Rapid spatio-temporal flood modelling via hydraulics-based graph neural networks

Bentivoglio, Roberto; Isufi, Elvin; Jonkman, Sebastiaan Nicolas; Taormina, Riccardo

DOI
10.5194/hess-27-4227-2023

Publication date
2023

Document Version
Final published version

Published in
Hydrology and Earth System Sciences

Hydrol. Earth Syst. Sci., 27, 4227–4246, 2023
https://doi.org/10.5194/hess-27-4227-2023

Hydrology and Earth System Sciences
Open Access
EGU

Rapid spatio-temporal flood modelling via hydraulics-based graph neural networks

Roberto Bentivoglio[1], **Elvin Isufi**[2], **Sebastiaan Nicolas Jonkman**[3], **and Riccardo Taormina**[1]

[1]Department of Water Management, Faculty of Civil Engineering and Geosciences,
Delft University of Technology, Delft, the Netherlands
[2]Department of Intelligent Systems, Faculty of Electrical Engineering, Mathematics and Computer Science,
Delft University of Technology, Delft, the Netherlands
[3]Department of Hydraulic Engineering, Faculty of Civil Engineering and Geosciences,
Delft University of Technology, Delft, the Netherlands

Correspondence: Roberto Bentivoglio (r.bentivoglio@tudelft.nl)

Received: 19 February 2023 – Discussion started: 22 March 2023
Revised: 3 September 2023 – Accepted: 23 October 2023 – Published: 30 November 2023

Abstract. Numerical modelling is a reliable tool for flood simulations, but accurate solutions are computationally expensive. In recent years, researchers have explored data-driven methodologies based on neural networks to overcome this limitation. However, most models are only used for a specific case study and disregard the dynamic evolution of the flood wave. This limits their generalizability to topographies that the model was not trained on and in time-dependent applications. In this paper, we introduce shallow water equation–graph neural network (SWE–GNN), a hydraulics-inspired surrogate model based on GNNs that can be used for rapid spatio-temporal flood modelling. The model exploits the analogy between finite-volume methods used to solve SWEs and GNNs. For a computational mesh, we create a graph by considering finite-volume cells as nodes and adjacent cells as being connected by edges. The inputs are determined by the topographical properties of the domain and the initial hydraulic conditions. The GNN then determines how fluxes are exchanged between cells via a learned local function. We overcome the time-step constraints by stacking multiple GNN layers, which expand the considered space instead of increasing the time resolution. We also propose a multi-step-ahead loss function along with a curriculum learning strategy to improve the stability and performance. We validate this approach using a dataset of two-dimensional dike breach flood simulations in randomly generated digital elevation models generated with a high-fidelity numerical solver. The SWE–GNN model predicts the spatio-temporal evolution of the flood for unseen topographies with mean average errors in time of 0.04 m for water depths and $0.004\,m^2\,s^{-1}$ for unit discharges. Moreover, it generalizes well to unseen breach locations, bigger domains, and longer periods of time compared to those of the training set, outperforming other deep-learning models. On top of this, SWE–GNN has a computational speed-up of up to 2 orders of magnitude faster than the numerical solver. Our framework opens the doors to a new approach to replace numerical solvers in time-sensitive applications with spatially dependent uncertainties.

1 Introduction

Accurate flood models are essential for risk assessment, early warning, and preparedness for flood events. Numerical models can characterize how floods evolve in space and time, with the two-dimensional (2D) hydrodynamic models being the most popular (Teng et al., 2017). They solve a discretized form of the depth-averaged Navier–Stokes equations, referred to as shallow water equations (SWEs) (Vreugdenhil, 1994). Numerical models are computationally expensive, making them inapplicable for real-time emergencies and uncertainty analyses. Several methods aim to speed up the solution of these equations either by approximating them (Bates and De Roo, 2000) or by using high-performance computing and parallelization techniques (Hu et al., 2022;

Petaccia et al., 2016). However, approximate solutions are valid only for domains with low spatial and temporal gradients (Costabile et al., 2017), while high-performance computing methods are bound by the numerical constraints and the computational resources.

Data-driven alternatives speed up numerical solvers (Mosavi et al., 2018). In particular, deep learning outperforms other machine learning methods used for flood modelling in both speed and accuracy (Bentivoglio et al., 2022). Berkhahn et al. (2019) developed a multi-layer perceptron model for predicting urban floods given a rainfall event, achieving promising speed-ups and accuracy. Guo et al. (2021) and Kabir et al. (2020) developed convolutional neural networks (CNNs) for river flood inundation, while Jacquier et al. (2021) used deep learning to facilitate the reduced-order modelling of dam break floods and to provide uncertainty estimates. Also, Zhou et al. (2022) employed a CNN-based model to determine the spatio-temporal variation of flood inundation from a set of representative locations. These works explored the generalization of boundary conditions on a fixed domain. That is, they change the return period of the floods for a single case study, but they need retraining when applied to a new area, requiring more resources in terms of data, model preparation, and computation times.

To overcome this issue, the community is investigating the generalizability of deep-learning models to different study areas. Löwe et al. (2021) proposed a CNN model to estimate the maximum water depth of pluvial urban floods. They trained their model on part of their case study and then deployed it on the unseen parts, showing consistent performances. Guo et al. (2022) accurately predicted the maximum water depth and flow velocities for river floods in different catchments in Switzerland. To incorporate the variations in catchment size and shape, they divided the domain into patches. do Lago et al. (2023) proposed a conditional generative adversarial network that could predict the maximum water depth unseen rain events in unseen urban catchments. However, these approaches focus on a single maximum depth or velocity map, disregarding the dynamical behaviour. That is, no information is provided on the flood conditions over space and time, which is crucial for evacuation and the response to the flood.

To overcome this limitation, we propose SWE–GNN, a deep-learning model merging graph neural networks (GNNs) with the finite-volume methods used to solve the SWEs. GNNs generalize convolutional neural networks to irregular domains such as graphs and have shown promising results for fluid dynamics (e.g. Lino et al., 2021; Peng et al., 2022) and partial differential equations (e.g. Brandstetter et al., 2022; Horie and Mitsume, 2022). Hence, developing GNNs that follow the SWE equations is not only more physically interpretable but also allows better generalization abilities to unseen flood evolution, unseen breach locations, and unseen topographies. In particular, we exploit the geometrical structure of the finite-volume computational mesh by using its dual graph, obtained by connecting the centres of neighbouring cells via edges. The nodes represent finite-volume cells and edge fluxes across them. Following an explicit numerical discretization of the SWE, we formulate a novel GNN propagation rule that learns how fluxes are exchanged between cells, based on the gradient of the hydraulic variables. We set the number of GNN layers based on the time step between consecutive predictions, in agreement with the Courant–Friedrichs–Lewy conditions. The inputs of the model are the hydraulic variables at a given time, elevation, slope, area, length, and orientation of the mesh's cells. The outputs are the hydraulic variables at the following time step, evaluated in an auto-regressive manner. That is, the model is repeatedly applied using its predictions as inputs to produce extended simulations.

We tested our model on dike breach flood simulations due to their time-sensitive nature and the presence of uncertainties in topography and breach formation (Jonkman et al., 2008; Vorogushyn et al., 2009). Moreover, given the sensibility to floods in low-lying areas, fast surrogate models that generalize over all those uncertainties are required for probabilistic analyses. By doing so, our key contributions are threefold.

- We develop a new graph neural network model where the propagation rule and the inputs are taken from the shallow water equations. In particular, the hydraulic variables propagate based on their gradient across neighbouring finite-volume cells.
- We improve the model's stability by training it via a multi-step-ahead loss function, which results in stable predictions up to 120 h ahead using only the information of the first hour as initial hydraulic input.
- We show that the proposed model can serve as a surrogate for numerical solvers for spatio-temporal flood modelling in unseen topographies and unseen breach locations, with speed-ups of 2 orders of magnitude.

The rest of the paper is structured as follows. Section 2 illustrates the theoretical background; Sect. 3 describes the proposed methodology. In Sect. 4, we present the dataset used for the numerical experiments. Section 5 shows the results obtained with the proposed model and compares it with other deep-learning models. Finally, Sect. 6 discusses the results, analyses the current limitations of this approach, and proposes future research directions.

2 Theoretical background

In this section, we describe the theory supporting our proposed model. First, we discuss numerical models for flood modelling; then, we present deep-learning models, focusing on graph neural networks. Throughout the paper, we use the

standard vector notation, with a scalar, $\boldsymbol{a}$ vector, $\mathbf{A}$ matrix, and $\mathcal{A}$ tensor.

2.1 Numerical modelling

2.1.1 Shallow water equations

When assuming negligible vertical accelerations, floods can be modelled via the SWEs (Vreugdenhil, 1994). These are a system of hyperbolic partial differential equations that describe the behaviour of shallow flows by enforcing mass and momentum conservation. The two-dimensional SWE can be written as

$$\frac{\partial \boldsymbol{u}}{\partial t} + \nabla \mathbf{F} = \boldsymbol{s}, \tag{1}$$

with

$$\boldsymbol{u} = \begin{pmatrix} h \\ q_x \\ q_y \end{pmatrix}, \mathbf{F} = \begin{pmatrix} q_x & q_y \\ \frac{q_x^2}{h} + \frac{gh^2}{2} & \frac{q_x q_y}{h} \\ \frac{q_x q_y}{h} & \frac{q_y^2}{h} + \frac{gh^2}{2} \end{pmatrix},$$
$$\boldsymbol{s} = \begin{pmatrix} 0 \\ gh\left(s_{0x} - s_{\mathrm{f}x}\right) \\ gh\left(s_{0y} - s_{\mathrm{f}y}\right) \end{pmatrix}, \tag{2}$$

where $\boldsymbol{u}$ represents the conserved variable vector, $\mathbf{F}$ the fluxes in the x and y directions, and $\boldsymbol{s}$ the source terms. Here, h (m) represents the water depth, $q_x = uh$ ($\mathrm{m^2\,s^{-1}}$) and $q_y = vh$ ($\mathrm{m^2\,s^{-1}}$) are the averaged components of the discharge vector along the x and y coordinates, respectively, and g ($\mathrm{m\,s^{-2}}$) is the acceleration of gravity. The source terms in $\boldsymbol{s}$ depend on the contributions of bed slopes $\boldsymbol{s}_0$ and friction losses $\boldsymbol{s}_\mathrm{f}$ along the two coordinate directions.

2.1.2 Finite-volume method

The SWE cannot be solved analytically unless some simplifications are enforced. Thus, they are commonly solved via spatio-temporal numerical discretizations, such as the finite-volume method (e.g. Alcrudo and Garcia-Navarro, 1993). This method discretizes the spatial domain using meshes, i.e. geometrical structures composed of nodes, edges, and faces. We consider each finite-volume cell to be represented by its centre of mass, where the hydraulic variables h, q_x, and q_y are defined (see Fig. 1). The governing equations are then integrated over the cells, considering piece-wise constant variations. That is, the value of the variables at a certain time instant is spatially uniform for every cell. The SWE can be discretized in several ways in both space and time (e.g. Petaccia et al., 2013; Xia et al., 2017), but we focus on a first-order explicit scheme with a generic spatial discretization. For an arbitrary volume Ω_i and a discrete time step Δt, the SWE (Eq. 1) can be re-written as

$$\boldsymbol{u}_i^{t+1} = \boldsymbol{u}_i^t + \left(\boldsymbol{s}_i - \sum_{j=1}^{N_i} (\mathbf{F} \cdot \boldsymbol{n})_{ij} \frac{l_{ij}}{a_i} \right) \Delta t, \tag{3}$$

with $\boldsymbol{u}_i^t$ the hydraulic variables at time t and cell i, a_i the area of the ith cell, N_i the number of neighbouring cells, l_{ij} the length of the jth side of cell i, $\boldsymbol{s}_i$ the source terms, $\boldsymbol{n}_{ij} = [n_{xij}, n_{yij}]$ the outward unit normal vector in the x and y directions for side ij, and $(\mathbf{F} \cdot \boldsymbol{n})_{ij}$ the numerical fluxes across neighbouring cells.

In numerical models with explicit discretization, stability is enforced by satisfying the Courant–Friedrichs–Lewy (CFL) condition, which imposes the numerical propagation speed to be lower than the physical one (Courant et al., 1967). Considering v to be the propagation speed, the Courant number C can be evaluated as

$$C = \frac{v \Delta t}{\Delta x}, \tag{4}$$

where Δt and Δx represent the time step and the mesh size. This condition forces Δt to be sufficiently small to avoid a too-fast propagation of water in space that would result in a loss of physical consistency. Small time steps imply an increasing number of model iterations, which slow down numerical models over long time horizons. Deep learning provides an opportunity to accelerate this process.

2.2 Deep learning

Deep learning obtains non-linear high-dimensional representations from data via multiple levels of abstraction (LeCun et al., 2015). The key building blocks of deep-learning models are neural networks, which comprise linear and non-linear parametric functions. They take an input $\boldsymbol{x}$ and produce an estimate $\hat{\boldsymbol{y}}$ of a target representation $\boldsymbol{y}$ as $\hat{\boldsymbol{y}} = f(\boldsymbol{x}; \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ are the parameters (Zhang et al., 2021). The parameters are estimated to match predicted output with the real output by minimizing a loss function. Then, the validity of the model is assessed by measuring its performance on a set of unseen pairs of data, called the test set.

