Water Research 290 (2026) 125079

Contents lists available at ScienceDirect

Water Research

journal homepage: www.elsevier.com/locate/watres

Evaluation of graph neural networks for urban drainage metamodeling: Key components and transferability analysis

Alexander Garzón [a,*], Zoran Kapelan [a], Jeroen Langeveld [a,b], Riccardo Taormina [a]

[a] Delft University of Technology, Stevinweg 1, Delft, 2628 CN, The Netherlands
[b] Partners4UrbanWater, Graafseweg 274, Nijmegen, 6532 ZV, The Netherlands

ARTICLE INFO

Dataset link: https://doi.org/10.4121/7408b922-3916-4cf2-9fad-afcbabb08bb4, https://doi.org/10.4121/5cf6c6fe-f0eb-4d48-978b-4b139113b8f2, https://github.com/alextremo0205/SWMM_GNN_Component_Evaluation_and_Transferability_Analysis

Keywords:
Sewer networks
Surrogate modeling
Deep learning
Transfer learning
SWMM

ABSTRACT

Simulating urban drainage hydraulics is computationally demanding, limiting its application in tasks that require real-time or repeated simulations. Graph Neural Networks (GNNs) are promising metamodels, but the effect of their internal components and transferability potential remain underexplored. This study addresses these gaps through two main contributions: (1) a systematic evaluation of key architectural components, including graph layer type, processor depth, and prediction window with links to physical transport dynamics; and (2) transferability experiments across domains (across two distinct drainage networks) and tasks (from head to flow prediction). As case studies, we selected two combined sewer networks in The Netherlands that differ in their hydraulic dynamics. We find that metamodels with moderate depth and a ten-step prediction window achieve high accuracy (RMSE of 2–5 cm for hydraulic heads and 0.02 m^3/s for flowrates). They also reach speed-ups of up to four orders of magnitude higher compared to the physics-based model, SWMM, when executing parallel simulations in GPU. Based on our two case studies, we find that pre-trained metamodels with full fine-tuning effectively adapt to a new task within the same domain, whereas cross-domain transfer requires appropriate normalization and fine-tuning. Furthermore, joint training on both case studies enables the metamodel to capture representations of both systems, suggesting potential for more general applicability. These findings demonstrate that metamodel architecture can reflect physical system behavior and offer practical guidance for building fast, accurate, and generalizable GNN-based metamodels—establishing a foundation for their use in applications such as uncertainty analysis, design optimization, and nowcasting.

1. Introduction

Urban drainage systems (UDSs) protect cities from flooding and manage wastewater, becoming increasingly critical with climate change and urbanization (Yazdanfar and Sharma, 2015; Chocat et al., 2007). While accurate simulation is essential for design and management, traditional hydrodynamic models like SWMM are computationally expensive for real-time or iterative applications (Seyedashraf et al., 2021; Mahmoodian et al., 2018a). This has motivated development of metamodels as efficient alternatives (Razavi et al., 2012), including data-driven Gaussian processes (Mahmoodian et al., 2018b) and hybrid approaches that combine simulator components with data-driven methods (Mahmoodian et al., 2018a).

Machine learning (ML) has emerged as a powerful approach for metamodeling urban water networks (Garzón et al., 2022), with researchers primarily using Multi-Layer Perceptrons (MLPs) for various applications—from predicting overflow volumes (Kim and Han, 2020) and water quality variables (Latifi et al., 2019) to simulating partial networks through disaggregation methods (Seyedashraf et al., 2021). However, these approaches often sacrifice spatial detail for computational efficiency. To address this limitation, Palmitessa et al. (2022) developed a fully distributed surrogate model using physics-guided generalized residue MLPs, which reconstruct hydraulic heads and flowrates throughout the entire system while achieving one to two orders of magnitude speed improvement over high-fidelity models.

Traditional ML metamodels like MLPs face significant limitations: they suffer from the curse of dimensionality (i.e., rapidly growing data and training demands with increased task or domain complexity) and lack structural flexibility for spatial generalization. Consequently, they typically require complete retraining for each network variation, undermining the efficiency goal of metamodeling. Graph Neural Networks (GNNs), however, incorporate domain knowledge through topological

* Corresponding author.
E-mail addresses: J.A.GarzonDiaz@tudelft.nl (A. Garzón), Z.Kapelan@tudelft.nl (Z. Kapelan), J.G.Langeveld@tudelft.nl (J. Langeveld), R.Taormina@tudelft.nl (R. Taormina).

https://doi.org/10.1016/j.watres.2025.125079
Received 4 August 2025; Received in revised form 8 November 2025; Accepted 28 November 2025
Available online 6 December 2025

inductive biases that leverage the inherent relational structure of UDSs reducing their training requirements.

Furthermore, their structure helps them generalize and transfer across different network configurations. Recent studies have demonstrated GNNs' effectiveness as efficient alternatives to physics-based models for UDSs. Zhang et al. (2024) proposed a GNN-based surrogate model with attention mechanisms and node-edge fusion for flow balance, achieving speed-ups of two orders of magnitude over SWMM. Li et al. (2024) developed a Graph-WaveNet model combining temporal and graph convolutions for UDS state forecasting, though accuracy decreases with longer windows and for upstream components. Both approaches focus on short-term forecasting during single rainfall events rather than complete simulations. A significant limitation is their omission of pipe characteristics in model inputs, restricting transferability across different networks or configurations. Garzón et al. (2024b) developed an auto-regressive GNN metamodel for stormwater systems that estimates hydraulic heads and was later extended using task transfer to predict flowrates (Garzón et al., 2024a). While demonstrating high accuracy and transferability to an unseen section of the same network, the similarity between these sections was high, and transferability to completely different drainage systems and joint training were not explored. The metamodel achieved only one order of magnitude speed-up compared to SWMM due to its step-by-step prediction mechanism. It also showed limitations in accurately modeling complex flow dynamics in upstream nodes and outflow points, and its lack of dry weather flow inputs restricted its applicability to stormwater-only systems.

Therefore, despite advances in GNNs for urban drainage modeling, two challenges remain: (1) limited performance, particularly in upstream areas affected by backwater effects, and modest computational speed-ups (Zhang et al., 2024; Garzón et al., 2024b); and (2) underexplored transferability, despite its importance for efficient metamodel development and deployment.

Hyperparameters play a critical role in the performance of machine learning algorithms, and their optimization remains an active field of research (Bischl et al., 2023). For GNNs, performance is particularly sensitive to architectural choices such as the type and number of layers, as demonstrated in domain-specific studies like cheminformatics (Ebadi et al., 2025). Traditionally, machine learning models — including surrogate or metamodels — apply hyperparameter optimization methods such as random search, Bayesian optimization, or evolutionary algorithms to identify configurations that achieve the highest accuracy for a given problem. However, while these approaches are effective for performance tuning, they offer limited understanding of how hyperparameters relate to the model's underlying mechanisms or the physical processes it represents. Recent studies have begun to bridge this gap by revealing theoretical connections between certain hyperparameters and physical principles. For instance, the number of graph layers has been linked to the Courant–Friedrichs–Lewy (CFL) condition governing numerical stability in PDE solvers (Tesán et al., 2026), suggesting that GNN architectures may implicitly encode physical constraints. Building on this perspective, our study adopts a more systematic approach to analyze how hyperparameters influence GNN metamodels and to suggest more generalizable insights.

Moreover, transferability leads to less development time by leveraging previous training efforts and examples, and opens the possibility for extendable or even generalizable metamodels. Transfer learning refers to the process of leveraging knowledge gained from one or more domains or tasks to improve learning in a related setting (Zhuang et al., 2020). Transfer can occur across domains addressing the same task (domain transfer) or across tasks within the same domain (task transfer). These, in turn, can be categorized by the order of training as sequential or simultaneous. In the sequential setting, a model pre-trained on one domain or task is adapted to another, reflecting a parameter-transfer approach. In the simultaneous setting, training examples from both domains or tasks are available, allowing a joint model to be trained, which corresponds to an instance-transfer approach. These two approaches correspond to two of the main categories of transfer learning according to the classification of Pan and Yang (2010), which is based on what knowledge is transferred. In water resources, transfer learning has been applied to improve urban flood susceptibility modeling (Zhao et al., 2021), short-term water demand forecasting (Li et al., 2025), and detecting bursts in water distribution systems (Glynis et al., 2023). These applications highlight transfer learning's potential to mitigate limited data availability and high computational costs—challenges also faced in developing surrogate models for UDSs, where knowledge reuse across networks or predictive tasks could substantially improve efficiency and generalization.

Our work addresses these two gaps by systematically analyzing how graph layer type, processor depth, and prediction window influence both accuracy and computational efficiency. This analysis identifies architectural choices that improve metamodel accuracy and speed. Additionally, we provide a comprehensive assessment of transferability, examining domain transfer across structurally diverse networks and task transfer between hydraulic variables. Our results show effective sequential transfer when adapting pre-trained metamodels to new drainage systems and hydraulic prediction tasks. Simultaneous transfer further demonstrates the feasibility of training a more general metamodel capable of handling multiple case studies or hydraulic variables concurrently, offering a step toward broader applicability. Using two Dutch urban drainage networks with distinct layouts and slopes as case studies, we demonstrate how pre-trained metamodels can be effectively adapted to other networks and hydraulic variables.

The main contributions of this paper are:

- Systematic exploration of key GNN architectural components for urban drainage metamodels, including graph layer types, processor depth, and prediction window.
- Demonstration of significant performance gains, achieving higher accuracy and speed compared to previous metamodels and numerical simulators.
- Empirical evaluation of metamodel transferability: (1) across structurally different drainage networks, and (2) across related hydraulic variables.
- Evidence that shared representations can be learned across case studies and tasks, suggesting potential for broader generalization with more diverse training.

2. Reference metamodel

We build on the GNN-based SWMM metamodel by Garzón et al. (2024b), which first demonstrated transferability across sections of an UDS using GNNs. This metamodel incorporates pipe attributes and future rainfall as inputs for cross-network generalization, and uses an autoregressive approach capable of simulating indefinitely long events, unlike earlier GNN metamodels limited to short-term predictions.

The chosen metamodel works in an auto-regressive manner: it predicts the system state at time $t+1$ using inputs from $t-p$ to t, being p the length of the input time series, and iteratively feeds its own outputs to forecast the full time series. Simulations begin from a known initial condition, which can also be assumed to be a dry state, i.e., all water depths and flows equal to zero.

For this study, we modified the metamodel to incorporate both dry weather flow and runoff. This allows the metamodel to better represent combined drainage systems instead of only stormwater systems. However, this modification does not significantly affect the behavior of the metamodel, since dry weather flow is typically an order of magnitude smaller than runoff.

The node inputs to the metamodel are noted as

$$\mathbf{X} = [\mathbf{H}_{t-p:t} \parallel \mathbf{R}_{t-p:t+1} \parallel \mathbf{z}], \tag{1}$$

where $\mathbf{H}_{t-p:t}$ is a matrix containing the time series of hydraulic heads at previous time steps from $t-p$ to t for all nodes. At the start of the simulation, the metamodel initializes the hydraulic heads on all nodes based on user-specified conditions. If no initial conditions are provided, zero depths are assumed by default. From this initial state, the metamodel predicts the next time steps and subsequently uses its own predictions as inputs in an autoregressive manner. $\mathbf{R}_{t-p:t+1}$ is the matrix containing the time series of lateral inflow (runoff and dry weather flow), from $t-p$ to $t+1$, $\mathbf{z}$ is the vector of node invert levels, and $\|$ denotes vector concatenation.

We note that $\mathbf{R}_{t-p:t+1}$ includes the future lateral inflow at time $t+1$. Lateral inflow is assumed to be known for all relevant timesteps. The metamodel focuses on hydraulic computation rather than hydrological prediction, as the former constitutes the main computational bottleneck (Palmitessa et al., 2022). In practical applications, known rainfall events can be converted into lateral inflows using simplified models, SWMM, or fully end-to-end through differentiable hydrology (Colleoni et al., 2025), thereby capturing the rainfall-runoff process.

For each pipe that connects node i with node j, we consider its diameter ($D_{(i,j)}$) and length ($l_{(i,j)}$), and represent them as a feature vector $\mathbf{e}_{(i,j)}$ defined as

$$\mathbf{e}_{(i,j)} = [D_{(i,j)}, l_{(i,j)}]. \tag{2}$$

Since the graph is modeled as undirected, we have $\mathbf{e}_{(i,j)} = \mathbf{e}_{(j,i)}$. In the computational graph representation, undirected connections are encoded using two directed edges (one for each direction) to support bidirectional message passing. We denote the set of feature vectors for all pipes connected to node i as $\mathcal{E}_i$:

$$\mathcal{E}_i = \{\mathbf{e}_{(i,j)} \mid j \in \mathcal{N}(i)\}, \tag{3}$$

where $\mathcal{N}(i)$ is the set of neighboring nodes connected to node i.

