ASME Journal of Offshore Mechanics and Arctic Engineering
Online journal at:
https://asmedigitalcollection.asme.org/offshoremechanics

Marlon S. Mathias[1]
Escola Politécnica - Universidade de São Paulo,
São Paulo, CEP 05508-080, Brazil
e-mail: marlon.mathias@usp.br

Caio F. D. Netto
Escola Politécnica - Universidade de São Paulo,
São Paulo, CEP 05508-080, Brazil
e-mail: caio.netto@usp.br

Felipe M. Moreno
Escola Politécnica - Universidade de São Paulo,
São Paulo, CEP 05508-080, Brazil
e-mail: felipe.marino.moreno@usp.br

Jefferson F. Coelho
Escola Politécnica - Universidade de São Paulo,
São Paulo, CEP 05508-080, Brazil
e-mail: jfialho@usp.br

Lucas P. de Freitas
Escola Politécnica - Universidade de São Paulo,
São Paulo, CEP 05508-080, Brazil
e-mail: lfreitasp2001@usp.br

Marcel R. de Barros
Escola Politécnica - Universidade de São Paulo,
São Paulo, CEP 05508-080, Brazil
e-mail: marcel.barros@usp.br

Pedro C. de Mello
Escola Politécnica - Universidade de São Paulo,
São Paulo, CEP 05508-080, Brazil
e-mail: pcmello@usp.br

Marcelo Dottori
Instituto Oceanográfico - Universidade de São Paulo,
São Paulo, CEP 05508-120, Brazil
e-mail: fgcozman@usp.br

Fábio G. Cozman
Escola Politécnica - Universidade de São Paulo,
São Paulo, CEP 05508-080, Brazil
e-mail: mdottori@usp.br

Anna H. R. Costa
Escola Politécnica - Universidade de São Paulo,
São Paulo, CEP 05508-080, Brazil
e-mail: anna.reali@usp.br

A Physics-Informed Neural Operator for the Simulation of Surface Waves

We develop and implement a neural operator (NOp) to predict the evolution of waves on the surface of water. The NOp uses a graph neural network (GNN) to connect randomly sampled points on the water surface and exchange information between them to make the prediction. Our main contribution is adding physical knowledge to the implementation, which allows the model to be more general and able to be used in domains of different geometries with no retraining. Our implementation also takes advantage of the fact that the governing equations are independent of rotation and translation to make training easier. In this work, the model is trained with data from a single domain with fixed dimensions and evaluated in domains of different dimensions with little impact to performance. [DOI: 10.1115/1.4064676]

Keywords: neural networks, machine learning, computational fluid dynamics, wave modeling

[1]Corresponding author.

Contributed by the Ocean, Offshore, and Arctic Engineering Division of ASME for publication in the Journal of Offshore Mechanics and Arctic Engineering. Manuscript received July 14, 2023; final manuscript received January 22, 2024; published online February 26, 2024. Assoc. Editor: Ying Min Low.

Alberto C. Nogueira Junior
IBM Research,
São Paulo, CEP 04007-900, Brazil
e-mail: albercn@br.ibm.com

Edson S. Gomi
Escola Politécnica - Universidade de São Paulo,
São Paulo, CEP 05508-080, Brazil
e-mail: gomi@usp.br

Eduardo A. Tannuri
Escola Politécnica - Universidade de São Paulo,
São Paulo, CEP 05508-080, Brazil
e-mail: eduat@usp.br

1 Introduction

Fluid simulation is a notoriously complex and expensive task; it is, however, an essential step in the design and validation of offshore structures. In this work, we develop a physics-informed machine learning (PIML) model that can predict the temporal evolution of waves on the surface of water. The concept behind PIML is to combine data-driven machine learning (ML) models to physics-driven numerical simulations to benefit from the best characteristics of each methodology. We develop a neural operator (NOp) [1] that maps a snapshot of the water surface at a given instant to a prediction of its state in the future.

ML algorithms are often trained using data that are available for a certain situation. Relying solely on data to train the ML model often requires enormous datasets, which may or may not be available. By adding physical knowledge to the model, the total amount of data required may decrease dramatically [2]. Several means of mixing physical knowledge and data-based models are available [3], for instance, a physics-informed neural network can be used to solve a partial differential equation (PDE) even when no solution data are available, all it needs is the PDE itself in residual form and its boundary and initial conditions [4]. ML models can also be used to learn the biases and correct the outputs of physics-based solvers [5,6]. In our implementation, the NOp described by Li et al. [1] will be extended to include physical information on the boundary conditions, as well as to account for the rotational and translational invariance of the governing physics. Similarly, Baddoo et al. [7] have demonstrated that a physics-informed dynamic mode decomposition, which uses information such as translational invariance and symmetry, can outperform standard dynamic mode decomposition.

We apply this model to a virtual tank inspired by the hydrodynamic calibrator (HC) water test tank at the Numerical Offshore Tank (TPN) facilities at the University of São Paulo (USP) [8]; nonetheless, the model is intended to be general and could be easily adapted to other scenarios. The model is resolution-independent, and rotation and translation agnostic, which means that, once trained, it can be used for simulations of other grid resolution and even in tanks of different sizes and shapes.

We intend to build an operator that learns the mapping between two functions. Given an input function f, it will approximate an output function g. This type of operation has several practical uses besides the one illustrated in this paper. One such application is the solution of PDEs, which are found in a large array of physics and engineering scenarios, and often require long and expensive simulations, such as the Navier–Stokes equations, which are PDEs that govern the motion of fluids. Reference [1], for example, sets the forcing term of a PDE as f and its solution as g. The reader is referred to Refs. [9,10] for further reading on graph neural networks (GNNs) and on neural operators, use cases, and how they compare to other forms of machine learning.

The NOp works by distributing randomly sampled points along the domain and connecting them via a GNN. Initially, each node of the graph represents the current state of the tank at its location, in an encoded state. The edges connecting neighboring nodes contain encoded information on the physical distance between them. The GNN works by exchanging messages between the neighboring nodes; each exchange depends on the information contained in the edge, and updates the values contained in the nodes [11,12]. This message passing cycle is repeated a number of times; after which, the information in each node is decoded to the future state of the water surface. Both the kernel used for message passing and the encoding and decoding phases contain trainable parameters. Reference [12] shows that a GNN can perform image recognition based on point clouds, in contrast to convolutional neural networks (CNNs), that require structured grids. This illustrates the capability of GNNs when dealing with domains that require unstructured meshes. The Fourier neural operator described in Ref. [13] is an efficient alternative to the GNN-based NOp described in this work, but it requires regular domains.

The main contribution of this work is to implement the NOp in a way that leverages physical knowledge of the problem to create more general models, which are also easier to train. In our case, the physical problem is agnostic to rotation and translation; therefore, creating a model that shares this characteristics can dramatically increase its flexibility and ease its training when compared to models that are dependent on the absolute position on the body of water, as the model does not have to learn these characteristics itself, neither adapt to every possible combination of translation and rotation.

