We thank the Reviewer for the helpful comments and suggestions. We hereby address them individually. In this document we indicate the *Reviewer's comments in italic dark grey*, while text that was changed in the paper in blue.

This paper introduces a novel machine learning algorithm for the simulation of 2D surface flooding. The algorithm takes inspiration from scientific computing techniques to develop the machine learning architecture, to arrive at a setup that provides a time-dynamic and spatially distributed simulation, and a speedup in the order of factor 100 compared to a numerical simulation model. To my knowledge, this is the first application of this kind of methodology for flood problems. A number of questions therefore remain unsolved, and the considered case examples do not yet reflect all complexities that we encounter in the real world. On the other hand, this work is likely to inspire a variety of derivative works in hydrology, where the potential impact of these techniques so far is not really recognized. Because this work is a frontrunner, I have a number of suggestions that are mainly aimed at making the work accessible to a broader audience. I think these can be addressed in a minor revision, and I recommend the paper for publication, subject to these modifications.

We thank the Reviewer for the comments and for sharing our view on the potential impact of this paper. Regarding the accessibility to a broader audience, we will also release a tutorial notebook on how to use the model alongside with the code repository that will be shared upon paper publication.

Main comments:

Comment 1: Limitations - and important feature of numerical methods is that they (mostly) preserve e.g. mass and momentum. As far as I can see, this is not the case for the algorithm proposed here, and it is not straightforward to see how this can be implemented in the encoder/decoder architecture. This should be clearly mentioned as a limitation.

Answer 1: We thank the Reviewer for the comment. We included the following additional sentences in the discussion to emphasize this limitation and how it could be addressed.

Lines 415-417: "Contrarily 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 in the training loss function, as is commonly done with physics-informed neural networks."

Comment 2: A major challenge with this type of model is actually implementation. For example, efficient data pipelines for custom graph operators are not a straightforward problem. I would strongly suggest a section or appendix summarizing the main computational challenges and how you suggest to address them.

Answer 2: We understand the concern with graph operators not being commonly used in the water field, however there are now several libraries, such as pytorch geometric, which offer a great variety of implementation strategies which are relatively easy to follow using the documentation. Moreover, GNN-based solutions have been implemented in billion-scale graphs (D. Zheng et al., 2020) and there are also companies working on specific hardware for graph-based NNs (e.g., Graphcore). So, overall, we believe that this might not be so big of an issue.

We highlighted the implementation issue in lines 276-277 as follows:

"Pytorch Geometric is a library that allows easy implementation of graph neural networks, also for billion-scale graphs (D. Zheng et al., 2020)."

D. Zheng *et al*., "DistDGL: Distributed Graph Neural Network Training for Billion-Scale Graphs," *2020 IEEE/ACM 10th Workshop on Irregular Applications: Architectures and Algorithms (IA3)*, GA, USA, 2020, pp. 36-44, doi: 10.1109/IA351965.2020.00011.

https://www.graphcore.ai/posts/accelerating-pyg-on-ipus-unleash-the-power-of-graph-neural-networks?utm_content=243840010&utm_medium=social&utm_source=linkedin&hss_channel=lcp-10812092

Comment 3: Along the same lines, I appreciate that the authors have tried to keep the Methodology description generic. However, this also makes it very hard to read in some places. I think you could greatly help the readers with an Appendix that includes a detailed variant of Figure 2, where you include equations 5 to 10, and where hyperparameters (G, p, etc.) reflect that actual values used in your implementation.