The metamodel follows an Encoder–Processor–Decoder architecture. First, it computes enriched node and edge representations using MLP-based encoders. These are passed to the graph processor, which outputs a high-dimensional representation of the hydraulic state, $\mathbf{X}_i''$, as follows:

$$\mathbf{X}_i'' = \text{GraphLayer}\left(\phi_{\text{node-enc}}(\mathbf{X}_i),\, \phi_{\text{edge-enc}}(\mathcal{E}_i)\right), \tag{4}$$

where $\phi_{\text{node-enc}}$ and $\phi_{\text{edge-enc}}$ are MLPs applied to the node and edge features, respectively, and $\mathbf{X}_i$ is the input feature vector of node i.

The predicted hydraulic head at node i and time $t+1$ is computed autoregressively as

$$\hat{h}_i^{(t+1)} = c_i\left(\lambda_{\text{nodes}} \cdot \phi_{\text{node-dec}}(\mathbf{X}_i'') + (1-\lambda_{\text{nodes}}) \cdot \hat{d}_i^{(t)} + z_i; z_i; g_i\right), \tag{5}$$

where $\hat{h}_i^{(t+1)}$ is the predicted hydraulic head, $c_i(\cdot)$ is a clamping function that applies a simple min-max operation to ensure physically valid values, restricting $\hat{h}_i^{(t+1)}$ to remain between the invert level z_i and the terrain elevation g_i, $\phi_{\text{node-dec}}$ is the decoder MLP, and $\lambda_{\text{nodes}} \in [0,1]$ is a trainable gating parameter that balances the influence of the decoder and the previous state.

Here, the predicted water depth at the previous time step is

$$\hat{d}_i^{(t)} = \hat{h}_i^{(t)} - z_i, \tag{6}$$

where $\hat{h}_i^{(t)}$ is the predicted hydraulic head at time t, and z_i is the elevation of node i.

Building on this, the prediction of flow magnitudes was introduced by Garzón et al. (2024a) as an extension of the head prediction task, using transfer learning to adapt the metamodel for flow estimation. In this work, we reproduce that approach in the tests of task transferability, where the predicted flow rate in the pipe between nodes i and j at time $t+1$ is given by:

$$\hat{q}_{(i,j)}^{(t+1)} = \lambda_{\text{flows}} \cdot \phi_{\text{flow-dec}}\left(\mathbf{X}_i'' \,\|\, \mathbf{X}_j'' \,\|\, \mathbf{E}_{(i,j)}\right) + (1-\lambda_{\text{flows}}) \cdot \hat{q}_{(i,j)}^{(t)}, \tag{7}$$

where $\hat{q}_{(i,j)}^{(t+1)}$ is the predicted flow magnitude, $\phi_{\text{flow-dec}}$ is the decoder MLP used to predict flow rates, $\hat{q}_{(i,j)}^{(t)}$ is the flow magnitude at the previous time step, and $\lambda_{\text{flows}} \in [0,1]$ is the corresponding trainable gating parameter. The network representation includes two directed edges per pipe, an abstraction commonly used in graph neural network models to encode undirected connections. This allows information to propagate in both directions along a pipe and helps the model represent bidirectional interactions such as backwater effects.

In both expressions, the λ values act as memory terms, inspired by time integration schemes used in ordinary differential equation solvers. Similarly to these schemes, the metamodel predicts one timestep at a time and then uses this value to continue predicting further timesteps. This allows it to predict indefinitely long time series.

3. Methods

3.1. Overview of the methodology

The methodology comprises two main aspects: component evaluation and transferability testing.

The component evaluation focuses on systematically varying hyperparameters of the metamodel architecture to identify the effect of these components on accuracy and computational speed when simulating hydraulic heads under varying rainfall conditions. All component evaluation experiments, as well as domain transfer tests, are conducted using metamodels trained solely on hydraulic heads. The overall architecture and these key components, which are central to how the metamodel learns and predicts UDS behavior, are highlighted in Fig. 1.

Transferability testing assesses whether metamodels developed for one drainage system or hydraulic variable can be applied to others. We distinguish between domain transfer and task transfer. Domain transfer examines prediction performance across drainage systems, with and without fine-tuning, and includes a multi-domain learning setup in which metamodels are simultaneously trained on the two case studies. In these experiments, we also used the metamodel trained exclusively on hydraulic heads.

Task transfer, in contrast, evaluates whether metamodels trained for hydraulic heads can be reused to predict pipe flowrates. For this, we extended the previously developed metamodels for hydraulic head prediction (Garzón et al., 2024b) to also predict pipe flow rates, following the approach described in Section 2. The architecture was augmented with an edge decoder, which takes as input the learned pipe (edge) embeddings and the processed node embeddings at both ends of each pipe, and outputs the corresponding flow magnitude, as shown in Eq. (7). The associated loss function during training includes both heads and flows, enabling the metamodel to learn the new hydraulic variable while retaining accuracy for heads.

3.2. Component evaluation

We focused specifically on three components with the highest potential for improving both accuracy and computational efficiency: (1) the type of graph neural network layer, (2) the depth of the graph processor, and (3) the prediction window.

We identified metamodel configurations through a two-phased approach: first, an initial grid evaluation across the combinations of parameters to assess the general landscape, followed by controlled experiments varying one hyperparameter at a time while holding others constant. Each configuration was trained with five random seeds to ensure robust assessment of performance differences.

For the exploratory search, we systematically evaluated combinations of graph layers (GATv2Conv and GINEConv), processor depths (1, 5, 10, 15), and predictive windows (1, 10, 20, 30, 50, and 100).

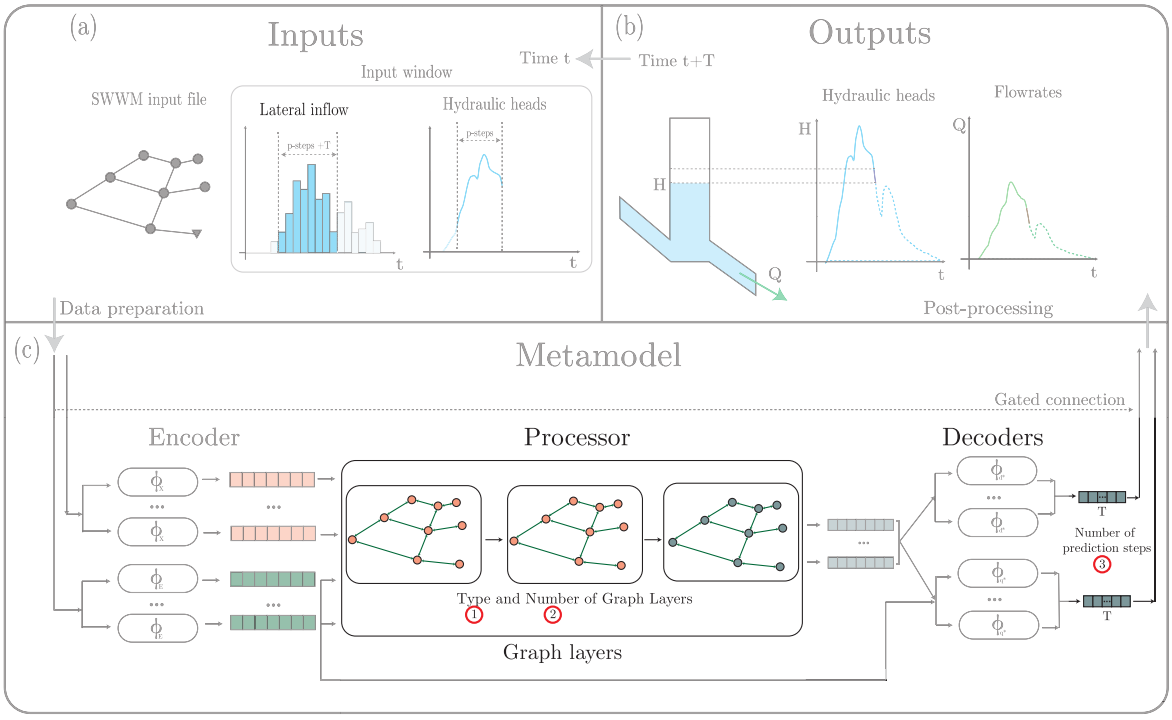


Fig. 1. Graph-based metamodel receiving system topology and partial time series of inflows and heads as input. The central Processor applies Graph Layers, and Decoders predict hydraulic heads and flowrates over time. The contrast in colors emphasize the modified components. Modified from Garzón et al. (2024a).

3.2.1. Type of graph layer

The choice of graph layers is crucial for effective information aggregation in UDSs, where both node and edge features play a role. This study evaluates two edge-aware graph layers that have been used in UDS metamodeling. The *Graph Isomorphism Network with Edge attributes* (GINEConv, Xu et al. (2019); applied by Garzón et al., 2024b,a) aggregates information from neighboring nodes and edges through a shared MLP. The *Graph Attention Network v2* (GATv2Conv, Brody et al., 2021) employs dynamic attention weights to adaptively prioritize connections based on their relevance, an approach that has been shown to be effective in UDSs (Zhang et al., 2024).

3.2.2. Depth of graph processor

The next component we varied was the depth of the graph processor, defined as the number of graph layers it contains. Instead of a single layer, stacking graph layer after graph layer results in a more complex model that handles further away connections. Considering $\mathbf{x}_i^{(l)}$ as an embedding of node i at layer l, where $l = 1, 2, \ldots, L$ and L is the total number of layers. The propagation rule for each layer can be expressed as:

$$\mathbf{x}_i^{(l+1)} = \text{GraphLayer}^{(l)}\left(\mathbf{x}_i^{(l)}, \{\mathbf{x}_j^{(l)} : j \in \mathcal{N}(i)\}, \{\mathbf{e}_{j,i} : j \in \mathcal{N}(i)\}\right) \tag{8}$$

where $\text{GraphLayer}^{(l)}$ is the layer-specific function, e.g., GINEConv or GATv2Conv, $\mathbf{x}_i^{(0)}$ represents the encoded node features, $\mathbf{e}_{j,i}^{(0)}$ represents the encoded edge features, and $\mathcal{N}(i)$ represents the neighbors of node i.

By stacking L layers, the final representation of node i is obtained as:

$$\mathbf{x}_i^{(L)} = \text{GraphLayer}^{(L)} \circ \text{GraphLayer}^{(L-1)} \cdots \circ \text{GraphLayer}^{(1)}\left(\mathbf{x}_i^{(0)}, \{\mathbf{x}_j^{(0)} : j \in \mathcal{N}(i)\}, \{\mathbf{e}_{j,i}^{(0)} : j \in \mathcal{N}(i)\}\right) \tag{9}$$

Here, $\circ$ is the composition operator, and the number of layers L determines the depth of the processor.

Graph processor depth critically impacts metamodel performance by enhancing compositional inductive bias, enabling more abstract representations through hierarchical processing. Deeper architectures allow information to propagate across multiple network hops (Gama et al., 2020), capturing complex hydraulic interactions beyond immediate neighbors. However, increasing depth introduces challenges like oversmoothing (Rusch et al., 2023) and computational overhead, creating an important trade-off in urban drainage modeling.

In our initial hyperparameter search, we tested graph processor depths of 1, 5, 10, and 15 layers. We then conducted a more refined exploration, evaluating depths from 1 to 15 in single-layer increments.

3.2.3. Prediction window

The prediction window refers to how many future timesteps the metamodel forecasts in a single forward pass. To enable this, we modified the decoder to predict K timesteps at once by extending its output dimension accordingly. The decoder equations are defined as:

$$\mathbf{x}''' = \phi_{\text{dec}}(\mathbf{x}_i^{(L)}, k), \quad k = 1, 2, \ldots, K \tag{10}$$

where $\mathbf{x}_i^{(L)}$ is the processed features of node i, and ϕ_{dec} is the decoding function. The decoder output for all K steps forms a matrix of size $n_{\text{nodes}} \times K$ for node decoders, and $n_{\text{pipes}} \times K$ for edge decoders.