Furthermore, by adding physical knowledge on the boundary conditions, the model can be reused for domains of various geometries without retraining. All boundaries in this case are considered as fixed walls, which, in terms of wave propagation, act virtually the same as a symmetry condition. In our implementation, we modify the graph to include this information by connecting graph nodes that are close to the walls to both real nodes, inside the domain, and to phantom nodes, which lie outside of the domain, but perfectly mirror the state inside of it. We illustrate the capabilities of our model by training it with data from a square tank and are inferring it in (i) the same square tank, (ii) a rectangular tank, (iii) a larger square tank, and (iv) a triangular tank. These shapes were chosen due to the possibility of generating reference solutions for them via the process described in Sec. 2.5.

This paper is organized as follows: In the next section, we describe our test scenario, the implementation of the NOp, the available hyperparameters, the graph generation process, and the computation of training and validation data. In Sec. 3, we present NOp results. Finally, in Sec. 4, we summarize our findings, and discuss other uses for this implementation of the NOp.

2 Methods

The HC-TPN is a square water tank, 14.04 m wide and 4.1 m deep. It has 148 flaps along its sides that can be actuated to generate waves of various shapes for the simulation of floating structures. In this work, we use a tank of the same dimensions and assume that the flaps are static.

Our goal is to train the NOp to take a snapshot of water surface at a given time and to output a prediction of this surface at a later instant. Two variables are needed to fully describe the state of the water surface, its height (η) and its vertical velocity (η_t); therefore, we define $f(x,\ y) = `\eta(t=t_1,\ x,\ y),\ \eta_t(t=t_1,\ x,\ y)'^T$ and $g(x,\ y) = `\eta(t=t_2,\ x,\ y),\ \eta_t(t=t_2,\ x,\ y)'^T$. The NOp will be trained to map between functions f and g, which describe the water surface at instants t_1 and t_2, respectively.

The code is implemented in PYTHON, using the PyTorch framework. It is available on GitHub.[2]

2.1 Implementation of the Neural Operator.

The neural operator is based on a GNN. This implementation is based on the work of Li et al. [1]. The GNN operates on a graph that is placed on the physical domain, in which each vertex represents a point in space and contains information on its physical state, and each edge contains information on the geometry of the domain regarding both connecting vertices. Initially, the information encoded in each vertex represents the input function f; after a set of operations, the nodes begin representing the values of the output function g. The NOp is built and trained in such a way that it is independent of the graph topology; therefore, one could train a NOp in a certain set of graphs, each with a correspondent vertex density in space (akin to the mesh resolution), and evaluate it in a different set of graphs, of varying vertex densities.

There are three stages in the evaluation of a NOp. In the first, the physical variables (f_i) in the domain at each vertex (i) are linearly transformed to a set of n internal states ($v_{i,0}$) by a trainable matrix P and a trainable vector p.

$$v_{i,0} = Pf_i + p \tag{1}$$

The second stage in evaluating the NOp is convolution. This step is repeated a fixed number of times, which represents the depth of the model, D. The internal state of each node i, $v_{i,j}$, is updated to $v_{i,j+1}$ by the following equation:

$$v_{i,j+1} = \sigma\left(Wv_{i,j} + \frac{1}{|N_i|}\sum_{k\in N_i}\kappa(e_{i,k})v_{k,j}\right) \tag{2}$$

where W is a trainable weight matrix, and N_i is the neighborhood of node i, i.e., the set of nodes that have an edge between itself and i. σ is an activation function, in our case, ReLU. κ is the convolution kernel, which is given by a trainable fully connected multilayer perceptron (MLP) that takes $e_{i,k}$ as input and outputs a $n\times n$ matrix. $e_{i,k}$ is the information contained in the edge between vertices i and k.

Finally, in the third stage, the internal state of each node i is linearly transformed by a matrix Q and a vector q to the physical variables of the output function:

$$g_i = Qv_{i,J} + q \tag{3}$$

In our implementation, each vertex holds two scalar values: η and η_t, which are, respectively, the surface height and its temporal derivative. Both values are enough to fully describe the snapshot of the water surface. Each edge contains a single scalar, D, which is the distance between both vertices it connects to. In this case, as there are no direction-dependent variables, such as horizontal velocities, there is no need to store the edges' orientation in any form, either as an angle or as components D_x and D_y.

A GNN can be thought of as a generalized version of a CNN. In a CNN, the internal states are laid in a matrix that is then convoluted with a kernel matrix at each step. The internal states matrix often represents the spatial relationship between points in the physical domain, i.e., adjacent values in the matrix represent physical points next to each other. One could use a GNN to operate at this same grid of points using each point as a vertex of the graph and defining its neighborhood as the adjacent vertices, as illustrated in Fig. 1. In this figure, a 3×3 kernel is used in the CNN, which translates to a graph where each vertex has a neighborhood containing a 3×3 grid around itself. The kernel matrix is replaced by a function κ. The CNN's kernel matrix has nine trainable weights, which the GNN replaces by an MLP. After training, this MLP should take the information in each edge and output the corresponding weight; for example, $\kappa(e_{i,n1})$ should output $k_{1,1}$. By representing space by a matrix, one is restricted to using physical points that are laid in a grid of equally spaced points. In case of the grid resolution changes, the CNN must be retrained. On the other hand, a graph has no such constraints; therefore, the points can be spread in the domain in a non-uniform manner, and there is no need for retraining in case the graph changes. This ability to deal with unstructured meshes is especially useful for more complex geometries, which are more easily represented by such meshes.

2.2 Boundary Conditions.

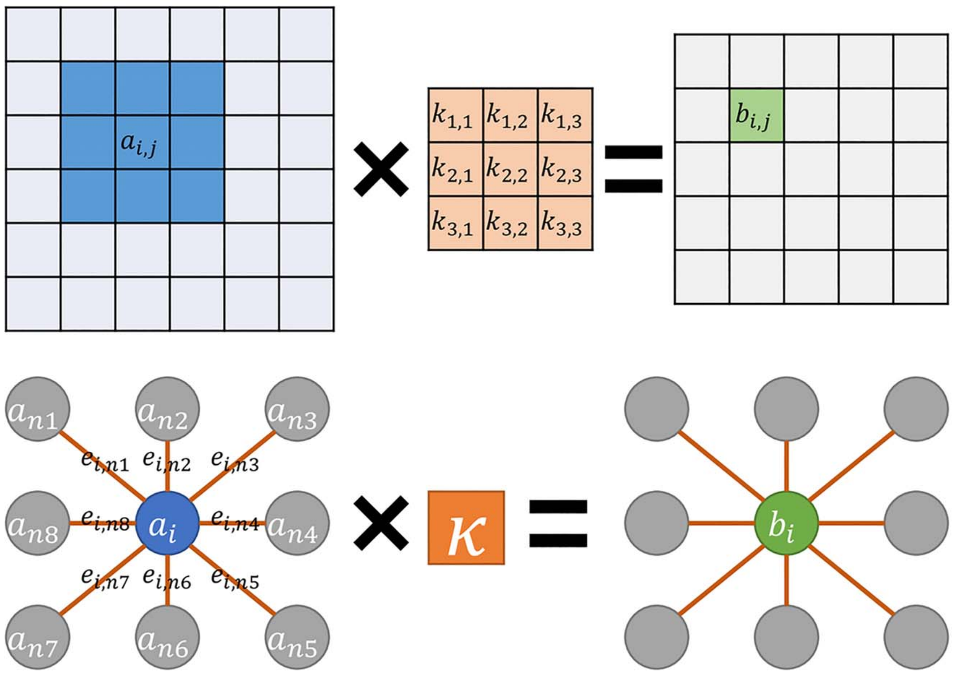
In the work of Li et al. [1], they have added the coordinates $(x,\ y)$ of each vertex to the input function f of the GNN. This was needed to convey information on the proximity to the boundary conditions, so the NOp would learn to account for their effect on the output function g. In this work, we want to take advantage of the fact that the governing equations of this system are translation and rotation invariant; therefore, we do not include any information regarding the coordinates in the GNN. This greatly eases the training, as the model will not need to learn the translational and rotational invariance by itself. Nonetheless, the boundary conditions still need to be accounted for in the solution. This is done by modifying the graph near the boundaries to emulate the physical effects of the boundary conditions.