The most general type of neural network is a multi-layer perceptron (MLP). It is formed by stacking linear models followed by a point-wise non-linearity (e.g. rectified linear unit, ReLU, $\sigma(x) = \max\{0, x\}$). For MLPs, the number of parameters and the computational cost increase exponentially with the dimensions of the input. This makes them unappealing to large-scale high-dimensional data typical of problems with relevant spatio-temporal features such as floods. MLPs are non-inductive: when trained for flood prediction on a certain topography, they cannot be deployed on a different one, thus requiring a complete retraining. To overcome this curse of dimensionality and to increase generalizability, models can include inductive biases that constrain their degrees of freedom by reusing parameters and exploiting symmetries in the data (Battaglia, 2018; Gama et al., 2020; Villar et al., 2023). For example, convolutional neural networks exploit translational symmetries via filters that share parameters in space (e.g. LeCun et al., 2015; Bronstein et al., 2021). However, CNNs cannot process data defined on irregular meshes, which are

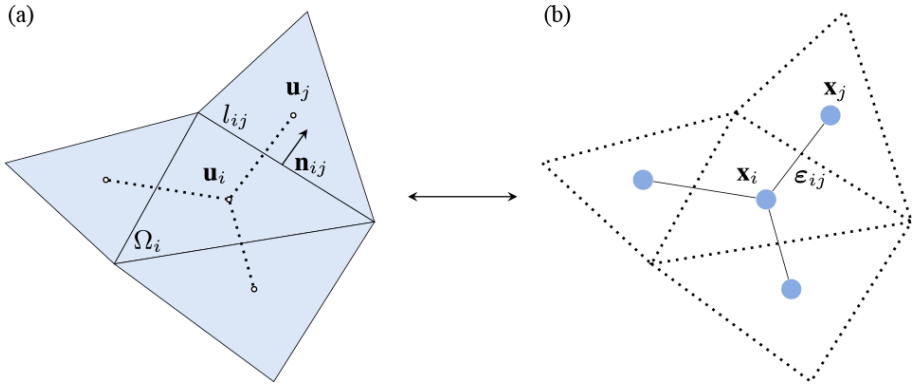


Figure 1. Schematic representation of an arbitrary triangular volume mesh and its dual graph. **(a)** A finite-volume cell Ω_i along with its neighbouring cells. Vectors $\boldsymbol{u}_i$ and $\boldsymbol{u}_j$ represent the cells' hydraulic variables, while l_{ij} and $\boldsymbol{n}_{ij}$, respectively, correspond to the length of the mesh side and the outward unit normal vector between cells i and j. **(b)** The dual graph of the mesh is obtained by considering each ith cell's centre as a node i, with features $\boldsymbol{x}_i$ and connecting neighbouring nodes, i and j, via edges ij, with features $\boldsymbol{\varepsilon}_{ij}$.

common for discretizing topographies with sparse details. Thus, we need a different inductive bias for data on meshes.

GNNs use graphs as an inductive bias to tackle the curse of dimensionality. This bias can be relevant for data represented via networks and meshes, as it allows these models to generalize to unseen graphs. That is, the same model can be applied to different topographies discretized by different meshes. GNNs work by propagating features defined on the nodes, based on how they are connected. The propagation rule is then essential in correctly modelling a physical system. However, standard GNNs do not include physics-based rules, meaning that the propagation rules may lead to unrealistic results.

3 Shallow-water-equation-inspired graph neural network (SWE–GNN)

We develop a graph neural network in which the computations are based on the shallow water equations. The proposed model takes as input both static and dynamic features that represent the topography of the domain and the hydraulic variables at time t, respectively. The outputs are the predicted hydraulic variables at time t+1. In the following, we detail the proposed model (Sect. 3.1) and its inputs and outputs (Sect. 3.2). Finally, we discuss the training strategy (Sect. 3.3).

3.1 Architecture

SWE–GNN is an encoder–processor–decoder architecture inspired by You et al. (2020) with residual connections that predicts auto-regressively the hydraulic variables at time $t+1$ as

$$\hat{\mathbf{U}}^{t+1} = \mathbf{U}^t + \Phi\left(\mathbf{X}_\mathrm{s}, \mathbf{U}^{t-p:t}, \mathbf{E}\right), \tag{5}$$

where the output $\hat{\mathbf{U}}^{t+1}$ corresponds to the predicted hydraulic variables at time t+1; $\mathbf{U}^t$ are the hydraulic variables at time t; $\Phi(\cdot)$ is the GNN-based encoder–processor–decoder model that determines the evolution of the hydraulic variables for a fixed time step; $\mathbf{X}_\mathrm{s}$ are the static node features; $\mathbf{U}^{t-p:t}$ are the dynamic node features, i.e. the hydraulic variables for time steps $t-p$ to t; and $\mathbf{E}$ are the edge features that describe the geometry of the mesh. The architecture detailed in the sequel is illustrated in Fig. 2.

3.1.1 Encoder

We employ three separate encoders for processing the static node features $\mathbf{X}_\mathrm{s} \in \mathbb{R}^{N \times I_{\mathrm{Ns}}}$, dynamic node features $\mathbf{X}_\mathrm{d} \equiv \mathbf{U}^{t-p:t} \in \mathbb{R}^{N \times O(p+1)}$, and edge features $\boldsymbol{\varepsilon} \in \mathbb{R}^{E \times I_\varepsilon}$, where I_{Ns} is the number of static node features, O the number of hydraulic variables (e.g. $O = 3$ if we consider water depth and the x and y components of the unit discharges), p the number of input previous time steps, and I_ε the number of input edge features. The encoded variables are

$$\mathbf{H}_\mathrm{s} = \phi_\mathrm{s}\left(\mathbf{X}_\mathrm{s}\right), \mathbf{H}_\mathrm{d} = \phi_\mathrm{d}\left(\mathbf{X}_\mathrm{d}\right), \mathbf{E}' = \phi_\varepsilon\left(\mathbf{E}\right), \tag{6}$$

where $\phi_\mathrm{s}(\cdot)$ and $\phi_\mathrm{d}(\cdot)$ are MLPs shared across all nodes that take an input $\mathbf{X} \in \mathbb{R}^{N \times I}$ and return a node matrix $\mathbf{H} \in \mathbb{R}^{N \times G}$, and $\phi_\varepsilon(\cdot)$ are MLPs shared across all edges that encode the edge features in $\mathbf{E}' \in \mathbb{R}^{E \times G}$. All MLPs have two layers, with a hidden dimension G followed by a parametric ReLU (PReLU) activation. The encoders expand the dimensionality of the inputs to allow for higher expressivity, with hyperparameter G being the dimension of the node embeddings. The ith rows of the node matrices $\mathbf{H}_\mathrm{s}$ and $\mathbf{H}_\mathrm{d}$ represent the encoded feature vectors associated with node i, i.e. $\boldsymbol{h}_{\mathrm{s}i}$ and $\boldsymbol{h}_{\mathrm{d}i}$, and the kth rows of the edge matrices $\mathbf{E}'$ represent the encoded feature vector associated with edge k.

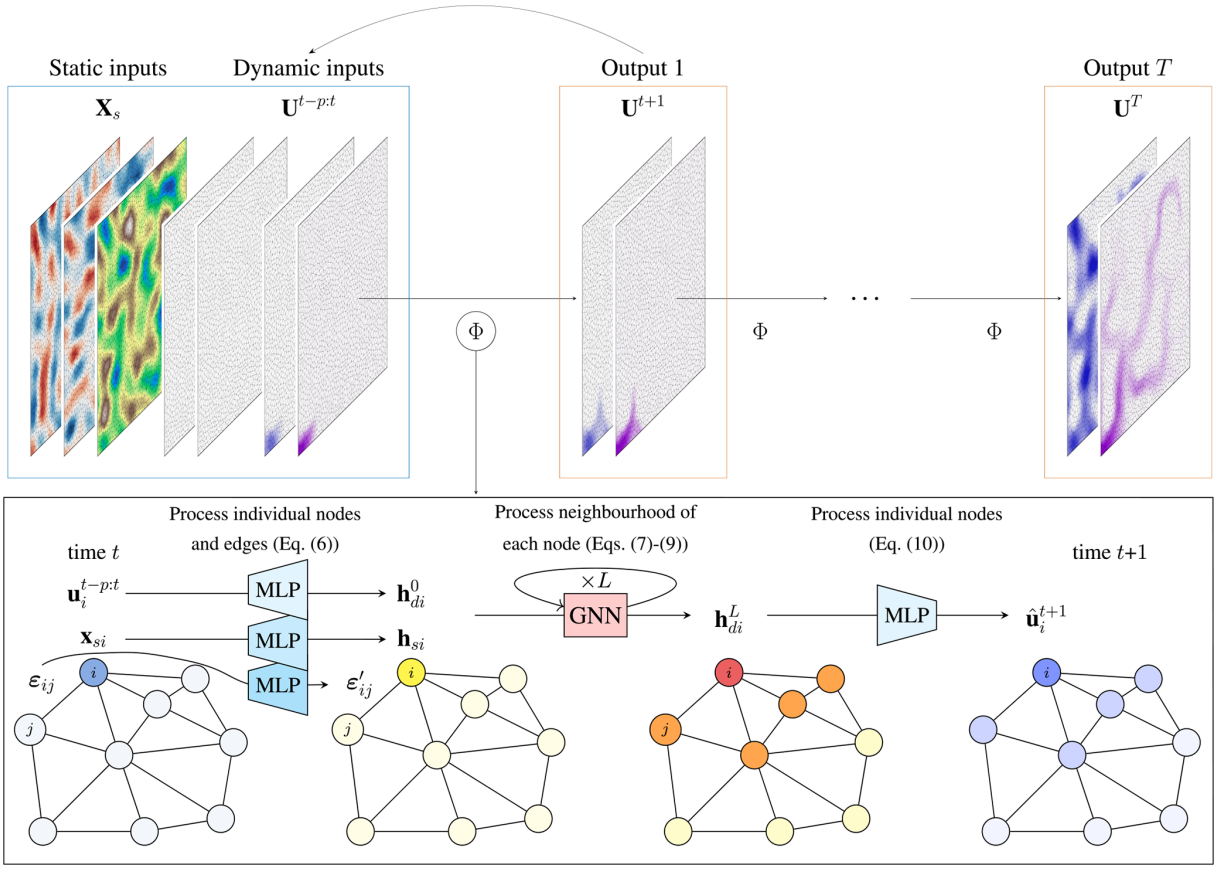


Figure 2. Overview of the proposed SWE–GNN model. The model Φ takes as input the mesh discretization of the static and dynamic input (blue box) and produces an estimate of their evolution in time (orange box). The model is then repeated auto-regressively, i.e. using its predictions as inputs, to determine the spatio-temporal evolution of the flood. The encoder–processor–decoder structure of the SWE–GNN model is shown in the bottom black box. The node inputs $\boldsymbol{x}_{si}$ and $\boldsymbol{u}_i^{t-p:t}$ represent static attributes, such as elevation and slopes, and dynamic attributes representing hydraulic variables, while the edge inputs $\boldsymbol{\varepsilon}_{ij}$ represent the mesh's geometry. The inputs are encoded into higher-dimensional embeddings $\boldsymbol{h}_{si}$, $\boldsymbol{h}_{di}^0$ (yellow nodes), and $\boldsymbol{\varepsilon}'_{ij}$ via three separate multi-layer perceptrons shared across nodes or edges. The embeddings, whose purpose is to increase the inputs' expressivity, are used as input for the L GNN layers. The output of the GNN $\boldsymbol{h}_{di}^L$ (red and orange nodes) is decoded via another shared multi-layer perceptron and is summed to the hydraulic variables at time t, i.e. $\boldsymbol{u}_i^t$. The final output $\hat{\boldsymbol{y}}_i$ (blue nodes) represents the prediction at time $t+1$, i.e. $\hat{\boldsymbol{u}}_i^{t+1}$.