This approach reduces the number of forward passes by a factor of K, improving computational efficiency while still enabling simulation of longer horizons through recursive feeding of predictions back as inputs. The metamodel moves forward through the time series in strides of K time steps, using the p historical values preceding each prediction window as inputs. To accommodate multi-step predictions, we extended the lateral inflow matrix $\mathbf{R}$ to include future values (from $t+1$ to $t+K$), ensuring the model has access to upcoming rainfall within the entire prediction window. When the total prediction length is not an exact multiple of K, the metamodel generates predictions for the next multiple of K steps and discards any extra steps beyond the requested horizon.

In our initial hyperparameter search, we evaluated prediction windows of 1, 10, 20, 30, 50, and 100 steps, followed by a refined exploration from 1 to 100 steps (in increments of 10). Though the

prediction window determines how many steps are forecast per forward pass, the metamodel can simulate indefinitely long horizons by autoregressively reusing outputs, retaining only the most recent inputs up to the metamodel's fixed input length.

3.3. Transferability testing

Using the best-performing configurations from the previous experiments, we assessed metamodel transferability along two dimensions: domain and task transferability. Domain transfer aimed to transfer knowledge from one drainage system to another, while task transfer focused on transferring knowledge gained from predicting hydraulic heads to predicting flowrates.

For both domain and task transfer, we further subdivided the experiments according to the learning setup: *sequential learning*, in which the model is trained on one domain or task before adapting to the target, and *simultaneous learning*, in which the model is trained jointly on multiple domains or tasks. This subdivision allows us to disentangle different mechanisms of transfer learning: sequential learning evaluates how effectively representations learned on a source domain or task can be reused and the extent of fine-tuning required, while simultaneous learning assesses the benefits of joint learning, where the model leverages shared structure across domains or tasks. Together, these setups provide insight into the relative contributions of pre-trained representations, additional data exposure, and joint learning to improvements in the target domain or task.

In the experiments, we used either five different pre-trained metamodel weights or five different random seeds for robustness, depending on the specific requirements of each experiment.

3.3.1. Domain transferability

To evaluate the effectiveness of domain transfer, we tested the following settings:

- **Sequential learning**: The metamodel is trained on one case study (the *source domain*) and tested on another (the *target domain*).
  - **Zero-shot transfer**: The pre-trained metamodel is directly applied to the target case study without additional training. We further examined the effect of input normalization by testing two strategies: (i) using the normalizer fitted on the source domain, and (ii) using a normalizer fitted on the target domain.
  - **Fine-tuned in target**: The metamodel is initialized with pre-trained weights from the source case study and then further trained (fine-tuned) using data from the target case study.
- **Simultaneous learning**: The metamodel is trained using data from both case studies and subsequently evaluated in each individual case study.
  - **Jointly trained**: The metamodels are trained and validated on the union of the training and validation sets from both domains. Since the combined dataset is approximately twice as large, training for the same number of epochs effectively yields about twice as many model updates as in the single-domain setups.
  - **Fine-tuned in target**: After training the metamodels on the joint dataset, they are further fine-tuned using data from a single case study. This adapts the general metamodel to the specifics of the target case, improving specialization. However, this approach may also increase the risk of overfitting and reduce performance in the other case study.

3.3.2. Task transferability

To evaluate the effectiveness of task transfer, we tested the following settings:

- **Sequential learning**: Metamodels are first trained on the task of predicting hydraulic heads (the *source task*) and then fine-tuned on predicting flowrates (the *target task*). The weights from the hydraulic head model are used as initialization for flowrate training.
  - **Target-task fine-tuning**: only the parameters relevant for the target task are updated, i.e. the newly added decoder; all other weights remain fixed.
  - **Full fine-tuning**: all weights are updated during training.
- **Simultaneous learning**: Metamodels are trained jointly on both tasks from random initialization to assess whether shared representations can be learned for both tasks. Training is performed with the original and double the number of epochs to separate the effect of transfer from that of extended training and to match the total training time of the sequential transfer scenarios.
  - **Standard training**: trained for the same number of epochs as in the single-task experiments.
  - **Extended training**: trained for twice the number of epochs.

3.4. Training strategy

All metamodels were trained using curriculum learning (Soviany et al., 2022) and the AdamW optimizer (Loshchilov and Hutter, 2019), minimizing the Huber loss (Gokcesu and Gokcesu, 2021). The Huber loss balances mean squared error (MSE) and mean absolute error (MAE), behaving like MSE for small residuals to provide smooth gradients, and like MAE for large residuals to reduce outlier influence. This loss is defined as

$$\mathcal{L}_\delta(a) = \begin{cases} \frac{1}{2}a^2 & \text{if } |a| \le \delta \\ \delta(|a| - \frac{1}{2}\delta) & \text{otherwise} \end{cases} \tag{11}$$

where $a = y - \hat{y}$ is the residual, and δ controls the transition from quadratic to linear behavior. The loss is normalized over batch size and timesteps for consistent comparison. With normalized data, most residuals fall below δ, so the loss behaves like MSE in general. Early in training, however, when residuals can be larger, the MAE component becomes active, guiding the optimization robustly, while the MSE component ensures smooth updates as predictions approach the training labels.

When training for both hydraulic heads and flowrates, we incorporated a Huber loss term for flow prediction into the training objective as follows

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{nodes}} + \mathcal{L}_{\text{edges}}. \tag{12}$$

During each epoch, the metamodel sweeps through every training simulation one timestep at a time. At each timestep, it takes the previous p timesteps as input and predicts a sequence of future values. This process effectively transforms each simulation into multiple training examples, one per timestep in the time series, excluding the initial p timesteps. Near the end of each simulation, the length of the future sequence is shortened to ensure that the metamodel does not exceed the available time series of the training simulations.

Curriculum learning was implemented by progressively increasing the length of the predicted sequence over training epochs. At the start of training, the metamodel was required to predict a timeseries as long as its predictive window, allowing it to first learn short-term forecasting. As training progressed, the prediction horizon was gradually extended, encouraging the metamodel to capture longer temporal dependencies

Table 1
Common hyperparameters used for all the experiments.

Model Hyperparameters	
Steps behind	9
Hidden dimension	32
MLP hidden layers	1
Non-linearity	PReLU[a]
Training Hyperparameters	
Learning rate	0.001
δ (Huber-loss)	1.0
Maximum weight decay (AdamW)	0.01
Batch size	20
Epochs	25
Curriculum update frequency	Every epoch
Initial prediction window (curriculum learning)	Number of prediction steps
Increase in steps ahead (curriculum learning)	Number of prediction steps
Maximum prediction window (curriculum learning)	400 steps

[a] While PReLU is listed here, the internal structure of the graph layers use other nonlinearities which were kept consistent with their original definitions.

and to maintain stability across consecutive predictions. During inference, such as in validation and testing, the metamodel is capable of processing time series of arbitrary length.

Model selection was based on the minimum validation loss observed during training, which identifies the point of best generalization before overfitting—equivalent to early stopping. While this metric is useful for selecting model weights, it is specific to the case study dataset, so we also employ additional metrics to compare performance across experiments and cases evaluated over the test sets.

We trained all models using a batch size of 20 events. Because the batch is defined at the event level, the number of prediction windows per batch varies depending on the length of the events. While larger batch sizes could accelerate training through increased parallelism, they tend to converge to sharp local minima with poorer generalization (Keskar et al., 2017). This conservative choice underutilized GPU memory during training, though larger batch sizes during inference enable simultaneous predictions for more rainfall events.

We fixed the remaining metamodel and training hyperparameters as defined in Garzón et al. (2024b). These include the number of steps behind (input sequence length), hidden dimension for embeddings, the number of hidden layers in the MLPs, and the activation function. MLPs are used throughout the architecture—in the encoder, decoder, and, when applicable, within the GINEConv layer. The specific values used in all experiments are listed in Table 1.

3.5. Performance metrics

3.5.1. Accuracy

Predictive accuracy is evaluated using four metrics: Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), Maximum Absolute Error (MaxAE), and the coefficient of determination (R^2). These metrics quantify different aspects of prediction error and are used to assess how closely the metamodel reproduces the target hydraulic variables.

The RMSE measures the average magnitude of the squared prediction errors, giving greater weight to larger errors:

$$\mathrm{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}, \tag{13}$$

The MAE provides a direct measure of the average error magnitude, treating all errors equally:

$$\mathrm{MAE} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|, \tag{14}$$

The coefficient of determination (R^2) measures how well the predictions explain the variance in the observed data:

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2}, \tag{15}$$

where $\bar{y}$ is the mean of the true values.

The MaxAE reports the single largest absolute error observed during an event at any node or timestep:

$$\mathrm{MaxAE} = \max_{i\in\{1,\ldots,N\}}|y_i - \hat{y}_i|, \tag{16}$$

In these equations, y_i is the true value, $\hat{y}_i$ is the predicted value, and N is the number of samples. In this context, the sample count is computed separately for hydraulic heads and flowrates, and corresponds to the total number of predicted values across all nodes or pipes and time steps.

For the coefficient of determination (R^2), to avoid numerical instability caused in time series with very low variance, we exclude timeseries of node hydraulic heads and pipe flowrates with variances lower than 10^{-4} and 10^{-6}, respectively. Many nodes exhibit nearly constant hydraulic heads or low flow variability, which can lead R^2 to penalize such nodes despite minimal absolute errors, resulting in low or even negative values. Thus, absolute metrics, which directly quantify physical prediction errors, provide more reliable assessments of metamodel performance.

All performance metrics are computed for each element's time series (hydraulic heads for nodes and flow rates for pipes) across all elements and all test events. Because the resulting distributions are often highly skewed, reporting simple averages could misrepresent the actual accuracy of the metamodels. Therefore, we summarize the distributions using quantile-based statistics.

3.5.2. Speed

The computational efficiency of a metamodel can be quantified through the speed-up and the number of simulations that need to be surrogated to compensate for the development of the metamodel, which accounts for both the time required to develop the metamodel and its runtime when replacing multiple simulations. Let $T_{\mathrm{reference}}$ denote the time required for a single SWMM simulation (assuming an average simulation time), and T_{dev} the total development time of the metamodel, including tasks such as data preparation, training, validation, and hyperparameter tuning. The term $T_{\mathrm{metamodel}}$ represents the effective time per simulation when using the metamodel. This value implicitly accounts for potential batching when multiple simulations can be executed in parallel within a single forward pass.

The break-even point (BEP) represents the number of simulations that need to be surrogated for the computational savings to compensate for the metamodel's development cost:

$$N_{\mathrm{BEP}} = \frac{T_{\mathrm{dev}}}{T_{\mathrm{reference}} - T_{\mathrm{metamodel}}}. \tag{17}$$

The maximum achievable speed-up, corresponding to the upper limit of efficiency, is given by

$$S_{\max} = \frac{T_{\mathrm{reference}}}{T_{\mathrm{metamodel}}}. \tag{18}$$

We use $S_{\max}$ as a reference metric to compare the theoretical maximum efficiency across different metamodel configurations. In contrast, the break-even point is used to assess the practical performance of the metamodel in the demonstration scenarios, accounting for development cost.

3.6. Technical specifications

Hydrodynamic simulations were performed using SWMM version 5.1.015 with the dynamic wave routing model. The metamodels were implemented and trained using Python 3.11.11, PyTorch (Paszke et al., 2019), and PyTorch Geometric (Fey and Lenssen, 2019). All training

experiments were conducted on GPUs, with performance improvements reported based on the GPU NVIDIA Tesla A100 (80 GB). Experiment tracking was managed using the Weights and Biases platform (Biewald, 2020).

4. Case studies

4.1. Urban drainage models

We selected two combined urban drainage networks previously studied and calibrated in Meijer et al. (2018). Both systems are located in The Netherlands and serve urban catchments of approximately 20 hectares. The systems convey stormwater and wastewater by gravity towards a downstream portion of the drainage system that is ultimately pumped out.

4.1.1. Tuindorp

Tuindorp, shown in Fig. 2(a), is part of the urban drainage network of Utrecht (Garzón et al., 2024b). The network has a looped structure with 345 pipes and 311 nodes, totaling 11.8 km in length. Pipe lengths range from 2.8 to 85 m, with diameters between 0.235 m and 1.35 m (0.45 m most common). Pipe slopes are generally flat, ranging from 0 to 0.09, with a median of 0.002. The network has a storage of approximately 14 mm and exhibits backwater effects, particularly in upstream nodes. Dry weather flows range between 2 and 8 L/s. Flows in the dataset can reach up to 2.5 m^3/s and depths up to 0.8 m.