All boundaries in our domain are defined as walls, which impose Neumann boundary conditions on both physical variables. Physically, this means that incoming waves are fully reflected at the walls, which is equivalent to having a mirrored version of the domain across the boundary. Therefore, by placing a mirrored copy of the graph across each boundary, one can impose the correct boundary conditions for this scenario. Naturally, this is recursive to each copy of the graph, which requires new copies across each of its boundaries, resulting in an infinite number of copies of the original graph. Fortunately, there is no need to store each of these infinite copies in memory, as they contain the same nodes and edges, each with the same internal states.

The implementation of these boundary conditions is done by manipulating the edges of vertices near the boundaries. Each vertex i has a neighborhood N_i, which contains nearby vertices, including itself. N_i can contain both vertices in the original domain or in any of the mirrored domains. From now on, vertices in any of the mirrored domains will be referred to as phantom vertices, while vertices in the original domain will be the real vertices. Each edge, besides its internal state, also carries the indices of both vertices connecting to it. In case an edge connects to a phantom vertex, it will contain the index of the corresponding real vertex, which holds the same internal state.

This is illustrated in Fig. 2. In the figure, we have marked two vertices in the real domain, 1 and 2; we have also marked their phantom counterparts in the mirrored domain, $1'$ and $2'$. Some edges connecting to vertex 1 have been marked A, B, and C. Edge A connects vertices 1 and 2, both real; this edge will contain the geometrical information between vertices 1 and 2, namely their distance, and will also hold the indices of both vertices. On the other hand, edge B connects vertices 1 and $2'$, one on each side of the boundary, which means that the distance embedded to this edge will be that between vertices 1 and $2'$; however, instead of holding the index of vertex $2'$ as one of its ends, this edge will hold that of vertex 2; therefore, there is no need to store the internal state of vertex $2'$, as this information is already held by vertex 2.

[2]https://github.com/C4AI/neural_operator_surface_waves



$a_{i,j}$
$k_{1,1}$ $k_{1,2}$ $k_{1,3}$
$k_{2,1}$ $k_{2,2}$ $k_{2,3}$
$k_{3,1}$ $k_{3,2}$ $k_{3,3}$
$b_{i,j}$
a_{n1} a_{n2} a_{n3} a_{n4} a_{n5} a_{n6} a_{n7} a_{n8}
$e_{i,n1}$ $e_{i,n2}$ $e_{i,n3}$ $e_{i,n4}$ $e_{i,n5}$ $e_{i,n6}$ $e_{i,n7}$ $e_{i,n8}$
a_i
κ
b_i

Fig. 1 Comparison between a convolutional neural network and a graph neural network

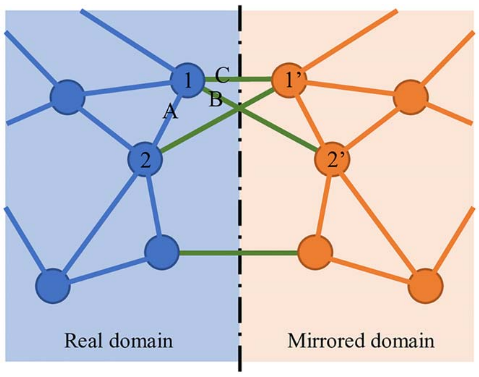


Fig. 2 Illustration of the graph near the domain boundaries. The region to the left shows the real domain, while the region to the right shows its mirrored version.

Similarly, edge C will hold the distance between vertices 1 and 1′ but will have both ends connected to the index of vertex 1.

By building the graph in this manner, the reflective boundary conditions are always met, and there is no need to embed the absolute coordinates of vertices or edges in the graph, which causes the translational independence of the governing equations to be met as well. This implementation does not require any changes to the convolution subroutine.

2.3 Graph Generation. Each function f and g must be represented by a graph in this implementation. These graphs are generated by the following procedure: N points are randomly placed on the domain, each becoming a vertex of the graph. The neighborhood of a vertex is constructed by placing edges connecting it to all other vertices within a certain radius R_n of it, including itself. Phantom nodes are also considered when building the neighborhoods, as explained in the previous subsection.

2.4 Hyperparameters. There are two groups of hyperparameters that must be chosen prior to training the NOp. The first group is relative to the structure of the NOp itself and has four items:

- W—model width: the amount of internal states in each graph node;
- D—model depth: number of convolutions in a forward pass of the GNN;
- K_W—kernel width: the number of neurons in each hidden layer of the kernel MLP;
- K_D—kernel depth: number of hidden layers of the kernel MLP.

The second group relates to the training of the NOp and has the following items:

- $N_{\min}$—minimum number of vertices in graphs used for training;
- $N_{\max}$—maximum number of vertices in graphs used for training;
- R_n—radius of the neighborhood of each vertex;
- LR—learning rate;
- S_n—scheduler step—number of epochs before reducing LR;
- S_γ—scheduler ratio—reduction factor of LR every S_n epochs;
- P—patience: number of epochs without improvement for early stopping.

2.5 Reference Solution. We assume that the water surface follows the wave equation for superficial waves propagation; therefore, the surface elevation η evolves according to

$$\frac{\partial^2\eta}{\partial t^2} = c^2\left(\frac{\partial^2\eta}{\partial x^2} + \frac{\partial^2\eta}{\partial y^2}\right) \tag{4}$$

with t representing time, and x and y, the spatial coordinates. c is the wave propagation velocity, which varies with the wavelength relation to the water depth following the relation:

$$c^2 = \frac{g}{k}\tanh(kh) \tag{5}$$

where g is the acceleration of gravity, h is the water depth, and k is the wave number, which depends on the wavelength λ:

$$k = \frac{2\pi}{\lambda} \tag{6}$$

When h is small relative to λ, c becomes constant with value $c = \sqrt{gh}$, which is known as the shallow-water approximation. Nonetheless, in this case, we are dealing with values of h and λ that may be in the same order of magnitude, which causes c to vary considerably for each wavelength.

In this paper, we assume η is very small compared to h, and that all linear relationships hold true. Therefore, we can use a mode separation technique to find the solution to $\eta(t, x, y)$. We separate η in a series of modes, each with a different value of λ, which causes c to be fixed within each mode; these solutions can be linearly combined to find a general solution to η. For simplification purposes, we will present the solution for the one-dimensional case, which is analogous to the two-dimensional case.