3.1.2 Processor

We employed as a processor an L-layer GNN that takes a high-dimensional representation of the static and dynamic properties of the system at time t given by the encoders and that produces a spatio-temporally propagated high-dimensional representation of the system's evolution from times t to $t+1$. The propagation rule is based on the shallow water equation. In the SWE, the mass and momentum fluxes, representative of the dynamic features, evolve in space as a function of the source terms representative of the static and dynamic features. Moreover, water can only propagate from sources of water, and the velocity of propagation is influenced by the gradients of the hydraulic variables. Thus, the GNN layer $\ell = 1, \ldots, L-1$ update reads as

$$\boldsymbol{s}_{ij}^{(\ell+1)} = \psi\left(\boldsymbol{h}_{si}, \boldsymbol{h}_{sj}, \boldsymbol{h}_{di}^{(\ell)}, \boldsymbol{h}_{dj}^{(\ell)}, \varepsilon'_{ij}\right) \odot \left(\boldsymbol{h}_{dj}^{(\ell)} - \boldsymbol{h}_{di}^{(\ell)}\right), \tag{7}$$

$$\boldsymbol{h}_{di}^{(\ell+1)} = \boldsymbol{h}_{di}^{(\ell)} + \sum_{j \in \mathcal{N}_i} \boldsymbol{s}_{ij}^{(\ell+1)} \mathbf{W}^{(\ell+1)}, \tag{8}$$

where $\psi(\cdot) : \mathbb{R}^{5G} \to \mathbb{R}^G$ is an MLP with two layers, with a hidden dimension $2G$ followed by a PReLU activation function; $\odot$ is the Hadamard (element-wise) product; and $\mathbf{W}^{(\ell)} \in \mathbb{R}^{G \times G}$ are parameter matrices. The term $\boldsymbol{h}_{dj}^{(\ell)} - \boldsymbol{h}_{di}^{(\ell)}$ represents the gradient of the hydraulic variables and enforces water-related variables $\boldsymbol{h}_d$ to propagate only if at least one of the interfacing node features is non-zero, i.e. has water. The function $\psi(\cdot)$, instead, incorporates both static and dynamic inputs and provides an estimate of the source terms acting on the nodes. Thus, vector $\boldsymbol{s}_{ij}$ represents the fluxes

exchanged across neighbouring cells, and their linear combination is used as in Eq. (3) to determine the hydraulic variables' variation for a given cell. In this way, Eq. (7) resembles how fluxes are evaluated at the cell's interface in the numerical model, i.e. $\delta\mathbf{F}(\boldsymbol{u})_{ij} = \tilde{\boldsymbol{J}}_{ij}(\boldsymbol{u}_j - \boldsymbol{u}_i)$, which enforces conservation across interface discontinuities (Martínez-Aranda et al., 2022). Based on this formulation, $\boldsymbol{s}_{ij}$ can also be interpreted as an approximate Riemann solver (Toro, 2013), where the Riemann problem at the boundary between computational cells is approximated by the function $\psi(\cdot)$ in place of equations (e.g. Roe, 1981). To reduce model instabilities, the output of $\psi(\cdot)$ is normalized along its embedding dimension. That is, it is divided by its norm $\|\psi(\cdot)\|$. This procedure is similar to other graph normalization techniques that improve training stability (Chen et al., 2022). The contribution of each layer is linearly multiplied by $\mathbf{W}^{(\ell)}$ (Eq. 7). From a numerical perspective, this is analogous to an L-order multi-time-step scheme with L being the number of layers, where the weights are learned instead of being assigned (e.g. Dormand and Prince, 1980).

The GNN's output represents an embedding of the predicted hydraulic variables at time $t+1$ for a fixed time step Δt. Instead of enforcing stability by limiting Δt, as is done in numerical models, we can obtain the same result by considering a larger portion of space, which results in increasing Δx (see Eq. 4). This effect can be achieved by stacking multiple GNN layers, as each layer will increase the propagation space, also called the neighbourhood size. The number of GNN layers is then correlated with the space covered by the flood for a given temporal resolution. We can then write the full processor for the L GNN layers as

$$
\begin{aligned}
&\boldsymbol{h}_{\mathrm{d}i}^{(0)} = \boldsymbol{h}_{\mathrm{d}i}\mathbf{W}^{(0)},\\
&\boldsymbol{s}_{ij}^{(\ell+1)} = \psi\left(\boldsymbol{h}_{\mathrm{s}i}, \boldsymbol{h}_{\mathrm{s}j}, \boldsymbol{h}_{\mathrm{d}i}^{(\ell)}, \boldsymbol{h}_{\mathrm{d}j}^{(\ell)}, \boldsymbol{\varepsilon}_{ij}'\right) \odot \left(\boldsymbol{h}_{\mathrm{d}j}^{(\ell)} - \boldsymbol{h}_{\mathrm{d}i}^{(\ell)}\right),\\
&\boldsymbol{h}_{\mathrm{d}i}^{(\ell+1)} = \boldsymbol{h}_{\mathrm{d}i}^{(\ell)} + \sum_{j\in\mathcal{N}_i} \boldsymbol{s}_{ij}^{(\ell+1)}\mathbf{W}^{(\ell+1)},\\
&\boldsymbol{h}_{\mathrm{d}i}^{(L)} = \sigma\left(\boldsymbol{h}_{\mathrm{d}i}^{(L-1)} + \sum_{j\in\mathcal{N}_i} \boldsymbol{s}_{ij}^{(L)}\mathbf{W}^{(L)}\right),
\end{aligned}
\tag{9}
$$

where we employ a Tanh activation function $\sigma(\cdot)$ at the output of the Lth layer to limit numerical instabilities resulting in exploding values. The embedding of the static node features $\boldsymbol{h}_{\mathrm{s}i}$ and of the edge features $\boldsymbol{\varepsilon}_{ij}'$ does not change across layers, as the topography and discretization of the domain do not change in time.

3.1.3 Decoder

Symmetrically to the encoder, the decoder is composed of an MLP $\varphi(\cdot)$, shared across all the nodes, that takes as input the output of the processor $\mathbf{H}_{\mathrm{d}}^{(L)} \in \mathbb{R}^{N\times G}$ and updates the hydraulic variables at the next time step, i.e. $\hat{\mathbf{U}}^{t+1} \in \mathbb{R}^{N\times O}$, via residual connections, as

$$
\hat{\mathbf{U}}^{t+1} = \mathbf{U}^{t} + \varphi\left(\mathbf{H}_{\mathrm{d}}^{(L)}\right). \tag{10}
$$

The MLP $\varphi(\cdot)$ has two layers, with a hidden dimension G, followed by a PReLU activation. Neither of the MLPs in the dynamic encoder and the decoder has the bias terms as this would result in adding non-zero values corresponding to dry areas that would cause water to originate from any node.

3.2 Inputs and outputs

We define input features on the nodes and edges based on the SWE terms (see Eq. 2). We divide node features into a static component that represents fixed spatial attributes and a dynamic component that represents the hydraulic variables.

Static node features are defined as

$$
\boldsymbol{x}_{\mathrm{s}i} = \left(a_i, e_i, \boldsymbol{s}_{0i}, m_i, w_i^t\right), \tag{11}
$$

where a_i is the area of the ith finite-volume cell, its elevation e_i, its slopes in the x and y directions $\boldsymbol{s}_{0i}$, and its Manning coefficient m_i. We also included the water level at time t, w_i^t, given by the sum of the elevation and water depth at time t as node inputs, since this determines the water gradient (Liang and Marche, 2009). The reason why we include w_i^t in the static attributes instead of the dynamic ones is that these features can also be non-zero without water due to the elevation term and would thus result in the same issue mentioned for the dynamic encoder and decoder.

Dynamic node features are defined as

$$
\begin{aligned}
&\boldsymbol{x}_{\mathrm{d}i} = \boldsymbol{u}_i^{t-p:t} = \left(\boldsymbol{u}_i^{t-p}, \ldots, \boldsymbol{u}_i^{t-1}, \boldsymbol{u}_i^{t}\right),\\
&\boldsymbol{u}_i^t = \left(h_i^t, |q|_i^t\right),
\end{aligned}
\tag{12}
$$

where $\boldsymbol{u}_i^t$ are the hydraulic variables at time step t and $\boldsymbol{u}_i^{t-p:t}$ are the hydraulic variables up to p previous time steps to leverage the information of past data and to provide a temporal bias to the inputs. In contrast to the definition of the hydraulic variables as in Eq. (2), we selected the modulus of the unit discharge $|q|$ as a metric of flood intensity in place of its x and y components to avoid mixing scalar and vector components and because, for practical implications, such as damage estimation, the flow direction is less relevant than its absolute value (e.g. Kreibich et al., 2009).

Edge features are defined as

$$
\boldsymbol{\varepsilon}_{ij} = \left(\boldsymbol{n}_{ij}, l_{ij}\right), \tag{13}
$$

where $\boldsymbol{n}_{ij}$ is the outward unit normal vector and l_{ij} is the cell sides' length. Thus, the edge features represent the geometrical properties of the mesh. We excluded the fluxes $\mathbf{F}_{ij}$ as additional features as they depend on the hydraulic variables $\boldsymbol{u}_i$ and $\boldsymbol{u}_j$, which are already included in the dynamic node features.

Outputs. The model outputs are the estimated water depth and unit discharge at time $t+1$, i.e. $\hat{\boldsymbol{u}}_i^{t+1} = (\hat{h}_i^{t+1}, |\hat{q}|_i^{t+1})$, resulting in an output dimension $O = 2$. The outputs are used

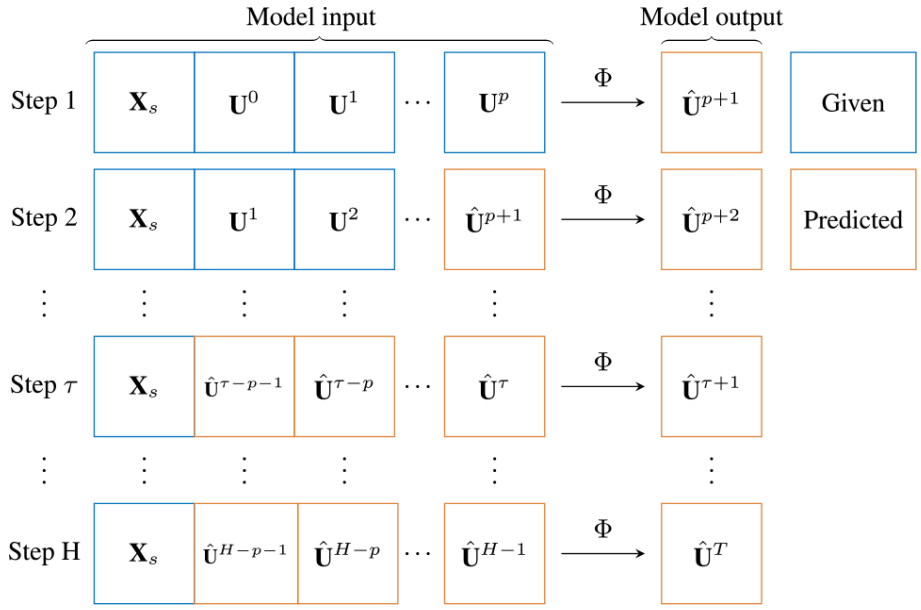


Figure 3. Example of auto-regressive prediction for p input previous time steps and H predicted steps ahead. The predictions at time τ are used as new inputs to predict the following time step and so on. The loss and the metrics are evaluated as the average over all steps H.

to update the input dynamic node features $\boldsymbol{x}_{\mathrm{d}i}$ for the following time step, as exemplified in Fig. 3. The same applies for the water level in the static attributes, i.e. $w_i^{t+1} = e_i + \hat{h}_i^{t+1}$.

3.3 Training strategy

The model learns from input–output data pairs. To stabilize the output of the SWE–GNN over time, we employ a multi-step-ahead loss function $\mathcal{L}$ that measures the accumulated error for multiple consecutive time steps, i.e.

$$\mathcal{L} = \frac{1}{HO} \sum_{\tau=1}^{H} \sum_{o=1}^{O} \gamma_o \| \hat{\boldsymbol{u}}_o^{t+\tau} - \boldsymbol{u}_o^{t+\tau} \|_2, \tag{14}$$

where $\boldsymbol{u}_o^{t+\tau} \in \mathbb{R}^N$ are the hydraulic variables over the whole graph at time $t+\tau$; H is the prediction horizon, i.e. the number of consecutive time instants; and γ_o are coefficients used to weight the influence of each variable on the loss. For each time step τ, we evaluate the model's prediction $\hat{\boldsymbol{u}}^{t+\tau}$ and then use the prediction recursively as part of the new dynamic node input (see Fig. 3). We repeat this process for a number of time steps H and calculate the root mean squared error (RMSE) loss as the average over all the steps. In this way, the model learns to correct its own predictions while also learning to predict a correct output, given a slightly wrong prediction, hence improving its robustness. After $p+1$ prediction steps, the inputs of the model are given exclusively by its predictions. During training, we limit the prediction horizon H instead of using the full temporal sequence due to memory constraints, since the back-propagation gradients must be stored for each time step.

To improve the training speed and stability, we also employed a curriculum learning strategy (Algorithm 1). This consists in progressively increasing the prediction horizon in Eq. (14) every fixed number of epochs up to H. The idea is first to learn the one-step-ahead or few-steps-ahead predictions to fit the short-term predictions and then to increase the number of steps ahead to stabilize the predictions (Wang et al., 2022).

Algorithm 1 Curriculum learning strategy.