4.1.2. Loenen

The Loenen network, shown in Fig. 2(b), serves a Dutch village of the same name. The system has a predominantly branched layout, consisting of 352 pipes and 337 nodes, totaling 12 km. Pipe lengths range from 1 to 91 m, with diameters between 0.125 m and 2.0 m (0.3 m most common). Pipe slopes vary between 0 and 0.52, with a median of 0.004. The network has a storage of approximately 10 mm. Dry weather flows range from 5 to 12 L/s. Flow conditions in the dataset can exceed 2 m^3/s, with depths reaching over 2.5 m. The outfall structure includes one main outlet and two combined sewer overflow (CSO) points regulated by weirs.

Although the network includes weirs, which are not explicitly represented in the current metamodel architecture, the Loenen case study was chosen to assess the metamodel's performance in the presence of such elements. These weirs have limited influence on the overall hydraulic behavior during most simulated timesteps but become active during sewer overflow conditions, affecting both hydraulic heads and flow rates.

To estimate how far water can travel within a single time step between nodes, we simulated a heavy synthetic rainfall event (60 mm/h for 60 min) in both UDSs. The maximum number of hydraulically connected upstream nodes within one 60-sec step was five in both networks. However, Loenen shows a higher average connectivity, with many nodes influenced by three or more upstream nodes, whereas most nodes in Tuindorp have neighborhoods smaller than three. This analysis, while neglecting backwater effects, illustrates differences in flow propagation and network connectivity under high-flow conditions.

4.2. Simulation databases

The dataset of events consists of a mix of real and synthetic rainfall events. The real rainfall data were derived from the radar rainfall dataset of 5-minute precipitation depths at a 1-km grid from the Royal Netherlands Meteorological Institute (KNMI, 2022). We selected the real events following Garzón et al. (2024b) such that each time series included at least one rainfall peak exceeding 10 mm and a subsequent dry period of five hours. The synthetic events were generated using the alternating blocks method (Te Chow et al., 1988) with variable intensities (up to 130 mm) and durations (1-34 h) to complement the real events and expand the range of extreme conditions.

For each case study, we used 160 rainfall events (125 real and 35 synthetic), divided into training (80 real, 20 synthetic), validation (15 real, 15 synthetic), and testing (30 real) subsets. The real events capture naturally occurring rainfall patterns, while the synthetic ones emulate design storms. All rainfalls were assumed to be uniformly distributed over the catchment. For training and validation, we did not include dry weather flow, allowing the metamodel to learn from the full range of depths and flows during wet weather conditions. For testing, we included both wet and dry weather flow to evaluate the metamodel's performance under these conditions and its ability to generalize. Table 2 provides an overview of the main rainfall and simulation characteristics. Detailed values for individual characteristics, including maximum and mean intensity, total depth, duration, and simulation time, are presented in Tables S3–S8 of the Supplementary Materials.

For both Tuindorp and Loenen, we employed SWMM models adapted from the InfoWorks models in Meijer et al. (2018). Each rainfall event was simulated independently, resulting in 160 simulations per case study. We used a routing time step of 1 s and a reporting interval of 1 min. All simulations were run with a single computational thread to ensure consistent runtime comparisons; running and processing the complete set of simulations for each case required approximately two hours. Additional information on simulation parameters is provided in the Supplementary Materials (Tables S1 and S2).

5. Results

5.1. Component evaluation

This section presents the results of the initial grid evaluation of the metamodels, which focused exclusively on predicting hydraulic heads. Building on these results, we then examine the effects of graph type and depth together, followed by an evaluation of different prediction window lengths. Finally, we summarize the best-performing architectures identified through these experiments.

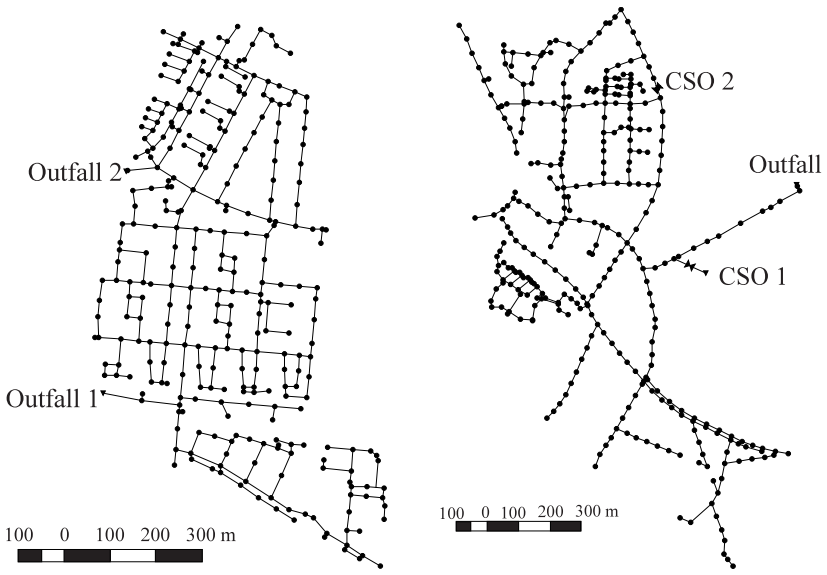
5.1.1. Initial grid evaluation

An initial grid search explored combinations of graph layer type, processor depth, and prediction window. As shown in Fig. 3, both case studies achieved minimum validation losses around ten prediction steps, independent of the graph configuration. This consistent minimum guided subsequent experiments to fix the prediction window at ten steps while systematically varying layer type and depth.

5.1.2. Type and depth of graph processor

A focused evaluation with a fixed ten-step prediction window revealed that for both systems, the metamodels achieved their best performance with GINEConv layers and moderate processor depths—four layers for Tuindorp (approximately 16.5k parameters) and five for Loenen (approximately 18.6k parameters), as illustrated in Figs. 4(a) and 4(b). These results indicate that deeper GNNs are more effective, which reflects the impact of reinforcing the relational and compositional inductive biases in the metamodels, allowing it to capture interactions across distant node neighborhoods (Battaglia et al., 2018). Furthermore, the effective graph depth matches with the spatial extent of hydraulic influence within each network, which agrees with previous findings on the effect of GNN depth for solving PDEs (Tesán et al., 2026). In this case, increasing depth initially improved accuracy but eventually degraded performance due to oversmoothing and higher training cost.

In terms of computational speed, Figs. 4(c) and 4(d) show that the metamodel speed-up decreases exponentially with increasing GNN depth, reflecting the additional sequential operations required for message passing. The speed up is highly influenced by the batch size. Using a batch size of 30 (the full validation dataset), simulations ran roughly

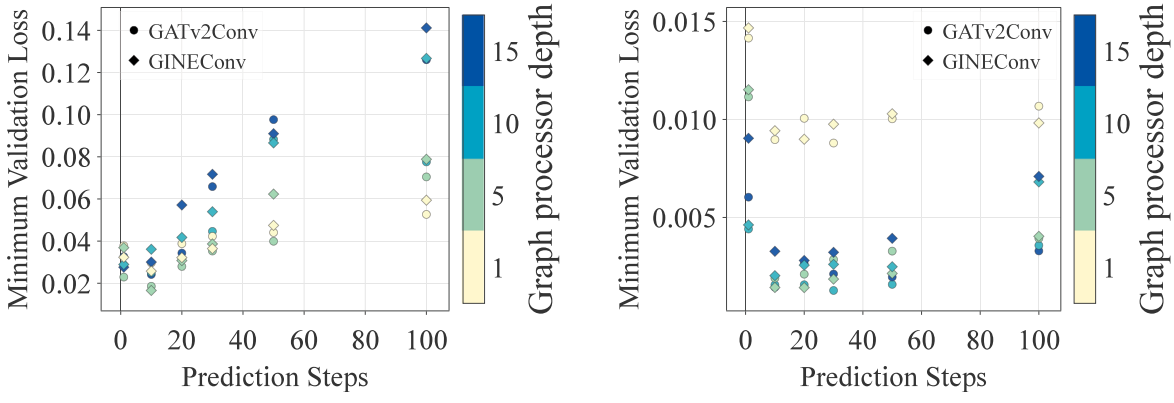


(a) Tuindorp urban drainage network (b) Loenen urban drainage network

Fig. 2. Layouts of the two urban drainage networks used as case studies. CSO stands for combined sewer overflow.

Table 2
Characteristics of rainfall events used in the simulation databases.

Characteristic	Tuindorp	Loenen
Total number of events	160 (125 real, 35 synthetic)	160 (125 real, 35 synthetic)
Temporal resolution	5 min	5 min
Duration range	2–64 h (median: 8.75 h)	1–20 h (median: 2.9 h)
Peak intensity range	10.63–345 mm/h (median: 36.48 mm/h)	10–168 mm/h (median: 44.13 mm/h)
Total rainfall depth range	1.5–327 mm (median: 17.56 mm)	0.83–277.78 mm (median: 39.13 mm)
Total simulated time (sum of all event durations)	3426 h (2041/523/861)	2913 h (1929/478/504)
Time to run simulations (wall-clock time)	90 min	26 min

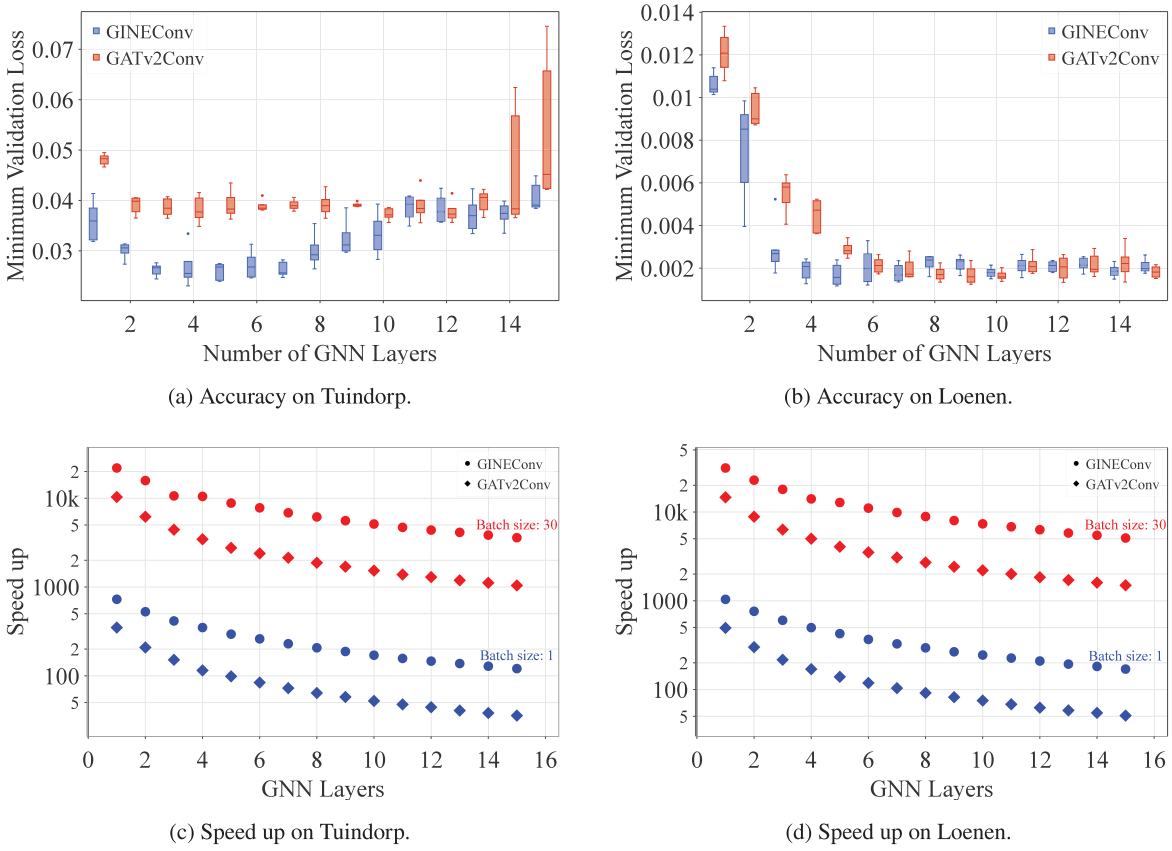


(a) Tuindorp (b) Loenen

Fig. 3. Validation loss as a function of prediction window length for both case studies and graph configurations.

28 times faster than when the metamodel was executed sequentially (batch size of 1). In terms of type of graph layer, GINEConv consistently achieved faster inference than GATv2Conv — more than two times faster — owing to its simpler aggregation scheme and roughly 25% fewer trainable parameters.

