Symmetry Constraints Enhance Long-term Stability and Accuracy in Unsupervised Learning of Geophysical Fluid Flows

Yunfei Huang[1, 2], **David S. Greenberg**[1, 2]

[1]Institute of Coastal Systems, Helmholtz-Zentrum Hereon, Geesthacht, Germany
[2]Helmholtz AI

Key Points:

- We construct group-equivariant convolutional networks to respect transformation symmetries of fluid dynamical PDEs as hard constraints.
- We apply these networks to unsupervised learning of a semi-implicit shallow water scheme with a physics-derived loss function.
- Symmetry constraints improve accuracy, stability and generalization performance as well as representation of mass, momentum and energy.

Corresponding author: David S. Greenberg, david.greenberg@hereon.de

Abstract

Fluid dynamical systems are well described by discretized partial differential equations, but computational costs limit accuracy, duration and/or resolution in numerical integrations. Recent studies showed that deep neural networks trained on simulations or PDE-derived losses can improve cost-accuracy tradeoffs, but purely data-centric approaches discard physical and mathematical insights and require computationally costly training data. Here we draw on advances in geometric deep learning to design solver networks that respect PDE symmetries as hard constraints. We construct equivariant convolutional layers for mixed scalar-vector input fields in order to capture the symmetries inherent to specific PDEs. We demonstrate our approach on a challenging 1D semi-implicit shallow water scheme with closed boundaries, applying unsupervised learning with a physics-derived loss function. We report strong improvements in accuracy and stability of equivariant solvers compared to standard convolutional networks with the same architectures and parameter counts. Solver equivariance also improves performance on new initial conditions not encountered during training, and suppresses error accumulation in global momentum and energy. Strikingly, these benefits do not reduce loss values during training, but appear later during ML-assisted rollouts over time steps. Our results suggest that symmetry constraints could improve deep learning performance across a wide range of fluid dynamical tasks, learning algorithms and neural architectures.

Plain Language Summary

The complex fluid dynamics of the atmosphere and oceans can be described and simulated using partial differential equations (PDEs). However, accurately modeling these systems is too computationally expensive in many important scenarios. Recent studies have proposed using deep learning to quickly and accurately solve these PDEs, but this approach discards many of our physical and mathematical insights and can result in poor accuracy and stability as longer time scales are simulated. To address these limitations, we apply symmetry constraints that describe how PDE solutions should be transformed when initial and boundary conditions are shifted, flipped or rotated. These symmetries are imposed as hard constraints, meaning they are fully satisfied at every stage of the learning process. To achieve this, we design new computational modules to accept mixtures of scalar and vector fields as input, and mathematically prove that they possess the correct symmetries. We test this approach on the shallow water equations, revealing striking improvements in accuracy and stability compared to models with similar numbers of free parameters but no symmetry constraints. Symmetry-preserving networks also better represent global momentum and energy, and perform better on new scenarios not encountered during training.

Keywords: Fluid Dynamics, Equivariance, Convolutional Networks, Physics-informed Deep Learning, Deep PDE Solvers, Hybrid Models, Shallow Water Equations

1 Introduction

Partial differential equations (PDEs) are essential for understanding and simulating complex fluid dynamics. Examples include convection-diffusion (Egger & Schöberl, 2010), Euler (Euler, 1757) and Navier–Stokes equations (NS) (Temam, 2001; Constantin & Foias, 2020). The shallow water equations (SWEs) (De St Venant, 1871), derived by depth integration of NS, are mathematically simpler but widely employed as test cases to evaluate solution techniques for ocean, weather and climate applications (Dellar & Salmon, 2005; Bunya et al., 2009; Kärnä et al., 2011; Zängl et al., 2015; Klöwer et al., 2020; Korn et al., 2022).

PDEs describing geophysical fluid flows require numerical methods, for example, finite difference (Sadourny, 1975; Casulli, 1990), finite element (Zienkiewicz & Ortiz, 1995;

Bassi & Rebay, 1997), finite volume (Eymard et al., 2000), boundary element (Grilli et al., 1998), and spectral element methods (Taylor et al., 1997). While small spatial domains admit direct numerical simulation, geophysical applications require coarse grids with Reynolds-averaging (Reynolds, 1895) or large eddy simulation (LES) (Smagorinsky, 1963; Deardorff, 1970) to approximate unresolved scales. Explicit time stepping simplifies computations but requires small steps for stability, while (semi)implicit schemes take larger time steps but must iteratively solve a system of equations (Saad, 2003; Polyak, 1969; Saad & Schultz, 1986; Van der Vorst, 1992). However, these classical approaches incur heavy computational costs at high spatial and temporal resolutions.

Recent machine learning (ML) approaches aim to transcend these cost-accuracy tradeoffs by training a model to accurately and efficiently solve PDEs on modern computational hardware. *Supervised learning* uses simulations from a classical solver to train an ML model that uses larger space and time steps or skips the iterative computations of an implicit scheme. This approach has shown success in accelerating PDE solutions while maintaining accuracy, obeying conservation laws and preserving high frequency features (Kim et al., 2019; Wang, Kashinath, et al., 2020; Z. Li et al., 2020a; Wen et al., 2022; Gupta et al., 2021; Y. Li et al., 2022; Kohl et al., 2023), and has also been applied to mesh-free particle-based solvers (Z. Li et al., 2020b).

Unsupervised learning trains the model to satisfy the PDE without requiring training data. It is most effective for implicit schemes, since solving their equations iteratively is complex and costly but verifying a solution is simple and fast. Unsupervised learning avoids overfitting by training on its own outputs, but cannot avoid discretization errors for large the space or time steps. It has been used to solve several fluid dynamical PDEs (Wandel et al., 2020, 2021; Raissi et al., 2019; Bar & Sochen, 2019; Cai et al., 2020; Stanziola et al., 2021; Michelis & Katzschmann, 2022; Zhu et al., 2019; Sun et al., 2020; Geneva & Zabaras, 2020).

Hybrid models replace only part of a classical PDE solver with an ML model, leaving other components unchanged. Early work applied this to computer graphics (Grzeszczuk et al., 1998),while a later study demonstrated an approach combining a fluid solver with ML techniques to approximate NS in a Lagrangian framework using regression forests (Ladickỳ et al., 2015). More recently, (Um et al., 2020) used supervised learning to compute an additive correction to low-resolution incompressible NS, so that its evolution mimics a high-resolution model coarsened at each time step. An LSTM-based hybrid approach (Wiewel et al., 2019) with significant practical speed-ups has been presented for predicting pressure changes for incompressible flow, while (Tompson et al., 2017) proposed an ML-based approach for replacing the linear projection in the Eulerian fluid implicit simulation, (Wang, Kashinath, et al., 2020) combined two well-established turbulent flow simulation techniques with deep learning and (Obiols-Sales et al., 2020) developed an accelerated integrative ML solverto aid convergence of Reynolds Averaged Navier-Stokes simulations. Overall, hybrid methods allow us to effectively incorporate physical knowledge while simplifying the learning task, and can improve accuracy and generalization capabilities.

Major challenges remain for ML-based PDE solvers: long-term stability and accuracy are not guaranteed even for low loss values on training and testing data (Um et al., 2020; Nonnenmacher & Greenberg, 2021b; List et al., 2022; Wu et al., 2022), and generalization to new scenarios remains problematic (Thuerey et al., 2020; Kashinath et al., 2021; Lye et al., 2020; Fresca & Manzoni, 2022). A partial explanation is that training neural networks means choosing from a large, high-dimensional family of functions, many of which are physically or mathematically implausible. Narrowing the search by constraining the learned function has shown great promise: for example, conservation laws improve process representations in climate, weather, and ocean models (Yuval et al., 2021; Yuval & O'Gorman, 2020), while symmetry constraints aid image classification (T. Cohen & Welling, 2016) and segmentation (Tajbakhsh et al., 2020; Han et al., 2021). However, the potential benefit for fluid dynamics remains mostly unclear.

In this work we construct hybrid PDE solvers using equivariant neural networks that obey PDE symmetry constraints. We draw on previous work in geometric deep learning (T. Cohen & Welling, 2016; Gerken et al., 2023), but extend group equivariant convolutions to handle mixed scalar/vector inputs with the correct, PDE-specific transformation rules. We demonstrate the benefit of equivariant solver networks using an unsupervised learning task, in which the network is trained to integrate a semi-implicit scheme for 1-D shallow water equations. These equations exhibit challenging stiff dynamics due to closed boundaries and reflecting waves. Our experiments show significant improvements in long term accuracy and stability compared to standard CNNs, despite similar loss values during training. We also observe that symmetry constraints improve performance on initial conditions not encountered during training, as well as representations of global mass, momentum and energy.

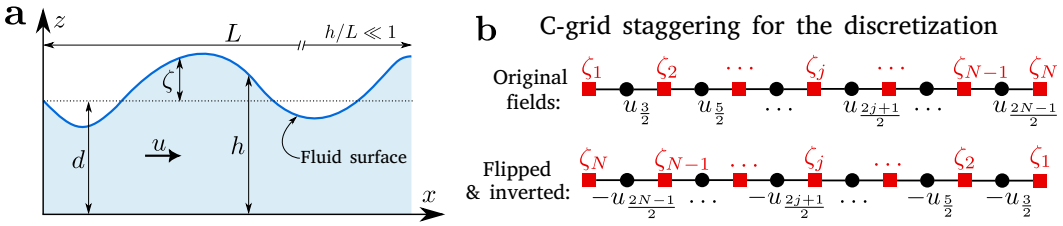


Figure 1. Schematic representation of a one-dimensional shallow water problem with C-grid staggering of discretized fields. (a) Shallow water system with domain length L. d and h are un-disturbed- and disturbed- water depth, ζ is fluid surface elevation and $h = \zeta + d$. u is the velocity along the space coordinate x. (b) Staggered grid for elevation and velocity. Top: fluid surface elevation ζ_j is represented at red squares while velocity $u_{j+1/2}$ is represented at black circles. Bottom: A flipped fluid surface elevation ζ_i, as well as a flipped-and-inverted velocity. These transformed fields solve the SWE with transformed initial and boundary conditions.

2 Numerical Integration of Fluid Dynamics

In this section we establish concepts and notation for PDE integration with classical numerical techniques, allowing us to describe our task and approach in the following section. We consider a general governing partial differential equation for fluid dynamics:

$$\frac{\partial q(t,x)}{\partial t} = \mathcal{F}[q] = f(t,x,q(t,x),\frac{dq}{dx},\frac{d^2q}{dx^2},\ldots), \quad x \in \Omega \tag{1}$$

$$q(x,t) = q_\Omega(x), \quad x \in \partial\Omega \tag{2}$$

$$q(x,0) = q^0(x) \tag{3}$$

x and t are space and time coordinates and $q(t,x)$ is the vector of modeled variable fields at one place and time, such as velocity and pressure in NS or velocity and height in SWE. $\mathcal{F}$ is a nonlinear operator computing time derivatives as nonlinear functions of functions of the fields and their spatial derivatives. The Dirichlet boundary conditions (BCs) $q_\Omega(x)$ on the boundary $\partial\Omega$ and initial conditions (ICs) $q^0(x)$ are given while $q(x,t)$ is the unknown quantity for which we solve the PDE. Eqs. 1-3 are a common form for governing a fluid flow, though other types of BCs and constraints (such as incompressibility) can also be used.

2.1 Spatial Discretization

We solve our PDEs with the classical finite difference methods with uniform time step Δt and all prognostic variables defined on a regular grid with space step Δx. For 1-D fields, and denoting the k-th variable field in q by z, we use $z_j^n = q(j\Delta x, n\Delta t)_k$ to denote the value of z at the j-th location on the n-th time step. We employ staggered representations of scalar fields and velocities using Arakawa C-grids (Akio & Vivian, 1977), and the notation $z_{1/2}^n, z_{3/2}^n, \ldots$ for shifted variables in Fig. 1b.

2.2 Time Stepping

Given the discretized variable fields q^n at time $t = n\Delta t$, a time stepping scheme is used to compute the next fields q^{n+1}. Here we consider the broad range of schemes in which each occurence of q in the definition of $\mathcal{F}$ is replaced by a weighted average of q^n and q^{n+1}, and the weighting may be different for each field and each term of the PDE. Thus when q^n is used in every case we have an explicit method, while using $(q^n+q^{n+1})/2$ in every case gives a Crank-Nicholson method (Crank & Nicolson, 1947). Denoting the discretized version of $\mathcal{F}$ by $\widetilde{\mathcal{F}}$, the scheme can be written as a system of equations

$$q_j^{n+1} = q_j^n + \Delta t \widetilde{\mathcal{F}}[q^n, q^{n+1}] \tag{4}$$

Because q^{n+1} appears on both sides of the equation, we must solve the equations to obtain it, for example by using iterative methods. Calculating $\widetilde{\mathcal{F}}$ for the discretized variable fields requires discretized versions of all spatial derivatives, which in general must be designed and tested for each PDE to ensure accuracy and stability. This class of time stepping schemes is widely used for fluid dynamical PDEs: examples include incompressible Navier Stokes (Turek, 1996; Fischer et al., 2005; Forti & Dedè, 2015) and certain shallow water solvers with land-water boundaries (Backhaus, 1983) (see below).

3 Unsupervised Learning of PDE Integration

3.1 Problem Statement

We aim to replace an expensive semi-implicit time scheme (Eq. 4) with a faster, neural-network based solver. Critically, we do not assume that we have access to simulation data for training purposes, but must train the network using only our knowledge of the PDE, spatial discretization and time stepping scheme.

Concretely, we wish to train (that is, optimize) the parameters ϕ of a flexible function approximator $\widehat{S}_\phi$, such that $\widehat{S}_\phi(q^n) \approx S(q^n) = q^{n+1}$. Here S denotes a single step of time integration using a classical numerical solver that acts as our target reference solution. We aim to achieve a close approximation between $\widehat{S}_\phi$ and S on PDE integrations with initial distributions drawn from a specified probability distribution $\Pi(q)$:

$$q^0 \sim \Pi(q) \tag{5}$$
$$q^n = S^{(n)}(q^0) \tag{6}$$
$$\widehat{q}^n = \widehat{S}_\phi^{(n)}(q^0) \tag{7}$$
$$\widehat{q}^n \approx q^n \tag{8}$$

Without access to simulation data, we cannot carry out supervised training of $\widehat{S}_\phi$ using input-output pairs (q^n, q^{n+1}). The motivation behind this problem formulation without access to training data is that it avoids expensive simulations, and does not require us to commit to a fixed set of simulated system states at the onset of training.

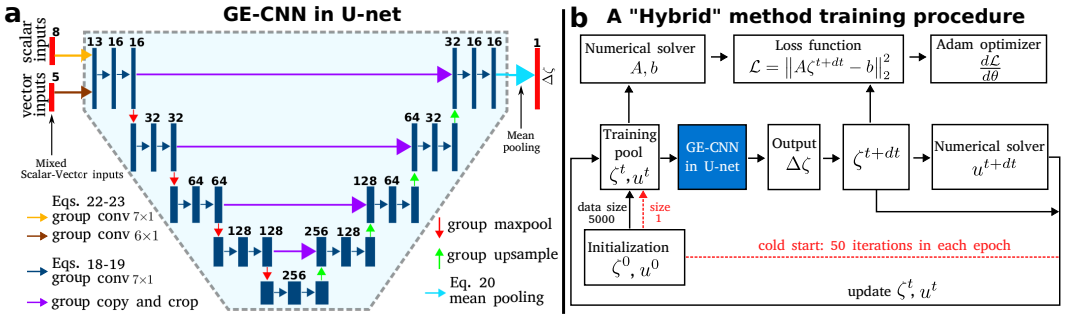


Figure 2. Schematic representation of equivariant U-net architecture and 'hybrid' training procedure for a deep SWE solver. (a) Equivariant U-Net architecture with an example of channels. Grid staggering leads to input channels with different sizes, so we use equivariant input layers with different kernel sizes to obtain a uniform size across channels in the first hidden layer. All activations except for inputs and outputs are defined as real-valued functions on the infinite discrete group H containing reflections and translations (light blue area). Network outputs provide a fluid surface update $\Delta\zeta$. (b) A 'hybrid' training loop adapted from (Wandel et al., 2020). A pool of system states is first filled with the initial conditions ζ^0, u^0. For a randomly selected batch of system states, the U-net then generates ζ^{t+dt} at the next time step while velocity is calculated as in a numerical solver. The unsupervised loss function summed over the batch, its gradients are used to update network parameters and the new states overwrite their own inputs in the pool.

3.2 Physics-derived Loss Function

To train $g_\phi \approx S$ without simulation data, we construct a physics-derived loss (Zhu et al., 2019; Wandel et al., 2020) that is zero if and only if the discretized PDE is precisely solved:

$$\mathcal{L}_{\text{PDE}}(q,\phi) = \left\| q + \Delta t \widetilde{\mathcal{F}}[q, g_\phi(q)] - \widehat{S}_\phi(q) \right\|_2^2 + \mathbb{1}_\Omega \cdot \left\| \widehat{S}_\phi(q) - q_\Omega \right\|_2^2 \tag{9}$$

The first loss term measures deviations from the prescribed PDE, and is clearly zero when $\widehat{S}_\phi = S$. The second term measures violation of the Dirichlet BCs, with $\mathbb{1}_\Omega$ an indicator function for the boundary. If $\widehat{S}_\phi$ is constructed to satisfy the BCs for any ϕ the second term can be dropped.

3.3 Training Algorithm

In principle, we could minimize $\mathcal{L}_{\text{PDE}}$ using any input fields q, but to obtain optimal results when ICs are drawn from $\Pi(q)$, we should train on fields likely to occur when time-integrating from those ICs. We therefore train $\widehat{S}_\phi$ on fields it has itself integrated, following the strategy of (Wandel et al., 2020) in Fig. 2b.