Answer 3: As suggested by the Reviewer, we included in Appendix A a more detailed description of the inputs and outputs used by the model, in the specific case of a temporal resolution of 1 hour and with only one previous time step as input. To better explain this, we added the following figure:

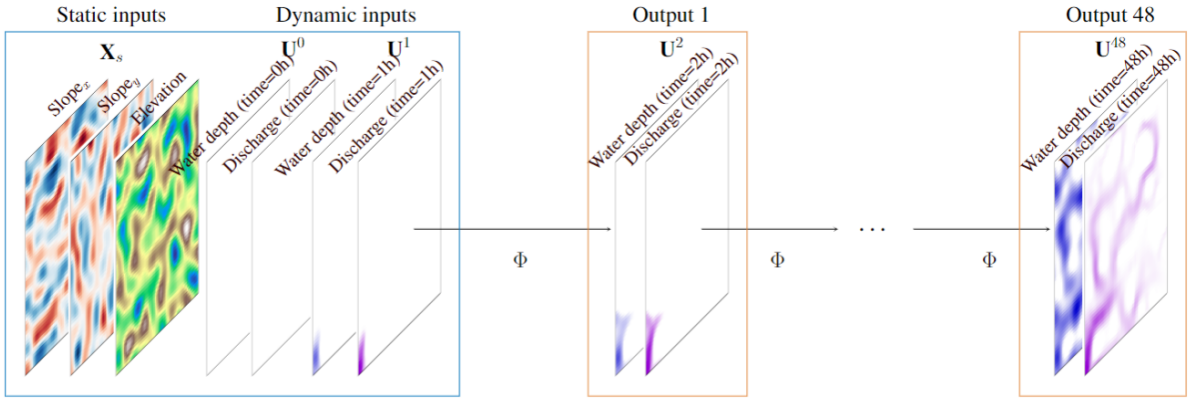


Figure A1. Detailed inputs and outputs used in the paper, considering a regular mesh, $p = 1$ previous time steps and a time resolution $\Delta t = 1h$. The initial inputs are dry bed conditions, i.e., $\mathbf{U}^{t=0h}$, and the first time step of the simulation, i.e., $\mathbf{U}^{t=1h}$, given by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

We describe Figure A1 in the appendix (lines 431-433) as follows:

"Fig. A1 shows the inputs employed by all models in Section 5.1. The static inputs Xs are given by the slopes in the x and y directions, and the elevation, while the initial dynamic inputs Xd = (U0, U1) are given by water depth and discharge at times t = 0h, i.e., the empty domain, and t = 1h."

Following also the minor comments (comment 7), we added the equation numbers directly in Figure 2, as later descripted.

Comment 4: Regarding the hyperparameters, I strongly miss a table that summarizes

the final set of hyperparameters. These are now spread out through the paper. In addition, the complexity of the different MLPs used throughout the methodology is currently entirely unclear.

Answer 4: We thank the Reviewer for the comment. We added a table summarizing the hyperparameters and their values' range in Appendix A. Lines 435-437 read as follows:

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

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

DL model	Hyperparameter name	Values' range (**best**)
All models	Initial learning rate	0.005
	Input previous time steps (p)	1
	Temporal resolution (Δt)	$1h$
	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

Regarding the MLP characterisation, we added the following sentences in Section 3.1.

Lines 153-154: "All MLPs have two layers, with hidden dimension G, followed by a PReLU activation."

Lines 168-169: "where $\psi(\cdot) : R5G \rightarrow RG$ is an MLP with two layers, with hidden dimension 2G, followed by a PReLU activation function, $\odot$ is the Hadamard (element-wise) product, and $W(\ell) \in RG \times G$ are parameter matrices."

Lines 192: "The MLP $\varphi(\cdot)$ has two layers, with hidden dimension G, followed by a PReLU activation."

Comment 5: The benchmark models are introduced in a few lines of text, and not easily understood. Also here, I suggest including an appendix that details these.

Answer 5: As requested also in point 3, we detailed all model architectures and benchmark models in Appendix A, including explanatory figures and equations.

Appendix A reads as follows:

“In this section, we further detail the different inputs and outputs, the hyperparameters, and the models’ architectures used in Section 5.1.

A1 Inputs, outputs, and hyperparameters

See answers 3 and 4

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, contrarily to the SWE-GNN, we stacked the node inputs into a single node feature matrix X = (Xd, Xs), which passes through an encoder MLP and then to the GNN.