The observed difference in speed-up between the two case studies is substantial, with the metamodel in Tuindorp running almost twice as fast as the one in Loenen. This discrepancy arises from differences in the simulation conditions of the numerical model, the physical characteristics of each system, and the duration of the simulated time series. Because the speed-up is defined relative to the runtime of the



(a) Accuracy on Tuindorp.

(b) Accuracy on Loenen.

(c) Speed up on Tuindorp.

(d) Speed up on Loenen.

Fig. 4. Performance comparison on Tuindorp and Loenen: (a,b) accuracy, (c,d) maximum speed up (S_{max}) in the validation set.

numerical simulator, it reflects not only the computational efficiency of the metamodel itself but also the runtime characteristics of the reference simulations, which can vary considerably under different drainage systems and simulation conditions.

5.1.3. Prediction window

We investigated the effect of prediction window length using the best-performing layers from the previous experiment: GINEConv models with four layers for Tuindorp and five for Loenen (Fig. 5). The expectation was that increasing the number of predicted steps would enhance computational efficiency at the cost of accuracy. This trend was largely observed; however, an improvement in accuracy occurred when increasing the window from one to ten steps. This suggests that short multi-step prediction windows enhance temporal consistency and stabilize learning, before error accumulation dominates at longer windows. Computational efficiency, meanwhile, increased almost linearly with window length, achieving speed-ups of up to four orders of magnitude compared with SWMM, surpassing previous one-step-at-a-time architectures (Garzón et al., 2024b; Palmitessa et al., 2022). Overall, a ten-step window provides a balanced configuration, offering both improved accuracy and computational gains.

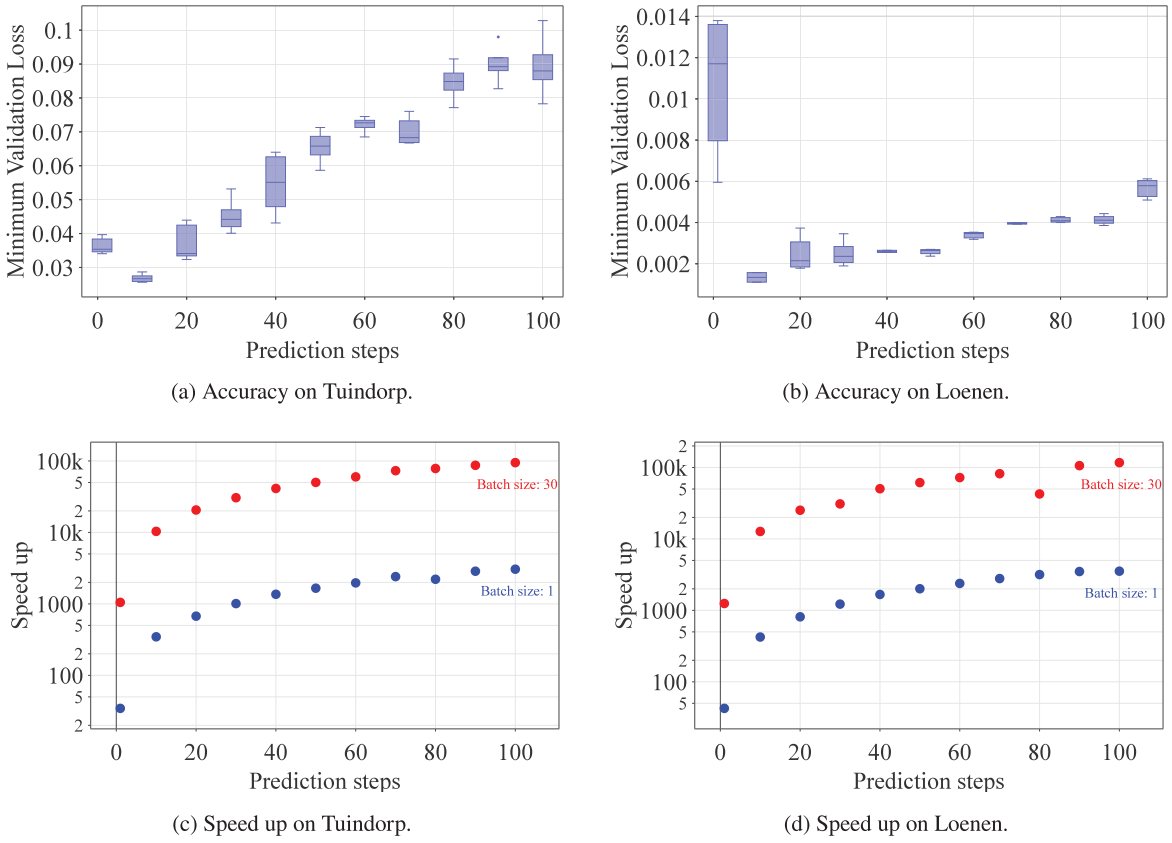
The observed pattern results from two opposing effects within the temporal bundling mechanism implemented in the decoder (Brandstetter et al., 2022), which predicts K consecutive time steps per forward pass. On the one hand, bundling improves numerical stability and temporal coherence by requiring the model to produce longer, temporally consistent sequences—effectively encouraging accurate yet stable predictions across multiple steps. On the other hand, as the window grows longer, prediction errors accumulate across successive steps, eventually offsetting these benefits and causing the gradual decline in accuracy observed for larger windows.

5.2. Summary of best-performing metamodels

The experiments described in the previous subsections were designed to investigate how different metamodel components affect predictive performance. From these analyses, the best-performing hyperparameter values were identified for each case study based on the lowest minimum validation loss. For each case, the selected metamodel corresponds to the configuration that achieved the lowest validation loss during training, balancing accuracy and generalization. Both models employed the GINEConv operator and were trained to predict 10 future timesteps. The architecture in Loenen comprised five graph layers, whereas the best-performing Tuindorp model used four.

Table 3 summarizes the performance of these metamodels in Tuindorp and Loenen, evaluated on their respective test datasets. Overall, the metamodels demonstrate high predictive accuracy across both case studies. In both systems, the median RMSE and MAE are below 2 cm, indicating a good absolute fit. The median R^2 in Tuindorp (85%) shows that the metamodel effectively reproduces temporal dynamics. In Loenen, this value drops to 66%, reflecting the greater difficulty in accurately reproducing time series in this network and the sensitivity of the metric. This can be attributed to the sloped nature of the system, which relies more on water transport process that is challenging for the metamodel to capture.

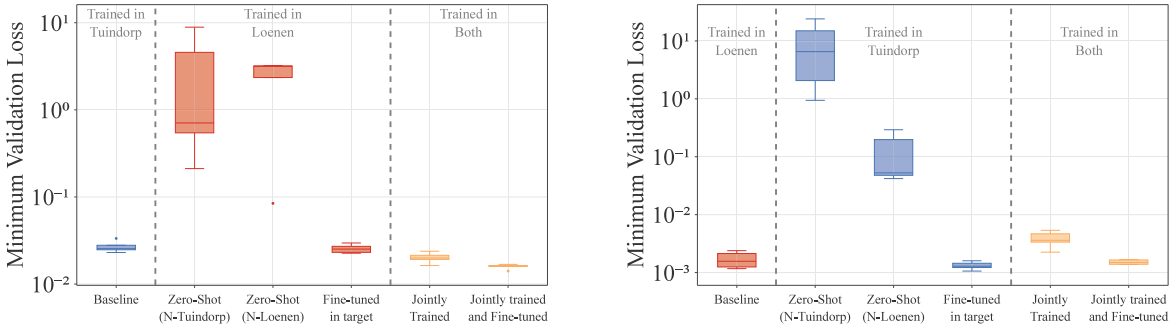
For MaxAE, most nodes exhibit errors below 10 cm, although maximum errors can reach up to 1 m, representing worst-case conditions for a small fraction of nodes. Spatially, these large errors primarily occur at nodes dominated by water transport, particularly where flow traverses multiple nodes without lateral inflow, while temporally, they arise during rapid hydraulic head changes, such as the filling and emptying of main peaks, when the metamodel can mistime the peak by a few minutes, either early or late. Distributions of all metrics across nodes



(a) Accuracy on Tuindorp. (b) Accuracy on Loenen.

(c) Speed up on Tuindorp. (d) Speed up on Loenen.

Fig. 5. Performance comparison on Tuindorp and Loenen: (a,b) accuracy, (c,d) maximum speed up (S_{max}) in the validation set.



(a) Tuindorp as target domain (b) Loenen as target domain

Fig. 6. Accuracy of metamodels under domain transfer. The evaluation domains are (a) Tuindorp and (b) Loenen. Each plot compares four configurations: the baseline (trained and evaluated on the same domain), two zero-shot transfer settings without retraining (trained on the other domain, using either Tuindorp or Loenen normalizers), and one configuration with fine-tuning on the target domain (using the target domain's normalizer). "N" denotes the origin of the normalizer.

and test events are reported in Figures S1 and S2 in Supplementary Materials.

These best-performing metamodels are used as the reference configurations in the subsequent sections, where their transferability is evaluated under different experimental conditions.

5.3. Transferability testing

In this section, we present the results of training the metamodels under different transfer learning settings. All models use the best-performing architectures described in Section 5.2. For domain transfer, the metamodels focus on predicting hydraulic heads, as in the previous

Table 3
Performance of the best metamodels in Tuindorp and Loenen. Median (Q1–Q3) over all nodes and simulations in the test sets.

Case study	RMSE [cm]	MAE [cm]	R^2	MaxAE [cm]
Tuindorp	1.4 (0.9–2.3)	1.0 (0.6–1.6)	0.85 (0.56–0.94)	6.7 (4.2–12.2)
Loenen	1.6 (0.9–3.5)	1.2 (0.7–2.4)	0.66 (0.21–0.85)	4.1 (2.2–9.7)

experiments, while for task transfer, the metamodels are extended to predict both hydraulic heads and flow rates.

5.3.1. Domain transferability

Fig. 6 presents the performance of metamodels under different scenarios of domain transfer. Figs. 6(a) and 6(b) are divided into three sections. The first sections of each figure show the performance of the best metamodels of each case study using training data from the very same case study; we use these values as the baseline in order to compare the performance of the other experiments. The second sections show the performance when doing a cross-domain transfer, this is, the metamodels were trained in one case study and then tested in the other. The third sections show the performance of the metamodels when simultaneously trained in the two domains. For this joint model, we selected a depth of five layers, which was the best-performing choice for Loenen, while for Tuindorp the difference between four and five layers is minimal, as shown in Fig. 4.

Overall, the zero-shot validation losses were substantially higher than in the other experiments, requiring a logarithmic scale on the y-axis for proper visualization. In contrast, the results when fine-tuning the metamodel with information from the target domain match the results of the baseline. Furthermore, when jointly training the metamodels in both case studies, the performance is better in Tuindorp and similar to the baseline in Loenen. In addition, when further training with the specific case study's information, the performance can be equal or even better than the baseline.

In particular, the zero-shot transfer between case studies generally resulted in losses two orders of magnitude higher than the baseline models. This is the only setting in which the metamodels do not train on examples from the target case study, this setting is analog to using the metamodel for extrapolation, which ANNs typically struggle with. The performance drop observed here was expected, given the substantial differences in topology, boundary conditions, and dynamics between Tuindorp and Loenen. Transfer learning is generally more successful when the source and target domains are more closely aligned, which was not the case in these experiments. We observed that fine-tuning restores performance to baseline levels, underscoring the importance of adapting the metamodels when transferring

We evidenced that while the metamodel does not transfer well across networks when trained on a single network, when trained in both networks, it manages to learn representations of both urban drainage systems. This setting is analog to using the metamodel for interpolation, which ANNs typically thrive on. Consequently, this opens the possibility for a more general metamodel that can learn and work in multiple urban drainage systems.

Table 4 reports the best metamodel accuracies under different domain transfer scenarios. Overall, the results confirm the trends observed in Fig. 6.

Notably, the jointly trained metamodels exhibited better performance metrics on the Tuindorp dataset compared to the Loenen dataset. This disparity stems from the difference in the magnitude of validation losses. Specifically, Loenen's larger range in elevations leads to smaller normalized depths and consequently, smaller loss magnitudes. Since the training process inherently prioritizes the larger losses (from Tuindorp), it adjusts less effectively to Loenen. This imbalance could be effectively mitigated by implementing a weighted loss function or, as detailed in Table 4, by fine-tuning the metamodel using Loenen-specific data.