The 1D governing equation is given by

$$\frac{\partial^2 \eta}{\partial t^2} = c^2 \frac{\partial^2 \eta}{\partial x^2} \tag{7}$$

And the solution is assumed to be of the form

$$\eta(t, x) = \sum_{n=0}^{\infty} A_n(t) \cos\left(\frac{n\pi}{L} x\right) \tag{8}$$

where L is the domain length. We have specifically chosen this series of cosines because it always follows the boundary conditions, regardless of the values of A_n.

Substituting Eq. (8) in (7):

$$\frac{\partial^2}{\partial t^2} \sum_{n=0}^{\infty} A_n(t) \cos\left(\frac{n\pi}{L} x\right) = c^2 \frac{\partial^2}{\partial x^2} \sum_{n=0}^{\infty} A_n(t) \cos\left(\frac{n\pi}{L} x\right) \tag{9}$$

which can be separated in distinct equations:

$$\frac{\partial^2 A_n}{\partial t^2} = c_n^2 A_n \frac{\partial^2}{\partial x^2} \cos\left(\frac{n\pi}{L} x\right) \tag{10}$$

where c_n is the wave propagation velocity for each term n, which is fixed and given by

$$c_n^2 = \frac{gL}{n\pi} \tanh\left(\frac{n\pi}{L} h\right) \tag{11}$$

Substituting Eq. (11) in (10), we have that the governing equation becomes

$$\frac{\partial^2 A_n}{\partial t^2} = -\frac{gn\pi}{L} \tanh\left(\frac{n\pi}{L} h\right) A_n \tag{12}$$

which has the analytical solution

$$A_n(t) = A_{n,0} \cos(\alpha_n t + \phi_n) \tag{13}$$

with

$$\alpha_n^2 = g k_n \tanh(h k_n) \tag{14}$$

and

$$k_n = \frac{n\pi}{L} \tag{15}$$

$A_{n,0}$ and ϕ_n are constants that can be computed based on a known snapshot, such as the initial condition. In practice, this solution must be truncated to a finite series of N terms.

Analogously, the solution for the two-dimensional case is given by

$$\eta(t, x, y) = \sum_{m=0}^{\infty} \sum_{n=0}^{\infty} A_{m,n}(t) \cos\left(\frac{m\pi}{L_x} x\right) \cos\left(\frac{n\pi}{L_y} y\right) \tag{16}$$

with coefficients $A_{m,n}$ given by

$$A_{m,n}(t) = A_{m,n,0} \cos(\alpha_{m,n} t + \phi_{m,n}) \tag{17}$$

with

$$\alpha_{m,n}^2 = g k_{m,n} \tanh(h k_{m,n}) \tag{18}$$

and

$$k_{m,n} = \pi \sqrt{\left(\frac{m}{L_x}\right)^2 + \left(\frac{n}{L_y}\right)^2} \tag{19}$$

with L_x and L_y representing the lengths in x and y directions, respectively. Coefficients $A_{m,n,0}$ and $\phi_{m,n}$ must be computed with the available data, such as the initial conditions.

This solution can be verified by comparing it to the one presented by Mello et al. [8]. In their work, a pulse is introduced at the surface; its propagation happens at different velocities, depending on the frequency. Differently from the present work, for this experiment, their domain is considered infinitely large; therefore, no wave reflections are present.

Figure 3 compares the solution obtained by Mello et al. [8] (top) and ours (bottom). In the reference case, a pulse with a maximum frequency of $3Hz$ was introduced at time $t = 10s$ and location $x = 0$. The water depth is set to $h = 4.1m$. In the image, we can observe the dispersion of waves as longer wavelengths travel faster than shorter wavelengths. In our reproduction, we have taken a snapshot of the reference solution at $t = 40s$ and used it to compute the $A_{n,0}$ and ϕ_n coefficients for Eq. (13); with these values, the equation was used to reconstruct the flow in the $0 \le t \le 100$ interval.

Away from either end of the domain, the agreement between both solutions is excellent and can be down to the rounding error if enough terms are used in the series. Near the ends, the differences between both approaches become clear. For $t > 80s$, our solution shows the wave being reflected at the end of the domain, which does not happen in the reference. On the other hand, for $t < 10s$, our solution displays incoming waves that collapse in a peak at $t = 10s$. This happens because our solution does not account for external disturbances to the flow; therefore, it has continued to propagate the wave back in time. Nonetheless, as there will be no external disturbances added to the surface in this work and the reflections are the expected behavior of the boundary conditions, the solution presented in this section is adequate for our purposes.

2.6 Reference Data Generation. We have generated 100 pairs of f and g samples. Eighty of them are used for training and 20 for validation. The time delta between f and g is set to 1 s. Thirty percent of the dataset was generated with a Gaussian function as the initial height, given by

$$\eta(t = t_0, x, y) = e^{-\alpha\left[(x-x_0)^2+(y-y_0)^2\right]} \tag{20}$$

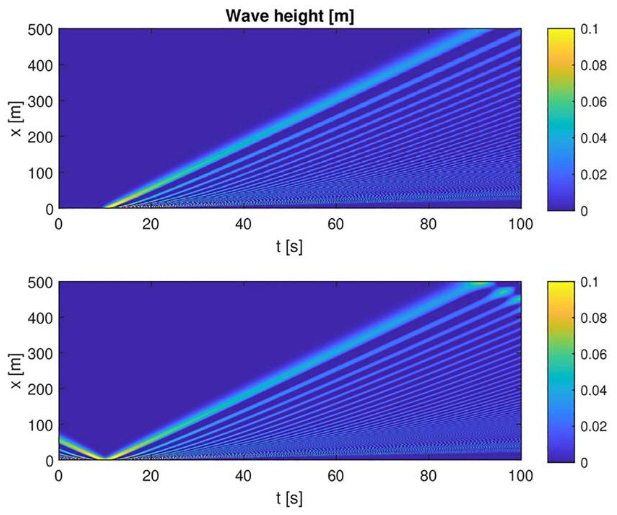


Fig. 3 Comparison between the solution by Mello et al. [8] (top) and ours (bottom) for a pulse introduced at $t = 10$ s

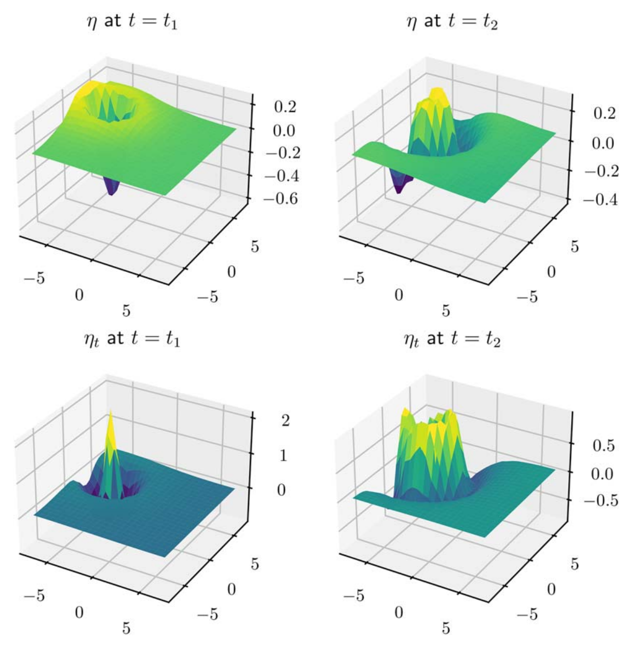


Fig. 4 Values of η (top) and η_t (bottom) at $t = t_1$ (left) and $t = t_2$ (right) for a randomly selected sample generated by a Gaussian function

where α is chosen at random in the range $0.1 \leq \alpha \leq 1$ and $(x_0,\ y_0)$ is a random point in the domain. $\eta_t(t = t_0,\ x,\ y)$ is null. An accommodation period of 1 s is used between t_0 and t_1. This is important to allow time for non-physical features present in the initial distribution to dissipate. Therefore, we compute the coefficients $A_{m,n}$ for Eq. (17) that follow Eq. (20) at $t = 0$, then, we use Eq. (17) to compute f at $t = 1s$ and g at $t = 2s$. Figure 4 shows the values of η and η_t for a randomly selected sample.