Graph Convolutional Neural Network (GCN) employs the normalized Laplacian connectivity matrix to define the edge weights sij. The layer propagation rule reads as:

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

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

where I is the identity matrix, A is the adjacency matrix, which has non-zero entries in correspondence of edges, and D is the diagonal matrix.

Graph Attention Network (GAT) employs an attention-based mechanism to define the edge weights s_{ij} based on their importance in relation to the target node. The layer propagation rule reads as:

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

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

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

A3 CNN

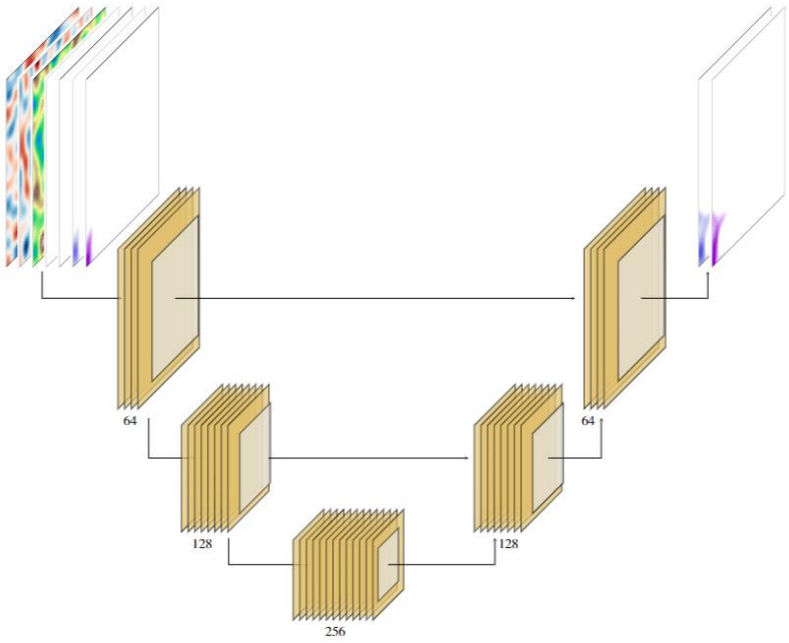


Figure A2. U-NET based CNN architecture employed in the experiments, with first embedding dimension of 64 and three encoding blocks. Each block is composed of one convolutional layer, followed by a batch normalization layer, a *PReLU* activation function, another convolutional layer, and finally a pooling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

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 theirs 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, i.e., 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:

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

where Yk is the output feature map for the k-th filter, X is the input image, Wk is the weight matrix for the k-th filter, $*$ denotes the 2D convolution operation, and σ is an activation function.”

Detailed comments:

Comment 6:line 106: a is already used as symbol for area, use v for velocity?

Answer 6: We corrected the symbol to *v*, as suggested by the Reviewer, to avoid confusion with the symbol for area.

Comment 7: Figure 2:

-top part of the figure

-mention what are the static inputs and the outputs in your work in the figure

-include a recursive arrow from output 1 back to dynamic inputs, to make it clear that the model prediction is recycled

-bottom part of the figure

-include the following captions above MLP, GNN and MLP: "process individual nodes", "process neighborhood of each node", "process individual nodes"

-consider referencing the equation numbers inside the figure, to make it clear which illustration relates to what step