Initialize:
$H = 1$
CurriculumSteps $= 15$
$\gamma_1 = 1$ (water depth h)
$\gamma_2 = 3$ (unit discharge q)
for epoch $= 1$ to MaxEpochs **do**
$\hat{\mathbf{U}}^{t+1} = \mathbf{U}^t + \Phi(\mathbf{X}_s, \mathbf{U}^{t-p:t}, \mathbf{E})$
$\mathcal{L} = \frac{1}{HO} \sum_{\tau=1}^{H} \sum_{o=1}^{O} \gamma_o \| \hat{\boldsymbol{u}}_o^{t+\tau} - \boldsymbol{u}_o^{t+\tau} \|_2$,
Update the parameters
if epoch $>$ CurriculumSteps$*H$ **then**
$H = H + 1$
end if
end for

4 Experimental setup

4.1 Dataset generation

We considered 130 numerical simulations of dike breach floods run on randomly generated topographies over two squared domains of sizes 6.4×6.4 and $12.8 \times 12.8\,\mathrm{km}^2$ representative of flood-prone polder areas.

We generated random digital elevation models using the Perlin noise generator (Perlin, 2002) as its ups and downs reflect plausible topographies. We opted for this methodology, instead of manually selecting terrain patches, to automatize the generation process, thus allowing for an indefinite number of randomized and unbiased training and testing samples.

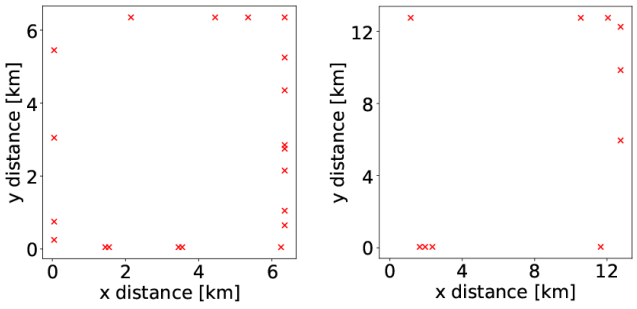
We employed a high-fidelity numerical solver, Delft3D-FM, which solves the full shallow water equations using an implicit scheme on staggered grids and adaptive time steps (Deltares, 2022). We used a dry bed as the initial condition and a constant input discharge of $50\,\mathrm{m}^3\,\mathrm{s}^{-1}$ as the boundary condition, equal to the maximum dike breach discharge. We employed a single boundary condition value for all the simulations as our focus is on showing generalizability over different topographies and breach locations. The simulation output is a set of temporally consecutive flood maps with a temporal resolution of 30 min.

We created three datasets with different area sizes and breach locations as summarized in Table 1. We selected a rectangular domain discretized by regular meshes to allow for a fairer comparison with other models that cannot work with meshes or cannot incorporate edge attributes. Furthermore, we considered a constant roughness coefficient m_i for all the simulations, meaning that we use the terrain elevation and the slopes in the x and y directions as static node inputs.

1. The first dataset consists of 100 DEMs over a squared domain of 64×64 grids of length 100 m and a simulation time of 48 h. This dataset is used for training, validation, and testing. We used a fixed testing set of 20 simulations, while the remaining 80 simulations are used for training (60) and validation (20).

2. The second dataset consists of 20 DEMs over a squared domain of 64×64 grids of length 100 m and a simulation time of 48 h. The breach location changes randomly across the border with a constant discharge of $50\,\mathrm{m}^3\,\mathrm{s}^{-1}$ (Fig. 4a). This dataset is used to test the generalizability of the model to unseen domains and breach locations.

3. The third dataset consists of 10 DEMs over a squared domain of 128×128 grids of length 100 m. The boundary conditions are the same as for the second dataset. Since the domain area is 4 times larger, the total simulation time is 120 h to allow for the flood to cover larger parts of the domain. This dataset is used to test the generalizability of the model to larger unseen domains, unseen breach locations, and longer time horizons.



(a) Dataset 2 (b) Dataset 3

Figure 4. Distribution of the breach locations (red crosses) for datasets 2 and 3.

Unless otherwise mentioned, we selected a temporal resolution of $\Delta t = 1$ h as a trade-off between detail and speed. When the beginning of the flood is relevant (e.g. for real-time forecasts), higher temporal resolutions are better. By contrast, if the final flood state is relevant, lower temporal resolutions may be better.

4.2 Training setup

We trained all models via the Adam optimization algorithm (Kingma and Ba, 2014). We employed a varying learning rate with 0.005 as a starting value and a fixed step decay of 90 % every seven epochs. The training was carried out for 150 epochs with early stopping. We used a maximum prediction horizon $H = 8$ steps ahead during training as a trade-off between model stability and training time, as later highlighted in Sect. 5.4. There is no normalization pre-processing step and, thus, the values of water depth and unit discharge differ in magnitude by a factor of 10. Since for application purposes discharge is less relevant than water depth (Kreibich et al., 2009), we weighted the discharge term by a factor of $\gamma_2 = 3$ (see Eq. 14) while leaving the weight factor for water depths as $\gamma_1 = 1$. Finally, we used one previous time step as input, i.e. $\mathbf{X}_\mathrm{d} = (\mathbf{U}^{t=0}, \mathbf{U}^{t=1})$, where the solution at time $t = 0$ corresponds to dry bed conditions.

We trained all the models using Pytorch (version 1.13.1) (Paszke et al., 2019) and Pytorch Geometric (version 2.2) (Fey and Lenssen, 2019). In terms of hardware, we employed an Nvidia Tesla V100S-PCIE-32GB for training and deployment (DHPC, 2022) and an Intel(R) Core(TM) i7-8665U @1.9 GHz CPU for deployment and for the execution of the numerical model. We run the models on both GPUs and CPUs to allow for a fair comparison with the numerical models.

4.3 Metrics

We evaluated the performance using the multi-step-ahead RMSE (Eq. 14) over the whole simulation. However, for test-

Table 1. Summary of the datasets employed for training (TR), validation (VA), and testing (TE). The uncertainty accounts for the variability across the different simulations in each dataset.

Dataset and use	Number of simulations	Size (km^2)	Random breach location	Simulation duration (h)	Execution time of the numerical model (s)
1 (TR, VA, TE)	100	6.4×6.4	No	48	29.5 ± 9.1
2 (TE)	20	6.4×6.4	Yes	48	32.5 ± 5.1
3 (TE)	10	12.8×12.8	Yes	120	185.5 ± 29.9

ing, we calculated the RMSE for each hydraulic variable o independently as

$$\mathrm{RMSE}_o = \frac{1}{H} \sum_{\tau=1}^{H} \|\hat{\boldsymbol{u}}_o^\tau - \boldsymbol{u}_o^\tau\|_2. \tag{15}$$

Analogously, we evaluated the mean average error (MAE) for each hydraulic variable o over the whole simulation as

$$\mathrm{MAE}_o = \frac{1}{H} \sum_{\tau=1}^{H} \|\hat{\boldsymbol{u}}_o^\tau - \boldsymbol{u}_o^\tau\|_1. \tag{16}$$

The prediction horizon H depends on the total simulation time and temporal resolution. For example, predicting 24 h with a temporal resolution of 30 min results in $H = 48$ steps ahead. We also measured the spatio-temporal error distribution of the water depth using the critical success index (CSI) for threshold values of 0.05 and 0.3 m, as in Löwe et al. (2021). The CSI measures the spatial accuracy of detecting a certain class (e.g. flood or no-flood) and, for a given threshold, it is evaluated as

$$\mathrm{CSI} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP} + \mathrm{FN}}, \tag{17}$$

where TP is the true positives, i.e. the number of cells where both the model and simulation predict flood; FP is the false positives, i.e. the number of cells where the model wrongly predicts flood; and FN is the false negatives, i.e. the number of cells where the model does not recognize a flooded area. We selected this measure as it discards the true negatives, i.e. when both the model and simulation predict no flood, as this condition is over-represented, especially for the initial time steps. Thus, including true negatives may give an overconfident performance estimate. We measured the computational speed-up as the ratio between the computational time required by the numerical model and the inference time of the deep-learning model. Both times refer to the execution of the complete flood simulation but do not include the time required to simulate the initial time steps.

5 Numerical results

5.1 Comparison with other deep-learning models

The proposed SWE–GNN model is compared with other deep-learning methods, including the following.

- CNN: encoder–decoder convolutional neural network based on U-Net (Ronneberger et al., 2015). The CNN considers the node feature matrix $\mathbf{X}$ reshaped as a tensor $\mathcal{X} \in \mathbb{R}^{g \times g \times I_N}$, where g is the number of grid cells, i.e. 64 for datasets 1 and 2 and 128 for dataset 3, and I_N is the number of static and dynamic features. This baseline is used to highlight the advantages of the mesh dual graph as an inductive bias in place of an image.

- GAT: graph attention network (Veličković et al., 2017). The weights in the propagation rule are learned considering an attention-based weighting. This baseline is considered to show the influence of learning the propagation rule with an attention mechanism. For more details, see Appendix A.

- GCN: graph convolutional neural network (Defferrard et al., 2016). This baseline is considered to show the influence of not learning the edge propagation rule in place of learning it. For more details, see Appendix A.

- SWE–$\mathrm{GNN_{ng}}$: SWE–GNN without the gradient term $\boldsymbol{x}_{\mathrm{d}j} - \boldsymbol{x}_{\mathrm{d}i}$. This is used to show the importance of the gradient term in the graph propagation rule.

We also evaluated MLP-based models, but their performance was too poor and we do not report it. All the models consider the same node feature inputs $\mathbf{X} = (\mathbf{X}_\mathrm{s}, \mathbf{X}_\mathrm{d})$, produce the same output $\hat{\mathbf{Y}} = \mathbf{U}^{t+1}$, produce extended simulations by using the predictions as input (as in Fig. 3), and use the same training strategy with the multi-step-ahead loss and curriculum learning. For the GNN-based models, we replaced the GNN in the processor while keeping the encoder–decoder structure as in Fig. 2. We conducted a thorough hyperparameter search for all the models, and we selected the one with the best validation loss. For the CNN architecture, the best model has three down- and up-scaling blocks, with 64 filters in the first encoding block. Interestingly, we achieved

good results only when employing batch normalization layers, PReLU as an activation function, and no residual connections. All other standard combinations resulted in poor performances, which we did not report as they are outside the scope of the paper. For the GNN-based architectures, all hyperparameter searches resulted in similar best configurations, i.e. $L = 8$ GNN layers and an embedding size of $G = 64$.

In Table 2, we report the testing RMSE and MAE for water depth and discharges as well as the CSI scores for all the models. The proposed SWE–GNN model and the U-Net-based CNN perform consistently better than all the other models, with no statistically significant difference in performance according to the Kolmogorov–Smirnov test (p value less than 0.05). The CNN performs similarly to the SWE–GNN because the computations on a regular grid are similar to those of a GNN. Nonetheless, there are valuable differences between the two models. First, SWE–GNN is by definition more physically explainable as water can only propagate from wet cells to neighbouring cells, while in the CNN there is no such physical constraint, as exemplified by Fig. 5b. Second, as emphasized in the following section, the SWE–GNN results in improved generalization abilities. Moreover, in contrast to CNNs, GNNs can also work with irregular meshes. Regarding the other GNN-based models, we noticed that the GAT model had the worse performance, indicating that the propagation rule cannot be learned efficiently via attention mechanisms. Moreover, the GCN and the SWE–$\mathrm{GNN_{ng}}$ achieved comparable results, meaning that the gradient term makes a relevant contribution to the model as its removal results in a substantial loss in performance. We expected this behaviour as, without this term, there is no computational constraint on how water propagates.

5.2 Generalization to breach locations and larger areas

We further tested the already trained models on datasets 2 and 3, with unseen topographies, unseen breach locations, larger domain sizes, and longer simulation times, as described in Table 1. In the following, we omit the other GNN-based models, since their performance was poorer, as highlighted in Table 2.

Table 3 shows that all the metrics remain comparable across the various datasets for the SWE–GNN, with test MAEs of approximately 0.04 m for water depth and $0.004\,\mathrm{m^2\,s^{-1}}$ for unit discharges, indicating that the model has learned the dynamics of the problems. The speed-up on the GPU of the SWE–GNN over dataset 3 increased further with respect to the smaller areas of datasets 1 and 2, reaching values twice as high, i.e. ranging from 100 to 600 times faster than the numerical model on the GPU. We attribute this to the deep-learning models' scalability and better exploitation of the hardware for larger graphs.

In Fig. 5, we see two examples of a SWE–GNN and a CNN on test datasets 2 and 3. The SWE–GNN model predicts better the flood evolution over time for unseen breach locations, even on bigger and unseen topographies, thanks to its hydraulic-based approach. On the other hand, the CNN strongly over- or under-predicts the flood extents unless the breach location is close to that of the training dataset, indicating that it lacks the correct inductive bias to generalize floods. For both models, the predictions remain stable even for time horizons 2.5 times longer than those in training.

5.3 SWE–GNN model analysis

Over the entire test part of dataset 1, the model achieves MAEs of 0.04 m for water depth and $0.004\,\mathrm{m^2\,s^{-1}}$ for unit discharges with respect to maximum water depths and unit discharges of 2.88 m and $0.55\,\mathrm{m^2\,s^{-1}}$, respectively, and average water depths and unit discharges of 0.62 m and $0.037\,\mathrm{m^2\,s^{-1}}$.