The rest of the dataset (70%) is generated by a random spectrum in the following manner: a grid of n_x by n_y points is generated and populated with random numbers, following a Gaussian distribution. Its two-dimensional Fourier transform is computed, and all coefficients above a certain threshold are set to zero, which is a form of a low-pass filter. The inverse Fourier transform is used to compute η and η_t at $t = 0$. Once more, an accommodation period is used to damp non-physical features from the domain. In this work, we have limited the number of modes to 5 in each direction. The accommodation period is set to 5 s. Therefore, snapshots for f and g are taken at $t = 5s$ and $t = 6s$, respectively. Figure 5 exemplifies η and η_t generated by this procedure.

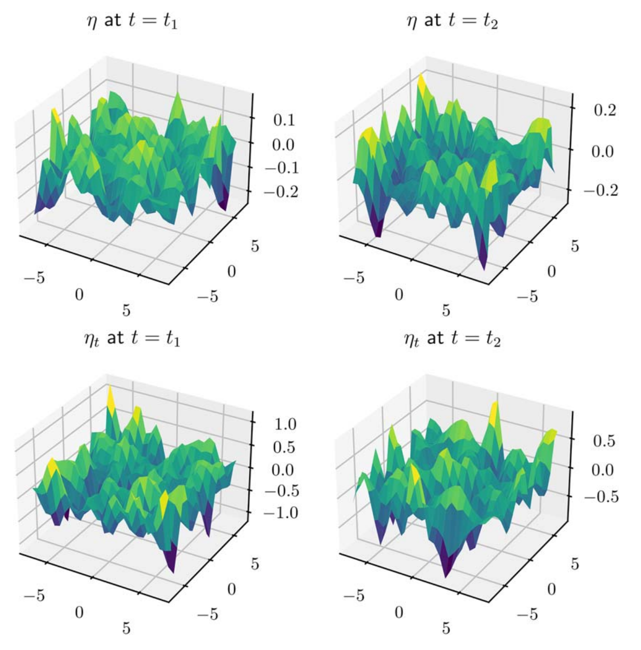


Fig. 5 Values of η (top) and η_t (bottom) at $t = t_1$ (left) and $t = t_2$ (right) for a sample generated by a random spectrum

2.7 Summary of the Neural Operator. The input to the system is a function f which describes the state of the water surface at some initial time. The output is g, a function that describes the water surface at a later time. Algorithm ?? summarizes the graph generation phase and the implementation of the neural operator.

Algorithm 1 Physics-informed neural operator

```
1: Place nodes in the physical domain
2: Compute position of phantom nodes                    ▷See Sec. 2
3: for $i$ in $0 \ldots n-1$                           ▷Loop through all nodes
4:     Compute neighborhood of each node
5:     Create edges linking the node to its neighborhood
6:     Compute the length of each edge
7: end for
8: Assign the physical variables of each node to $f_i = [\eta, \eta_t]^T$
9: Compute the internal state $v_{i,0}$ with Eq. (1)
10: for $j$ in $0 \ldots D-1$ do                      ▷Repeat for each convolution
11:     Perform the convolution with Eq. (2)
12: end for
13: Retrieve physical variables $g_i = [\eta, \eta_t]^T$ with Eq. (3)
```

Training is performed with known pairs of f and g generated as described in Secs. 2.5 and 2.6.

3 Results

3.1 Training and Hyperparameter Tuning. For training, the loss is defined as the mean square error (MSE) between the prediction and the ground truth. The Adam optimizer was used with a batch size of one. In Sec. 2.4, we present 11 hyperparameters; four of them are related to the NOp structure, while the other seven relate to the training procedure. For simplification, we have fixed the number of hidden layers in the kernel to two ($K_D = 2$), the scheduler step to $S_n = 100$, and the scheduler ratio to $S_\gamma = 0.9$. All other hyperparameters were tuned by the Weights and Biases tool [14] by means of its Bayesian optimization routine. A total of 295 different combinations of parameters were tried. They are summarized in Fig. 6. A subset with the best runs is highlighted; those are identified by a mean average error (MAE) in the validation set below 0.04 and a training time below 1 h. Two of the hyperparameters were not optimized directly: the LR was optimized via its $\log_{10}$ so that the distribution became exponential; and the minimum number of graph nodes was defined as a ratio of the maximum number. This was done to avoid invalid cases with $N_{\min} > N_{\max}$.

These results show that some hyperparameters are key to obtaining optimal results, while others are much more lenient. A learning rate above $10^{-2.4}$ would always lead to sub-optimal results; similarly, most of the runs in the optimal range had a patience of, at least, 70. In the graph generation phase, a radius close to $3m$ was clearly favored, this is likely due to the fact that our validation dataset was constructed with a radius of $3m$. For the number of vertices in the training graphs, the optimal training datasets contain values of $N_{\max}$ between 1789 and 2801, and $N_{\min}$ between 285 and 1209; differently from the neighborhood radius hyperparameter, the optimum number of vertices for training did not closely reproduce the values used to create the validation dataset, which had graphs between 500 and 3000 vertices.

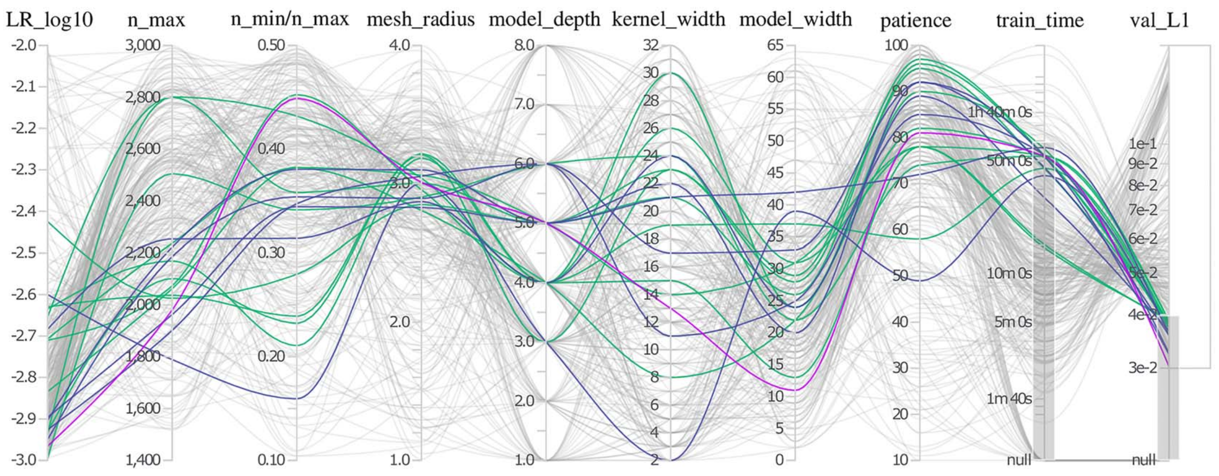


Fig. 6 Summary of the hyperparameter optimization

The model depth of the optimal results would usually be either 4 or 5, while the model width was usually between 20 and 30. The kernel width was not found to be a strong predictor for model quality, as most possible values are represented within the subset of models with best results.