We first initialize a pool of 5000 simulations with ICs drawn randomly from $\Pi(q)$. For each gradient step, a batch of simulations is randomly selected from the pool, and stepped forward using $\widehat{S}_\phi$. The fields at the old and new time steps for this batch are used to compute $\mathcal{L}_{\text{PDE}}$, and the resulting gradients are used to update ϕ. The updated simulations are then stored in the simulation pool, where they overwrite their own previous values. After every 50 gradient steps, a randomly selected simulation is overwritten with a new initialization from $\Pi(q)$. We use a default batch size of $b = 100$, 60 epochs, and the Adam optimizer (Kingma & Ba, 2014) with an initial learning rate of 0.001.

3.4 Hybrid Solvers

For a fully data-driven architecture, the time stepping function $\widehat{S}_\phi$ can be fully specified by a deep neural network or other general function approximator. However, several studies have shown that combining deep learning and numerical physics in a single model can provide better results than either approach alone (Tompson et al., 2017; Long et al., 2019; Obiols-Sales et al., 2020; Kochkov et al., 2021; Wandel et al., 2021). A particular focus of this hybrid approach has been semi-implicit numerical schemes that require a system of equations to be solved at each time step. Classical numerical solvers for these schemes often use variable substitution to reduce the number of equations and unknowns: examples include elimination of velocity when solving for fluid height in the SWEs (Backhaus, 1983) and the pressure projection step for incompressible NS. Hybrid approaches therefore train a deep neural network to efficiently solve the reduced set of equations, after which the remaining output variables are calculated using formulae from the original numerical solver. We describe this approach in detail for a SWE scheme in Fig. 2b.

This hybrid approach can offer several benefits compared to a learning a fully data-driven time stepping scheme. By replacing only expensive computations, it retains some inductive biases of the original scheme, and ensures that the full set of updated fields are accurate when the learned computations are correct. Fewer input and output channels for the trained model also reduce parameter counts and improve optimization and data efficiencies.

3.5 Neural Architectures

To learn (hybrid) time stepping for spatially structured fields, we employ the U-net architecture (Ronneberger et al., 2015). The U-net is a convolutional encoder-decoder network. In the encoder, the number of channels increases with depth while spatial resolution decreases, while the decoder enacts the opposite transformations while receiving skip connections from the encoder at each resolution (Fig. 2a). At each resolution, the encoder and decoder employ two convolution layers with kernel size 7. For input fields with C-grid staggering, different kernel sizes are used to achieve a uniform spatial extent for output fields in the encoder's first convolution layer. The final outputs of the U-net are interpreted as updates Δq, and added to the corresponding input fields q^n to produce the time-step fields q^{n+1}. The default resolution of input is 200 for mass grids and 199 for velocity grids. The number of parameters for the network is changed by using a multiplier for the channel counts of all hidden layers (Fig. 2 shows a multiplier of 16).

4 Geometric Constraints

Many PDEs have symmetries: certain spatial transformations of initial and boundary conditions lead to a corresponding transformation of the system state at future time points. We aim to improve ML-based PDE solvers by endowing them with these properties as a hard constraint built into the neural architecture. This effectively narrows the class of functions through which we are searching for an effective and efficient solver, by filtering out functions inconsistent with the symmetry.

4.1 Equivariance

Suppose we have a finite group of symmetries $g \in G$ acting on a set of spatially extended fields q by transformations $q \to \mathcal{T}_g(q)$, with $\mathcal{T}_{g_1 g_2} = \mathcal{T}_{g_1} \circ \mathcal{T}_{g_2}$. A function $\Psi(q)$ is equivariant when transforming its inputs is equivalent to transforming its outputs. Concretely, for each $g \in G$ the transformations $\mathcal{T}_g, \mathcal{T}'_g$ act on Ψ's inputs and out-

puts respectively, and

$$\forall g, q : \Psi(\mathcal{T}_g q) = \mathcal{T}'_g \Psi(q) \tag{10}$$

For example, let Ψ denote time integration of the 2D heat equation $\frac{\partial q}{\partial t} = \kappa \left(\frac{\partial^2 q}{\partial x_1^2} + \frac{\partial^2 q}{\partial x_2^2} \right)$. This Ψ is equivariant to rotations, reflections, and translations of the heat field q. In this case both the input and output transformations are simply the same point-to-point mappings of the scalar heat fields, but for more complex PDEs involving vector fields the transformations can be more involved (see below).

4.2 Equivariant Convolutions

We now describe the construction of convolutional networks with equivariance as a hard constraint, meaning that the $\widehat{S}_\phi$ is equivariant for any ϕ. Throughout this section we follow (T. Cohen & Welling, 2016), but simplify notation by describing a single input and output channel, both of which are n-D fields of the same size. We denote by $\mathbb{Y} \subset \mathbb{Z}^n$ the regular grid of integer valued coordinates on which the input and output channels are defined.

A standard convolutional layer applies an n-D convolutional filter W to a spatially extended scalar input field q to produce a scalar output field $q \star W$:

$$[q \star W](x) = \sum_{y \in \mathbb{Y}} q(y) W(y - x) + b \tag{11}$$

where $W(y) = 0$ for y outside the spatial extent of the filter. q, W, and $q \star W$ are simply real-valued functions on $\mathbb{Y}$, while b is a scalar. Standard convolutions are equivariant with respect to translations, but not other symmetries.

In contrast, equivariant convolutional layers produce outputs that are real-valued functions of an extended discrete group H generated by the symmetry group G of interest as well as translations in $\mathbb{R}^n$. The first such layer takes standard scalar fields as input:

$$[q \star W](h) = \sum_{y \in \mathbb{Y}} q(y) W(h^{-1} y) + b \tag{12}$$

Subsequent layers use functions on H as both inputs and outputs:

$$[\gamma \star W](h) = \sum_{h' \in \mathbb{H}} \gamma(h') W(h^{-1} h') + b \tag{13}$$

γ, W, and $\gamma \star W$ are real-valued functions of H. As shown in (T. Cohen & Welling, 2016), Eqs. 12-13 satsify equivariance (Eq. 10). The input transformation in Eq. 12 is simply G's action on $\mathbb{Z}^n$ described by G, while other transformations act on real-valued functions of H. For any such function $\alpha(h)$ we have

$$[T_h \alpha](h') = \alpha(h^{-1} h') \tag{14}$$

We visualize functions on H as collections of maps over $\mathbb{Y}$, with one map (i.e. real-valued function on $\mathbb{Y}$) for each $g \in G$ (Fig. 3). Since the nonzero regions of q, γ and W are bounded, the outputs' nonzero regions are as well. In Eq. 12, W is defined on a patch of $\mathbb{Z}^n$, but in Eq. 13 W is a function on H. To include multiple input output and channels, we simply sum over inputs for each output in Eqs. 11-13, and note that the arrays storing W acquire two additional dimensions. The bias b is then indexed by the output channel, but not by location or group element.

To build an equivariant convolutional network, a sequence of equivariant convolutional layers is interspersed with pointwise nonlinearities. To obtain an equivariant final output on $\mathbb{Y}$ instead of H, a pooling operation (e.g. a mean or maximum) operates

'along the G-axis' of the G-indexed collection of maps on $\mathbb{Y}$. Concretely, we take the mean or maximum over all elements of H that share the same translational component.

For a symmetry group of size $|G|$, a standard convolution with c input and output channels has as many parameters as an equivariant layer with $c/\sqrt{|G|}$ channels. When $\mathbb{Y}$ is a D-dimensional grid with N points per axis, the computational complexity of the forward and backward passes is $O(N^D K^D c^2)$ in both cases, since the equivariant network has $|G|$ times fewer input-output channel pairs but each input channel must be convolved with a transformed slice of the filter bank $|G|$ times.

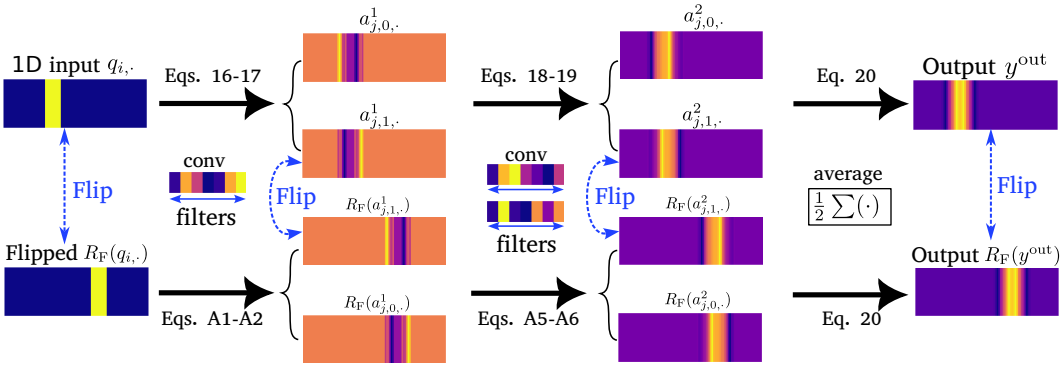


Figure 3. Reflection-equivariant 1D convolutional neural network (GE-CNNs). For illustrative purposes, a network with only one channel in each layers is shown. Blue arrows indicate a pair of fields that are reflections of each other, while black arrows indicate equivariant convolution layers. The 1D input and its flipped duplicate are shown on the left side. Activations in each layer are computed by applying both a standard and flipped version of the convolutional filter to the previous layer. The final output is obtained by averaging over reflected and non-reflected version of each channel pooling on the geometrical two features.

4.2.1 Reflection-equivariant 1D Convolutions

Having introduced equivariant convolutions for any finite symmetry group G acting on a discrete grid $\mathbb{Y}$, we next focus on the concrete example of 1-D reflective symmetries. This two-element group is the only nontrivial symmetry group and the main focus of this paper. Here $\mathbb{Y} = \{-N, \ldots, N\}$, G contains the identity and a reflection, and H consists of either element of G followed by any translation. In this case, a convolutional time stepping network $\widehat{S}_\phi : q^n \to \hat{q}^{n+1}$ is equivariant if $R_F(\widehat{S}_\phi(q)) = \widehat{S}_\phi(R_F(q))$, for any input field(s) q and where the 'mirroring' operator R_F reflects the fields on the spatial axis, i.e. $[R_F(q)](x) = q(-x)$.

A standard 1D convolutional layer in Eq. 11 with c_{in} inputs q and c_{out} outputs a is defined as:

$$a_{j,\cdot} = \sum_{i=1}^{c_{in}} W_{j,i,\cdot} \star q_{i,\cdot} + b_j \tag{15}$$

where the $\cdot$ symbol denotes all values along a given axis. W is an $c_{in} \times c_0 \times K$ array for filter size K, while b is a c_{out}-element vector.

In 1D reflection-equivariant networks, the first layer is a special case with c_{in} input channels q defined on $\mathbb{Y}$ and c_1 outputs a^1 defined on H. Eq. 12 thus becomes:

$$a^1_{j,0,\cdot} = \sum_{i=1}^{c_{\text{in}}} W^1_{j,i,\cdot} \star q_{i,\cdot} \qquad + b^1_j \tag{16}$$

$$a^1_{j,1,\cdot} = \sum_{i=1}^{c_{\text{in}}} R_{\text{F}} \left(W^1_{j,i,\cdot} \right) \star q_{i,\cdot} + b^1_j \tag{17}$$

While q, W^1, b^1 have the same size here as in standard convolutional layers, a^1 gains a third dimension that indexes the elements of G.

For subsequent layers, both the $c_{\ell-1}$ input channels $a^{\ell-1}$ and c_ℓ outputs a^ℓ are defined on H and are stored in 3-D arrays. Eq. 13 becomes:

$$a^\ell_{j,0,\cdot} = \sum_{i=1}^{c_{\ell-1}} W_{j,i,0,\cdot} \star a^{\ell-1}_{i,0,\cdot} \qquad + W_{j,i,1,\cdot} \star a^{\ell-1}_{i,1,\cdot} \qquad + b^\ell_j \tag{18}$$

$$a^\ell_{j,1,\cdot} = \sum_{i=1}^{c_{\ell-1}} R_{\text{F}} \left(W_{j,i,1,\cdot} \right) \star a^{\ell-1}_{i,0,\cdot} + R_{\text{F}} \left(W_{j,i,0,\cdot} \right) \star a^{\ell-1}_{i,1,\cdot} + b^\ell_j \tag{19}$$

The filter bank W now has four dimensions, the third of which indexes G. When computing results at the second index along the second, G-indexing dimension of each output channel (Eq. 19), the filters are flipped on the spatial axis and permuted on the G axis (Fig. 3). While the equivariance of these layers follows as a special case of the results in (T. Cohen & Welling, 2016), we include simple proofs for the case of 1D reflections in the Appendices A1-A2.

Finally, to produce a network output that is defined on a simple 1D grid (not as a function on H), we use a mean pooling operation over the symmetry dimension

$$y^{\text{out}}_{j,\cdot} = \left(a^L_{j,0,\cdot} + a^L_{j,1,\cdot} \right) / 2 \tag{20}$$

which also obviously has the desired equivariance property. Thus, by chaining together these input, internal, and output layers, our entire network Ψ is reflection equivariant.

4.2.2 Extension to Mixed Scalar-Vector Inputs

Unfortunately, the equivariance defined for convolutions above does not match the reflection symmetries of many PDEs, since it fails to account for differences in how vector and scalar fields are affected by rotation and reflection. For a scalar field, the value of the transformed field (e.g. heat) is simply the value of the original field at a different point. But for vector fields (e.g. velocity) both a location change and a reflection/rotation of the vector at the corrected location are required. Simply transforming each component of the velocity field in a PDE solution as a separate scalar would yield a new field that does not solve the PDE.

For reflections of 1-D vector field u, the necessary transformation is

$$[R_{\text{F}} u](x) = -u[-x] \tag{21}$$

To implement the proper transformation when q contains both scalar fields ζ and vector fields u, we define the following input layer (compare to Eq. 19):

$$a^1_{j,0,\cdot} = \sum_{i=1}^{c^\zeta_{in}} W^\zeta_{j,i,\cdot} \star \zeta_{i,\cdot} \qquad + \sum_{i=1}^{c^u_{in}} W^u_{j,i,\cdot} \star u_{i,\cdot} \qquad + b^\ell_j \tag{22}$$

$$a^1_{j,1,\cdot} = \sum_{i=1}^{c^\zeta_{in}} R_{\text{F}}(W^\zeta_{j,i,\cdot}) \star \zeta_{i,\cdot} + \sum_{i=1}^{c^u_{in}} -R_{\text{F}}(W^u_{j,i,\cdot}) \star u_{i,\cdot} + b^\ell_j \tag{23}$$

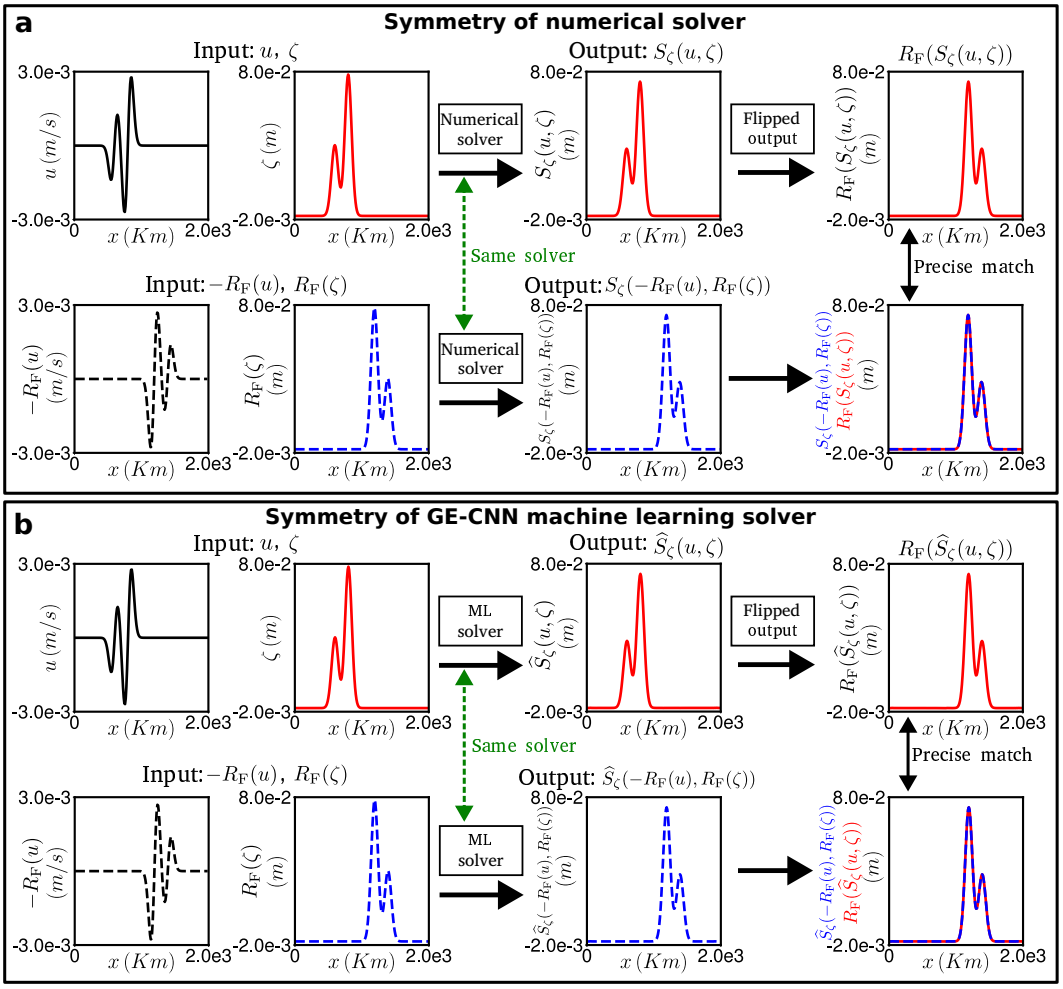


Figure 4. Empirical verification of identical PDE and GE-CNN symmetries for the shallow water equations. (a) ICs, ζ, and u for the numerical solver (top left). A flipped version is shown with inversion of u (bottom left). One step of numerical integration produces a new ζ (top, third column), from which we compute a flipped version. Integration from the flipped/inverted ICs produces a different ζ (bottom, third column), which precisely matches the flipped output from the non-flipped ICs (bottom right). (b) As in 'a,' but for a trained U-net based solver. Note however that the equivariance of the network is not learned, but exists as a hard constraint throughout the training process.