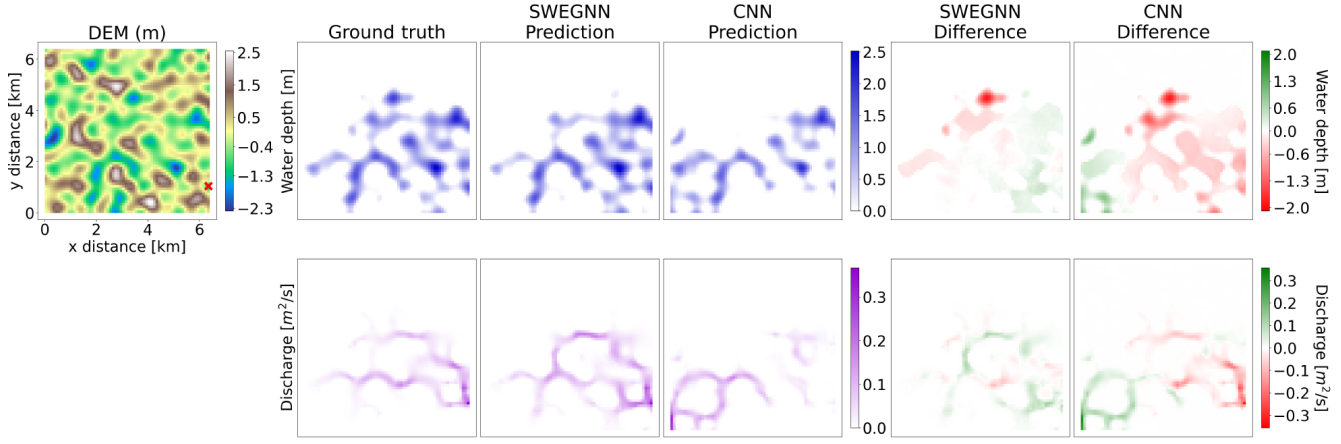
We illustrate the spatio-temporal performance of the model on a test sample in Fig. 6. Water depth and discharges evolve accurately over time, overall matching the ground-truth numerical results. The errors are related to small over- or under-predictions, a few incorrect flow routes, and lags in the predictions resulting in delays or anticipations that are corrected by the successive model iterations. In particular, the model struggles to represent discharges corresponding to ponding phenomena, i.e. when an area gets filled with water and then forms a temporary lake, as exemplified in the bottom-left part of the domain in Fig. 6b. This is because of the lower contribution of the discharges to the training loss. Nonetheless, the error does not propagate over time, thanks to the multi-step-ahead loss employed during training. In fact, the model updates the solution for the entire domain at each time step. Consequently, it exploits information on newly flooded neighbourhoods to recompute better values for the cells that were flooded before.

We also observe the average performance of the different metrics over time, for the whole test dataset 1, in Fig. 7. The CSI is consistently high throughout the whole simulation, indicating that the model correctly predicts where water is located in space and time. On the other hand, both MAE and RMSE increase over time. This is partially due to the evaluation of both metrics via a spatial average, which implies that, in the first time steps, where the domain is mostly dry, the error will naturally be lower. Nonetheless, the errors increase linearly or sub-linearly, implying that they are not prone to exploding exponentially.

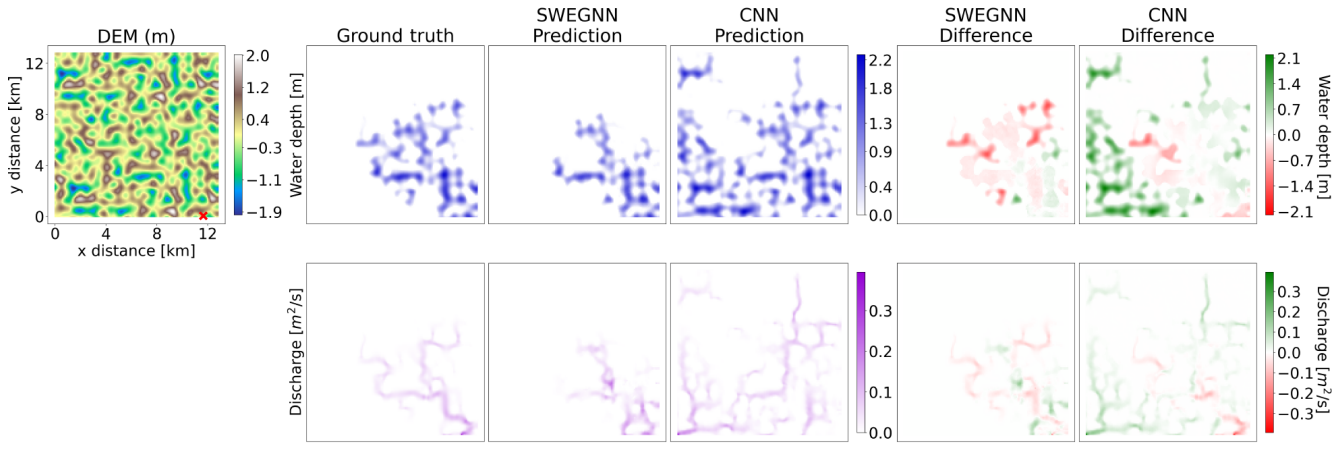
Next, we analysed the relationship between the number of GNN layers and the temporal resolution to validate the hypothesis that the number of layers is correlated with the time steps. Following the CFL condition, we can expand the computational domain by increasing the number of GNN layers in the model instead of decreasing the time steps. We considered several models with an increasing number of GNN layers targeting temporal resolutions of $\Delta t = 30$, 60, 90, and 120 min. Figure 8 shows that lower temporal resolutions (e.g. 120 min) require more GNN layers to reach the same perfor-

Table 2. Performance of the deep-learning models over test dataset 1. The provided uncertainty estimates account for the variability across the different simulations in the dataset. Bold results indicate the best performances considering a statistical significance with a p value of 0.05.

DL model	RMSE		MAE		CSI_τ (%)	
	h (m) $[10^{-2}]$	$\lvert q\rvert$ ($m^2\,s^{-1}$) $[10^{-2}]$	h (m) $[10^{-2}]$	$\lvert q\rvert$ ($m^2\,s^{-1}$) $[10^{-2}]$	$\tau = 0.05$ m	$\tau = 0.3$ m
CNN	$\mathbf{10.97 \pm 5.11}$	$\mathbf{1.33 \pm 0.57}$	$\mathbf{3.87 \pm 1.29}$	$\mathbf{0.42 \pm 0.13}$	$\mathbf{75.64 \pm 9.40}$	$\mathbf{73.42 \pm 9.26}$
GAT	25.78 ± 7.23	1.96 ± 0.61	9.27 ± 0.73	5.78 ± 0.11	34.50 ± 10.91	27.07 ± 8.63
GCN	16.49 ± 6.91	1.65 ± 0.55	6.05 ± 1.62	0.57 ± 0.11	61.14 ± 13.34	58.89 ± 11.90
SWE–GNN_ng	16.24 ± 6.65	1.71 ± 0.66	6.10 ± 1.56	0.63 ± 0.07	58.61 ± 11.97	57.91 ± 12.62
SWE–GNN	$\mathbf{11.15 \pm 5.11}$	$\mathbf{1.22 \pm 0.42}$	$\mathbf{3.93 \pm 1.63}$	$\mathbf{0.37 \pm 0.10}$	$\mathbf{75.85 \pm 9.30}$	$\mathbf{73.44 \pm 9.28}$

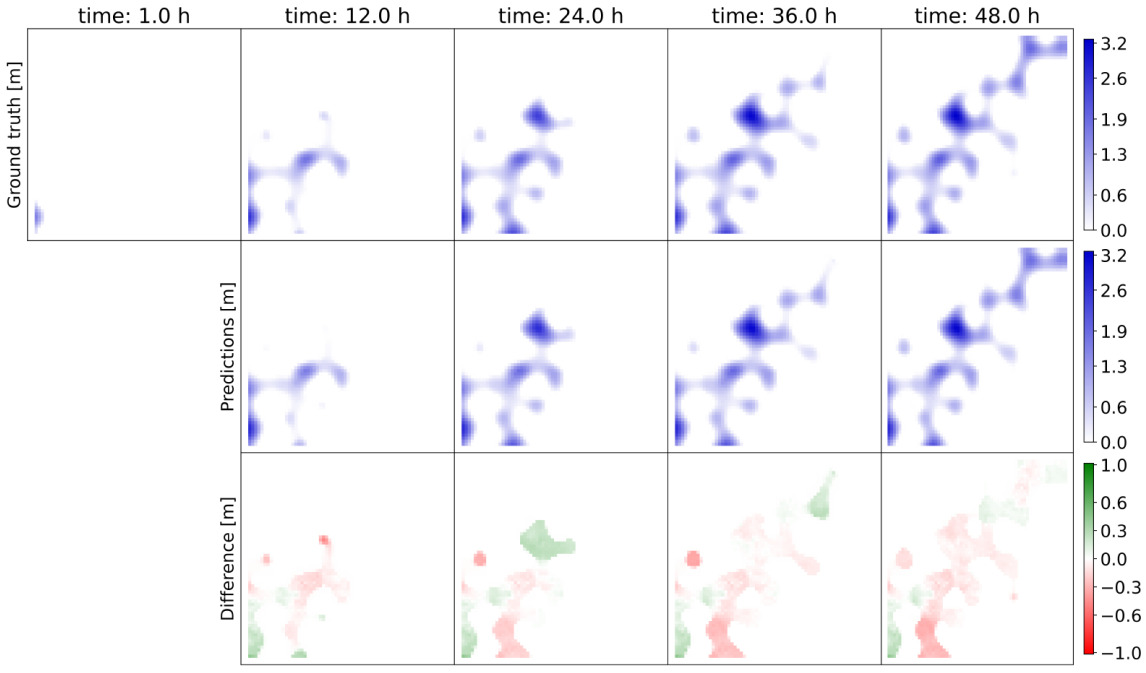


(a) Dataset 2

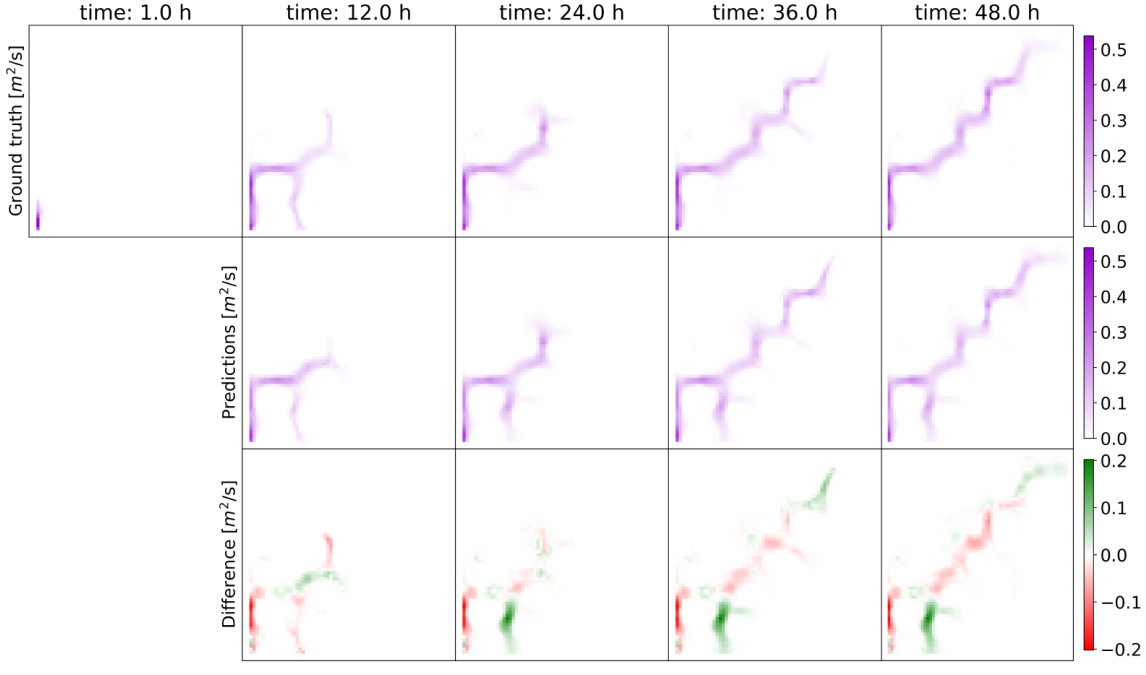


(b) Dataset 3

Figure 5. Comparison of the proposed SWE–GNN model against the CNN for two examples in test datasets 2 **(a)** and 3 **(b)**. In each panel, the top-left image represents the digital elevation model (DEM) along with a red cross corresponding to the breach location. The following blocks represent, respectively, the ground-truth numerical results, the SWE–GNN predictions, and the CNN predictions for water depth and unit discharges at the last time instant of the simulation (i.e. 48 h for dataset 2 and 120 h for dataset 3).



(a) Water depths



(b) Discharges

Figure 6. SWE–GNN model predictions for water depth **(a)** and discharges **(b)**. The results are displayed over time for a test topography in dataset 1 comparing the ground-truth output of the numerical simulation (top row) with the predictions (middle row). The difference (bottom row) is evaluated as the predicted value minus the ground-truth one; thus, positive values correspond to model over-predictions, while negative values correspond to under-predictions. The legends refer to the maximum values throughout the whole simulation. The top-left panels in both sub-figures represent the initial hydraulic conditions given as input to the DL model along with the dry bed conditions at time $t = 0$.