5.3.2. Task transferability

Fig. 7 shows the validation losses for different approaches to task transference. Figs. 7(a) and 7(b) are divided into two sections. The first sections show the performance when doing a sequential transfer, this is, the metamodels were trained in the task of predicting hydraulic heads and then repurposed for also predicting flowrates. The second sections show the performance of the metamodels when simultaneously trained in the two tasks initializing from scratch the trainable weights. As the figures for each case study show the same main patterns, we analyze them jointly.

The first sections of the figures show that fine-tuning all the weights in the metamodel leads to better performance than freezing the pre-trained weights, as done in Garzón et al. (2024a). In this setting, the trainable parameters can adapt to the target data, allowing the metamodels to adjust their internal parameters in the graph layers to improve performance for both tasks. The second sections of the figures show that when simultaneously training on both tasks, the metamodels can perform better than training in a sequential setting but this requires more training effort and results in larger variability.

Table 5 reports the test performance metrics for all task transfer experiments for the best metamodel in each setting. Overall, the results are consistent with the trends observed in Fig. 7, but some differences arise between case studies. Tuindorp appears less sensitive to the transfer setting than Loenen, showing smaller variability across experiments and relatively stable R^2 values. In contrast, Loenen exhibits larger fluctuations in R^2, indicating that model generalization across tasks is more challenging in this case.

For hydraulic heads, both case studies reach performance levels close to their baselines, with most mean absolute errors below 3 cm and R^2 values between 0.5 and 0.9. For flow rates, Tuindorp shows relatively small errors (around 0.5 to 1 L/s MAE) and moderate R^2 values between 0.4 and 0.88, whereas Loenen exhibits larger deviations (0.7 to 7.2 L/s MAE) and a wider spread in R^2 values ranging from 0.5 to 0.9. These larger errors in Loenen can be attributed to the outlet pipe, where the metamodel must predict high discharge rates in a transport-dominated section of the network. This is a scenario in which it performs less accurately. Nevertheless, even in Loenen, these errors remain small compared with the maximum flow capacity of approximately 2 m^3/s.

Even though we kept the same metamodel complexity for comparison purposes, it is worth noting that the original architecture was focused for head prediction rather than the dual head-flow prediction task. An architecture specifically designed for this dual task, perhaps with additional GNN layers or enlarged hidden dimensions, might better accommodate the joint learning objective given the physical coupling between hydraulic heads and flowrates.

5.4. Metamodels demonstration

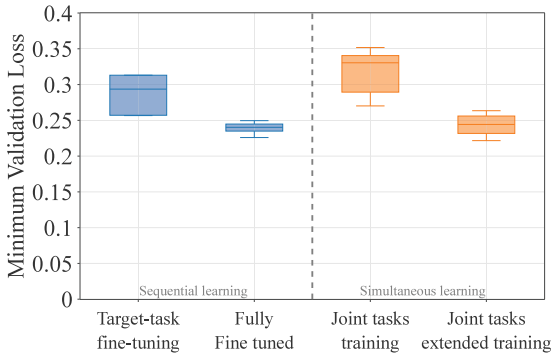
To illustrate the performance of the best found metamodels, Fig. 8 and Fig. 9 present their spatial and temporal responses during representative test events in Tuindorp and Loenen, respectively. In both case studies, the metamodels demonstrate strong predictive accuracy across the majority of nodes in the network. We used the extended metamodel developed through full fine-tuning in the task transfer experiment, as described in Section 5.3.2.

We selected representative nodes in each case study based on their hydraulic importance to demonstrate hydraulic head and flowrate predictions. In both systems, node A represents main-line locations where water accumulates from upstream. Nodes B highlight points of particular interest: in Tuindorp, the selected node is a branch node which occasionally presents backwater effect, while in Loenen it corresponds to the secondary weir. We added the predictions given by the GNN-based metamodel from Garzón et al. (2024a), referred to as the shallow metamodel, for comparison.

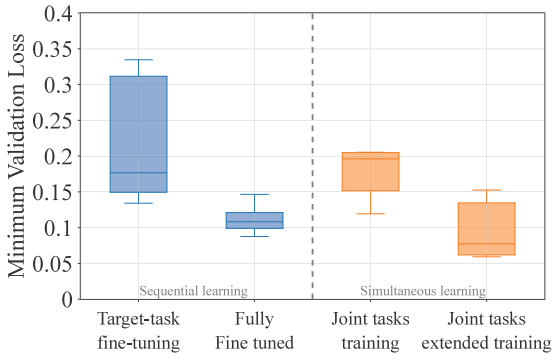
In Tuindorp, Fig. 8(a) shows consistently low errors across the network. For point A (Figs. 8(b), 8(d)) and point B (Figs. 8(c), 8(e)), both the proposed and shallow metamodels demonstrate good agreement with SWMM timeseries. Both metamodels accurately capture flow peaks, with their performance quality determined by the accuracy of nodal predictions. The shallow model tends to overpredict responses, while the deeper metamodel responds primarily to larger peaks, though these variations can occur due to the stochastic nature of training.

Table 4
Best metamodel performances in Tuindorp and Loenen in domain transfer settings. Median (Q1–Q3) over all nodes and simulations in the test sets.

Tuindorp				
Experiment	RMSE [cm]	MAE [cm]	R^2	MaxAE [cm]
Baseline	1.4 (0.9–2.3)	1.0 (0.6–1.6)	0.85 (0.56–0.94)	6.7 (4.2–12.2)
Zero-shot (Norm. Tuindorp)	4.9 (1.9–10.9)	2.8 (1.1–6.2)	−0.25 (−0.90–0.04)	22.2 (9.4–50.0)
Zero-shot (Norm. Loenen)	4.9 (2.1–11.6)	2.8 (1.1–6.4)	−0.49 (−1.25–0.16)	23.6 (10.2–57.9)
Fine-tuning	1.5 (0.9–2.5)	1.1 (0.6–1.8)	0.85 (0.55–0.94)	6.5 (4.1–12.9)
Jointly trained	1.9 (1.1–3.5)	1.3 (0.7–2.2)	0.72 (0.27–0.89)	9.1 (5.1–23.9)
Jointly trained (further fine-tuned)	1.4 (0.9–2.2)	1.1 (0.6–1.6)	0.87 (0.60–0.95)	6.1 (3.9–12.0)
Loenen				
Experiment	RMSE [cm]	MAE [cm]	R^2	MaxAE [cm]
Baseline	1.6 (0.9–3.5)	1.2 (0.7–2.4)	0.66 (0.21–0.85)	4.1 (2.2–9.7)
Zero-shot (Norm. Tuindorp)	5.1 (2.8–11.5)	4.3 (2.4–9.4)	−2.57 (−3.37–1.93)	10.0 (5.5–24.7)
Zero-shot (Norm. Loenen)	5.0 (2.6–14.3)	4.2 (2.2–10.3)	−2.75 (−3.92–2.02)	9.7 (4.9–41.8)
Fine-tuning	1.7 (1.0–3.3)	1.3 (0.8–2.4)	0.69 (0.33–0.85)	3.8 (2.4–8.9)
Jointly trained	2.1 (1.3–4.5)	1.7 (1.1–3.4)	0.37 (−0.38–0.68)	4.6 (2.9–12.0)
Jointly trained (further fine-tuned)	1.7 (0.9–3.7)	1.3 (0.7–2.7)	0.64 (0.23–0.82)	4.5 (2.4–9.9)



(a) The effect of different types of weight initialization in Tuindorp.



(b) The effect of different types of weight initialization in Loenen.

Fig. 7. Task transferability comparison for the case studies Tuindorp (a) and Loenen (b), showing the combined minimum validation loss for heads and flowrates across different weight initialization strategies.

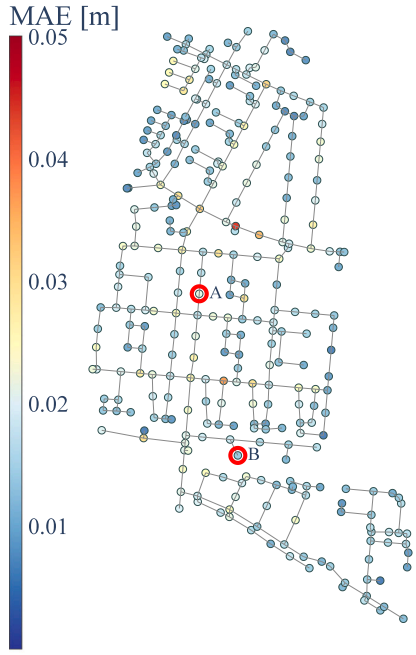
In Loenen, Fig. 9(a) shows strong agreement between the metamodel and SWMM across most of the network, with higher errors in downstream nodes of the main pipe—the most challenging prediction locations as they are furthest from runoff sources. For node A (Figs. 9(b), 9(d)), the metamodel accurately replicates SWMM's timeseries, while the shallow metamodel fails to capture the dynamics. At node B (Figs. 9(c), 9(e)), the deep metamodel captures the pattern of hydraulic heads but underpredicts their magnitude, likely due to information attenuation from distant upstream nodes and the lack of explicit weir modeling—the metamodel treats it as a regular pipe, so it cannot fully capture its hydraulic behavior. Although training helps partially compensate for this, it still struggles to reproduce the correct flow response, leading to systematic underestimation.

For demonstrating the functioning of the metamodels under the tested transfer settings, Figures S3–S14 in the supplementary materials show example time series and spatial performance for all domain transfer conditions, including hydraulic heads at nodes A and B, as well as nodes C and D (corresponding in Tuindorp to an additional backwater node and the main outlet, and in Loenen to upstream branches at different locations). For all task transfer experiments, Figures S15–S22 present the same type of plots for nodes A and B together with their respective downstream flow rates. These show that, although zero-shot settings did not generalize across networks, the transfer learning settings did not induce noticeable changes in the overall spatial or temporal patterns. Some settings, however, were more effective at extracting information from the training data or leveraging pretraining, which is reflected in improved predictions for the more challenging parts of the system—specifically, regions dominated by rapid water transport and nodes with backwater conditions.

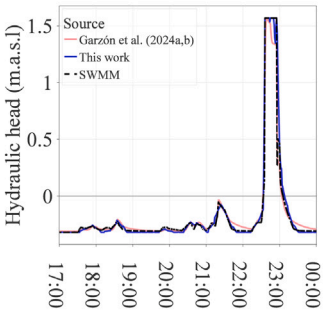
The speed-ups reach up to four orders of magnitude faster than SWMM while maintaining full spatial resolution. These improvements stem from the ability to predict multiple time steps in a single forward pass and to efficiently batch simulations on GPUs.

The break-even points (BEPs) were estimated by considering the dataset generation time (2 h) and an average training time of 4 h per metamodel. Given average SWMM simulation times of 22 s for Tuindorp and 12 s for Loenen, and metamodel simulation times of 50 ms per event (single simulation per batch), the BEPs were calculated as 984 simulations for Tuindorp and 1807 for Loenen. These values represent the number of simulations that must be replaced by the metamodel to offset the initial time investment. Even for these relatively fast simulations, only a modest number of runs is required to compensate for the development effort, which is typically achievable in optimization, uncertainty analysis, or real-time applications. However, the accuracy trade-off inherent to surrogate models should also be considered when evaluating their practical benefit in such contexts. The BEPs are highly sensitive to the SWMM simulation time, with slower simulations further increasing the benefit of using a surrogate model.

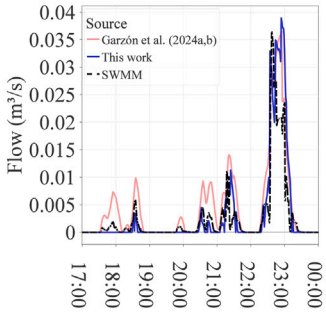
These results demonstrate that the metamodels not only perform well on validation data but generalize effectively to unseen test events, making them suitable for applications requiring rapid and repeated



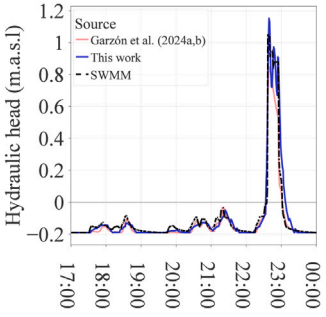
(a) Nodal distribution of RMSE for a rainfall test event in Tuindorp. The nodes plotted on the right are marked as A and B.