Since the output of this layer is a real-valued function on H, subsequent equivariant layers can be used without modification in Eq. 13. Defining an equivariant output layer to produce vector fields is straightforward, but because we construct hybrid solvers (see below) scalar outputs are sufficient for our purposes.

We prove the equivariance of our input layer in A3. For an empirical confirmation of this, Fig. 4 shows the equivariance of a classical PDE solver S and a trained equivariant convolutional network $\widehat{S}_\phi$ for the shallow water equations, which govern scalar height and vector velocity fields.

4.3 One-dimensional shallow water equations

We evaluated our learning strategies using the 1D SWEs, composed of momentum and continuity equations:

$$\frac{\partial u}{\partial t} = -C_D \frac{1}{h} u|u| - g\frac{\partial \zeta}{\partial x} \tag{24}$$

$$\frac{\partial \zeta}{\partial t} = -\frac{\partial (hu)}{\partial x} \tag{25}$$

with spatial coordinate $x \in [0, L]$, time t, velocity u, surface disturbance ζ, total depth $h = d+\zeta$, bottom drag C_D, and gravitational acceleration g. SWEs are commonly used to describe large-scale flows in coasts, oceans, estuaries, and rivers, based on the assumption that fluid depth is well below the length scale of horizontal motion, as illustrated in Fig. 1a. Our dry BCs, common in riverine and coastal models (Backhaus, 1983; Casulli & Walters, 2000; Zijlema & Stelling, 2008), mean that no fluid enters or escapes:

$$u(x = 0) = u(x = L) = 0 \tag{26}$$

$$\zeta(x = 0) = \zeta(x = L) = 0 \tag{27}$$

By default we use 'Gaussian bell' ICs:

$$u(x, 0) = 0 \tag{28}$$

$$\zeta(x, 0) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-\mu)^2/\sigma^2} \tag{29}$$

with μ and σ uniformly distributed on $[100\,Km, 1900\,Km]$ and $[10\,Km, 100\,Km]$ respectively.

Substituting $\zeta \leftarrow R_{\mathrm{F}}(\zeta), u \leftarrow -R_{\mathrm{F}}(u)$ into Eqs. 24-27 demonstrates reflection equivariance of the SWEs, which we confirmed empirically in Fig. 4a.

Since closed, wave-reflecting boundaries tend to require minuscule time steps for explicit schemes, we used a semi-implicit scheme (Backhaus, 1983) to generate reference simulations and to construct loss functions (Eq. 9) for unsupervised learning. This finite difference method stores velocities and surface elevations on staggered grids (details in the Appendix B, simulation parameters in Table 1). Its computational cost is dominated by solving a tridiagonal linear system

$$A\zeta^{n+1} = b \tag{30}$$

Where A, b are functions of ζ^n and u^n. The relative costs of calculating the coefficients of A and b or computing u^{n+1} given ζ^{n+1} are negligible.

Table 1. Simulation parameters used for SWEs

Parameters	Explanation	Value
L	simulation domain	$2000\,(Km)$
d	undisturbed water depth	$100\,(m)$
C_D	bottom drag coefficient	$1.0e-3$
g	acceleration due to gravity	$9.81\,(m/s^2)$
Δx	space step	$10\,(Km)$
Δt	time step	$300\,(s)$
$w_{\mathbf{imp}}$	implicit weighting	0.5

5 Evaluation Metrics

We compare trained solvers $\widehat{S}_\phi$ to a reference numerical method S with four error measures described previously (Huang et al., 2020). We calculate these metrics for each individual field $\widehat{z}^n$ estimating $z^n \in q^n$.

- *Normalized Root Mean Square Error* (NRMSE) describes a relative difference between estimated and reference solutions:

$$\text{NRMSE} = \frac{\|\widehat{z}^n - z^n\|_2}{\|z^n\|_2} \tag{31}$$

  Note that all estimated fields $\widehat{z}^n$ are integrated n time steps from the ICs of the reference solution.
- *Time-averaged Normalized Root Mean Square Error* ($\mathbb{E}_t$NRMSE) averages the NRMSE over the full duration of a simulation (in this work, 1200 time steps spanning 100 simulated hours).
- *Pearson's correlation* $\rho(\widehat{z}^n, z^n)$ of reference and learned solutions.
- *Training Success Rate* (TSR) is the probability that training will converge to a $\mathbb{E}_t\text{NRMSE} < 10$. This measure allows us to incorporate the stochastic aspect of deep learning in evaluating performance (both the initial weights and the order of ICs differ across runs). In this work, we quantify TSR by using $\mathbb{E}_t\text{NRMSE} - \zeta$ for SWEs.

6 Software implementation

Classical numerical and machine learning solvers are implemented in Pytorch and Numpy. Code for equivariant convolutions is partly adapted from GrouPy at `https://github.com/jornpeters/GrouPy/tree/pytorch\_p4\_p4m\_gconv/groupy/gconv`, while code for training on evolving simulations is based partly on code published in (Wandel et al., 2020), `https://github.com/wandeln/Unsupervised_Deep_Learning_of_Incompressible_Fluid_Dynamics`. The code for both numerical- and ML solver are publicly at `https://github.com/m-dml/GE-CNN_learning_SWEs`.

7 Experiments

We carried out unsupervised training of a hybrid PDE integration scheme for the SWEs, to determine whether hard symmetry constraints improve long-term accuracy and stability. Our reference simulations (see section 4.3) used a staggered grid with 200 height and 199 velocity points shown in Fig. 1b.

The neural network inferred surface height ζ_n from the system state at step $t-1$. Following (Wandel et al., 2020), we provide 13 input channels describing u_{n-1}, z_{n-1} and the BCs:

$$\text{input} = \left(\zeta, h, m^b_\zeta, m_\zeta, m^b_\zeta \cdot \zeta, m_\zeta \cdot \zeta, m^b_\zeta \cdot h, m_\zeta \cdot h, u, m^b_u, m_u, m^b_u \cdot u, m_u \cdot u\right) \tag{32}$$

Here, m_ζ is a mask for ζ. It is zero for boundary values and one for interior values while $m^b_\zeta = 1 - m_\zeta$, and m_u, m^b_u are the same for velocities. To deal with different spatial dimensions across input channels, we used kernel size 6 for ζ-sized inputs and kernel size 7 for u-sized inputs, and added the results in Fig. 2a.

In our hybrid scheme, the neural network replaced the expensive tridiagonal solve in Eq. 30 to compute ζ^{n+1}, while the numerical scheme computes coefficients of the tridiagonal system and updates u^{n+1} while imposing BCs. We can therefore drop the second term in Eq. 9 and replace the first with:

$$\mathcal{L}_{\text{SWE}} = \left\| A\zeta^{n+1} - b \right\|_2^2 \tag{33}$$

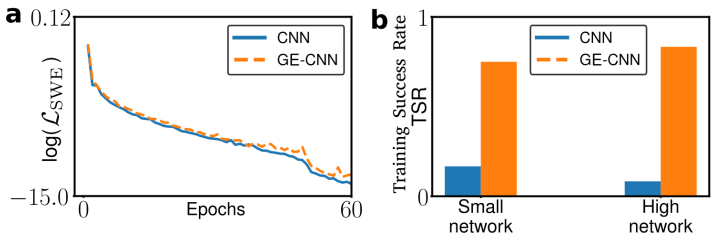


Figure 5. Progress and success rate of training for standard and reflection-equivariant convolutional networks. (a) Training loss curves for standard CNN and GE-CNN solvers. During training the loss of GE-CNN solver is a little larger than one of CNN solver. (b) Training success rate standard and GE-CNNs. Here, high- and small network respectively have about 148M and 1.6M parameters.

We compared equivariant networks (GE-CNN) to standard convolutional U-nets with the same architecture, loss and training procedure. For both network types, we adjusted the number of trainable parameters by scaling the number of output channels for all convolutions except the final layer, and for our default configuration as shown in Fig. 2a. This resulted in 148M parameters for the equivariant convolutions and 149M for standard convolutions.

7.1 Equivariant Networks are More Accurate and Easier to Train

We compared default configurations of our standard convolutional neural networks (CNN) to reflection-equivariant versions (GE-CNN), see Fig. 2a. We trained on a library of Gaussian bells ICs with occasional resets, as shown in section 3.3. Both standard and equivariant convolutional networks achieved low loss values and accurately predicted how the next time step for the SWEs (Fig. 5a). Since the standard CNN architecture describes a less restricted function class than the GE-CNN, it achieves a slightly lower loss value during training, at the cost of failing to respect symmetry in autoregressive predictions.

To test whether respecting symmetry would improve long term-accuracy, we therefore evaluated we compared the CNN and GE-CNN after training in autoregressive rollouts. Network outputs were used to define inputs for the next time step, and the results were compared to reference numerical solutions over 1200 time steps (100 simulated hours, Fig. 6a). The reference solutions describe waves propagating outward from the initial Gaussian bell before reflecting off the domain boundary five times (Fig. 6a, Fig. 6d black).

Individual CNN rollouts successfully reproduced the propagation and reflection of waves, but exhibited gradually increasing errors that increased when the waves were reflected by the closed boundaries (Fig. 6b). Over the CNN rollout the waves broadened, developed additional peaks in ζ not present in the reference simulation. Compared to to reference simulations, CNN rollouts exhibited higher spatial frequencies, a positive velocity bias and spatially asymmetric errors (Fig. 6d, blue). By the end of the rollout the magnitude of errors reached the amplitude of the simulated wave heights and velocities.

The GE-CNN followed the reference solution more closely, with errors at least one order of magnitude smaller than the simulated signals, and difficult to discern visually (Fig. 6c). The shape and width of the propagating and reflecting waves and surrounding smaller ripples closely matched the reference simulation. By the end of the simulation, errors appear as additional undulations in ζ, while u continues to to follow the reference simulation closely (Fig. 6d, orange).

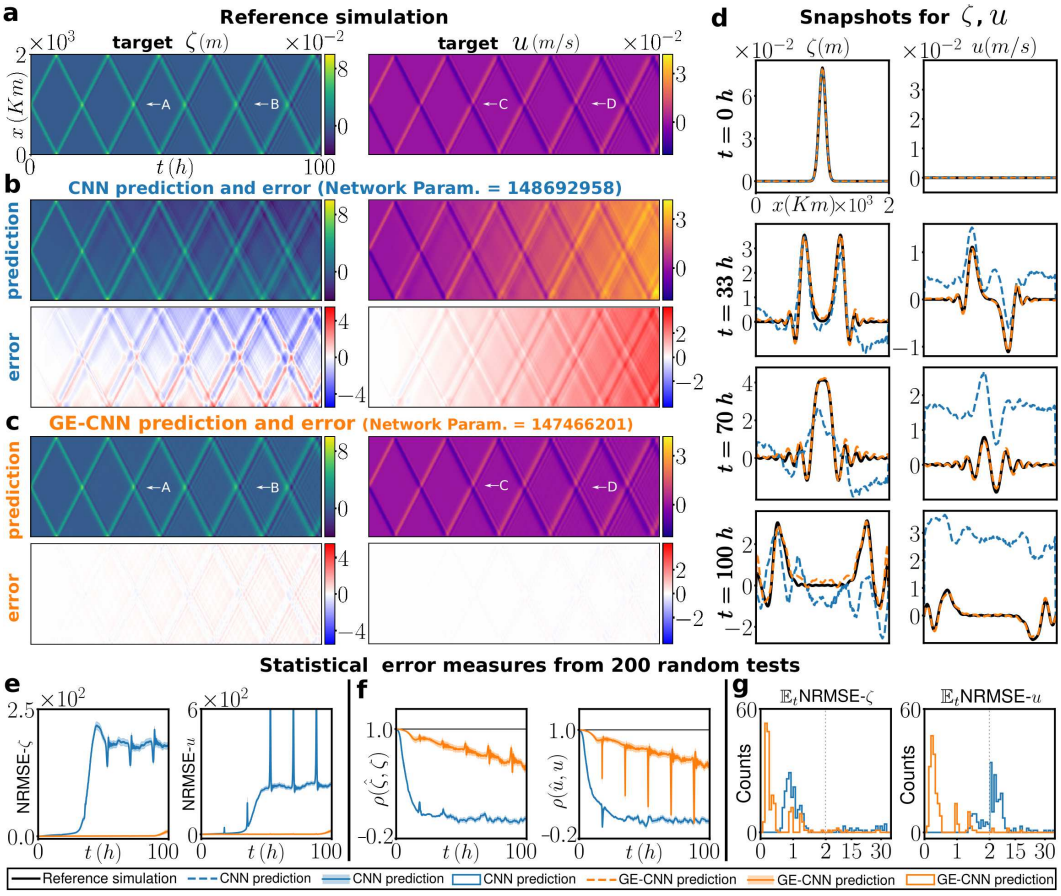


Figure 6. GE-CNNs produce accurate SWE rollouts from Gaussian bell ICs. (a) Reference simulation of surface elevation ζ and velocity u. (b) Rollouts from CNN solver for ζ and u, with errors (prediction – reference). (c) As in 'b,' but for GE-CNNs. (d) Snapshots of ζ and u obtained from the reference solver (black), CNN (blue) and GE-CNN orange. (e) Mean over ICs of NRMSE for ζ and u as a function of time from the start of the simulation, for CNNs (blue) and GE-CNNs. (f) As in 'e,' but for correlation. (g) Histograms of time averaged NRMSE in ζ and u for CNNs (blue) and GE-CNNs (orange).

The training success rate, defined as the probablity over multiple training runs and ICs of achieving low time-averaged error (in Section 5), was 10/12 for the GE-CNN but only 2/12 for the CNN (Fig. 5b). When using networks ~ 100 times smaller, we observed 9/12 successes for the GE-CNN (1.6M parameters) and 1/12 for the CNN (1.7M).

We further measured how accuracy of CNN and GE-CNN rollouts varied over time and ICs by computing rollouts for each over 200 ICs. NRMSE (see Section 5) in both ζ and u increased more quickly for the CNN, reaching average values 1-2 order of magnitude higher (Fig. 6e). Correlation coefficients between rollouts and the reference simulation followed a similar trend, with a decrease over time but clear superiority for the GE-CNN (Fig. 6f). We further examined the distribution of time-averaged NRMSE across ICs, computing histograms for u- and ζ-errors on logarithmic scales (Fig. 6g). The GE-CNN error distributions peaked near zero, while CNN errors peaked around the targeted signals' amplitude, with a long-tailed distribution.

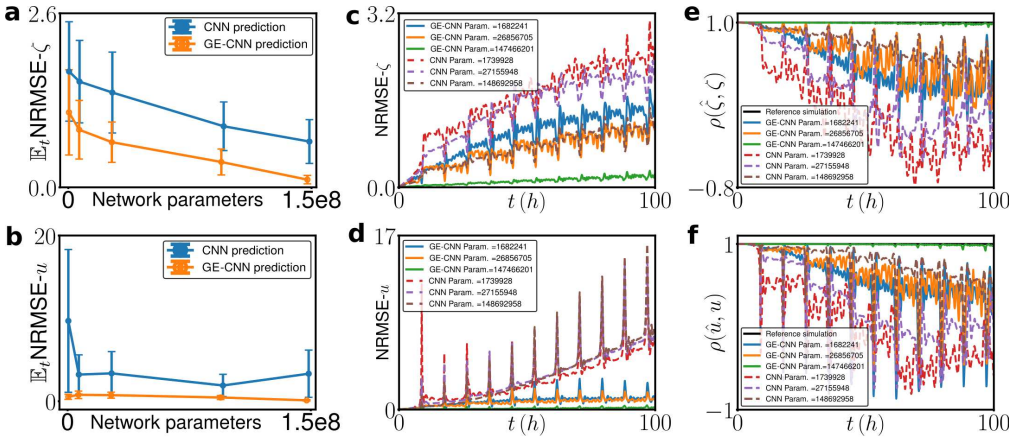


Figure 7. Reflection equivariance improves accuracy for all network sizes. (a-b) $\mathbb{E}_t$NRMSE-ζ and $\mathbb{E}_t$NRMSE-u as functions of network parameter counts in CNNs and GE-CNNs. Error bars show standard deviations which are obtained by using the predictions in 100 h. (c-f) Plots of NRMSE-ζ, NRMSE-u, $\rho(\hat{\zeta}, \zeta)$, and $\rho(\hat{u}, u)$ as functions of integration time for several sizes of CNN and GE-CNN.