Table 3. Performance of the deep-learning models over test datasets 2 and 3, respectively, composed of unseen domains with unseen breach locations and unseen domains 4 times bigger than the training ones, also with unseen breach locations. The provided uncertainty estimates account for the variability across different simulations. Bold results indicate the best performances, considering a statistical significance with a p value of 0.05.

Test dataset	DL model	RMSE		MAE		CSI_τ (%)	
		h (m) $[10^{-2}]$	$\lvert q\rvert$ ($m^2\,s^{-1}$) $[10^{-2}]$	h (m) $[10^{-2}]$	$\lvert q\rvert$ ($m^2\,s^{-1}$) $[10^{-2}]$	$\tau = 0.05$ m	$\tau = 0.3$ m
2	CNN	15.74 ± 7.00	1.69 ± 0.47	6.50 ± 2.37	0.54 ± 0.13	51.90 ± 20.25	47.82 ± 18.42
	SWE–GNN	$\mathbf{11.11 \pm 4.65}$	$\mathbf{1.31 \pm 0.44}$	$\mathbf{4.84 \pm 1.87}$	$\mathbf{0.48 \pm 0.13}$	$\mathbf{73.62 \pm 8.04}$	$\mathbf{68.46 \pm 7.13}$
3	CNN	16.86 ± 3.12	$\mathbf{1.21 \pm 0.16}$	6.07 ± 1.77	$\mathbf{0.36 \pm 0.10}$	42.16 ± 15.63	40.92 ± 15.96
	SWE–GNN	$\mathbf{11.38 \pm 3.95}$	$\mathbf{1.12 \pm 0.30}$	$\mathbf{3.77 \pm 1.98}$	$\mathbf{0.31 \pm 0.12}$	$\mathbf{68.53 \pm 10.18}$	$\mathbf{64.53 \pm 11.20}$

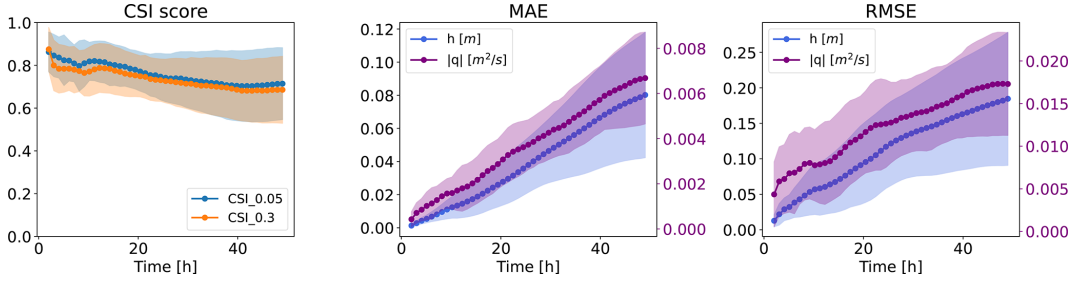


Figure 7. Temporal evolution of CSI scores, MAE, and RMSE for test dataset 1. The confidence bands refer to 1 standard deviation from the mean.

mance as that of higher temporal resolutions (e.g. 30 min). One reason why the number of layers does not increase linearly with the temporal resolution may be that the weighting matrices $\mathbf{W}_\ell$ (see Eq. 7) improve the expressive power of each layer, leading to fewer layers than needed otherwise.

Finally, we explored different model complexity combinations, expressed by the number of GNN layers and the latent space size, to determine a Pareto front for validation loss and speed-up, which results in a trade-off between fast and accurate models. Figure 9 shows that increasing the complexity reduces both errors and speed-ups while improving the CSI, as expected. While for the GPU the number of hidden features does not influence the speed-up, the performance on the CPU depends much more on it, with bigger models being slower, implying different trade-off criteria for deployment.

5.4 Sensitivity analysis of the training strategy

Finally, we performed a sensitivity analysis of the role of the multi-step-ahead function (see Eq. 14) and the curriculum learning (Algorithm 1) in the training performance. Sensitivity analysis is a technique that explores the effect of varying hyperparameters to understand their influence on the model's output. Figure 10a shows that increasing the number of steps ahead improves the performance. Increasing the number of steps implies higher memory requirements and longer training times. Because of the best performances and GPU availability, we selected eight steps ahead in all the experiments. However, when performing bigger hyperparameter searches or when limited by hardware, choosing fewer steps ahead can result in an acceptable performance. Similar considerations can also be made for the CNN model.

Figure 10b shows that increasing the interval of curriculum steps linearly reduces the training times while also improving the performance. The decrease in performance associated with bigger values is probably caused by the number of total training epochs, i.e. 150, which is insufficient to cover the whole prediction horizon H. Increasing the total number of epochs should increase both the performance and the training time, but we avoided this analysis and chose an interval of 15 epochs for the curriculum learning strategy as a trade-off between performance and training times. Moreover, models with curriculum steps between 0 and 15 suffered from spurious instabilities during training that were compensated for with early stopping, while models with more curriculum steps were generally more stable. This is due to sudden variations in the loss function that limit a smoother learning process.

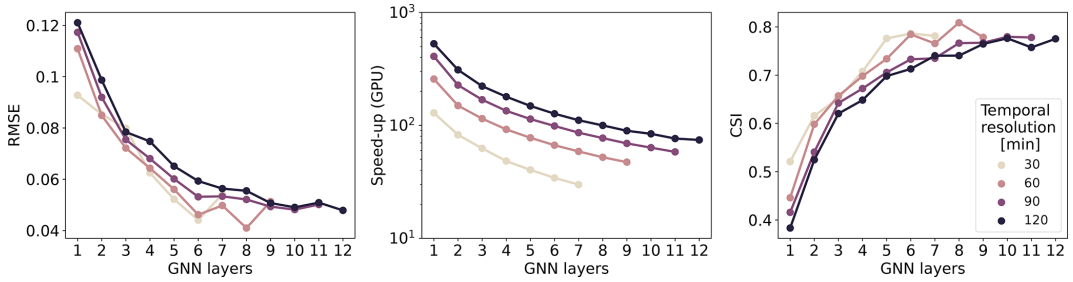


Figure 8. Relationship between the number of GNN layers and different temporal resolutions in terms of the validation RMSE and validation CSI. As the temporal resolution decreases and, conversely, as the time step increases, the optimal number of GNN layers, in terms of the desired performance level, increases.

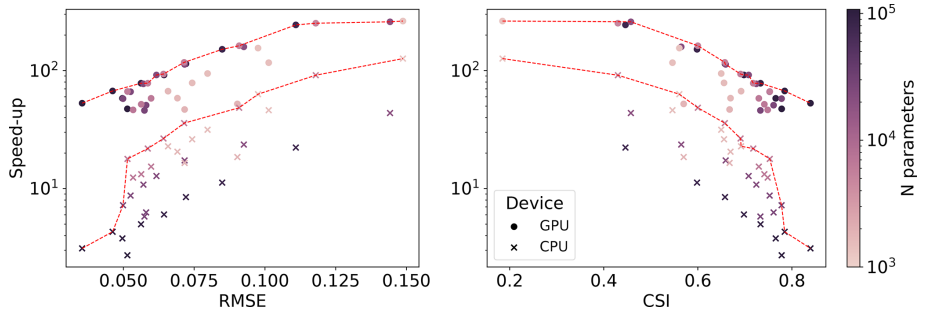


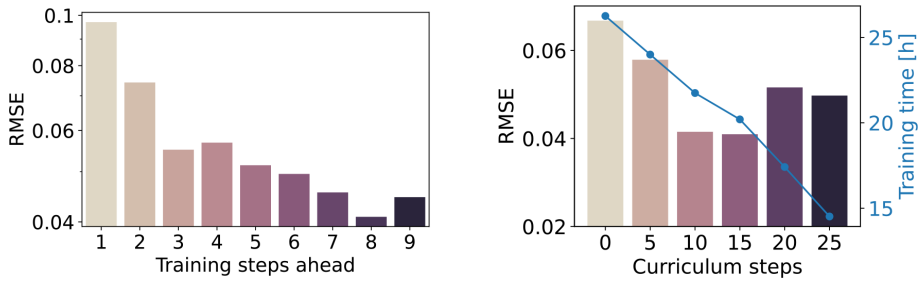
Figure 9. Pareto fronts (red-dotted lines) in terms of speed-ups, RMSE, and CSI for a varying number of parameters, for both CPUs and GPUs, for a temporal resolution of $\Delta t = 1$ h.

6 Concluding remarks

We proposed a deep-learning model for rapid flood modelling, called SWE–GNN, inspired by shallow water equations (SWEs) and graph neural networks (GNNs). The model takes the same inputs as a numerical model, i.e. the spatial discretization of the domain, elevation, slopes, and initial values of the hydraulic variables, and predicts their evolution in time in an auto-regressive manner. The results show that the SWE–GNN can correctly predict the evolution of water depth and discharges with mean average errors in time of 0.04 m and 0.004 $m^2 s^{-1}$, respectively. It also generalizes well to previously unseen topographies with varying breach locations, bigger domains, and longer time horizons. SWE–GNN is up to 2 orders of magnitude faster than the underlying numerical model. Moreover, the proposed model achieved consistently better performances with respect to other deep-learning models in terms of water depth and unit discharge errors as well as CSI.

In line with the hypothesis, GNNs proved to be a valuable tool for spatio-temporal surrogate modelling of floods. The analogy with finite-volume methods is relevant for three reasons. First, it improves the deep-learning model's interpretability, as the weights in the graph propagation rule can be interpreted as an approximate Riemann solver and multiple GNN layers can be seen as intermediate steps of a multi-step method such as Runge–Kutta. Second, the analogy also provides an existing framework to include conservation laws in the model and links two fields that can benefit from each other's advances. For example, multiple spatial and temporal resolutions could be jointly used in place of a fixed one, similarly to Liu et al. (2022). Third, the methodology is applicable to any flood modelling application where the SWE holds, such as storm surges and river floods. The same reasoning can also be applied to other types of partial differential equations where finite-volume methods are commonly used, such as in computational fluid dynamics.

The current analysis was carried out under a constant breach inflow as a boundary condition. Further research should extend the analysis to time-varying boundary conditions to better represent complex real-world scenarios. One solution is to employ ghost cells typical of numerical models (LeVeque, 2002) for the domain boundaries, assigning known values in time. It should be noted that our model cannot yet completely replace numerical models as it requires the first time step of the flood evolution as input. This challenge could be addressed by directly including boundary conditions in the model's inputs. In contrast to physically based numerical methods, the proposed model does not strictly enforce conservation laws such as mass balance. Future work could address this limitation by adding conservation equations to the training loss function, as is commonly done with physics-informed neural networks. Finally, while we empirically showed that the proposed model along with the multi-



(a) Multi-step-ahead loss

(b) Curriculum learning

Figure 10. Influence of **(a)** the number of training steps ahead on the validation RMSE and **(b)** the update interval in the curriculum learning.

step-ahead loss can sufficiently overcome numerical stability conditions, we provide no theoretical guarantee that stability can be enforced for an indefinite number of time steps.

Future research should investigate the new modelling approach in flood risk assessment and emergency preparation. This implies creating ensembles of flood simulations to reflect uncertainties, flood warning and predicting extreme events, and exploring adaptive modelling during floods by incorporating real-time observations. The model should also be validated in real case studies featuring linear elements such as secondary dikes and roads typical of polder areas. Further work could also address breach uncertainty in terms of timing, size, growth, and number of breaches. Moreover, future works should aim at improving the model's Pareto front. To improve the speed-up, one promising research direction would be to employ multi-scale methods that allow one to reduce the number of message-passing operations while still maintaining the same interaction range (e.g. Fortunato et al., 2022; Lino et al., 2022). On the other hand, better enforcing physics and advances in GNNs with spatio-temporal models (e.g. Sabbaqi and Isufi, 2022) or generalizations to higher-order interactions (e.g. Yang et al., 2022) may further benefit the accuracy of the model. Overall, the SWE–GNN marks a valuable step towards the integration of deep learning for practical applications.

Appendix A: Architecture details

In this Appendix, we further detail the different inputs and outputs, the hyperparameters, and the models' architectures used in Sect. 5.1.

A1 Inputs, outputs, and hyperparameters

Figure A1 shows the inputs employed by all the models in Sect. 5.1. The static inputs $\mathbf{X}_s$ are given by the slopes in the x and y directions as well as the elevation, while the initial dynamic inputs $\mathbf{X}_d = (\mathbf{U}^0, \mathbf{U}^1)$ are given by water depth and discharge at times $t = 0$ h, i.e. the empty domain, and $t = 1$ h.

Table A1 shows the hyperparameters employed for each model. Some hyperparameters are common to all the models, such as learning rate, number of maximum training steps ahead, and optimizer, while other change depend on the model, such as embedding dimensions and the number of layers.