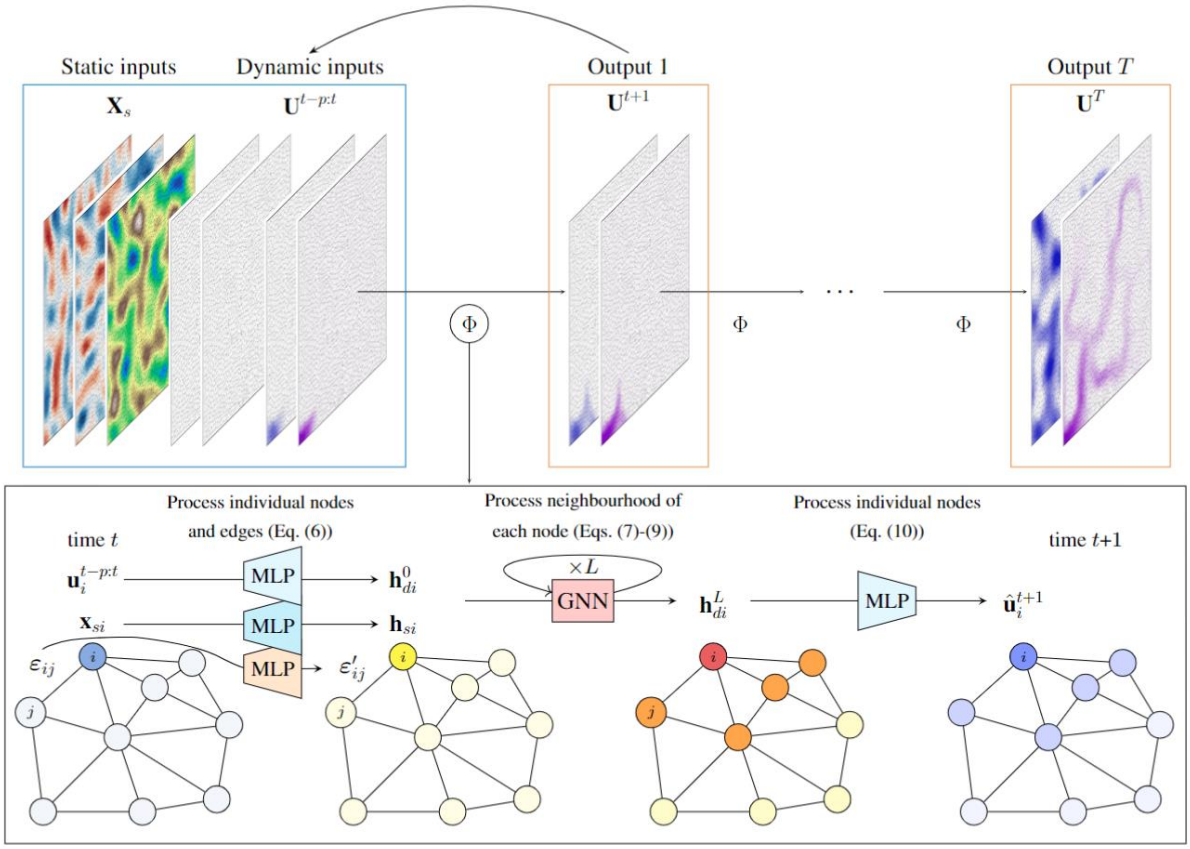
-include j in one of the orange circles

-edge feature encodings should receive and output arrow from the MLPs as well. As a whole, maybe this figure would be easier to understand if you distinguish h_si, h_di (why not h_dy?) and eps_ij separately (three sets of arrows)

-caption

-h_si and h_di are not clear from the caption (and explained somewhere much later in the main text)

Answer 7: We greatly thank the Reviewer for the suggestions. We modified figure 2 as follows:



We also modified the caption to better explain what h_si and h_di are. The caption of Figure 2 now reads as:

"… .The node inputs x_{si} and $u^{t-p:t}_i$ represent static attributes, such as elevation and slopes, and dynamic attributes, representing hydraulic variables, while the edge inputs ε_{ij} represent the mesh's geometry. The inputs are encoded into higher-dimensional embeddings h_{si}, h^0_{di} (yellow nodes), and ε′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. …"

Regarding the terminology used for h_di, we avoided using h_dy as the second subscript term, i.e., *i*, refers to node i, while d stands for dynamic. We clarified this by modifying lines 156-158 as follows:

"The ith rows of the node matrices Hs and Hd represent the encoded feature vectors associated to node i, i.e., hsi and hdi, and the kth rows of the edge matrices E′ represents the encoded feature vector associated to edge k."

Comment 8: line 138: explain that input features and hyperparameters will be explained in section 3.2

Answer 8: We modified lines 137-138 as suggested by the Reviewer as follows:

"In the following, we detail the proposed model (Section 3.1) and its inputs and outputs (Section 3.2). Finally, we discuss the training strategy (Section 3.3)."