For both CNN and GE-CNN rollouts, $\mathbb{E}_t$NRMSE-ζ decreases with parameter count of the trained networks, and is lower for GE-CNNs (Fig. 7a-b). The largest CNNs computed ζ as accurately as the smallest GE-CNNs, but for u even the smallest GE-CNNs outperformed CNNs of all sizes tested. The same trend was observed at individual time points for NRMSEs (Fig. 7c-d) and correlation values (Fig. 7e-f). Overall, these results show that the long-rollout accuracy improvement provided by equivariance is robust to the choice of network size and accuracy metric.

7.2 Generalization Capabilities After Training

We next examined how well PDEs solvers trained using Gaussian bell ICs would perform on conditions beyond their training data.

We first measured rollout accuracy for an ICs described by an isosceles triangle in ζ a the domain center with 400 km base and 0.12 m height (Fig. 8a). This system state was never encountered during training, as it contains a discontinuous first spatial derivative in contrast to the smooth Gaussian bell. Rollout errors we were considerably higher than for a novel Gaussian bell scenario, though the GE-CNN was again more accurate, especially for u (Fig. 8b-c). In the CNN rollout the propagating wave dissipated into many high-frequency ripples, while for the GE-CNN waves propagated with the correct shape but too slowly, leading to a position mismatch with reference simulations (Fig. 8d). CNN and GE-CNN rollout performance for triangular ICs with random height, width and position (uniform on 0.09-0.36 m, 200-300 km and 0-200 km respectively) showed similar trends to Gaussian bells. GE-CNN was uniformly superior, its errors grew more slowly over time, and its error distribution peaked near zero while the CNN's peaked above 4 times the estimated signals (Fig. 8g).

For a more challenging generalization task, we used a sum of 3 Gaussian bells as an initial condition. The reference simulation (Fig. 9a) shows 6 propagating and reflecting waves that form a complex interference pattern. As previously, in CNN rollouts the waves were distorted and dissipated over time with a positive bias emerging for u, while the GE-CNN maintained the correct shapes but introduced timing errors (Fig. 9c), and was more accurate at every time point (Fig. 9d). We also computed accuracy measures

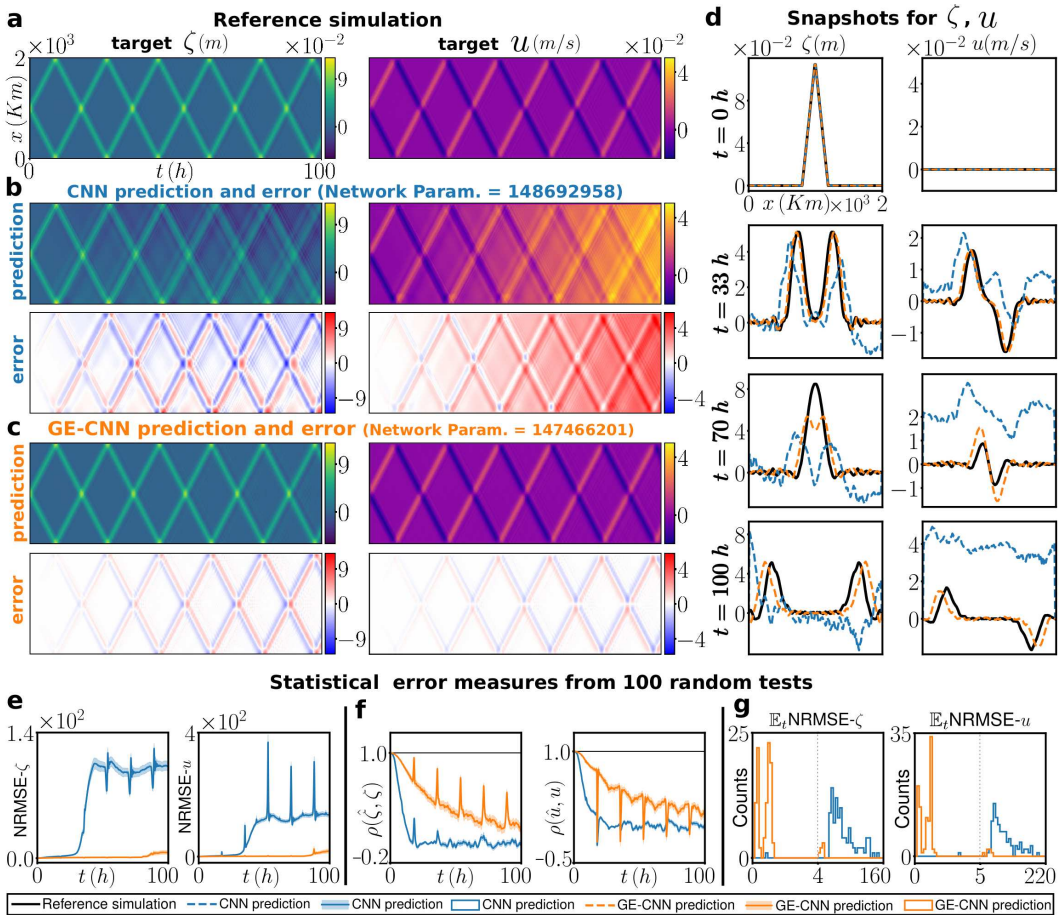


Figure 8. GE-CNN solver trained on Gaussian bell ICs generalizes to triangular ICs. (a) Reference simulation of surface elevation ζ and velocity u from a triangular IC. (b) Rollouts from CNN solver for ζ and u, with errors (prediction − reference). (c) As in 'b,' but for GE-CNNs. (d) Snapshots of ζ and u obtained from the reference solver (black), CNN (blue), and GE-CNN (orange). (e) Mean over ICs of NRMSE for ζ and u as a function of time from the start of the simulation, for CNNs (blue) and GE-CNNs. (f) As in 'e,' but for correlation. (g) Histograms of time averaged NRMSE in ζ and u for CNNs (blue) and GE-CNNs (orange).

for triple-bell ICs (Fig. 9e-g) with random means (uniform on 100-1900 km) and widths (10-100 km). The GE-CNN yielded better NRMSE and correlation values for all time delays, and a distribution of time-averagd NRMSE that peaked near zero and showed little overlap with CNN results.

7.3 Learned Representations of Global Mass, Momentum and Energy

A challenge for ML-based PDE solvers is that their predictions do not always satisfy conservation laws, even when these laws are manifested in their unsupervised loss or training data (Mohan et al., 2023; Kochkov et al., 2021). Our reference SWE solver conserves mass, conserves energy except for bottom drag, and conserves momentum except for bottom drag and boundary effects. Each reflection of a propagating wave from the closed boundaries involves a temporary conversion of kinetic to potential energy.

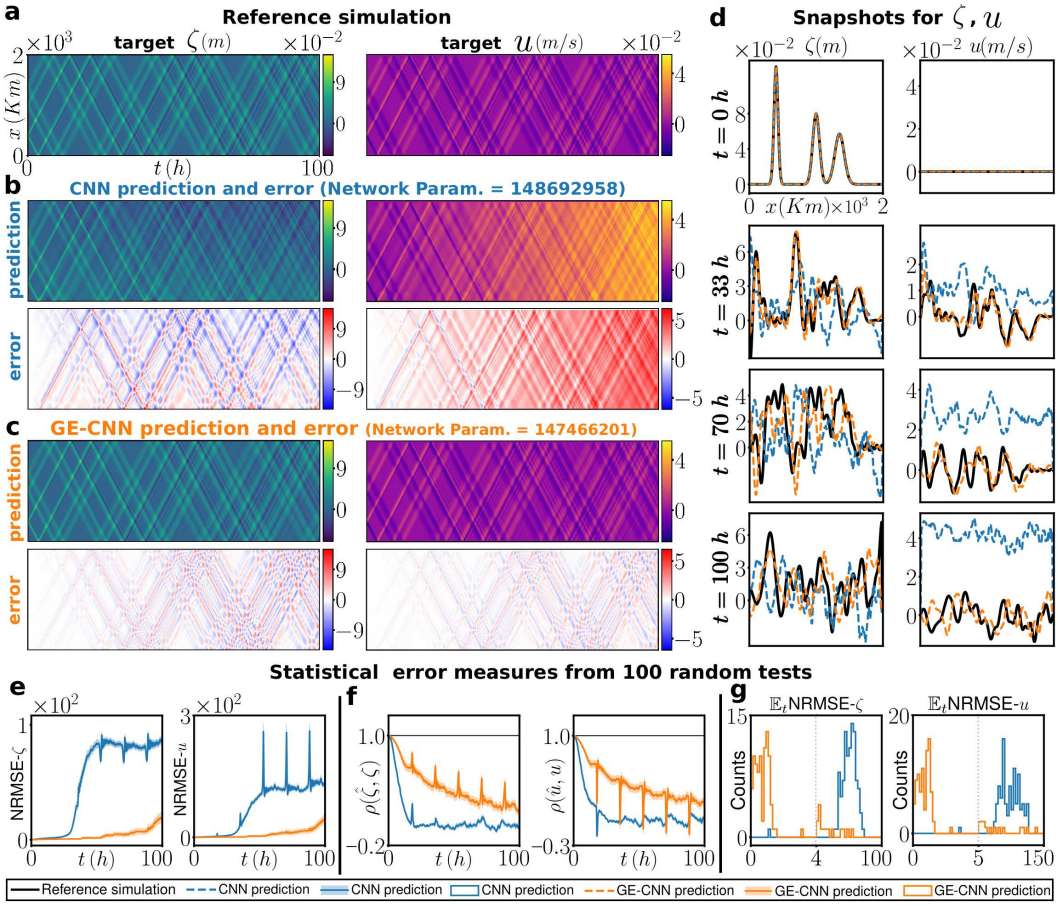


Figure 9. GE-CNN solver trained on Gaussian bell ICs generalizes to multi-bell ICs. (a) Reference simulation of surface elevation ζ and velocity u from a multi-bell IC (centers 400, 1000, 1350 km; widths 3, 5, 7 km). (b) Rollouts from CNN solver for ζ and u, with errors (prediction – reference). (c) As in 'b,' but for GE-CNNs. (d) Snapshots of ζ and u obtained from the reference solver (black), CNN (blue) and GE-CNN orange. (e) Mean over ICs of NRMSE for ζ and u as a function of time from the start of the simulation, for CNNs (blue) and GE-CNNs. (f) As in 'e,' but for correlation. (g) Histograms of time averaged NRMSE in ζ and u for CNNs (blue) and GE-CNNs (orange).

We investigated how well trained networks represent 4 quantities: mass $\sum h_i$, momentum $\sum h_i u_i$, kinetic energy $\frac{1}{2}\sum h_i u_i^2$, and potential energy $\frac{1}{2}\sum g h_i$. Note that we employed the disturbed water depth h to represent mass. We computed these for CNN and GE-CNN rollouts and compared to the reference solver. In individual held-out Gaussian bell ICs (Fig. 10a, upper row), CNN rollouts show a rapid error accumulation in all 4 quantities, while GE-CNNs exhibited a slow drift in total mass that produced a drift in potential energy, and negligible errors in momentum and kinetic energy. When averaging over many ICs, we found that average values of the conserved quantities matched closely for GE-CNN and the reference simulation until almost 100 hours, while the CNN showed clear differences after 50 h (Fig. 10a, lower row). Similar results were observed for triangular (Fig. 10b) and multi-bell ICs (Fig. 10c); for these ICs errors grew more quickly but the GE-CNN matched the reference simulation more closely.

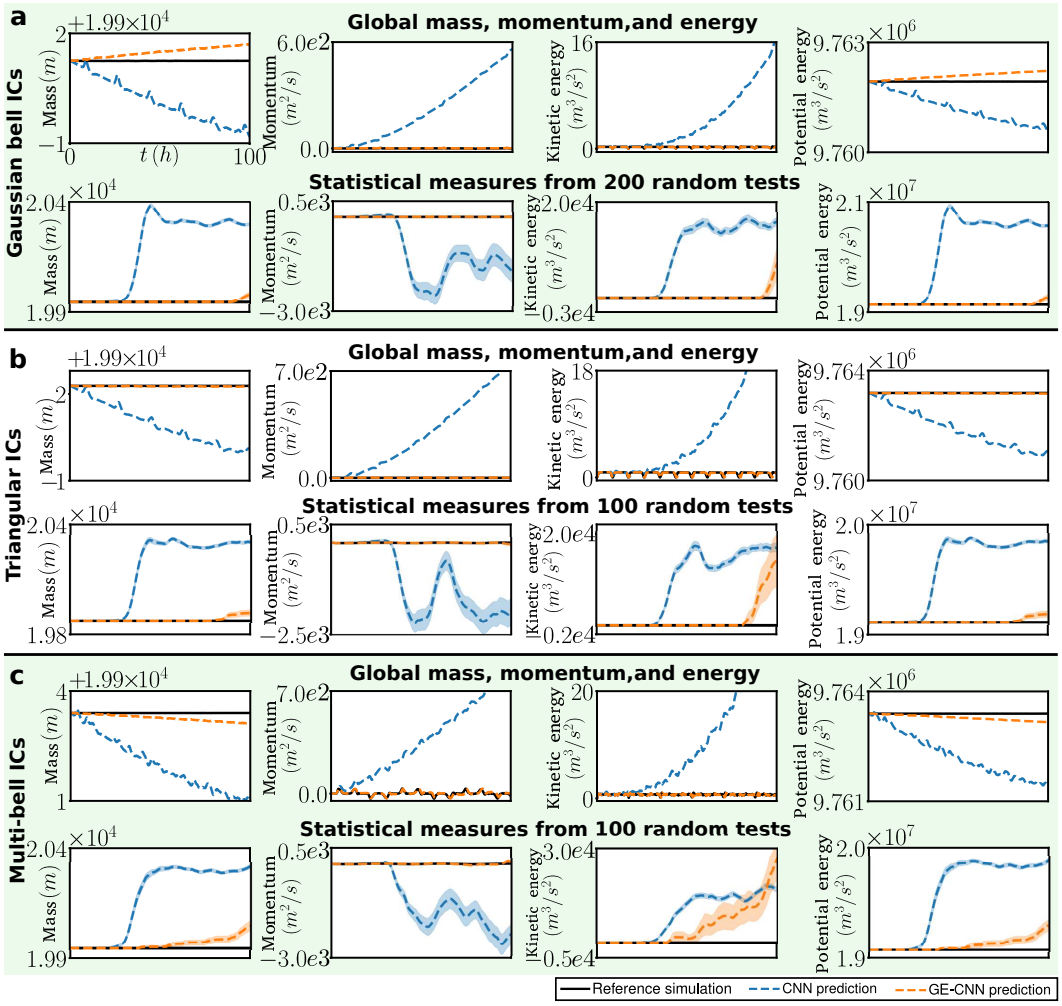


Figure 10. GE-CNN solver robustly predicts mass, momentum, and energy. (a) Global mass, momentum, kinetic- and potential energy from the reference solver, CNN- and GE-CNN rollouts (upper; same IC as Fig. 6). Mean and standard error of the mean for global mass, momentum, kinetic- and potential energy over 200 Gaussian bell ICs. (b) As in 'a,' but for triangular ICs. (c) As in 'a,' but for multi-bell ICs.

8 Discussion

We developed reflection-equivariant 1D convolutional networks for mixed vector-scalar inputs, and trained them to solve the SWEs with an unsupervised loss. We showed how these networks can be endowed with the same symmetries and the targeted PDEs, and our experiments showed how they improve accuracy and stability over standard CNNs with similar parameter counts, over a broad range of scenarios and tests. GE-CNNs matched reference simulations more closely at all time points, performed on new IC types and more faithfully represented mass, momentum, and energy. A remarkable aspect of these equivariant networks is that their advantages first become apparent when generating and evaluating longer rollouts, with no differences from standard CNNs apparent during training. Our results show that equivariant architectures offer significant benefits for long-term accuracy and physical consistency, with no modifications to the loss function or training procedures.

8.1 Related Work

Our work builds on and complements existing studies seeking to exploit symmetries for solving PDEs. (Brandstetter et al., 2022) use PDE symmetries to design data augmentations for use during training, instead of making their networks equivariant. (Wang, Walters, & Yu, 2020) built steerable CNNs (T. S. Cohen & Welling, 2016) and demonstrated their utility predicting the evolution of incompressible NS and an advected temperature field, but do not consider mixed scalar/vector inputs and examine their predictions only 10 time steps into the future.

In (Smets et al., 2022), the authors use network layers that solve a specific PDE to build convolutional networks, instead of constructing layers to match the symmetry groups of a specific PDE as we do here. (Marcos et al., 2017) use rotation equivariant convolutional layers to operate on vector fields, but do not consider mixed input types or solve PDEs.

The equivariant convolution layers we have developed for mixed scalar/vector inputs could also be realized using steerable convolutions (T. S. Cohen & Welling, 2016) with the correct combination of scalar and vector capsules. Instead of transforming filter banks, steerable convolutions are based on optimizing convolution weights within a pre-computed linear subspace that satisfies the desired constraints. While this approach is flexible and efficient, we believe our layers can provide considerable utility through their mathematical simplicity. Some studies have also reported successful implementation of equivariant network through filter bank transformation, but could not achieve the same results through steerable convolutions (Helwig et al., 2023).

8.2 Future Outlook

In future work, we intend to extend our results to higher dimensional and more complex systems, and to combine geometric and physical constraints (Guan et al., 2022; Ross et al., 2023). We also anticipate that by offering a combination of long-rollout performance and automatic differentiability, equivariant deep PDE solvers could prove useful for solving inverse problems (Nonnenmacher & Greenberg, 2021a; Holzschuh et al., 2023). The observed performance gains for long rollouts could also find useful applications in climate, weather and ocean modeling, which require stability and accuracy over far longer time intervals than commonly evaluated scenarios for deep PDE solvers.