For the rest of the paper, we have selected the model generated by the following set of parameters, which has produced the best results in the sweep under 1 h: model width $W = 11$, model depth $D = 5$, kernel width $K_W = 13$, and kernel depth $K_D = 2$. The dataset for training was composed of graphs between $N_{\min} = 888$ and $N_{\max} = 1981$ vertices, with a neighborhood radius of $R_n = 3m$. The training was performed with a learning rate $\mathrm{LR} = 1.1 \cdot 10^{-3}$, which is multiplied by the scheduler ratio $S_\gamma = 0.9$ every $S_n = 100$ epochs; the patience was set to 81, meaning that training halts if 81 epochs pass with no improvement on the loss. A total of 1300 epochs were used in training, which took 53 min in our NVIDIA GeForce RTX3090 GPU, allocating a maximum of 13 GB of memory. The tuning process took 4 days to complete using a workstation with two NVIDIA GeForce RTX3090 GPUs.

3.2 Physical Interpretation of Hyperparameters.

Some of the hyperparameters can be directly related to physical quantities in the domain. The number of vertices in the graphs, given by the range between $N_{\min}$ and $N_{\max}$, directly corresponds to the discretization of the continuous functions f and g. Larger numbers, corresponding to finer meshes, can depict the snapshots of the surface more accurately.

The wave equation assumes that information in the domain propagates at a given velocity, denoted by c, as defined in Eq. (5). Therefore, for propagating the solution by a time $\Delta t = t_2 - t_1$, each point in the function can only be influenced by its vicinity with a radius given by $\Delta t c$. Equation (5) shows that the maximum velocity a wave can propagate in this scenario is given by $c_{\max} = \sqrt{gh}$, in the limit of the wavenumber k going to zero, i.e., for very long waves. This means that, to have complete knowledge of the state of a point at time t_2, one must know a solution field of radius $r = \Delta t\sqrt{gh}$ around it at time t_1. With a depth of $4.1m$ and $\Delta t = 1s$, this radius becomes $r = 6.34m$. In the NOp, information is propagated spatially by each convolution of the graph, when each vertex exchanges information with its neighbors. The convolution happens D times during the forward pass, and the radius of the neighborhood is given by R_n; therefore, information can travel, at most, a distance of DR_n in the forward pass of the model. This means that to have enough information to accurately predict the solution Δt in the future, the following relation must hold:

$$DR_n \geq \Delta t\sqrt{gh} \tag{21}$$

For the hyperparameters chosen, the product $DR_n = 15m$, which is well above the threshold of $6.34m$.

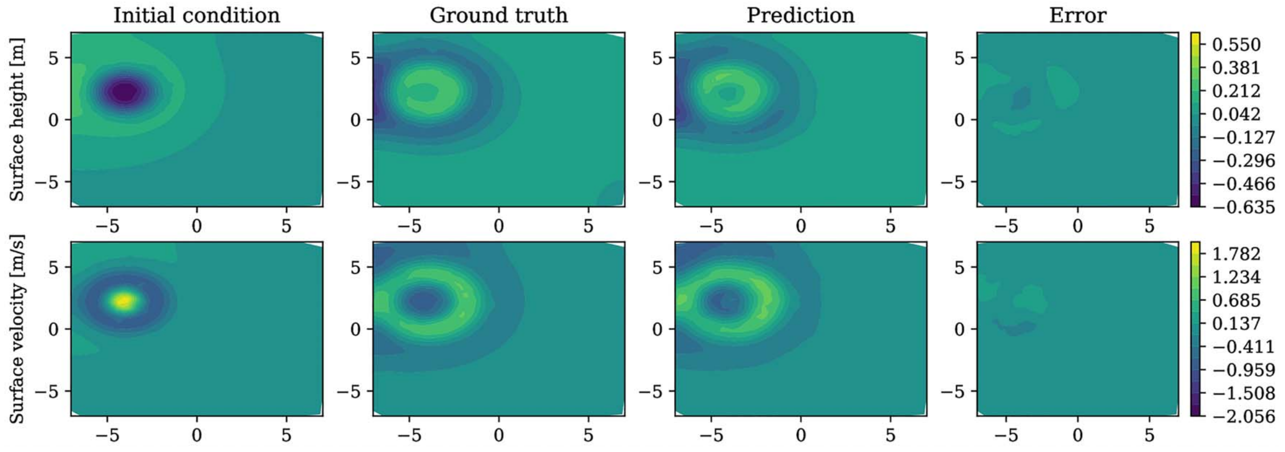


Fig. 7 Neural operator results for a Gaussian initial condition: (far-left) the initial condition, (center-left) the ground truth, (center-right) the model prediction, and (far-right) the prediction error for (top) the surface height η and (bottom) the surface vertical velocity η_t

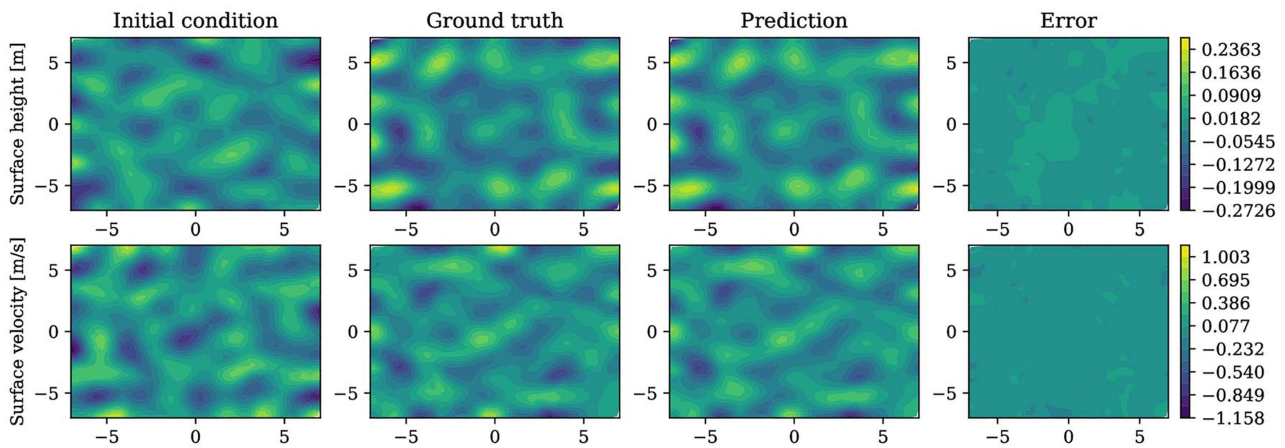


Fig. 8 Neural operator results for a random initial condition: (far-left) the initial condition, (center-left) the ground truth, (center-right) the model prediction, and (far-right) the prediction error for (top) the surface height η and (bottom) the surface vertical velocity η_t