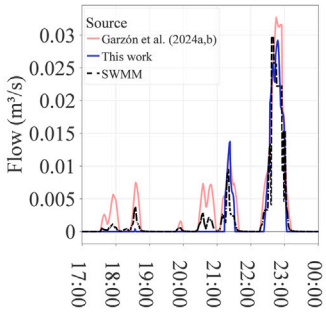
(b) Head prediction in A



(d) Flow predictions downstream A

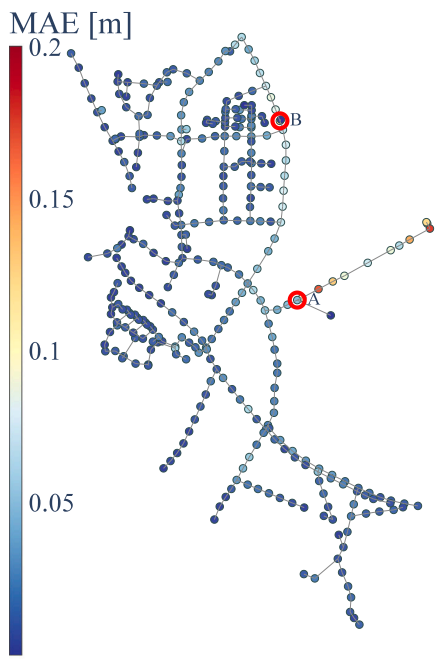


(c) Head prediction in B

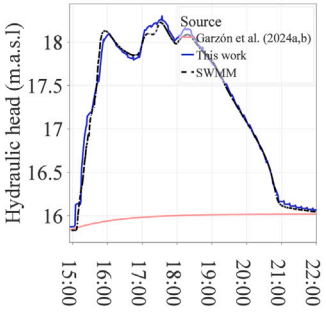


(e) Flow predictions downstream B

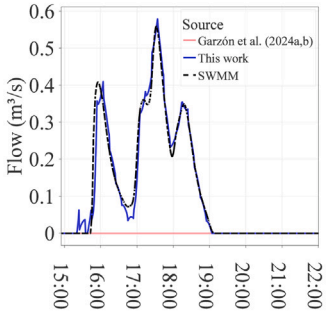
Fig. 8. Metamodel demonstration in Tuindorp.



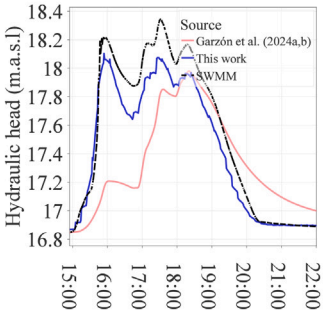
(a) Nodal distribution of RMSE for a rainfall test event in Loenen. The nodes plotted on the right are marked as A and B, respectively



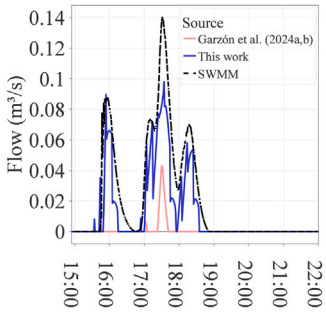
(b) Head prediction in A



(d) Flow predictions downstream A



(c) Head prediction in B



(e) Flow predictions downstream B

Fig. 9. Metamodel demonstration in Loenen.

Table 5
Best metamodel performances in Tuindorp and Loenen in task transfer settings. Median (Q1–Q3) over all nodes and simulations in the test sets.

Tuindorp – Heads				
Experiment	RMSE [cm]	MAE [cm]	R^2	MaxAE [cm]
Target-task fine tuning	1.4 (0.9–2.3)	1.0 (0.6–1.6)	0.89 (0.74–0.95)	6.7 (4.2–12.1)
Fully fine tuned	1.4 (0.9–2.3)	1.0 (0.6–1.6)	0.90 (0.76–0.96)	6.3 (3.9–12.1)
Joint tasks	1.6 (1.0–2.7)	1.2 (0.7–1.9)	0.86 (0.68–0.94)	7.0 (4.5–12.3)
Joint tasks (double epochs)	1.3 (0.9–2.3)	1.0 (0.6–1.6)	0.91 (0.78–0.96)	6.0 (3.8–11.2)
Tuindorp – Flowrates				
Experiment	RMSE [L/s]	MAE [L/s]	R^2	MaxAE [L/s]
Target-task fine tuning	1.3 (0.5–3.4)	0.5 (0.2–1.3)	0.62 (0.42–0.80)	10.4 (4.3–27.0)
Fully fine tuned	1.2 (0.5–2.7)	0.4 (0.2–1.0)	0.75 (0.52–0.88)	9.1 (4.0–22.8)
Joint tasks	1.3 (0.5–2.9)	0.4 (0.2–1.1)	0.71 (0.49–0.86)	9.7 (4.1–23.9)
Joint tasks (double epochs)	1.2 (0.5–2.7)	0.4 (0.2–1.0)	0.73 (0.52–0.88)	9.1 (4.0–22.2)
Loenen – Heads				
Experiment	RMSE [cm]	MAE [cm]	R^2	MaxAE [cm]
Target-task fine tuning	1.6 (0.9–3.5)	1.2 (0.7–2.4)	0.70 (0.35–0.87)	4.1 (2.2–9.7)
Fully fine tuned	1.5 (0.9–3.5)	1.2 (0.7–2.4)	0.73 (0.50–0.87)	3.8 (2.1–10.2)
Joint tasks	2.3 (1.3–4.9)	1.8 (1.0–3.7)	0.53 (−0.33–0.80)	5.8 (3.2–13.6)
Joint tasks (double epochs)	1.7 (1.1–3.2)	1.4 (0.8–2.4)	0.71 (0.41–0.86)	4.5 (2.6–9.7)
Loenen – Flowrates				
Experiment	RMSE [L/s]	MAE [L/s]	R^2	MaxAE [L/s]
Target-task fine tuning	3.6 (1.5–10.8)	2.4 (1.0–7.2)	0.54 (−0.15–0.81)	11.2 (4.5–34.5)
Fully fine tuned	2.3 (1.0–6.1)	1.6 (0.7–4.1)	0.85 (0.61–0.93)	7.7 (3.3–21.5)
Joint tasks	3.9 (1.4–10.0)	2.6 (0.9–6.7)	0.63 (0.03–0.84)	12.0 (4.1–31.7)
Joint tasks (double epochs)	3.0 (1.2–7.5)	2.0 (0.8–4.8)	0.79 (0.40–0.90)	9.3 (3.7–27.5)

simulations, such as preliminary scenario analysis, uncertainty quantification, flood risk screening, real-time nowcasting, and design optimization.

5.5. Limitations

While the proposed GNN-based metamodels show strong performance in modeling urban drainage systems, several limitations remain.

5.5.1. Component evaluation

The training process revealed complex interactions between hyperparameters. For example, the effectiveness of a particular graph layer type was influenced by network depth. However, due to computational constraints, several parameters — such as the number of hidden units or the input sequence length — were fixed, possibly leading to underperforming configurations.

We showed that GINEConv seems to work better than GATv2Conv, but these two architectures are not the only schemes to process graph-based information, and more recent and future types of graph layers could yield higher performance, especially if they are tailored to the underlying system that they are simulating, in this case, the UDS physics-based model.

We identified a relationship between metamodel performance as a function of graph depth and the maximum number of neighbors water traverses per timestep, linking this to the CFL condition, which reflects the drainage system's physical characteristics. Future interpretability analyses should explore similar relationships and assess how consistently metamodel hyperparameters are influenced by system attributes, ideally producing human-understandable explanations.

Such insights could accelerate metamodel development by guiding the pre-selection of high-performing hyperparameters and simultaneously improve generalization to new drainage systems, allowing interpretable relations to directly enhance transfer learning. Additionally, techniques like feature attribution and input sensitivity analyses can reveal which inputs drive the metamodel's predictions, aiding both researchers in model refinement and practitioners in building trust for real-world adoption.

5.5.2. Transferability

Our experiments demonstrate limited domain generalization through one-to-one case study transfer. While zero-shot transfer showed poor performance, fine-tuning pre-trained weights yielded better accuracy than training from scratch or partial retraining. This suggests that practitioners could develop system-specific base metamodels that can be adapted to their own cases or variables of interest.

It should be noted, however, that these findings are based on only two drainage systems with distinct but limited topological and hydraulic characteristics. As a result, the observed patterns may not fully capture the variability present in more diverse networks. While our results consistently indicate that transfer learning across domains requires careful normalization and fine-tuning, further studies on additional and more varied drainage systems are necessary to confirm the generality of these conclusions.

True generalization across diverse drainage systems requires training on many more networks to encompass broader structural and dynamic variability. To advance towards case-agnostic foundational metamodels, future research should explore many-source, many-target training approaches, as well as strategies to enhance robustness to differences in system characteristics, such as domain adaptation techniques or invariant feature representations. These strategies can be guided by the interpretability analyses suggested above, using insights about hyperparameter-system relationships and input sensitivities to identify features that remain consistent across systems and thus facilitate more reliable transfer.

The architecture used for the transfer task was designed for predicting hydraulic heads. Applying the same model complexity to multi-output tasks involving both heads and flows assumes a one-size-fits-all approach, likely limiting performance by not tuning the architecture for both tasks.

5.5.3. Modeling assumptions and simplifications

A key modeling constraint is the limited scope of UDS elements included in the metamodels. The current architecture handles only basic junctions and pipes, while excluding critical infrastructure such as pumps, tanks, and weirs. Although these components often operate locally, they can exert system-wide effects and are essential for capturing

realistic hydraulic behavior. For instance, the Loenen network contains two weirs that are not explicitly modeled; nevertheless, the metamodel was able to produce accurate predictions across the network in the simulated scenarios, suggesting some robustness to these unmodeled components.

Another limitation lies in the absence of embedded physical constraints, such as mass conservation. As the metamodels are purely data-driven, they may produce physically implausible predictions, especially in edge cases or under extrapolation. This limits their reliability in real-world operational settings, where compliance with physical laws is essential.

For the estimation of development time, we disregarded two components that can substantially affect the total effort: code development or adaptation for specific case studies, and hyperparameter tuning. These were excluded due to their high variability depending on the required adaptations and the allocated effort to optimize the metamodel. Integrating a surrogate model into an application pipeline typically requires additional coding and testing, while hyperparameter tuning involves repeated training under different configurations. We aim to reduce these overheads through the open-source release accompanying this publication and by illustrating the effects of the hyperparameters on metamodel performance.

Finally, this study also assumes spatially uniform rainfall as input, which simplifies the data requirements but reduces realism. Urban catchments frequently experience spatially heterogeneous precipitation, and this simplification can result in underestimated or inaccurate responses during localized storm events.

Taken together, these limitations underscore the need for improvements in system component modeling, enforcement of physical constraints, interpretability, hyperparameter optimization, and metamodel adaptability across tasks.

6. Conclusions

In this work, we evaluated an urban drainage metamodel based on graph neural networks (GNNs) with a dual objective: (i) to investigate how its components influence accuracy and computational efficiency, making the metamodel more practical for real-world applications, and (ii) to explore its potential for transfer learning, focusing on both domain transfer and task transfer, to assess opportunities for reducing development time. The key takeaways from this study are as follows:

- **Component evaluation** — We assessed three key components of a reference auto-regressive GNN metamodel: the type of graph layer, the depth of the graph processor, and the size of the prediction window. The choice of graph layer had a small but noticeable effect, with GINEConv performing slightly better than an attention-based layer. In terms of accuracy, increasing the graph depth improved performance up to a point — four layers for Tuindorp and five for Loenen — beyond which further layers provided little to no additional benefit. This behavior aligns with existing literature relating graph depth to flow dynamics and network structure, analogous to the CFL condition in numerical PDE solvers. In terms of computational performance, increasing the prediction window improved efficiency by reducing how often the metamodel needed to be run. Interestingly, predicting a few steps ahead gave slightly better accuracy than single-step prediction, but accuracy decreased as the prediction window became longer.
- **Domain transferability** — Zero-shot transfer of parameters across structurally different case studies resulted in low accuracy, indicating that fine-tuning with target case data is necessary. In contrast, simultaneous domain transfer successfully produced metamodels capable of handling both case studies at the same time, which could then be further fine-tuned for additional improvements. These results demonstrate that it is possible to learn more general metamodels, and that transferability can be further enhanced by training on a larger and more diverse set of networks.
- **Task transferability** — Pre-trained models for hydraulic head prediction could be successfully fine-tuned to also predict flow rates, highlighting the potential for efficient reuse of learned representations within the same domain. Comparable results were obtained when training jointly. Since the selected model architecture was based on a single task, more complex architectures may be better suited for multi-task learning.