Comment 9: line 145: Ut−p:t are the dynamic node features --> Ut−p:t are the dynamic node features (hydraulic variables) for time steps t-p to t

Answer 9: We corrected line 145 following the Reviewer's suggestion.

Line 145: "… Ut−p:t are the dynamic node features, i.e., the hydraulic variables for time steps t-p to t, …"

Comment 10: line 148: define I_epsilon

Answer 10: We have defined I_epsilon in line 151, just below it. Thanks for pointing this out.

Comment 11: line 154: explain that G is a hyperparameter

Answer 11: We thank the Reviewer for the comment. We explicitly mentioned that G is a hyperparameter in line 155-156 as:

"The encoders expand the dimensionality of the inputs to allow for higher expressivity, with the hyperparameter G being the dimension of the node embeddings."

Comment 12: Equation 7: At this point it would be really good to know how terrain differences come into play, and you don't explain it until Section 3.2. I think a short explanation is needed here, because it actually hinders the understanding of the methodology

Answer 12: We thank the Reviewer for the suggestion. Terrain differences come into play in the physical model but not explicitly in this deep learning model. Along with the other static variables and the hydraulic variables, however, they create an estimate of the contribution of the source terms used in the numerical models. This is linked with the interpretation of the MLP in Eq. 7 as a Riemann solver, according to which the output of the MLP is an approximation of the Jacobian of the fluxes. We added a short explanation on the role of function ψ in equation 7 in lines 171-172 as:

"The function ψ(·), instead, incorporates both static and dynamic inputs and provides an estimate of the source terms acting on the nodes."

Comment 13: line 184: is the activation function only applied on the final graph layer?

Answer 13: Yes, the nonlinearity is applied only in the final graph layer because preliminary experiments showed comparable or worse performances when using multiple activations.

Comment 14: line 191: Do the neural networks in the graph layers include bias terms? You refer to sparsity in multiple places in the paper, but you never explain it and why it is relevant (a large area of the image is 0 and does not need to be processed?)

Answer 14: Yes, the MLPs inside the graph layers (cfr. Eq. 7) use bias terms as most input terms in the function ψ already have non-zero values and thus excluding these

terms would have no marked positive influence. This is not the case for dynamic node features since adding a bias term would result in elements without water, and thus with only zero values, to be non-zero.

To clarify, we modified lines 194-196 as follows: "Both the MLPs in the dynamic encoder and the decoder do not have the bias terms as this would result in adding non-zero values in correspondence of dry areas that would cause water to originate from any node."

We also modified lines 204-205, that mentioned the term sparsity, as:

"The reason why we include w_i^t in the static attributes instead of the dynamic ones is that this features can be non-zero also without water, due to the elevation term, and would thus result in the same issue mentioned for the dynamic encoder and decoder."

Comment 15: Eq. 12: it is confusing that u is not the same vector as when introducing the SWE. I have no good suggestion for improving this though.

Answer 15: We thank the Reviewer for the suggestion. We clarified the difference between the u used in our deep learning model and that defined in Eq. 2. Lines 211-214 now read as follows:

"Contrarily 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)."

Comment 16: line 212: define unit normal vector (probably already needed in the context of Fig. 2)

Answer 16: We defined the unit normal vector in line 102 where it appears for the first time.

Comment 17: Section 3.3: Consider merging this with 4.2. Searching for the actual parameters used in training is yet another unnecessary complication

Answer 17: We understand the Reviewer's concern. However, we decided to keep the two sections separate as Section 3.3 describes a generic methodology that can be applied as well to different applications, while 4.2 describes the specific implementations used in the papers' setting.

Algorithm 1: Nice!

Thanks!

Comment 18: line 266: I don't understand why H is now fixed (previously, you introduced the curriculum). Is this the maximum of H considered?

Answer 18: We thank the Reviewer for comment. Indeed with this H we intended the maximum value that can be reached during training. Hence, we modified lines 271-272 as:

"We used a maximum prediction horizon H=8 steps ahead during training as a trade-off between model stability and training time."