Table 1 Error metrics for four different tanks

Size (m) (vertices)	Condition	MSE	MAE	IOA
14.04 × 14.04	Gauss	2.8×10^{-4}	1.0×10^{-2}	99.6%
(2000)	Random	3.4×10^{-4}	1.4×10^{-2}	99.0%
10 × 30	Gauss	8.3×10^{-4}	1.8×10^{-2}	98.9%
(2000)	Random	2.6×10^{-4}	1.2×10^{-2}	99.2%
50 × 50	Gauss	2.0×10^{-4}	6.8×10^{-3}	97.4%
(7000)	Random	6.9×10^{-4}	1.9×10^{-2}	98.2%
14.04 (triangular)	Gauss	5.1×10^{-4}	1.6×10^{-2}	98.4%
(1000)	Random	7.6×10^{-4}	2.0×10^{-2}	98.0%

Regarding the convergence at $R_n = 3m$ during the hyperparameter optimization, we have developed the following hypothesis: If the radius selected for training was shorter than 3 m, the Kernel's MLP would have to extrapolate when it encountered longer edges in the validation dataset, likely causing errors. On the other hand, if training was performed with radii larger than 3 m, the MLP would learn how to approximate the kernel for those larger edges; however, this information would not be needed during validation. Therefore, the additional difficulty of training for larger radii would bring no improvement on the validation accuracy.

Finally, we can also relate the hyperparameters of the convolution kernel, namely K_W and K_D, to the Green's function of the solution. The multilayer perceptron used as convolution kernel must be flexible enough to learn the Green's function of the solution, as demonstrated by Li et al. [1].

3.3 Approximations by the Neural Operator. In this subsection, we display predictions made by our model and compare them to the ground truth. First, we have selected a case from our validation dataset that was generated with a Gaussian initial condition. A graph with 2000 nodes was generated over the domain and used for predicting the surface state at a later time. Figure 7 shows (far-left) the initial condition, (center-left) the ground truth, (center-right) the model prediction, and (far-right) the prediction error for (top) the surface height η and (bottom) the surface vertical velocity η_t. There is great visual agreement between the prediction and the ground truth; the mean square error is $\text{MSE} = 2.8 \cdot 10^{-4}$, the mean average error is $\text{MAE} = 1.0 \cdot 10^{-2}$, and the index of agreement is $\text{IOA} = 99.6\%$. Similarly, Fig. 8 shows the prediction made for a random initial condition. The error metrics are similar, at $\text{MSE} = 3.4 \cdot 10^{-4}$, $\text{MAE} = 1.4 \cdot 10^{-2}$, and $\text{IOA} = 99.0\%$. Note that all errors are computed after normalizing the output values in a range between 0 and 1.

To test the capabilities of our model to adapt to new domains with no retraining, we have generated three new datasets, one for a rectangular tank with dimensions $10m \times 30m$; one square, with side $50m$; and one right-triangular tank, with $14.04m$ legs. A total of 2000 graph vertices were used for the modeling of the rectangular tank, 7000 vertices were used for the larger square tank, and 1000

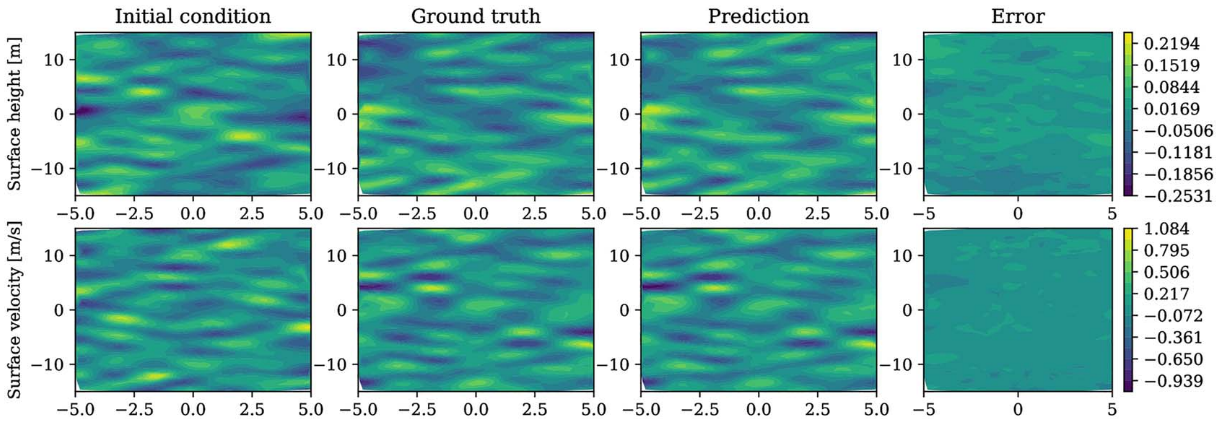


Fig. 9 Neural operator results for a random initial condition in the $10m \times 30m$ tank: (far-left) the initial condition, (center-left) the ground truth, (center-right) the model prediction, and (far-right) the prediction error for (top) the surface height η and (bottom) the surface vertical velocity η_t

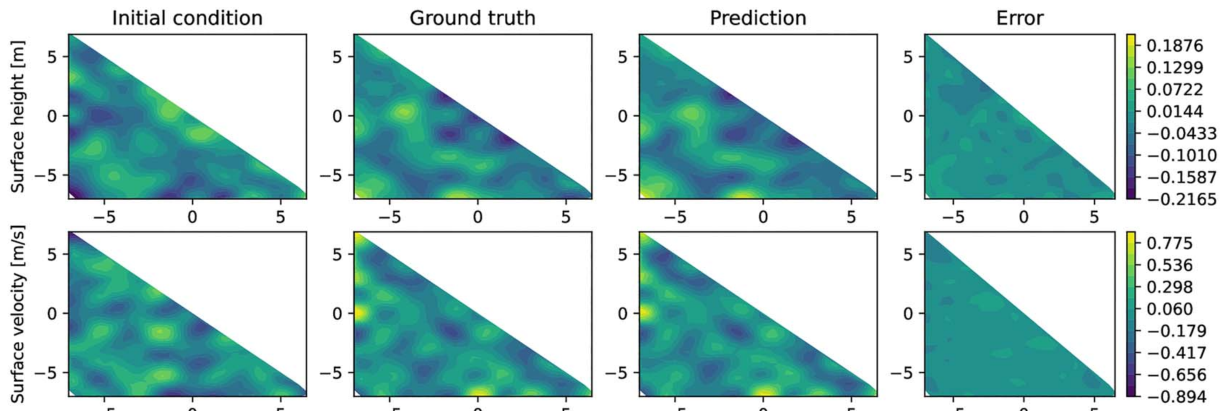


Fig. 10 Neural operator results for a random initial condition in the triangular tank: (far-left) the initial condition, (center-left) the ground truth, (center-right) the model prediction, and (far-right) the prediction error for (top) the surface height η and (bottom) the surface vertical velocity η_t