For researchers, this work illustrates the capabilities of GNN-based models for urban drainage, highlighting design choices that optimize performance and the effectiveness of transfer learning strategies. Future work should consider integrating hydraulic structures such as pumps and weirs, enforcing physical constraints like mass balance, supporting spatially variable rainfall, employing automated architecture search, and relating hyperparameters to physical characteristics of the drainage system or the numerical model to further enhance generalization.

For practitioners, these advancements open new possibilities for tasks traditionally challenging for hydrodynamic simulators, including rapid scenario analysis, uncertainty quantification, flood risk screening, real-time nowcasting, and design optimization.

CRediT authorship contribution statement

Alexander Garzón: Writing – review & editing, Writing – original draft, Visualization, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Zoran Kapelan:** Writing – review & editing, Supervision. **Jeroen Langeveld:** Writing – review & editing, Supervision. **Riccardo Taormina:** Writing – review & editing, Supervision, Methodology, Funding acquisition, Conceptualization.

Declaration of Generative AI and AI-assisted technologies in the writing process

During the preparation of this work the authors used ChatGPT in order to improve the language and readability of the manuscript. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the content of the publication.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work is supported by the TU Delft AI Labs programme.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.watres.2025.125079.

Data availability

The dataset, which includes SWMM network configuration files, synthetic and observed rainfall time series, simulation outputs, experiment configuration files, and supporting machine-learning objects for both case studies, is available at https://doi.org/10.4121/7408b922-3916-4cf2-9fad-afcbabb08bb4. The code repository regarding this study is available at https://doi.org/10.4121/5cf6c6fe-f0eb-4d48-978b-4b139113b8f2 and https://github.com/alextremo0205/SWMM_GNN_Component_Evaluation_and_Transferability_Analysis.

References

Battaglia, P.W., Hamrick, J.B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., Gulcehre, C., Song, F., Ballard, A., Gilmer, J., Dahl, G., Vaswani, A., Allen, K., Nash, C., Langston, V., Dyer, C., Heess, N., Wierstra, D., Kohli, P., Botvinick, M., Vinyals, O., Li, Y., Pascanu, R., 2018. Relational inductive biases, deep learning, and graph networks. pp. 1–40, arXiv, URL: https://arxiv.org/abs/1806.01261.

Biewald, L., 2020. Experiment tracking with weights and biases. URL: https://www.wandb.com/.

Bischl, B., Binder, M., Lang, M., Pielok, T., Richter, J., Coors, S., Thomas, J., Ullmann, T., Becker, M., Boulesteix, A.-L., Deng, D., Lindauer, M., 2023. Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges. WIREs Data Min. Knowl. Discov. 13 (2), e1484. http://dx.doi.org/10.1002/widm.1484, URL: https://wires.onlinelibrary.wiley.com/doi/10.1002/widm.1484.

Brandstetter, J., Worrall, D., Welling, M., 2022. Message passing neural PDE solvers. pp. 1–27, CoRR abs/2202.0, arXiv:2202.0, URL: http://arxiv.org/abs/2202.03376.

Brody, S., Alon, U., Yahav, E., 2021. How attentive are graph attention networks? URL: http://arxiv.org/abs/2105.14491.

Chocat, B., Ashley, R., Marsalek, J., Matos, M.R., Rauch, W., Schilling, W., Urbonas, B., 2007. Toward the sustainable management of urban storm-water. Indoor Built Environ. 16 (3), 273–285. http://dx.doi.org/10.1177/1420326X07078854.

Colleoni, F., Huynh, N.N.T., Garambois, P.-A., Jay-Allemand, M., Organde, D., Renard, B., De Fournas, T., El Baz, A., Demargne, J., Javelle, P., 2025. SMASH v1.0: A differentiable and regionalizable high-resolution hydrological modeling and data assimilation framework. Geosci. Model. Dev. 18 (19), 7003–7034. http://dx.doi.org/10.5194/gmd-18-7003-2025, URL: https://gmd.copernicus.org/articles/18/7003/2025/.

Ebadi, A., Kaur, M., Liu, Q., 2025. Hyperparameter optimization and neural architecture search algorithms for graph Neural Networks in cheminformatics. Comput. Mater. Sci. 254, 113904. http://dx.doi.org/10.1016/j.commatsci.2025.113904, URL: https://linkinghub.elsevier.com/retrieve/pii/S0927025625002472.

Fey, M., Lenssen, J.E., 2019. Fast graph representation learning with pytorch geometric, (1). pp. 1–9, CoRR abs/1903.0, arXiv:1903.0, URL: http://arxiv.org/abs/1903.02428.

Gama, F., Isufi, E., Leus, G., Ribeiro, A., 2020. Graphs, convolutions, and neural networks: From graph filters to graph neural networks. IEEE Signal Process. Mag. 37 (6), 128–138. http://dx.doi.org/10.1109/MSP.2020.3016143.

Garzón, A., Kapelan, Z., Langeveld, J., Taormina, R., 2022. Machine learning-based surrogate modelling for Urban Water Networks: Review and future research directions. Water Resour. Res. http://dx.doi.org/10.1029/2021WR031808, e2021WR031808, https://onlinelibrary.wiley.com/doi/full/10.1029/2021WR031808, https://onlinelibrary.wiley.com/doi/abs/10.1029/2021WR031808, https://agupubs.onlinelibrary.wiley.com/doi/10.1029/2021WR031808.

Garzón, A., Kapelan, Z., Langeveld, J., Taormina, R., 2024a. Accelerating urban drainage simulations: A data-efficient GNN metamodel for SWMM flowrates †. Eng. Proc. 69 (1), http://dx.doi.org/10.3390/engproc2024069137.

Garzón, A., Kapelan, Z., Langeveld, J., Taormina, R., 2024b. Transferable and data efficient metamodeling of storm water system nodal depths using auto-regressive graph neural networks. Water Res. http://dx.doi.org/10.1016/j.watres.2024.122396.

Glynis, K., Kapelan, Z., Bakker, M., Taormina, R., 2023. Leveraging transfer learning in LSTM neural networks for data-efficient burst detection in water distribution systems. Water Resour. Manag. 37 (15), 5953–5972. http://dx.doi.org/10.1007/s11269-023-03637-3, URL: https://link.springer.com/10.1007/s11269-023-03637-3.

Gokcesu, K., Gokcesu, H., 2021. Generalized huber loss for robust learning and its efficient minimization for a robust statistics. URL: http://arxiv.org/abs/2108.12627.

Keskar, N.S., Mudigere, D., Nocedal, J., Smelyanskiy, M., Tang, P.T.P., 2017. On large-batch training for deep learning: Generalization gap and sharp minima.

Kim, H., Han, K., 2020. Urban flood prediction using deep neural network with data augmentation. Water (Switzerland) 12 (3), http://dx.doi.org/10.3390/w12030899.

KNMI, 2022. Precipitation - 5 minute precipitation accumulations from climatological gauge-adjusted radar dataset for The Netherlands (1 km) in KNMI HDF5 format. URL: https://dataplatform.knmi.nl/dataset/rad-nl25-rac-mfbs-5min-2-0.

Latifi, M., Rakhshandehroo, G., Nikoo, M.R., Sadegh, M., 2019. A game theoretical low impact development optimization model for urban storm water management. J. Clean. Prod. 241, http://dx.doi.org/10.1016/j.jclepro.2019.118323.

Li, Z., Liu, G., Liu, H., Savic, D., Fu, G., 2025. Usage of multisource data through transfer learning to improve short-term water demand forecasting. J. Water Resour. Plan. Manag. 151 (10), 04025053. http://dx.doi.org/10.1061/JWRMD5.WRENG-6950.

Li, M., Shi, X., Lu, Z., Kapelan, Z., 2024. Predicting the urban stormwater drainage system state using the graph-WaveNet. Sustain. Cities Soc. 105877. http://dx.doi.org/10.1016/j.scs.2024.105877, URL: https://linkinghub.elsevier.com/retrieve/pii/S2210670724007017.

Loshchilov, I., Hutter, F., 2019. Decoupled weight decay regularization. In: 7th International Conference on Learning Representations. ICLR 2019.

Mahmoodian, M., Carbajal, J.P., Bellos, V., Leopold, U., Schutz, G., Clemens, F., 2018a. A hybrid surrogate modelling strategy for simplification of detailed urban drainage simulators. Water Resour. Manag. 32 (15), 5241–5256. http://dx.doi.org/10.1007/s11269-018-2157-4.

Mahmoodian, M., Torres-Matallana, J.A., Leopold, U., Schutz, G., Clemens, F.H., 2018b. A data-driven surrogate modelling approach for acceleration of short-term simulations of a dynamic urban drainage simulator. Water (Switzerland) 10 (12), http://dx.doi.org/10.3390/w10121849.

Meijer, D., Bijnen, M.v., Langeveld, J., Korving, H., Post, J., Clemens, F., 2018. Identifying critical elements in sewer networks using graph-theory. Water (Switzerland) 10 (2), http://dx.doi.org/10.3390/w10020136.

Palmitessa, R., Grum, M., Engsig-Karup, A.P., Löwe, R., 2022. Accelerating hydrodynamic simulations of urban drainage systems with physics-guided machine learning. Water Res. 223 (May), http://dx.doi.org/10.1016/j.watres.2022.118972.

Pan, S.J., Yang, Q., 2010. A survey on transfer learning. IEEE Trans. Knowl. Data Eng. 22 (10), 1345–1359. http://dx.doi.org/10.1109/TKDE.2009.191.

Paszke, A., Lerer, A., Killeen, T., Antiga, L., Yang, E., Gross, S., Bradbury, J., Massa, F., Steiner, B., 2019. PyTorch : An imperative style , high-performance deep learning library. (NeurIPS), CoRR abs/1912.0, arXiv:1912.0, URL: https://arxiv.org/abs/1912.01703.

Razavi, S., Tolson, B.A., Burn, D.H., 2012. Review of surrogate modeling in water resources. Water Resour. Res. 48 (7), http://dx.doi.org/10.1029/2011WR011527.

Rusch, T.K., Bronstein, M.M., Mishra, S., 2023. A survey on oversmoothing in graph neural networks. URL: http://arxiv.org/abs/2303.10993.

Seyedashraf, O., Bottacin-Busolin, A., Harou, J.J., 2021. A disaggregation-emulation approach for optimization of large urban drainage systems. Water Resour. Res. 57 (8), 1–18. http://dx.doi.org/10.1029/2020WR029098.

Soviany, P., Ionescu, R.T., Rota, P., Sebe, N., 2022. Curriculum learning: A survey. Int. J. Comput. Vis. 130 (6), 1526–1565. http://dx.doi.org/10.1007/s11263-022-01611-x.

Te Chow, V., Maidment, D., Mays, L., 1988. Applied Hydrology. In: Civil Engineering, McGraw-Hill, URL: https://books.google.nl/books?id=cmFuQgAACAAJ.

Tesán, L., Iparraguirre, M.M., González, D., Martins, P., Cueto, E., 2026. On the under-reaching phenomenon in message passing neural pde solvers: revisiting the cfl condition. Computer Methods in Applied Mechanics and Engineering 449, 118476. http://dx.doi.org/10.1016/j.cma.2025.118476.

Xu, K., Jegelka, S., Hu, W., Leskovec, J., 2019. How powerful are graph neural networks? In: 7th International Conference on Learning Representations. ICLR 2019, pp. 1–17.

Yazdanfar, Z., Sharma, A., 2015. Urban drainage system planning and design - Challenges with climate change and urbanization: A review. Water Sci. Technol. 72 (2), 165–179. http://dx.doi.org/10.2166/wst.2015.207.

Zhang, Z., Tian, W., Lu, C., Liao, Z., Yuan, Z., 2024. Graph neural network-based surrogate modelling for real-time hydraulic prediction of urban drainage networks. Water Res. 263, http://dx.doi.org/10.1016/j.watres.2024.122142.

Zhao, G., Pang, B., Xu, Z., Cui, L., Wang, J., Zuo, D., Peng, D., 2021. Improving urban flood susceptibility mapping using transfer learning. J. Hydrol. 602, 126777. http://dx.doi.org/10.1016/j.jhydrol.2021.126777.

Zhuang, F., Qi, Z., Duan, K., Xi, D., Zhu, Y., Zhu, H., Xiong, H., He, Q., 2020. A comprehensive survey on transfer learning. http://dx.doi.org/10.48550/arXiv.1911.02685, arXiv, arXiv:1911.02685, arXiv:1911.02685.