Acknowledgments

We thank Kai Logemann for providing code and valuable insights into the semi-implicit shallow water solver. We also thank Vien Minh Nguyen-Thanh, Ali Can Bekar, Tobias Schanz, Shivani Sharma, Vadim Zinchenko, and Andrey Vlasenko for useful discussions and comments on the manuscript.

References

Akio, A., & Vivian, R. L. (1977). Computational design of the basic dynamical processes of the ucla general circulation model. In J. Chang (Ed.), *General circulation models of the atmosphere* (Vol. 17, p. 173-265). Elsevier. Retrieved from `https://www.sciencedirect.com/science/article/pii/B9780124608177500094` doi: https://doi.org/10.1016/B978-0-12-460817-7.50009-4

Backhaus, J. O. (1983). A semi-implicit scheme for the shallow water equations for application to shelf sea modelling. *Cont. Shelf Res.*, *2*(4), 243–254.

Bar, L., & Sochen, N. (2019). Unsupervised deep learning algorithm for pde-based forward and inverse problems. *arXiv preprint arXiv:1904.05417*.

Bassi, F., & Rebay, S. (1997). A high-order accurate discontinuous finite element method for the numerical solution of the compressible navier–stokes equations. *J. Comput. Phys.*, *131*(2), 267–279.

Brandstetter, J., Welling, M., & Worrall, D. E. (2022). Lie point symmetry data augmentation for neural pde solvers. In *International conference on machine learning* (pp. 2241–2256).

Bunya, S., Kubatko, E. J., Westerink, J. J., & Dawson, C. (2009). A wetting and drying treatment for the runge–kutta discontinuous galerkin solution to the shallow water equations. *Comput. Methods Appl. Mech. Eng.*, *198*(17-20), 1548–1562.

Cai, Z., Chen, J., Liu, M., & Liu, X. (2020). Deep least-squares methods: An unsupervised learning-based numerical method for solving elliptic pdes. *J. Comput. Phys.*, *420*, 109707.

Casulli, V. (1990). Semi-implicit finite difference methods for the two-dimensional shallow water equations. *J. Comput. Phys.*, *86*(1), 56–74.

Casulli, V., & Walters, R. A. (2000). An unstructured grid, three-dimensional model based on the shallow water equations. *Int. J. Numer. Methods Fluids*, *32*(3), 331–348.

Cohen, T., & Welling, M. (2016). Group equivariant convolutional networks. In *International conference on machine learning* (pp. 2990–2999).

Cohen, T. S., & Welling, M. (2016). Steerable cnns. *arXiv preprint arXiv:1612.08498*.

Constantin, P., & Foias, C. (2020). *Navier-stokes equations*. University of Chicago Press.

Crank, J., & Nicolson, P. (1947). A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type. In *Mathematical proceedings of the cambridge philosophical society* (Vol. 43, pp. 50–67).

Deardorff, J. W. (1970). A numerical study of three-dimensional turbulent channel flow at large reynolds numbers. *J. Fluid Mech.*, *41*(2), 453–480.

Dellar, P. J., & Salmon, R. (2005). Shallow water equations with a complete coriolis force and topography. *Phys. Fluids*, *17*(10), 106601.

De St Venant, B. (1871). Theorie du mouvement non-permanent des eaux avec application aux crues des rivers et a l'introduntion des marees dans leur lit. *Academic de Sci. Comptes Redus*, *73*(99), 148–154.

Egger, H., & Schöberl, J. (2010). A hybrid mixed discontinuous galerkin finite-element method for convection–diffusion problems. *SIAM J. Numer. Anal.*, *30*(4), 1206–1234.

Euler, L. (1757). Principes généraux du mouvement des fluides. *Mémoires de l'académie des sciences de Berlin*, 274–315.

Eymard, R., Gallouët, T., & Herbin, R. (2000). Finite volume methods. *Handbook of numerical analysis*, *7*, 713–1018.

Fischer, P. F., Hecht, F., & Maday, Y. (2005). A parareal in time semi-implicit approximation of the navier-stokes equations. *Lect. Notes Comput. Sci. Eng.*, *40*, 433–440.

Forti, D., & Dedè, L. (2015). Semi-implicit bdf time discretization of the navier–stokes equations with vms-les modeling in a high performance computing framework. *Comput. Fluids*, *117*, 168–182.

Fresca, S., & Manzoni, A. (2022). Pod-dl-rom: Enhancing deep learning-based reduced order models for nonlinear parametrized pdes by proper orthogonal decomposition. *Comput. Methods Appl. Mech. Eng.*, *388*, 114181.

Geneva, N., & Zabaras, N. (2020). Modeling the dynamics of pde systems with physics-constrained deep auto-regressive networks. *J. Comput. Phys.*, *403*, 109056.

Gerken, J. E., Aronsson, J., Carlsson, O., Linander, H., Ohlsson, F., Petersson, C., & Persson, D. (2023, December). Geometric deep learning and equiv-

ariant neural networks. *Artif. Intell. Rev.*, *56*(12), 14605–14662. Retrieved 2023-10-20, from https://doi.org/10.1007/s10462-023-10502-7 doi: 10.1007/s10462-023-10502-7

Grilli, S., Pedersen, T., & Stepanishen, P. (1998). A hybrid boundary element method for shallow water acoustic propagation over an irregular bottom. *Eng. Anal. Bound. Elem.*, *21*(2), 131–145.

Grzeszczuk, R., Terzopoulos, D., & Hinton, G. (1998, July). NeuroAnimator: fast neural network emulation and control of physics-based models. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques* (pp. 9–20). New York, NY, USA: Association for Computing Machinery. Retrieved 2020-07-16, from https://doi.org/10.1145/280814.280816 doi: 10.1145/280814.280816

Guan, Y., Subel, A., Chattopadhyay, A., & Hassanzadeh, P. (2022). Learning physics-constrained subgrid-scale closures in the small-data regime for stable and accurate les. *Phys. D: Nonlinear Phenom.*, 133568.

Gupta, G., Xiao, X., & Bogdan, P. (2021). Multiwavelet-based operator learning for differential equations. *Adv. Neural Inf. Process Syst.*, *34*, 24048–24062.

Han, J., Ding, J., Xue, N., & Xia, G.-S. (2021). Redet: A rotation-equivariant detector for aerial object detection. In *Proceedings of the ieee/cvf conference on computer vision and pattern recognition* (pp. 2786–2795).

Helwig, J., Zhang, X., Fu, C., Kurtin, J., Wojtowytsch, S., & Ji, S. (2023, July). Group equivariant fourier neural operators for partial differential equations. In *Proceedings of the 40th International Conference on Machine Learning* (Vol. 202, pp. 12907–12930). Honolulu, Hawaii, USA: JMLR.org.

Holzschuh, B. J., Vegetti, S., & Thuerey, N. (2023). *Score matching via differentiable physics.*

Huang, Y., Gompper, G., & Sabass, B. (2020). A bayesian traction force microscopy method with automated denoising in a user-friendly software package. *Comput. Phys. Commun.*, *256*, 107313.

Kärnä, T., De Brye, B., Gourgue, O., Lambrechts, J., Comblen, R., Legat, V., & Deleersnijder, E. (2011). A fully implicit wetting–drying method for dg-fem shallow water models, with an application to the scheldt estuary. *Comput. Methods Appl. Mech. Eng.*, *200*(5-8), 509–524.

Kashinath, K., Mustafa, M., Albert, A., Wu, J., Jiang, C., Esmaeilzadeh, S., ... others (2021). Physics-informed machine learning: case studies for weather and climate modelling. *Philos. Trans. R. Soc. A*, *379*(2194), 20200093.

Kim, B., Azevedo, V. C., Thuerey, N., Kim, T., Gross, M., & Solenthaler, B. (2019). Deep fluids: A generative network for parameterized fluid simulations. In *Computer graphics forum* (Vol. 38, pp. 59–70).

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Klöwer, M., Düben, P., & Palmer, T. (2020). Number formats, error mitigation, and scope for 16-bit arithmetics in weather and climate modeling analyzed with a shallow water model. *J. Adv. Model. Earth Syst.*, *12*(10), e2020MS002246.

Kochkov, D., Smith, J. A., Alieva, A., Wang, Q., Brenner, M. P., & Hoyer, S. (2021). Machine learning–accelerated computational fluid dynamics. *Proc. Natl. Acad. Sci. U. S. A.*, *118*(21), e2101784118.

Kohl, G., Chen, L.-W., & Thuerey, N. (2023, September). *Turbulent Flow Simulation using Autoregressive Conditional Diffusion Models.* arXiv. Retrieved 2023-10-20, from http://arxiv.org/abs/2309.01745 (arXiv:2309.01745 [physics]) doi: 10.48550/arXiv.2309.01745

Korn, P., Brüggemann, N., Jungclaus, J. H., Lorenz, S., Gutjahr, O., Haak, H., ... others (2022). Icon-o: The ocean component of the icon earth system model—global simulation characteristics and local telescoping capability. *J. Adv. Model. Earth Syst.*, *14*(10), e2021MS002952.

Ladickỳ, L., Jeong, S., Solenthaler, B., Pollefeys, M., & Gross, M. (2015). Data-driven fluid simulations using regression forests. *ACM Trans. Graph.*, *34*(6), 1–9.

Li, Y., Xu, L., & Ying, S. (2022). Dwnn: Deep wavelet neural network for solving partial differential equations. *Mathematics*, *10*(12), 1976.

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., & Anandkumar, A. (2020a). Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*.

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., & Anandkumar, A. (2020b). Neural operator: Graph kernel network for partial differential equations. *arXiv preprint arXiv:2003.03485*.

List, B., Chen, L.-W., & Thuerey, N. (2022). Learned turbulence modelling with differentiable fluid solvers. *arXiv preprint arXiv:2202.06988*.

Long, Z., Lu, Y., & Dong, B. (2019). Pde-net 2.0: Learning pdes from data with a numeric-symbolic hybrid deep network. *J. Comput. Phys.*, *399*, 108925.

Lye, K. O., Mishra, S., & Ray, D. (2020). Deep learning observables in computational fluid dynamics. *J. Comput. Phys.*, *410*, 109339.

Marcos, D., Volpi, M., Komodakis, N., & Tuia, D. (2017). Rotation equivariant vector field networks. In *Proceedings of the ieee international conference on computer vision* (pp. 5048–5057).

Michelis, M. Y., & Katzschmann, R. K. (2022). Physics-constrained unsupervised learning of partial differential equations using meshes. *arXiv preprint arXiv:2203.16628*.

Mohan, A. T., Lubbers, N., Chertkov, M., & Livescu, D. (2023). Embedding hard physical constraints in neural network coarse-graining of three-dimensional turbulence. *Phys. Rev. Fluids*, *8*(1), 014604.

Nonnenmacher, M., & Greenberg, D. S. (2021a). Deep emulators for differentiation, forecasting, and parametrization in earth science simulators. *J. Adv. Model. Earth Syst.*, *13*(7), e2021MS002554.

Nonnenmacher, M., & Greenberg, D. S. (2021b). Learning Implicit PDE Integration with Linear Implicit Layers. In *The Symbiosis of Deep Learning and Differential Equations.*

Obiols-Sales, O., Vishnu, A., Malaya, N., & Chandramowliswharan, A. (2020). Cfdnet: A deep learning-based accelerator for fluid simulations. In *Proceedings of the 34th acm international conference on supercomputing* (pp. 1–12).

Polyak, B. T. (1969). The conjugate gradient method in extremal problems. *USSR Comput. Math. & Math. Phys.*, *9*(4), 94–112.

Raissi, M., Perdikaris, P., & Karniadakis, G. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.*, *378*, 686-707. Retrieved from `https://www.sciencedirect.com/science/article/pii/S0021999118307125` doi: https://doi.org/10.1016/j.jcp.2018.10.045

Reynolds, O. (1895). Iv. on the dynamical theory of incompressible viscous fluids and the determination of the criterion. *Philos. Trans. Royal Soc. A*(186), 123–164.

Ronneberger, O., Fischer, P., & Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *International conference on medical image computing and computer-assisted intervention* (pp. 234–241).

Ross, A., Li, Z., Perezhogin, P., Fernandez-Granda, C., & Zanna, L. (2023). Benchmarking of machine learning ocean subgrid parameterizations in an idealized model. *J. Adv. Model. Earth Syst.*, *15*(1), e2022MS003258.

Saad, Y. (2003). *Iterative methods for sparse linear systems.* SIAM.

Saad, Y., & Schultz, M. H. (1986). Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Comput.*, *7*(3), 856–869.

Sadourny, R. (1975). The dynamics of finite-difference models of the shallow-water equations. *J. Atmos. Sci.*, *32*(4), 680–689.

Smagorinsky, J. (1963). General circulation experiments with the primitive equations: I. the basic experiment. *Mon. Weather Rev.*, *91*(3), 99–164.

Smets, B. M., Portegies, J., Bekkers, E. J., & Duits, R. (2022). Pde-based group equivariant convolutional neural networks. *J. Math. Imaging Vis.*, 1–31.

Stanziola, A., Arridge, S. R., Cox, B. T., & Treeby, B. E. (2021). A helmholtz equation solver using unsupervised learning: Application to transcranial ultrasound. *J. Comput. Phys.*, *441*, 110430.

Sun, L., Gao, H., Pan, S., & Wang, J.-X. (2020). Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data. *Comput. Methods Appl. Mech. Eng.*, *361*, 112732.

Tajbakhsh, N., Jeyaseelan, L., Li, Q., Chiang, J. N., Wu, Z., & Ding, X. (2020). Embracing imperfect datasets: A review of deep learning solutions for medical image segmentation. *Med. Image Anal.*, *63*, 101693.

Taylor, M., Tribbia, J., & Iskandarani, M. (1997). The spectral element method for the shallow water equations on the sphere. *J. Comput. Phys.*, *130*(1), 92–108.

Temam, R. (2001). *Navier-stokes equations: theory and numerical analysis* (Vol. 343). American Mathematical Soc.

Thuerey, N., Weißenow, K., Prantl, L., & Hu, X. (2020). Deep learning methods for reynolds-averaged navier–stokes simulations of airfoil flows. *AIAA Journal*, *58*(1), 25–36.

Tompson, J., Schlachter, K., Sprechmann, P., & Perlin, K. (2017). Accelerating eulerian fluid simulation with convolutional networks. In *International conference on machine learning* (pp. 3424–3433).

Turek, S. (1996). A comparative study of time-stepping techniques for the incompressible navier-stokes equations: from fully implicit non-linear schemes to semi-implicit projection methods. *Int. J. Numer. Methods Fluids.*, *22*(10), 987–1011.

Um, K., Brand, R., Fei, Y. R., Holl, P., & Thuerey, N. (2020). Solver-in-the-loop: Learning from differentiable physics to interact with iterative pde-solvers. *Adv. Neural Inf. Process. Syst.*, *33*, 6111–6122.

Van der Vorst, H. A. (1992). Bi-cgstab: A fast and smoothly converging variant of bi-cg for the solution of nonsymmetric linear systems. *SIAM J. Sci. Comput.*, *13*(2), 631–644.

Wandel, N., Weinmann, M., & Klein, R. (2020). Learning incompressible fluid dynamics from scratch–towards fast, differentiable fluid models that generalize. *arXiv preprint arXiv:2006.08762*.

Wandel, N., Weinmann, M., & Klein, R. (2021). Teaching the incompressible navier–stokes equations to fast neural surrogate models in three dimensions. *Phys. Fluids*, *33*(4), 047117.

Wang, R., Kashinath, K., Mustafa, M., Albert, A., & Yu, R. (2020). Towards physics-informed deep learning for turbulent flow prediction. In *Proceedings of the 26th acm sigkdd international conference on knowledge discovery & data mining* (pp. 1457–1466).

Wang, R., Walters, R., & Yu, R. (2020). Incorporating symmetry into deep dynamics models for improved generalization. *arXiv preprint arXiv:2002.03061*.

Wen, G., Li, Z., Azizzadenesheli, K., Anandkumar, A., & Benson, S. M. (2022). U-fno—an enhanced fourier neural operator-based deep-learning model for multiphase flow. *Adv. Water Resour.*, *163*, 104180.

Wiewel, S., Becher, M., & Thuerey, N. (2019). Latent space physics: Towards learning the temporal evolution of fluid flow. In *Computer graphics forum* (Vol. 38, pp. 71–82).

Wu, T., Maruyama, T., & Leskovec, J. (2022). Learning to accelerate partial differential equations via latent global evolution. *arXiv preprint arXiv:2206.07681*.

Yuval, J., O'Gorman, P. A., & Hill, C. N. (2021). Use of neural networks for stable, accurate and physically consistent parameterization of subgrid atmospheric processes with good performance at reduced precision. *Geophys. Res. Lett.*, *48*(6), e2020GL091363.

Yuval, J., & O'Gorman, P. A. (2020). Stable machine-learning parameterization of subgrid processes for climate modeling at a range of resolutions. *Nat. Commun.*, *11*(1), 3295.

Zängl, G., Reinert, D., Rípodas, P., & Baldauf, M. (2015). The icon (icosahedral non-hydrostatic) modelling framework of dwd and mpi-m: Description of the non-hydrostatic dynamical core. *Quarterly Journal of the Royal Meteorological Society*, *141*(687), 563–579.

Zhu, Y., Zabaras, N., Koutsourelakis, P.-S., & Perdikaris, P. (2019). Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. *J. Comput. Phys.*, *394*, 56 - 81. doi: https://doi.org/10.1016/j.jcp.2019.05.024

Zienkiewicz, O., & Ortiz, P. (1995). A split-characteristic based finite element model for the shallow water equations. *Int. J. Numer. Methods Fluids.*, *20*(8-9), 1061–1080.