A2 GNN benchmarks

We compared the proposed model against two benchmark GNNs that employ different propagation rules. Since those models cannot independently process static and dynamic attributes, in contrast to the SWE–GNN, we stacked the node inputs into a single node feature matrix $\mathbf{X} = (\mathbf{X}_\mathrm{d}, \mathbf{X}_\mathrm{s})$, which passes through an encoder MLP and then to the GNN.

The GCN employs the normalized Laplacian connectivity matrix to define the edge weights s_{ij}. The layer propagation rule reads as

$$s_{ij} = \left(\mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}\right)_{ij}, \tag{A1}$$

$$\boldsymbol{h}_i^{(\ell+1)} = \sum_{j \in \mathcal{N}_i} s_{ij} \mathbf{W}^{(\ell)} \boldsymbol{h}_j^{(\ell)}, \tag{A2}$$

where $\mathbf{I}$ is the identity matrix; $\mathbf{A}$ is the adjacency matrix, which has non-zero entries corresponding to edges; and $\mathbf{D}$ is the diagonal matrix.

GAT employs an attention-based mechanism to define the edge weights $\boldsymbol{s}_{ij}$ based on their importance in relation to the target node. The layer propagation rule reads as

$$\boldsymbol{s}_{ij} = \frac{\exp(\mathrm{LeakyReLU}\left(\boldsymbol{a}^T \left[\mathbf{W}^{(\ell)}\boldsymbol{h}_i^{(\ell)} || \mathbf{W}^{(\ell)}\boldsymbol{h}_k^{(\ell)}\right]\right)}{\sum_{k \in N_i} \exp\left(\mathrm{LeakyReLU}\left(\boldsymbol{a}^T \left[\mathbf{W}^{(\ell)}\boldsymbol{h}_i^{(\ell)} || \mathbf{W}^{(\ell)}\boldsymbol{h}_k^{(\ell)}\right]\right)\right)}, \tag{A3}$$

$$\boldsymbol{h}_i^{(\ell+1)} = \sum_{j \in \mathcal{N}_i} \boldsymbol{s}_{ij} \mathbf{W}^{(\ell)} \boldsymbol{h}_j^{(\ell)}, \tag{A4}$$

where $\boldsymbol{a} \in \mathbb{R}^{2G}$ is a weight vector, $\boldsymbol{s}_{ij}$ are the attention coefficients, and $||$ denotes concatenation.

A3 CNN

The encoder–decoder convolutional neural network is an architecture composed of two parts (Fig. A2). The encoder

extracts high-level features from the input images while reducing their extent via a series of convolutional and pooling layers, while the decoder extracts the output image from the compressed signal, again via a series of convolutional layers and pooling layers. The U-Net version of the architecture also features residual connections between images with the same dimensions. That is, the output of an encoder block is summed to the inputs of the decoder block with the same dimensions, as shown in Fig. A2. The equation for a single 2D convolutional layer is defined as

$$\mathbf{Y}_k = \sigma\left(\mathbf{W}_k * \mathbf{X}\right), \tag{A5}$$

where $\mathbf{Y}_k$ is the output feature map for the kth filter, $\mathbf{X}$ is the input image, $\mathbf{W}_k$ is the weight matrix for the kth filter, $*$ denotes the 2D convolution operation, and σ is an activation function.

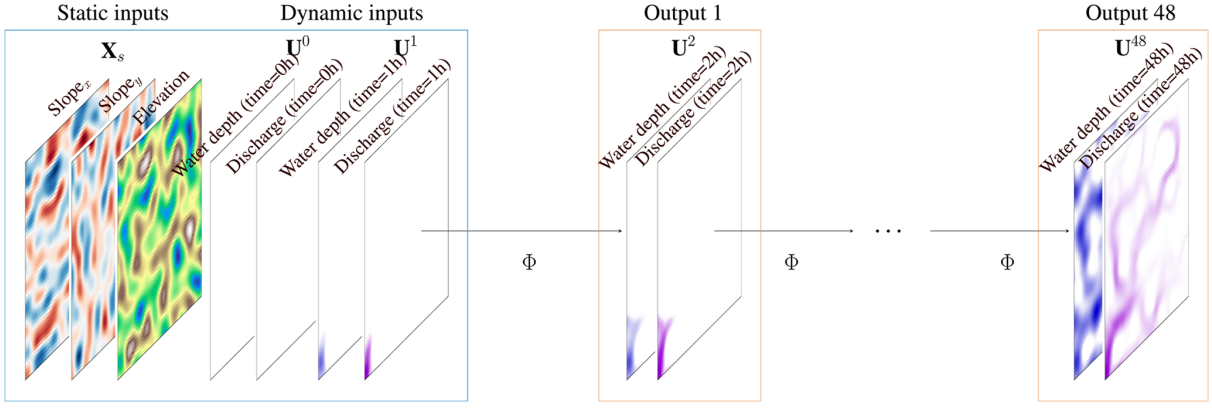


Figure A1. Detailed inputs and outputs used in the paper considering a regular mesh, $p = 1$ previous time steps, and a time resolution $\Delta t = 1$ h. The initial inputs are dry bed conditions, i.e. $\mathbf{U}^{t=0\,\mathrm{h}}$, and the first time step of the simulation, i.e. $\mathbf{U}^{t=1\,\mathrm{h}}$, given by the numerical model.

Table A1. Summary of the hyperparameters and related value ranges employed for the different deep-learning models. The bold values indicate the best configuration in terms of validation loss.

DL model	Hyperparameter name	Value range (**best**)
All models	Initial learning rate	0.005
	Input previous time steps (p)	1
	Temporal resolution (Δt)	1 h
	Maximum training steps ahead (H)	8
	Optimizer	Adam
GNN models	Embedding dimension (G)	8, 16, 32, **64**
	Number of GNN layers (L)	1, 2, 3, 4, 5, 6, 7, **8**, 9
	Batch size	8
CNN	First embedding dimension	16, 32, **64**, 128
	Number of encoding blocks	1, 2, **3**, 4
	Activation function	ReLU, **PReLU**, no activation
	Batch size	64

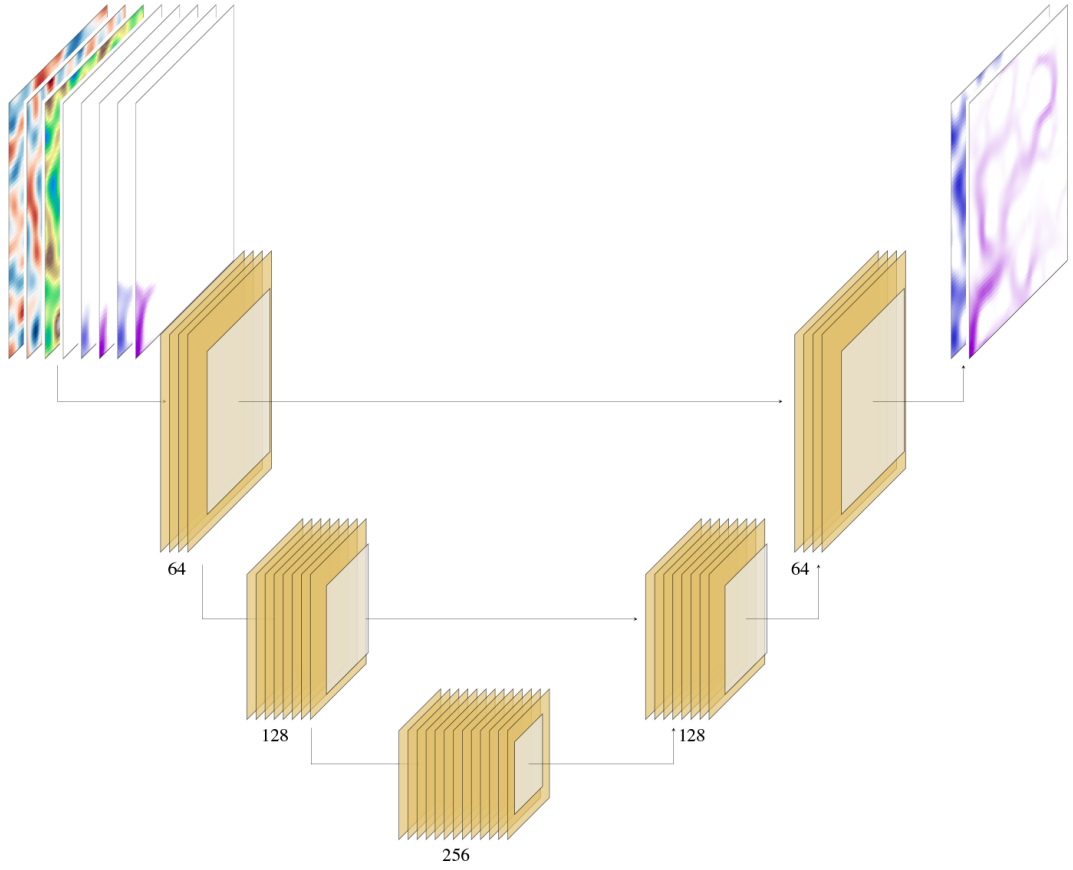


Figure A2. U-Net-based CNN architecture employed in the experiments, with the first embedding dimension of 64 and three encoding blocks. Each block is composed of one convolutional layer, followed by a batch normalization layer, a PReLU activation function, another convolutional layer, and finally a pooling layer. All the blocks with the same dimensions are connected by residual connections indicated by the horizontal lines.

Appendix B: Pareto front for dataset 3

We employed the models trained with different combinations of the number of GNN layers and embedding sizes (Sect. 5.3) on test dataset 3. Figure B1 shows that the models perform better in terms of speed with respect to the smaller areas, achieving similar CPU speed-ups and GPU speed-ups around 2 times higher than those in datasets 1 and 2.

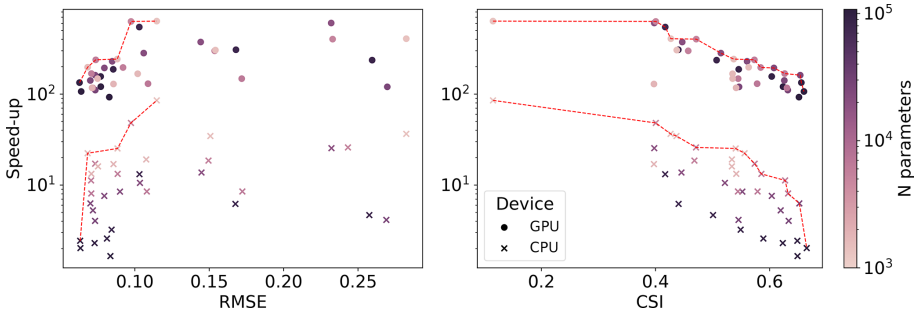


Figure B1. Pareto fronts on test dataset 3 (red-dotted lines) in terms of speed-ups, RMSE, and CSI for a varying number of parameters for a temporal resolution of $\Delta t = 1$ h.

Code and data availability. The employed dataset can be found at https://doi.org/10.5281/zenodo.7764418 (Bentivoglio and Bruijns, 2023). The code repository is available at https://doi.org/10.5281/zenodo.10214840 (Bentivoglio, 2023a) and https://github.com/RBTV1/SWE-GNN-paper-repository- (RBTV1, 2023).

Video supplement. The simulations on test datasets 1, 2, and 3, run with the presented model, can be found at https://doi.org/10.5281/zenodo.7652663 (Bentivoglio, 2023b).

Author contributions. RB: conceptualization, methodology, software, validation, data curation, writing – original draft preparation, visualization, writing – review and editing. EI: supervision, methodology, writing – review and editing, funding acquisition. SNJ: supervision, writing – review and editing. RT: conceptualization, supervision, writing – review and editing, funding acquisition, project administration.

Competing interests. The contact author has declared that none of the authors has any competing interests.

Disclaimer. Publisher's note: Copernicus Publications remains neutral with regard to jurisdictional claims made in the text, published maps, institutional affiliations, or any other geographical representation in this paper. While Copernicus Publications makes every effort to include appropriate place names, the final responsibility lies with the authors.

Acknowledgements. This work is supported by the TU Delft AI Initiative programme. We thank Ron Bruijns for providing the dataset to carry out the preliminary experiments. We thank Deltares for providing the license for Delft3D to run the numerical simulations.

Review statement. This paper was edited by Albrecht Weerts and reviewed by Daniel Klotz and one anonymous referee.

References

Alcrudo, F. and Garcia-Navarro, P.: A high-resolution Godunov-type scheme in finite volumes for the 2D shallow-water equations, Int. J. Numer. Meth. Fluids, 16, 489–505, 1993.

Bates, P. D. and De Roo, A. P.: A simple raster-based model for flood inundation simulation, J. Hydrol., 236, 54–77, https://doi.org/10.1016/S0022-1694(00)00278-X, 2000.

Battaglia, P. W. E. A.: Relational inductive biases, deep learning, and graph networks, arXiv [preprint], https://doi.org/10.48550/arXiv.1806.01261, 2018.

Bentivoglio, R.: Code repository for paper "Rapid Spatio-Temporal Flood Modelling via Hydraulics-Based Graph Neural Networks", Zenodo [code], https://doi.org/10.5281/zenodo.10214840, 2023a.

