
	TRAJ.	ENERGY	MASS	TRAJ.	ENERGY	MASS	TRAJ.	ENERGY	MASS
ENO	**6.47**	**7.03**	-	**6.13**	**6.23**	-	**5.82**	**5.92**	-
VANILLA NO	10.82	11.09	-	6.88	8.31	-	6.61	6.79	-
DEEPONET	9.72	11.86	-	6.33	8.01	-	6.99	6.62	-
FNO	8.83	10.63	-	6.78	7.23	-	5.90	6.12	-

(b) 2D Cahn-Hilliard equation: Multiplied by 10^2 for TRAJ. and by 10^2 for ENERGY.

	$(N_x, N_t) = (10^2, 10)$			$(N_x, N_t) = (15^2, 15)$			$(N_x, N_t) = (25^2, 25)$		
	TRAJ.	ENERGY	MASS	TRAJ.	ENERGY	MASS	TRAJ.	ENERGY	MASS
ENO	**23.63**	**28.91**	**48.33**	**1.42**	**2.76**	**1.56**	**0.51**	**1.81**	**1.04**
VANILLA NO	56.17	33.81	59.26	3.91	7.97	7.79	1.01	3.58	3.76
DEEPONET	43.44	32.24	56.49	4.10	5.46	7.99	0.86	2.23	3.00
FNO	57.34	34.76	56.84	3.02	4.12	2.53	0.71	2.22	3.01

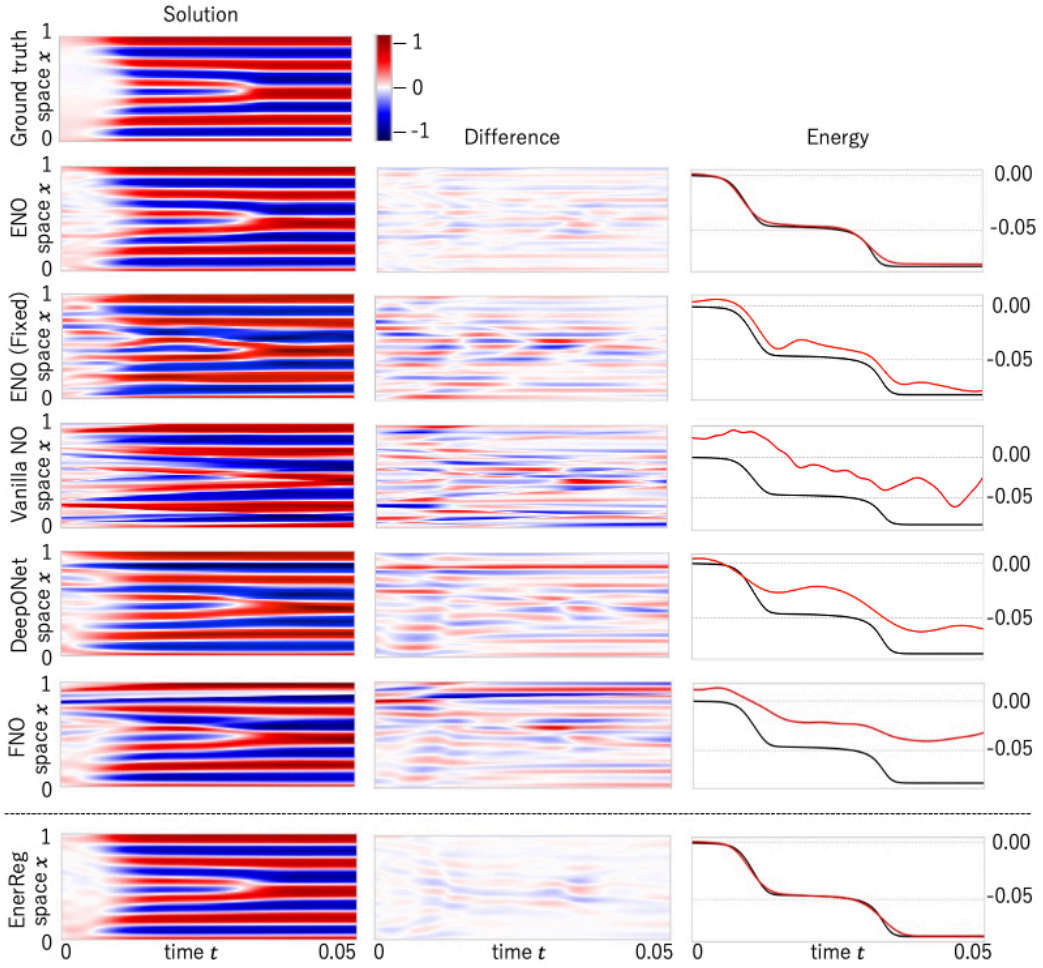


Figure 6: Results for Cahn-Hilliard equation ($(N_{\mathrm{x}}, N_{\mathrm{t}}) = (15, 15)$). First column is a visualization of predicted solutions. Second column is a difference between ground truth and its prediction. Third column provides a comparison between the true energy (black line) and its estimate (red line).