Zijlema, M., & Stelling, G. (2008). Efficient computation of surf zone waves using the nonlinear shallow water equations with non-hydrostatic pressure. *Coast. Eng.*, *55*(10), 780–790.

Appendix A Proof of Equivariance for Convolution Layers

For completeness, we first provide proofs for the equivariance of the original scalar-field-only convolution layers of (T. Cohen & Welling, 2016) in our notation. We then prove equivariance for our mixed scalar/vector input layers.

A1 Proof of Equivariance for Scalar-input Convolutional Input Layer

The outputs of the first (input) layer of an equivariant convolutional network $a^1_{j,0,\cdot}$ and $a^1_{j,1,\cdot}$ are defined in Eq. (16) and Eq. (17). We flip the input $R_{\mathrm{F}}(q_{i,\cdot})$. Applying this layer to a fliped input using the same weights and biases gives:

$$\tilde{a}^1_{j,0,\cdot} = \sum_{i=1}^{c_{\mathrm{in}}} W^1_{j,i,\cdot} \star R_{\mathrm{F}}\left(q_{i,\cdot}\right) + b^1_j \tag{A1}$$

$$\tilde{a}^1_{j,1,\cdot} = \sum_{i=1}^{c_{\mathrm{in}}} R_{\mathrm{F}}\left(W^1_{j,i,\cdot}\right) \star R_{\mathrm{F}}\left(q_{i,\cdot}\right) + b^1_j \tag{A2}$$

To prove the equivariance in this layer, we flip the first layer of output. This flipping operator can be moved into the convolution. Then, we obtain

$$R_{\mathrm{F}}(\tilde{a}^1_{j,0,\cdot}) = R_{\mathrm{F}}\left(\sum_{i=1}^{c_{\mathrm{in}}} W^1_{j,i,\cdot} \star R_{\mathrm{F}}\left(q_{i,\cdot}\right) + b^1_j\right) = \sum_{i=1}^{c_{\mathrm{in}}} R_{\mathrm{F}}\left(W^1_{j,i,\cdot}\right) \star q_{i,\cdot} + b^1_j = a^1_{j,1,\cdot} \tag{A3}$$

$$R_{\mathrm{F}}(\tilde{a}^1_{j,1,\cdot}) = R_{\mathrm{F}}\left(\sum_{i=1}^{c_{\mathrm{in}}} R_{\mathrm{F}}\left(W^1_{j,i,\cdot}\right) \star R_{\mathrm{F}}\left(q_{i,\cdot}\right) + b^1_j\right) = \sum_{i=1}^{c_{\mathrm{in}}} W^1_{j,i,\cdot} \star q_{i,\cdot} + b^1_j = a^1_{j,0,\cdot} \tag{A4}$$

Now, these two equations satisfy the definition of the group equivariance in Eq. (10). Thus, we finish the proof. An example plot for the group equivariance of this layer is shown in Fig. (3).

A2 Proof of Equivariance for Non-Input Convolution Layers

The output of subsequent layers, for which inputs and output channels are both real-valued functions on H, is given in eq. 19. A flipped input $R_{\mathrm{F}}(x_{i,\cdot})$ gives the output

$$\tilde{a}^\ell_{j,0,\cdot} = \sum_{i=1}^{c_{\ell-1}} W_{j,i,0,\cdot} \star \tilde{a}^{\ell-1}_{i,0,\cdot} + W_{j,i,1,\cdot} \star \tilde{a}^{\ell-1}_{i,1,\cdot} + b^\ell_j \tag{A5}$$

$$\tilde{a}^\ell_{j,1,\cdot} = \sum_{i=1}^{c_{\ell-1}} R_{\mathrm{F}}(W_{j,i,0,\cdot}) \star \tilde{a}^{\ell-1}_{i,1,\cdot} + R_{\mathrm{F}}(W_{j,i,1,\cdot}) \star \tilde{a}^{\ell-1}_{i,0,\cdot} + b^\ell_j \tag{A6}$$

Flipping outputs gives

$$R_{\mathrm{F}}(\tilde{a}^\ell_{j,0,\cdot}) = R_{\mathrm{F}}\left(\sum_{i=1}^{c_{\ell-1}} W_{j,i,0,\cdot} \star \tilde{a}^{\ell-1}_{i,0,\cdot} + W_{j,i,1,\cdot} \star \tilde{a}^{\ell-1}_{i,1,\cdot} + b^\ell_j\right) \tag{A7}$$

$$R_{\mathrm{F}}(\tilde{a}^\ell_{j,1,\cdot}) = R_{\mathrm{F}}\left(\sum_{i=1}^{c_{\ell-1}} R_{\mathrm{F}}(W_{j,i,0,\cdot}) \star \tilde{a}^{\ell-1}_{i,1,\cdot} + R_{\mathrm{F}}(W_{j,i,1,\cdot}) \star \tilde{a}^{\ell-1}_{i,0,\cdot} + b^\ell_j\right) \tag{A8}$$

Then, we move the flipping operator R_{F} into the conversation features. We not only need to flip the weights but also switch the non-flipped input. Thus, Eqs. (A7-A8) can be written as

$$R_{\mathrm{F}}(\tilde{a}^\ell_{j,0,\cdot}) = \sum_{i=1}^{c_{\ell-1}} R_{\mathrm{F}}\left(W_{j,i,1,\cdot}\right) \star a^{\ell-1}_{i,0,\cdot} + R_{\mathrm{F}}\left(W_{j,i,0,\cdot}\right) \star a^{\ell-1}_{i,1,\cdot} + b^\ell_j = a^\ell_{j,1,\cdot} \tag{A9}$$

$$R_{\mathrm{F}}(\tilde{a}^\ell_{j,1,\cdot}) = \sum_{i=1}^{c_{\ell-1}} W_{j,i,0,\cdot} \star a^{\ell-1}_{i,0,\cdot} + W_{j,i,1,\cdot} \star a^{\ell-1}_{i,1,\cdot} + b^\ell_j = a^\ell_{j,0,\cdot} \tag{A10}$$

Therefore, according to the definition of equivariance, we have proven the equivariance convolution in subsequent layers. The example plot is also illustrated in Fig. (3).

A3 Proof of Equivariance for Mixed Scalar-Vector Convolution Layers

The first layer's outputs for mixed scalar-vector inputs are shown in Eqs. (22-23). Here, we prove the equivariance in this layers. According to the symmetry of the vector field shown in Eq. (21), we transform the input as $R_{\mathrm{F}}(\zeta_{i,\cdot})$ and $-R_{\mathrm{F}}(u_{i,\cdot})$. Thus, the first layer of output using the flipping input is written as

$$\tilde{a}^1_{j,0,\cdot} = \sum_{i=1}^{c^{\zeta}_{in}} W^{\zeta}_{j,i,\cdot} \star R_{\mathrm{F}}(\zeta_{i,\cdot}) + \sum_{i=1}^{c^{u}_{in}} W^{u}_{j,i,\cdot} \star -R_{\mathrm{F}}(u_{i,\cdot}) + b^{\ell}_{j} \tag{A11}$$

$$\tilde{a}^1_{j,1,\cdot} = \sum_{i=1}^{c^{\zeta}_{in}} R_{\mathrm{F}}(W^{\zeta}_{j,i,\cdot}) \star R_{\mathrm{F}}(\zeta_{i,\cdot}) + \sum_{i=1}^{c^{u}_{in}} -R_{\mathrm{F}}(W^{u}_{j,i,\cdot}) \star -R_{\mathrm{F}}(u_{i,\cdot}) + b^{\ell}_{j} \tag{A12}$$

Next, we flip these outputs

$$R_{\mathrm{F}}(\tilde{a}^1_{j,0,\cdot}) = R_{\mathrm{F}}\left(\sum_{i=1}^{c^{\zeta}_{in}} W^{\zeta}_{j,i,\cdot} \star R_{\mathrm{F}}(\zeta_{i,\cdot}) + \sum_{i=1}^{c^{u}_{in}} -W^{u}_{j,i,\cdot} \star R_{\mathrm{F}}(u_{i,\cdot}) + b^{\ell}_{j} \right) \tag{A13}$$

$$R_{\mathrm{F}}(\tilde{a}^1_{j,1,\cdot}) = R_{\mathrm{F}}\left(\sum_{i=1}^{c^{\zeta}_{in}} R_{\mathrm{F}}(W^{\zeta}_{j,i,\cdot}) \star R_{\mathrm{F}}(\zeta_{i,\cdot}) + \sum_{i=1}^{c^{u}_{in}} R_{\mathrm{F}}(W^{u}_{j,i,\cdot}) \star R_{\mathrm{F}}(u_{i,\cdot}) + b^{\ell}_{j} \right) \tag{A14}$$

Now, we move the flipping operator into the convolution operator. The flipping weight and input feature can be changed as the following equations,

$$R_{\mathrm{F}}(\tilde{a}^1_{j,0,\cdot}) = \sum_{i=1}^{c^{\zeta}_{in}} R_{\mathrm{F}}(W^{\zeta}_{j,i,\cdot}) \star \zeta_{i,\cdot} + \sum_{i=1}^{c^{u}_{in}} -R_{\mathrm{F}}(W^{u}_{j,i,\cdot}) \star u_{i,\cdot} + b^{\ell}_{j} = a^1_{j,1,\cdot} \tag{A15}$$

$$R_{\mathrm{F}}(\tilde{a}^1_{j,1,\cdot}) = \sum_{i=1}^{c^{\zeta}_{in}} W^{\zeta}_{j,i,\cdot} \star \zeta_{i,\cdot} + \sum_{i=1}^{c^{u}_{in}} W^{u}_{j,i,\cdot} \star u_{i,\cdot} + b^{\ell}_{j} = a^1_{j,0,\cdot} \tag{A16}$$

Thus, according to the definition of equivariance of convolution, we have proven the equivariance for mixed scalar-vector convolution layers.

Appendix B Numerical Discretization of SWEs

Here we describe how the space- and time-discretized variable fields u^n and ζ^n of the SWE at the n-th time step are used to compute the (n+1)-th time step. We describe the procedures used for the semiiplict non-deep-learning-based classical numerical solver, which is a biased upwind scheme (Backhaus, 1983).

We discretize the momentum equation (eq. 24) as follows:

$$u^{n+1} = u^n - \Delta t C_D \frac{1}{h} u^n |u^n| - \Delta t g (1 - w_{\mathbf{imp}}) \frac{\partial \zeta^n}{\partial x} - \Delta t g w_{\mathbf{imp}} \frac{\partial \zeta^{n+1}}{\partial x} \tag{B1}$$

where $w_{\mathbf{imp}}$ is a fixed parameter controlling weighting between implicit and explicit time stepping. The mass equation (eq. 25) is discretized as:

$$\zeta^{n+1} = \zeta^n - \Delta t (1 - w_{\mathbf{imp}}) \frac{\partial h^n u^n}{\partial x} - \Delta t w_{\mathbf{imp}} \frac{\partial h^n u^{n+1}}{\partial x} \tag{B2}$$

Recall that $h = d + \zeta$ and d is the undisturbed water depth. Eq. (B1) is inserted into eq. (B2) to obtain

$$\zeta^{n+1} = \zeta^n - \Delta t (1 - w_{\mathbf{imp}}) \frac{\partial h^n u^n}{\partial x} - \Delta t w_{\mathbf{imp}} \frac{\partial h^n u^*}{\partial x} + \Delta t^2 w^2_{\mathbf{imp}} g \frac{\partial^2 h^n \zeta^{n+1}}{\partial x^2} \tag{B3}$$

where u^* is an 'interim solution' defined by

$$u^* = u^n - \Delta t c_D \frac{1}{h} u^n |u^n| - \Delta t g (1 - w_{\mathbf{imp}}) \frac{\partial \zeta^n}{\partial x} \tag{B4}$$

When calculating the product of two variables defined on the velocity and mass points of the Arakawa C-grid (Fig. 1b), we interpolate the mass variable to velocity grid points by averaging adjacent values. For a quantity α defined on the velocity or mass gird, the first spatial derivative is discretized as $\frac{\partial \alpha}{\partial x}_i = \frac{\alpha_{i+1/2} - \alpha_{i-1/2}}{\Delta x}$, with outputs staggered by $\Delta x/2$ from inputs. The second spatial derivative is discretized using the second order finite difference $\frac{\partial^2 \alpha}{\partial x^2}_i = \frac{\alpha_{i+1} - 2\alpha_i + \alpha_{i-1}}{\Delta x^2}$, with outputs on the same grid as inputs. Therefore, eq. (B3) can be written as

$$\zeta_i^{n+1} = \frac{1}{1 + c_E + c_W} \left[\zeta^n + \mathrm{div} + c_E \zeta_{i+1}^{n+1} + c_W \zeta_{i-1}^{n+1} \right] \tag{B5}$$

where $\mathrm{div} = -\Delta t (1 - w_{\mathbf{imp}}) \frac{\partial h^n u^n}{\partial x} - \Delta t w_{\mathbf{imp}} \frac{\partial h^n u^*}{\partial x}$, while c_E and c_W are defined as

$$c_E = \begin{cases} \frac{0.5 \Delta t^2 w_{\mathbf{imp}}^2 g (h_i^n + h_{i+1}^n)}{\Delta x^2} & if \;\; h_{i+1}^n > 0 \\ 0 \;\; \text{otherwise.} \end{cases}$$

$$c_W = \begin{cases} \frac{0.5 \Delta t^2 w_{\mathbf{imp}}^2 g (h_i^n + h_{i-1}^n)}{\Delta x^2} & if \;\; h_{i-1}^n > 0 \\ 0 \;\; \text{otherwise.} \end{cases}$$

Eq. (B5) describes a linear system of equations in ζ^{n+1} that can be written in matrix-vector form

$$A \zeta^{n+1} = b \tag{B6}$$

where A is a $N \times N$ tridiagonal matrix ($N = L/\Delta x$) with $A_{k,k} = 1$, $A_{k,k-1} = -\frac{c_W}{1 + c_E + c_W}$, $A_{k,k+1} = -\frac{c_E}{1 + c_E + c_W}$ and all other elements zero. $b \in \mathbb{R}^N$ with $b = \frac{\zeta^n + div}{1 + c_E + c_W}$. Following (Backhaus, 1983), we employ Gauss-Seidel iterations to solve eq. B6. Having obtained ζ^{n+1}, the new velocity u^{n+1} is calculated as

$$u^{n+1} = u^* - \Delta t g w_{\mathbf{imp}} \frac{\partial \zeta^{n+1}}{\partial x} \tag{B7}$$

Figure 1.

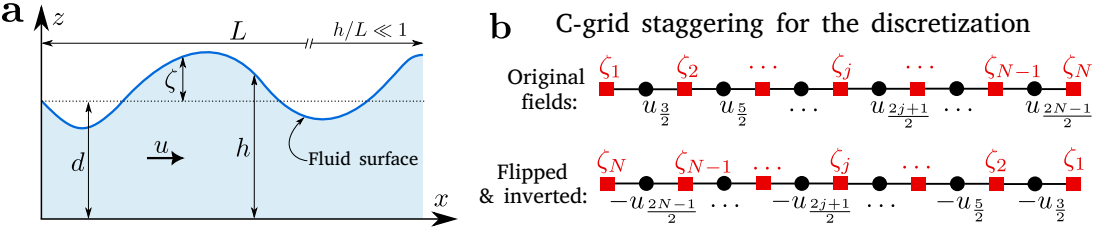

a
z
L
h/L ≪ 1
ζ
d
u
h
Fluid surface
x
b
C-grid staggering for the discretization
Original fields:
ζ1
ζ2
ζj
ζN−1
ζN
u 3/2
u 5/2
u (2j+1)/2
u (2N−1)/2
Flipped & inverted:
ζN
ζN−1
ζj
ζ2
ζ1
−u (2N−1)/2
−u (2j+1)/2
−u 5/2
−u 3/2

Figure 2.

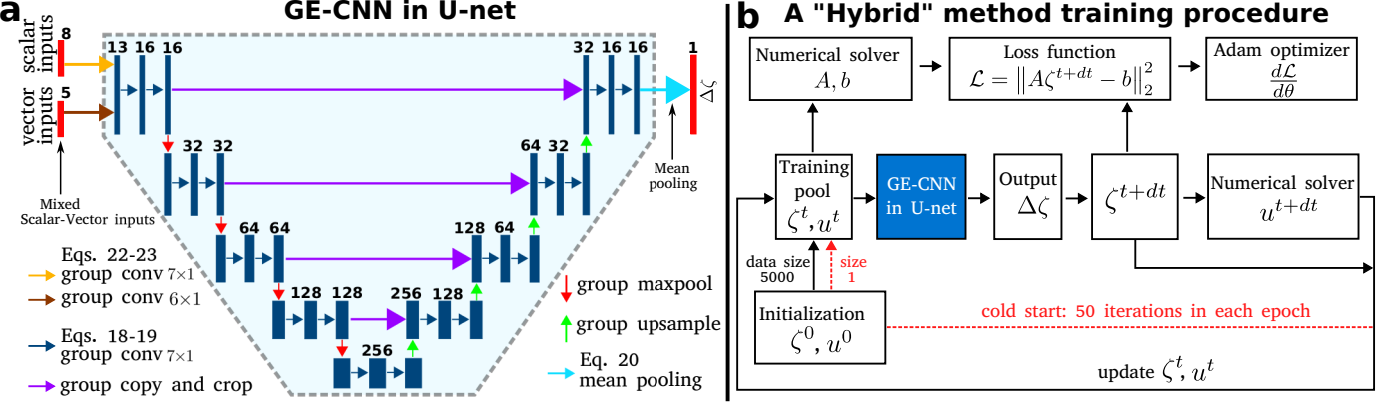

a
GE-CNN in U-net
scalar inputs
8
vector inputs
5
Mixed Scalar-Vector inputs
13 16 16
32 32
64 64
128 128
256
256 128
128 64
64 32
32 16 16
1
$\Delta\zeta$
Mean pooling
Eqs. 22-23
group conv 7×1
group conv 6×1
Eqs. 18-19
group conv 7×1
group copy and crop
group maxpool
group upsample
Eq. 20
mean pooling
b
A "Hybrid" method training procedure
Numerical solver
A, b
Loss function
$\mathcal{L} = \|A\zeta^{t+dt} - b\|_2^2$
Adam optimizer
$\frac{d\mathcal{L}}{d\theta}$
Training pool
ζ^t, u^t
GE-CNN in U-net
Output
$\Delta\zeta$
ζ^{t+dt}
Numerical solver
u^{t+dt}
data size 5000
size 1
Initialization
ζ^0, u^0
cold start: 50 iterations in each epoch
update ζ^t, u^t

Figure 3.