Comment 19: Section 4.3: I think it would be interesting to see some illustrations of how e.g. MAE evolves over simulation time. This would help us understand better if the method is stable

Answer 19: We thank the Reviewer for the suggestion. We included a new figure (Figure 7) to represent how MAE, RMSE, and CSI evolve in time over the whole testing dataset. We added lines 359-363 as:

"We also observe the average performance of the different metrics over time, for the whole test dataset 1, in Figure 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 explode exponentially."

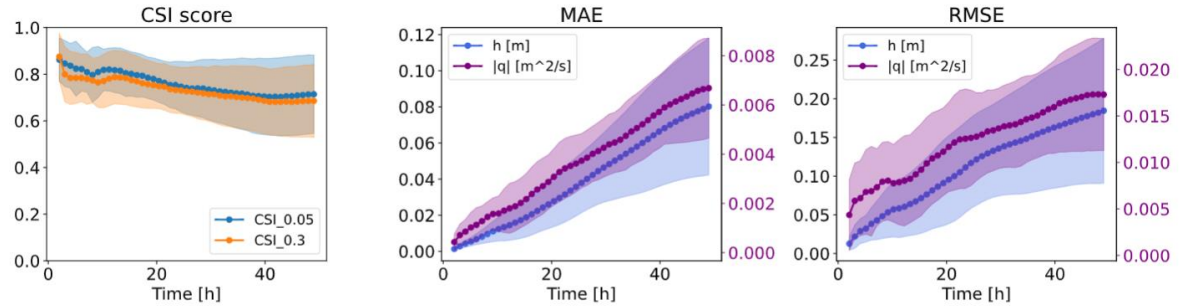


Figure 7. Temporal evolution of CSI scores, MAE, and RMSE for test dataset 1. The confidence bands refer to one standard deviation from the mean.

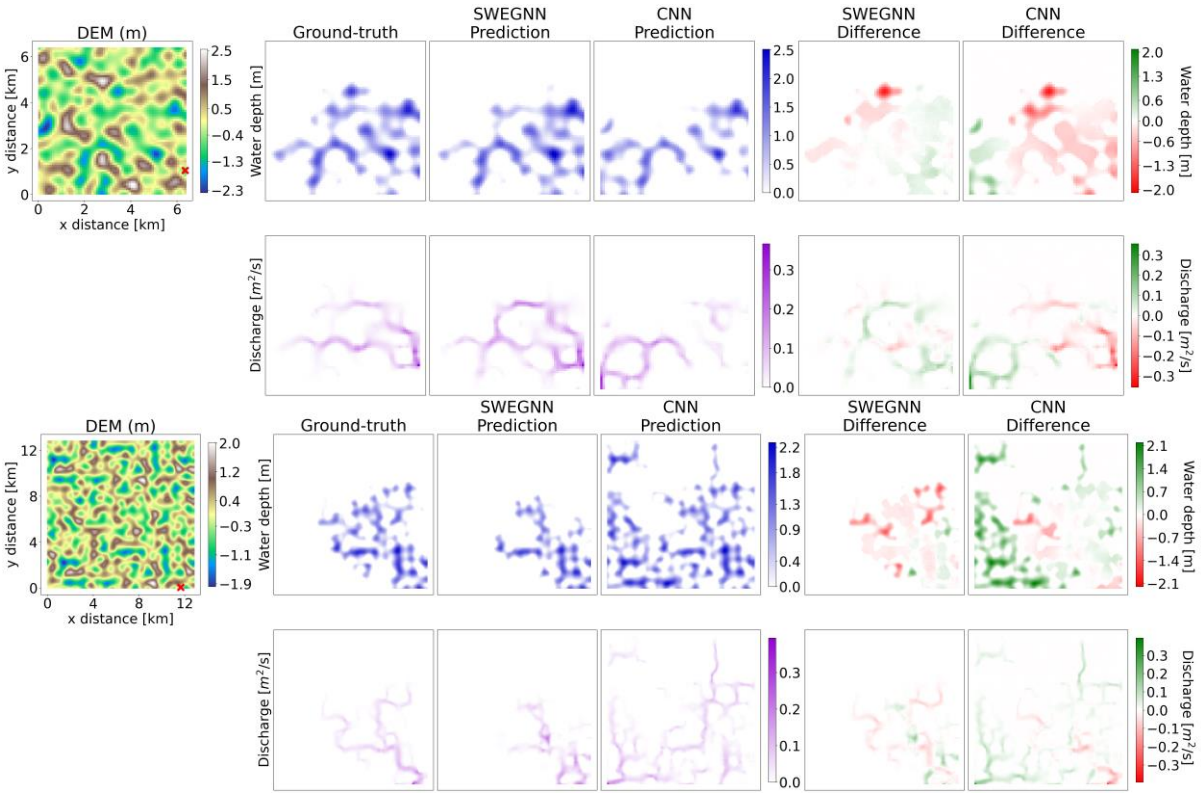
Comment 20: line 298: I'm not sure what propagation rule you refer to. As mentioned above, these model variations need to be documented better.