Bentivoglio, R.: Video simulations for paper "Rapid Spatio-Temporal Flood Modelling via Hydraulics-Based Graph Neural Networks", Zenodo [video supplement], https://doi.org/10.5281/zenodo.7652663, 2023b.

Bentivoglio, R. and Bruijns, R.: Raw datasets for paper "Rapid Spatio-Temporal Flood Modelling via Hydraulics-Based Graph Neural Networks", Zenodo [data set], https://doi.org/10.5281/zenodo.7764418, 2023.

Bentivoglio, R., Isufi, E., Jonkman, S. N., and Taormina, R.: Deep learning methods for flood mapping: a review of existing applications and future research directions, Hydrol. Earth Syst. Sci., 26, 4345–4378, https://doi.org/10.5194/hess-26-4345-2022, 2022.

Berkhahn, S., Fuchs, L., and Neuweiler, I.: An ensemble neural network model for real-time prediction of urban floods, J. Hydrol., 575, 743–754, https://doi.org/10.1016/j.jhydrol.2019.05.066, 2019.

Brandstetter, J., Worrall, D., and Welling, M.: Message passing neural PDE solvers, arXiv [preprint], https://doi.org/10.48550/arXiv.2202.03376, 2022.

Bronstein, M. M., Bruna, J., Cohen, T., and Veličković, P.: Geometric deep learning: Grids, groups, graphs, geodesics, and gauges, arXiv [preprint], arXiv:2104.13478, https://doi.org/10.48550/arXiv.2104.13478, 2021.

Chen, Y., Tang, X., Qi, X., Li, C.-G., and Xiao, R.: Learning graph normalization for graph neural networks, Neurocomputing, 493, 613–625, 2022.

Costabile, P., Costanzo, C., and Macchione, F.: Performances and limitations of the diffusive approximation of the 2-d shallow wa-

ter equations for flood simulation in urban and rural areas, Appl. Numer. Math., 116, 141–156, 2017.

Courant, R., Friedrichs, K., and Lewy, H.: On the partial difference equations of mathematical physics, IBM J. Res. Dev., 11, 215–234, 1967.

Defferrard, M., Bresson, X., and Vandergheynst, P.: Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering, in: Advances in Neural Information Processing Systems, vol. 29, edited by: Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., and Garnett, R., Curran Associates, Inc., 1–10, https://proceedings.neurips.cc/paper/2016/file/04df4d434d481c5bb723be1b6df1ee65-Paper.pdf (last access: 18 February 2023), 2016.

Deltares: Delft3D-FM User Manual, https://content.oss.deltares.nl/delft3d/manuals/D-Flow_FM_User_Manual.pdf (last access: 18 February 2023), 2022.

DHPC – Delft High Performance Computing Centre: DelftBlue Supercomputer (Phase 1), https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase1 (last access: 18 February 2023), 2022.

do Lago, C. A., Giacomoni, M. H., Bentivoglio, R., Taormina, R., Gomes, M. N., and Mendiondo, E. M.: Generalizing rapid flood predictions to unseen urban catchments with conditional generative adversarial networks, J. Hydrol., 618, 129276, https://doi.org/10.1016/j.jhydrol.2023.129276, 2023.

Dormand, J. R. and Prince, P. J.: A family of embedded Runge-Kutta formulae, J. Comput. Appl. Math., 6, 19–26, 1980.

Fey, M. and Lenssen, J. E.: Fast graph representation learning with PyTorch Geometric, arXiv [preprint], arXiv:1903.02428, https://doi.org/10.48550/arXiv.1903.02428, 2019.

Fortunato, M., Pfaff, T., Wirnsberger, P., Pritzel, A., and Battaglia, P.: Multiscale meshgraphnets, arXiv [preprint], arXiv:2210.00612, https://doi.org/10.48550/arXiv.2210.00612, 2022.

Gama, F., Isufi, E., Leus, G., and Ribeiro, A.: Graphs, convolutions, and neural networks: From graph filters to graph neural networks, IEEE Sig. Process. Mag., 37, 128–138, 2020.

Guo, Z., Leitao, J. P., Simões, N. E., and Moosavi, V.: Data-driven flood emulation: Speeding up urban flood predictions by deep convolutional neural networks, J. Flood Risk Manage., 14, e12684, https://doi.org//10.1111/jfr3.12684, 2021.

Guo, Z., Moosavi, V., and Leitão, J. P.: Data-driven rapid flood prediction mapping with catchment generalizability, J. Hydrol., 609, 127726, https://doi.org/10.1016/j.jhydrol.2022.127726, 2022.

Horie, M. and Mitsume, N.: Physics-Embedded Neural Networks: $\mathbf{E}(n)$-Equivariant Graph Neural PDE Solvers, arXiv [preprint], https://doi.org/10.48550/ARXIV.2205.11912, 2022.

Hu, R. L., Pierce, D., Shafi, Y., Boral, A., Anisimov, V., Nevo, S., and Chen, Y.-F.: Accelerating physics simulations with tensor processing units: An inundation modeling example, arXiv [preprint], https://doi.org/10.48550/arXiv.2204.10323, 2022.

Jacquier, P., Abdedou, A., Delmas, V., and Soulaïmani, A.: Non-intrusive reduced-order modeling using uncertainty-aware Deep Neural Networks and Proper Orthogonal Decomposition: Application to flood modeling, J. Comput. Phys., 424, 109854, https://doi.org/10.1016/j.jcp.2020.109854, 2021.

Jonkman, S. N., Kok, M., and Vrijling, J. K.: Flood risk assessment in the Netherlands: A case study for dike ring South Holland, Risk Ana., 28, 1357–1374, 2008.

Kabir, S., Patidar, S., Xia, X., Liang, Q., Neal, J., and Pender, G.: A deep convolutional neural network model for rapid prediction of fluvial flood inundation, J. Hydrol., 590, 125481, https://doi.org/10.1016/j.jhydrol.2020.125481, 2020.

Kingma, D. P. and Ba, J.: Adam: A method for stochastic optimization, arXiv [preprint], arXiv:1412.6980 https://doi.org/10.48550/arXiv.1412.6980, 2014.

Kreibich, H., Piroth, K., Seifert, I., Maiwald, H., Kunert, U., Schwarz, J., Merz, B., and Thieken, A. H.: Is flow velocity a significant parameter in flood damage modelling?, Nat. Hazards Earth Syst. Sci., 9, 1679–1692, https://doi.org/10.5194/nhess-9-1679-2009, 2009.

LeCun, Y., Bengio, Y., and Hinton, G.: Deep learning, Nature, 521, 436–444, 2015.

LeVeque, R. J.: Finite volume methods for hyperbolic problems, in: vol. 31, Cambridge University Press, ISBN 0521810876, 2002.

Liang, Q. and Marche, F.: Numerical resolution of well-balanced shallow water equations with complex source terms, Adv. Water Resour., 32, 873–884, 2009.

Lino, M., Cantwell, C., Bharath, A. A., and Fotiadis, S.: Simulating Continuum Mechanics with Multi-Scale Graph Neural Networks, arXiv [preprint], arXiv:2106.04900, https://doi.org/10.48550/arXiv.2106.04900, 2021.

Lino, M., Fotiadis, S., Bharath, A. A., and Cantwell, C. D.: Multi-scale rotation-equivariant graph neural networks for unsteady Eulerian fluid dynamics, Phys. Fluids, 34, 087110, https://doi.org/10.1063/5.0097679, 2022.

Liu, Y., Kutz, J. N., and Brunton, S. L.: Hierarchical deep learning of multiscale differential equation time-steppers, Philos. T. Roy. Soc. A, 38, 020210200, https://doi.org/10.1098/rsta.2021.0200, 2022.

Löwe, R., Böhm, J., Jensen, D. G., Leandro, J., and Rasmussen, S. H.: U-FLOOD – Topographic deep learning for predicting urban pluvial flood water depth, J. Hydrol., 603, 126898, https://doi.org/10.1016/j.jhydrol.2021.126898, 2021.

Martínez-Aranda, S., Fernández-Pato, J., Echeverribar, I., Navas-Montilla, A., Morales-Hernández, M., Brufau, P., Murillo, J., and García-Navarro, P.: Finite Volume Models and Efficient Simulation Tools (EST) for Shallow Flows, in: Advances in Fluid Mechanics, Springer, 67–137, https://doi.org/10.1007/978-981-19-1438-6_3, 2022.

Mosavi, A., Ozturk, P., and Chau, K.-W.: Flood Prediction Using Machine Learning Models: Literature Review, Water, 10, 1536, https://doi.org/10.3390/w10111536, 2018.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S.: Pytorch: An imperative style, high-performance deep learning library, arXiv [preprint], https://doi.org/10.48550/arXiv.1912.01703, 2019.

Peng, J.-Z., Wang, Y.-Z., Chen, S., Chen, Z.-H., Wu, W.-T., and Aubry, N.: Grid adaptive reduced-order model of fluid flow based on graph convolutional neural network, Phys. Fluids, 34, 087121, https://doi.org/10.1063/5.0100236, 2022.

Perlin, K.: Improving Noise, ACM Trans. Graph., 21, 681–682, https://doi.org/10.1145/566654.566636, 2002.

Petaccia, G., Natale, L., Savi, F., Velickovic, M., Zech, Y., and Soares-Frazão, S.: Flood wave propagation in steep mountain rivers, J. Hydroinform., 15, 120–137, 2013.

Petaccia, G., Leporati, F., and Torti, E.: OpenMP and CUDA simulations of Sella Zerbino Dam break on unstructured grids, Comput. Geosci., 20, 1123–1132, https://doi.org/10.1007/s10596-016-9580-5, 2016.

RBTV1: SWE-GNN-paper-repository-, GitHub [code], https://github.com/RBTV1/SWE-GNN-paper-repository- (last access: 29 November 2023), 2023.

Roe, P. L.: Approximate Riemann solvers, parameter vectors, and difference schemes, J. Comput. Phys., 135, 250–258, https://doi.org/10.1006/jcph.1997.5705, 1981.

Ronneberger, O., Fischer, P., and Brox, T.: U-net: Convolutional networks for biomedical image segmentation, in: International Conference on Medical image computing and computer-assisted intervention, arXiv [preprint], https://doi.org/10.48550/arXiv.1505.04597, 2015.

Sabbaqi, M. and Isufi, E.: Graph-Time Convolutional Neural Networks: Architecture and Theoretical Analysis, arXiv [preprint], https://doi.org/10.48550/arXiv.2206.15174, 2022.

Teng, J., Jakeman, A. J., Vaze, J., Croke, B. F., Dutta, D., and Kim, S.: Flood inundation modelling: A review of methods, recent advances and uncertainty analysis, Environ. Model. Softw., 90, 201–216, https://doi.org/10.1016/j.envsoft.2017.01.006, 2017.

Toro, E. F.: Riemann solvers and numerical methods for fluid dynamics: a practical introduction, Springer Science & Business Media, https://doi.org/10.1007/b79761, 2013.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y.: Graph attention networks, arXiv [preprint], https://doi.org/10.48550/arXiv.1710.10903, 2017.

Villar, S., Hogg, D. W., Yao, W., Kevrekidis, G. A., and Schölkopf, B.: The passive symmetries of machine learning, arXiv [preprint], arXiv:2301.13724, https://doi.org/10.48550/arXiv.2301.13724, 2023.

Vorogushyn, S., Merz, B., and Apel, H.: Development of dike fragility curves for piping and micro-instability breach mechanisms, Nat. Hazards Earth Syst. Sci., 9, 1383–1401, https://doi.org/10.5194/nhess-9-1383-2009, 2009.

Vreugdenhil, C. B.: Numerical methods for shallow-water flow, in: vol. 13, Springer Science & Business Media, ISBN 978-0-7923-3164-3, 1994.

Wang, X., Chen, Y., and Zhu, W.: A survey on curriculum learning, IEEE T. Pattern Anal. Mach. Intel., 44, 4555–4576, https://doi.org/10.1109/TPAMI.2021.3069908, 2022.

Xia, X., Liang, Q., Ming, X., and Hou, J.: An efficient and stable hydrodynamic model with novel source term discretization schemes for overland flow and flood simulations, Water Resour. Res., 53, 3730–3759, 2017.

Yang, M., Isufi, E., and Leus, G.: Simplicial Convolutional Neural Networks, in: ICASSP 2022–2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 23–27 May 2022, Singapore, Singapore, 8847–8851, https://doi.org/10.1109/ICASSP43922.2022.9746017, 2022.

You, J., Ying, Z., and Leskovec, J.: Design space for graph neural networks, Adv. Neural Inform. Process. Syst., 33, 17009–17021, 2020.

Zhang, A., Lipton, Z. C., Li, M., and Smola, A. J.: Dive into deep learning, arXiv [preprint], arXiv:2106.11342, https://doi.org/10.48550/arXiv.2106.11342, 2021.

Zhou, Y., Wu, W., Nathan, R., and Wang, Q.: Deep learning-based rapid flood inundation modelling for flat floodplains with complex flow paths, Water Resour. Res., 58, e2022WR033214, https://doi.org/10.1029/2022WR033214, 2022.