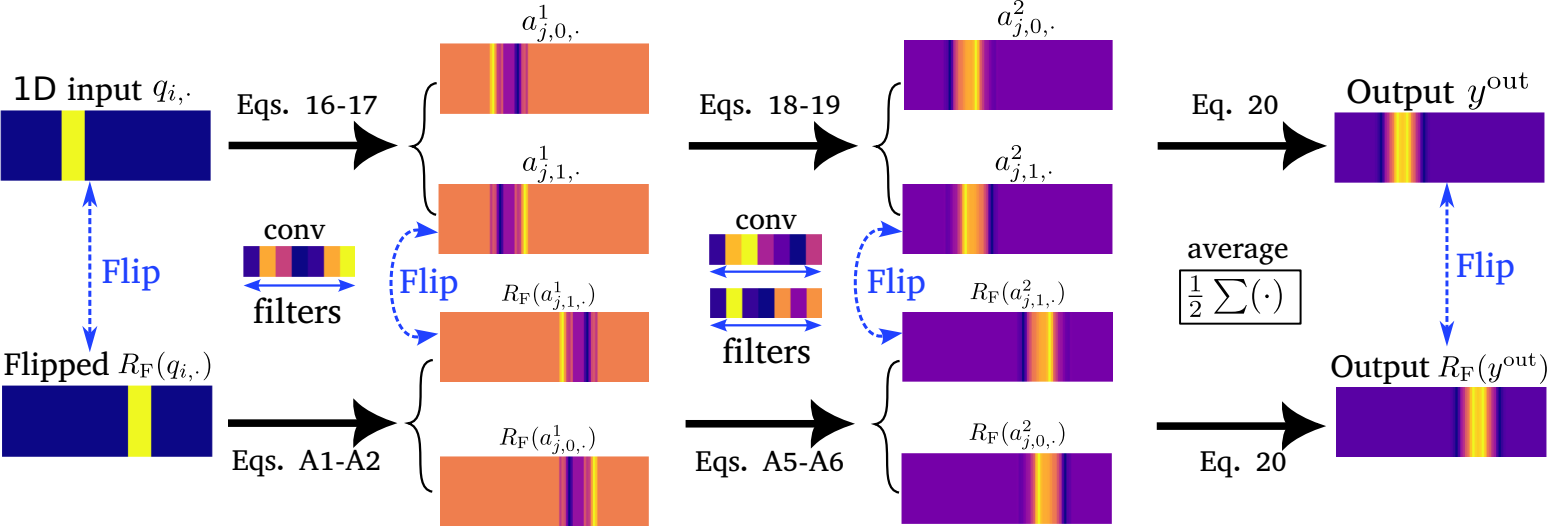

1D input $q_{i,\cdot}$
Flip
Flipped $R_{\mathrm{F}}(q_{i,\cdot})$
Eqs. 16-17
Eqs. A1-A2
conv
filters
Flip
$a^1_{j,0,\cdot}$
$a^1_{j,1,\cdot}$
$R_{\mathrm{F}}(a^1_{j,1,\cdot})$
$R_{\mathrm{F}}(a^1_{j,0,\cdot})$
Eqs. 18-19
Eqs. A5-A6
conv
filters
Flip
$a^2_{j,0,\cdot}$
$a^2_{j,1,\cdot}$
$R_{\mathrm{F}}(a^2_{j,1,\cdot})$
$R_{\mathrm{F}}(a^2_{j,0,\cdot})$
Eq. 20
Eq. 20
average
$\frac{1}{2}\sum(\cdot)$
Output y^{out}
Flip
Output $R_{\mathrm{F}}(y^{\mathrm{out}})$

Figure 4.

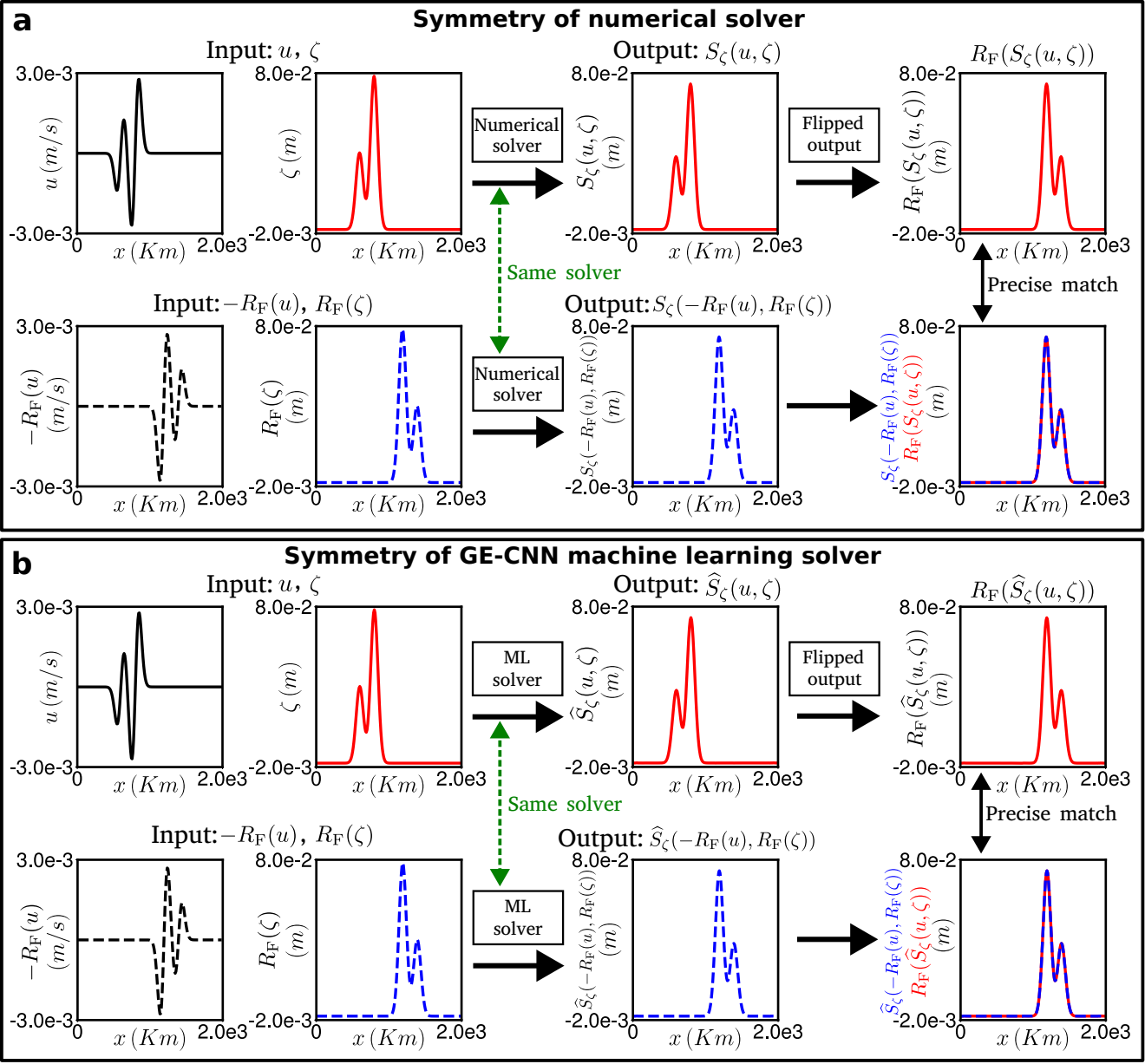

a
Symmetry of numerical solver
Input: u, ζ
Output: $S_\zeta(u,\zeta)$
$R_F(S_\zeta(u,\zeta))$
Numerical solver
Flipped output
Same solver
Precise match
Input: $-R_F(u)$, $R_F(\zeta)$
Output: $S_\zeta(-R_F(u), R_F(\zeta))$
Numerical solver
b
Symmetry of GE-CNN machine learning solver
Input: u, ζ
Output: $\widehat{S}_\zeta(u,\zeta)$
$R_F(\widehat{S}_\zeta(u,\zeta))$
ML solver
Flipped output
Same solver
Precise match
Input: $-R_F(u)$, $R_F(\zeta)$
Output: $\widehat{S}_\zeta(-R_F(u), R_F(\zeta))$
ML solver
$u\ (m/s)$
$\zeta\ (m)$
$-R_F(u)\ (m/s)$
$R_F(\zeta)\ (m)$
$x\ (Km)$
3.0e-3
-3.0e-3
8.0e-2
-2.0e-3
0
2.0e3

Figure 5.

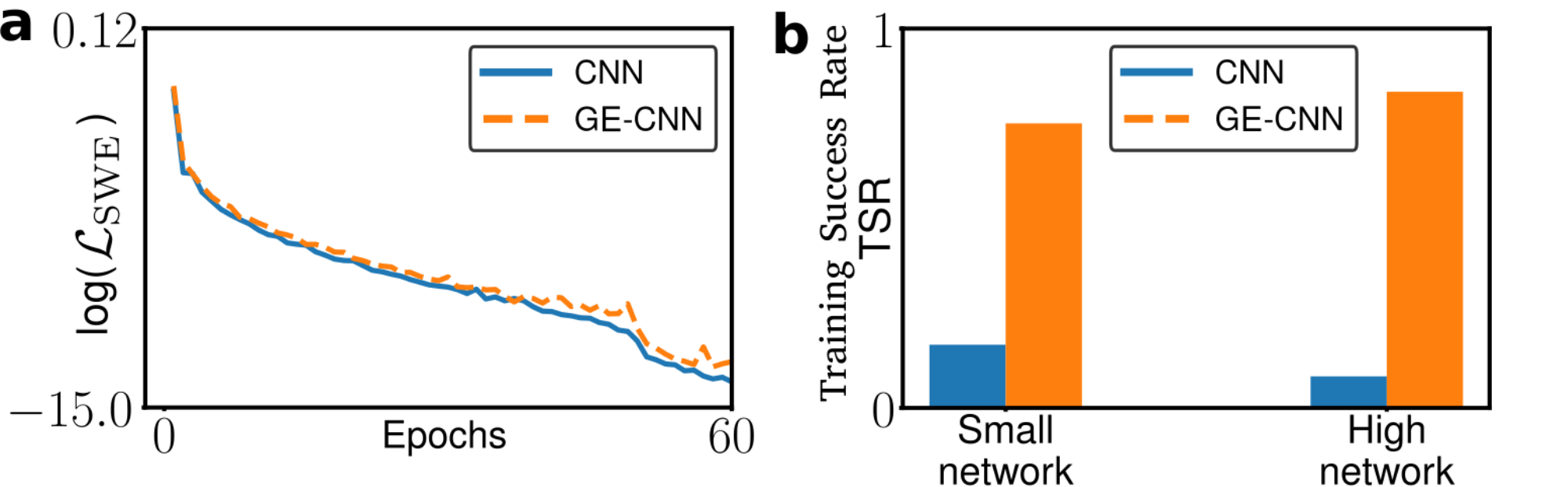

a
0.12
−15.0
log($\mathcal{L}_{\mathrm{SWE}}$)
0
Epochs
60
CNN
GE-CNN
b
Training Success Rate
TSR
1
0
Small network
High network
CNN
GE-CNN

Figure 6.

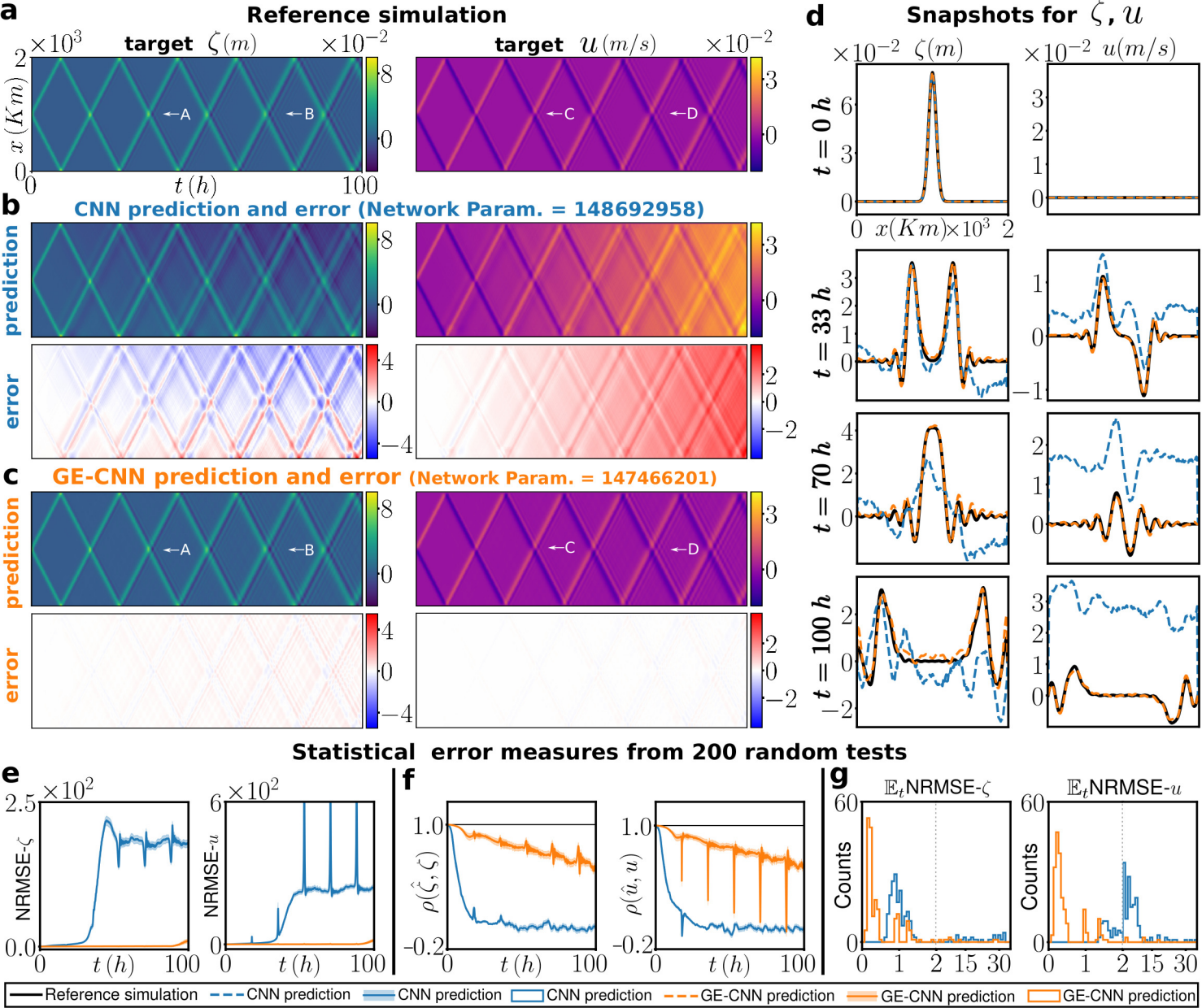

a
Reference simulation
target $\zeta(m)$
target $u\,(m/s)$
$\times 10^3$
$\times 10^{-2}$
$x\,(Km)$
$t\,(h)$
A
B
C
D
b
CNN prediction and error (Network Param. = 148692958)
prediction
error
c
GE-CNN prediction and error (Network Param. = 147466201)
prediction
error
d
Snapshots for ζ, u
$\zeta(m)$
$u(m/s)$
$x(Km)\times 10^3$
$t = 0\,h$
$t = 33\,h$
$t = 70\,h$
$t = 100\,h$
Statistical error measures from 200 random tests
e
NRMSE-ζ
NRMSE-u
$t\,(h)$
f
$\rho(\hat{\zeta}, \zeta)$
$\rho(\hat{u}, u)$
$t\,(h)$
g
$\mathbb{E}_t$NRMSE-ζ
$\mathbb{E}_t$NRMSE-u
Counts
Reference simulation
CNN prediction
CNN prediction
CNN prediction
GE-CNN prediction
GE-CNN prediction
GE-CNN prediction

Figure 7.