Answer 20: Since we added an appendix section with more details on the methods, we added the following sentences for both GAT and GCN in lines 305 and 307:

"For more details see Appendix A."

Comment 21: Figure 5: Show units also on the legends of the difference plots

Answer 21: We thank the Reviewer for the comment. We modified Figure 5 accordingly as:



Comment 22: Figure 7+8: From these figures, speedup in the order of factor 100 seems more realistic to me (not 600 as mentioned somewhere in the paper)

Answer 22: We agree with the Reviewer that for dataset 1 the order of 100 is a more realistic value than 600. The speedup range of 100-600 comes from running the same Pareto-front in Figure 8 on dataset 3. Here the model better exploits the bigger domain size and produces a higher range of speedup values. We included this figure, along with further details on it in Appendix B, which reads as follows:

"Appendix B: Pareto front for dataset 3

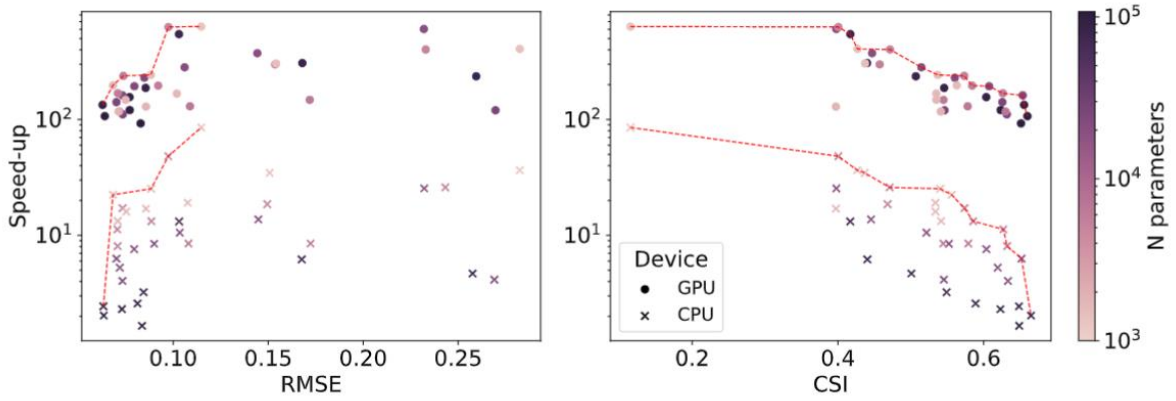


Figure B1. Pareto fronts on test dataset 3 (red-dotted lines) in terms of speed-ups, RMSE, and CSI for varying number of parameters for a temporal resolution Δt=1h.

We employed the models trained with different combinations of number of GNN layers and embedding size (Section 5.3) on test dataset 3. Figure B1 shows that the models performs better in terms of speed with respect to the smaller areas, achieving similar

CPU speedups and GPU speedups around two times higher than those in datasets 1 and 2.”