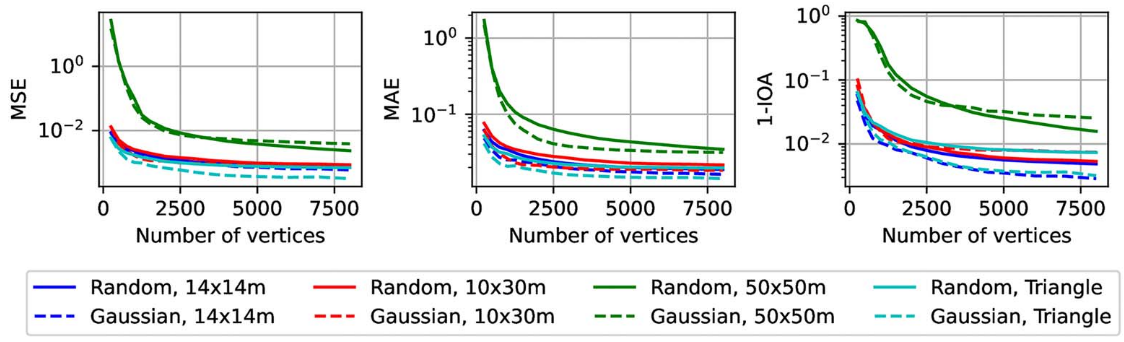


Fig. 11 MSE, MAE, and IOA of the model with graphs of various numbers of vertices for the $14.04m \times 14.04m$, $10m \times 30m$, $50m \times 50m$, and $14.04m \times 14.04m$ triangular tanks

vertices for the triangular tank. Table 1 shows the error metrics for all four tanks tested for both types of initial conditions. The predictions for the rectangular and triangular tanks with a random initial condition are shown in Figs. 9 and 10.

We have evaluated our model using graphs of various numbers of vertices to check how the performance of the model improves as the graph carries more information on the domain. Figure 11 summarizes MSE, MAE, and IOA for each of the three test domains, using both Gaussian and random initial conditions. The values in the figure are averaged from ten different graphs for each number of vertices. As the domain size increased, more vertices were needed to accurately represent the flow state in each snapshot, which directly impacted the model's performance.

4 Discussion

We have developed a novel PIML model that uses a GNN-based neural operator with physics-based boundary conditions and rotational and translational independence to simulate surface waves. The main contribution of our implementation is its ability to be used in other domains with no retraining. This is illustrated here by training the model with data from a single geometry and inferring it in different geometries.

By using physical knowledge to model the boundary conditions, the NOp does not need to learn about them. Moreover, removing the information regarding the boundary conditions from the NOp allows it to be invariant to translation and rotation, easing the training process. In this work, we have only used rectangular and triangular domains, as our analytical solution is only valid in this geometry for comparison purposes; nonetheless, in the future, we believe that the model could be extended to domains of different shapes, as long as the graph generating routine is adapted to follow the procedure illustrated in Fig. 2, although it is still unclear if this technique could be extended to curved boundaries.

In this work, we use one snapshot of the present state of the tank to predict a future state. By running the model in an autoregressive manner, it may be possible to generate predictions for a time series, although the numerical stability of this procedure is still unknown and must be subject to future investigation.

Acknowledgment

The authors would like to thank the Center for Artificial Intelligence (C4AI-USP), with support by the São Paulo Research Foundation (FAPESP) under Grant No. 2019/07665-4 and by the IBM Corporation. The authors would also like to thank the *Programa de Bolsas Itaú* (PBI) of the *Centro de Ciência de Dados* (C^{2}D), supported by *Itaú Unibanco S.A.* This work is also supported in part by FAPESP under Grant No. 2020/16746-5, the Brazilian National Council for Scientific and Technological Development (CNPq) under Grant Nos. 310085/2020-9, 310127/2020-3, and 305753/2022-3, and the Coordination for the Improvement of Higher Education Personnel (CAPES, Finance Code 001). The authors acknowledge ANP/Petrobras for sustaining long-term support in the development of the TPN-USP Hydrodynamic Calibrator (SIGITEC2021/00236-0).

Data Availability Statement

The data and information that support the findings of this article are freely available.[3]

[3]See Note 2.

References

[1] Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., and Anandkumar, A., 2020, "Neural Operator: Graph Kernel Network for Partial Differential Equations, 2003.03485," arXiv:2003.03485 [cs, math, stat].

[2] Karniadakis, G. E., Kevrekidis, I. G., Lu, L., Perdikaris, P., Wang, S., and Yang, L., 2021, "Physics-Informed Machine Learning," Nat. Rev. Phys., **3**(6), pp. 422–440.

[3] Raissi, M., Perdikaris, P., and Karniadakis, G. E., 2019, "Physics-Informed Neural Networks: A Deep Learning Framework for Solving Forward and Inverse Problems Involving Nonlinear Partial Differential Equations," J. Comput. Phys., **378**(1), pp. 686–707.

[4] Sun, L., Gao, H., Pan, S., and Wang, J.-X., 2020, "Surrogate Modeling for Fluid Flows Based on Physics-Constrained Deep Learning Without Simulation Data," Comput. Methods Appl. Mech. Eng., **361**(1), p. 112732.

[5] Xu, T., and Valocchi, A. J., 2015, "Data-Driven Methods to Improve Baseflow Prediction of a Regional Groundwater Model," Comput. Geosci., **85**(Part B), pp. 124–136.

[6] Marino Moreno, F., Deberaldini Netto, C., Barros, M., Fialho Coelho, J., Freitas, L., Mathias, M., and Neto, L., 2022, "Enhancing the Forecast of Ocean Physical Variables through Physics Informed Machine Learning in the Santos Estuary, Brazil," OCEANS 2022, Chennai, India, Feb. 21–24.

[7] Baddoo, P. J., Herrmann, B., McKeon, B. J., Kutz, J. N., and Brunton, S. L., 2021, "Physics-Informed Dynamic Mode Decomposition," Proc. Math. Phys. Eng. Sci., **479**(2271), pp. 1–23.

[8] Mello, P. C., Pérez, N., Adamowski, J. C., and Nishimoto, K., 2016, "Wave Focalization in a Wave Tank by Using Time Reversal Technique," Ocean Eng., **123**(1), pp. 314–326.

[9] Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., and Sun, M., 2020, "Graph Neural Networks: A Review of Methods and Applications," AI Open, **1**(1), pp. 57–81.

[10] Kovachki, N., Li, Z., Liu, B., Azizzadenesheli, K., Bhattacharya, K., Stuart, A., and Anandkumar, A., 2022, "Neural Operator: Learning Maps Between Function Spaces With Applications to PDEs," J. Mach. Learn. Res., **24**(89), pp. 1–97.

[11] Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E., 2017, "Neural Message Passing for Quantum Chemistry," International Conference on Machine Learning, Sydney, Australia, Aug. 6.

[12] Simonovsky, M., and Komodakis, N., 2017, "Dynamic Edge-Conditioned Filters in Convolutional Neural Networks on Graphs," 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honululu, HI, July.

[13] 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," International Conference on Learning Representations, Vienna, Austria, May.

[14] Biewald, L., 2020, Experiment Tracking with Weights and Biases, https://wandb.ai/.