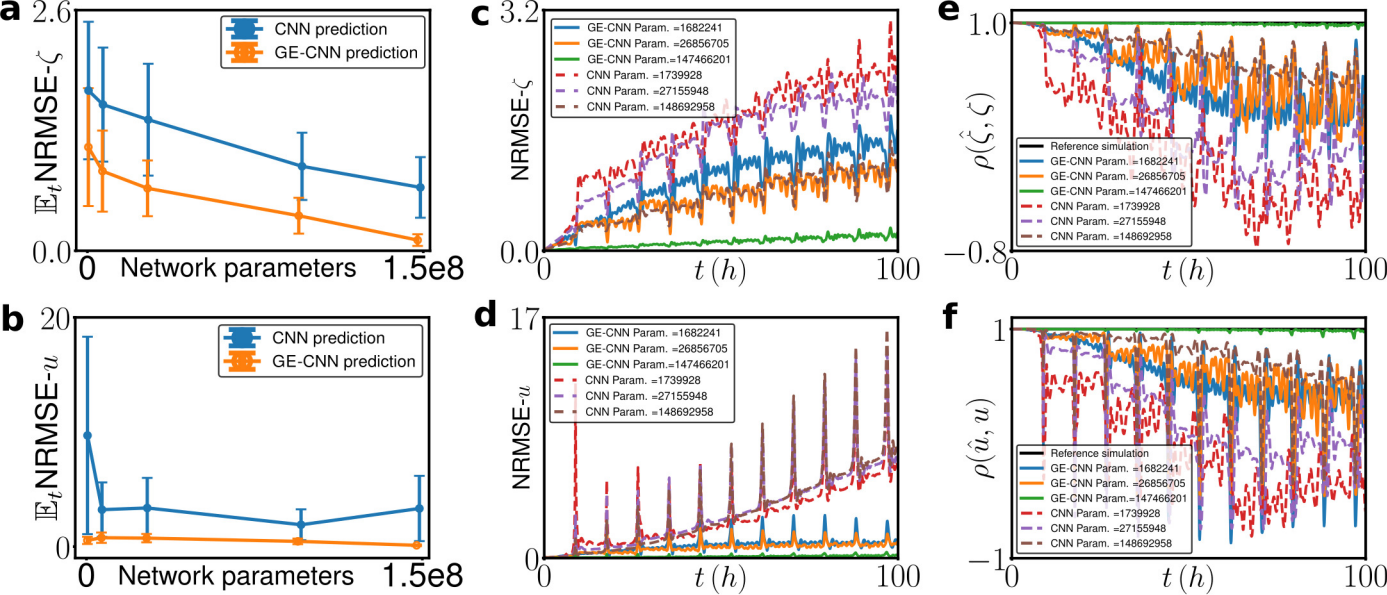

a
2.6
$\mathbb{E}_t$NRMSE-ζ
CNN prediction
GE-CNN prediction
0.0
0
Network parameters
1.5e8
b
20
$\mathbb{E}_t$NRMSE-u
CNN prediction
GE-CNN prediction
0
0
Network parameters
1.5e8
c
3.2
NRMSE-ζ
GE-CNN Param. =1682241
GE-CNN Param. =26856705
GE-CNN Param.=147466201
CNN Param. =1739928
CNN Param. =27155948
CNN Param. =148692958
0.0
0
$t\,(h)$
100
d
17
NRMSE-u
GE-CNN Param. =1682241
GE-CNN Param. =26856705
GE-CNN Param.=147466201
CNN Param. =1739928
CNN Param. =27155948
CNN Param. =148692958
0
0
$t\,(h)$
100
e
1.0
$\rho(\hat{\zeta}, \zeta)$
Reference simulation
GE-CNN Param. =1682241
GE-CNN Param. =26856705
GE-CNN Param.=147466201
CNN Param. =1739928
CNN Param. =27155948
CNN Param. =148692958
−0.8
0
$t\,(h)$
100
f
1
$\rho(\hat{u}, u)$
Reference simulation
GE-CNN Param. =1682241
GE-CNN Param. =26856705
GE-CNN Param.=147466201
CNN Param. =1739928
CNN Param. =27155948
CNN Param. =148692958
−1
0
$t\,(h)$
100

Figure 8.

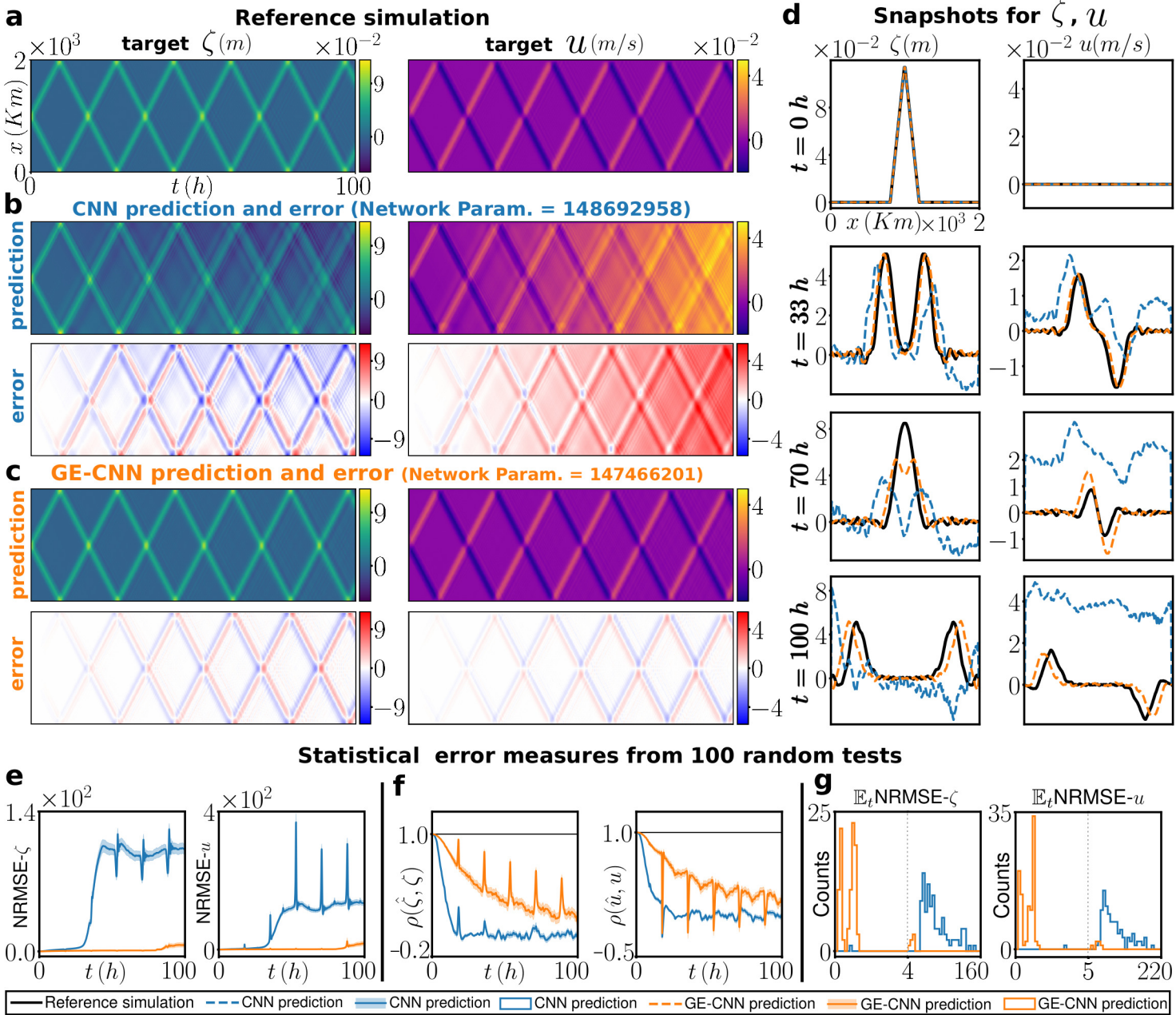

a
Reference simulation
target $\zeta(m)$
target $u(m/s)$
$\times 10^3$
$\times 10^{-2}$
$x(Km)$
$t(h)$
b
CNN prediction and error (Network Param. = 148692958)
prediction
error
c
GE-CNN prediction and error (Network Param. = 147466201)
prediction
error
d
Snapshots for ζ, u
$\times 10^{-2}$ $\zeta(m)$
$\times 10^{-2}$ $u(m/s)$
$t = 0\ h$
$t = 33\ h$
$t = 70\ h$
$t = 100\ h$
$x(Km)\times 10^3$
Statistical error measures from 100 random tests
e
NRMSE-ζ
NRMSE-u
$\times 10^2$
$t(h)$
f
$\rho(\hat{\zeta}, \zeta)$
$\rho(\hat{u}, u)$
$t(h)$
g
$\mathbb{E}_t$NRMSE-ζ
$\mathbb{E}_t$NRMSE-u
Counts
Reference simulation
CNN prediction
CNN prediction
CNN prediction
GE-CNN prediction
GE-CNN prediction
GE-CNN prediction

Figure 9.

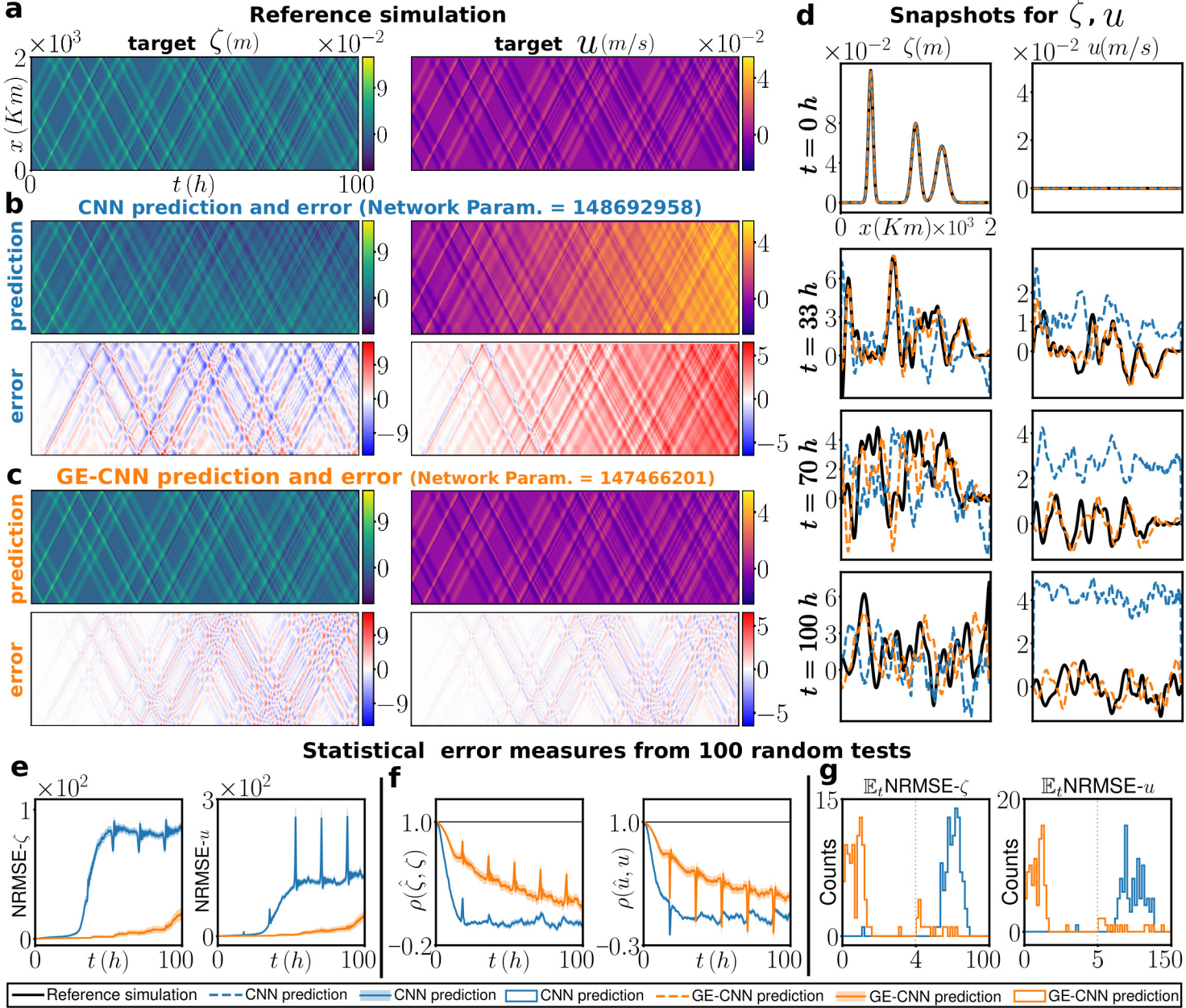
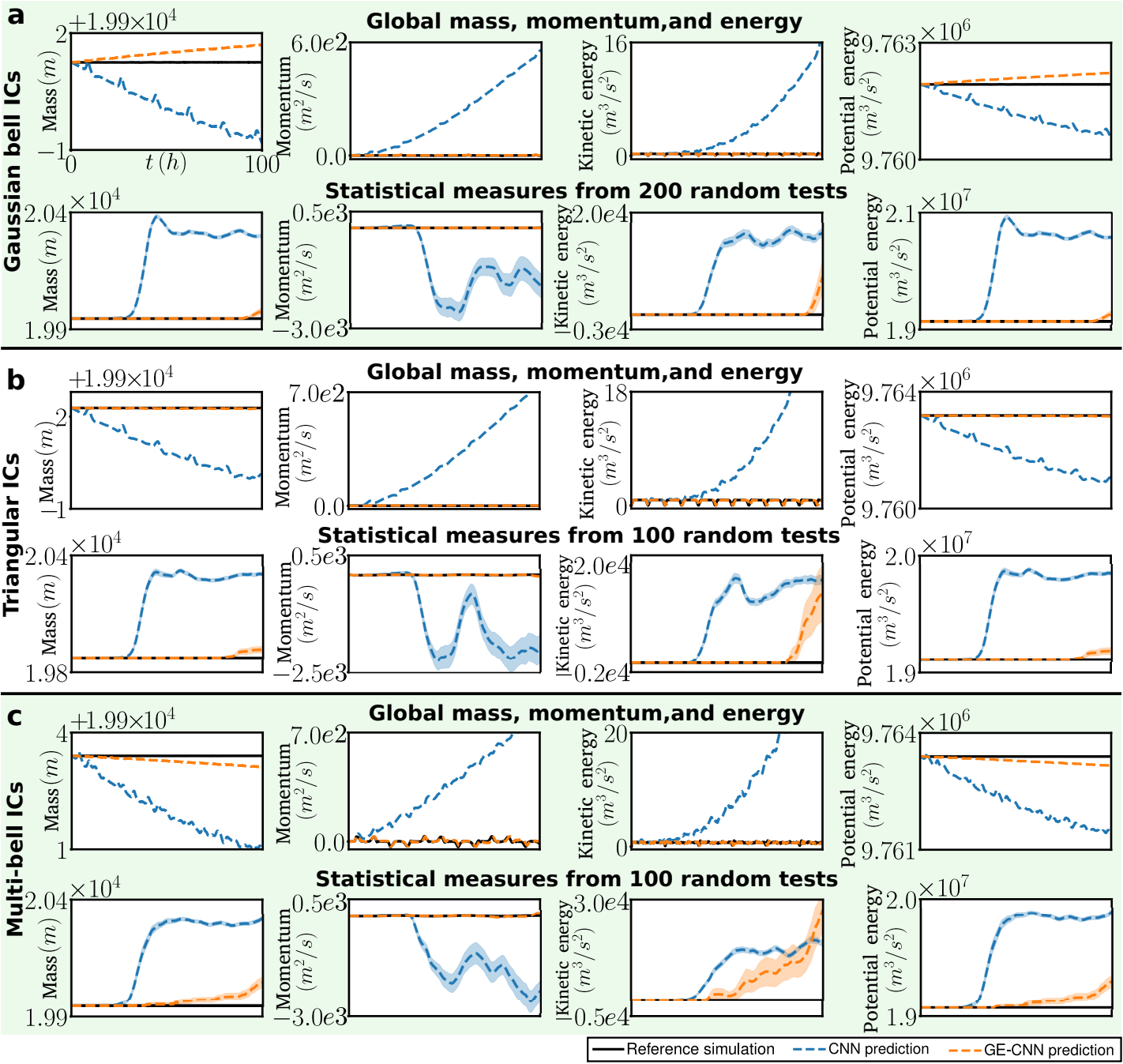

a
Reference simulation
target $\zeta(m)$
target $u(m/s)$
$\times 10^3$
$\times 10^{-2}$
$x\,(Km)$
$t\,(h)$
b
CNN prediction and error (Network Param. = 148692958)
prediction
error
c
GE-CNN prediction and error (Network Param. = 147466201)
prediction
error
d
Snapshots for ζ, u
$\times 10^{-2}$ $\zeta(m)$
$\times 10^{-2}$ $u(m/s)$
$t = 0\ h$
$t = 33\ h$
$t = 70\ h$
$t = 100\ h$
$x(Km)\times 10^3$
e
Statistical error measures from 100 random tests
NRMSE-ζ
NRMSE-u
$\times 10^2$
$t\,(h)$
f
$\rho(\hat{\zeta}, \zeta)$
$\rho(\hat{u}, u)$
g
$\mathbb{E}_t$NRMSE-ζ
$\mathbb{E}_t$NRMSE-u
Counts
Reference simulation
CNN prediction
CNN prediction
CNN prediction
GE-CNN prediction
GE-CNN prediction
GE-CNN prediction

Figure 10.


a
Gaussian bell ICs
Global mass, momentum,and energy
Statistical measures from 200 random tests
b
Triangular ICs
Global mass, momentum,and energy
Statistical measures from 100 random tests
c
Multi-bell ICs
Global mass, momentum,and energy
Statistical measures from 100 random tests
Mass (m)
Momentum (m²/s)
|Momentum (m²/s)
Kinetic energy (m³/s²)
|Kinetic energy (m³/s²)
Potential energy (m³/s²)
t (h)
Reference simulation
CNN prediction
GE-CNN prediction