Transferable and data efficient metamodeling of storm water system nodal depths using auto-regressive graph neural networks

Garzón, Alexander; Kapelan, Zoran; Langeveld, Jeroen; Taormina, Riccardo

DOI
10.1016/j.watres.2024.122396

Publication date
2024

Document Version
Final published version

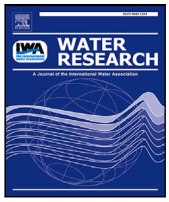
Published in
Water Research

Water Research 266 (2024) 122396

Contents lists available at ScienceDirect

Water Research

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



Transferable and data efficient metamodeling of storm water system nodal depths using auto-regressive graph neural networks

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/fec1e3de-9586-4a61-b3a1-02382592e52c, https://doi.org/10.4121/989a0d3d-3b4d-47c7-8677-31c5975f9dec, https://github.com/alextremo0205/SWMM_GNN_Repository_Paper_version

Keywords:
Urban drainage
Surrogate modeling
Machine learning
Deep learning
Transfer learning
SWMM

ABSTRACT

Storm water systems (SWSs) are essential infrastructure providing multiple services including environmental protection and flood prevention. Typically, utility companies rely on computer simulators to properly design, operate, and manage SWSs. However, multiple applications in SWSs are highly time-consuming. Researchers have resorted to cheaper-to-run models, i.e. metamodels, as alternatives of computationally expensive models. With the recent surge in artificial intelligence applications, machine learning has become a key approach for metamodelling urban water networks. Specifically, deep learning methods, such as feed-forward neural networks, have gained importance in this context. However, these methods require generating a sufficiently large database of examples and training their internal parameters. Both processes defeat the purpose of using a metamodel, i.e., saving time. To overcome this issue, this research focuses on the application of inductive biases and transfer learning for creating SWS metamodels which require less data and retain high performance when used elsewhere. In particular, this study proposes an auto-regressive graph neural network metamodel of the Storm Water Management Model (SWMM) from the Environmental Protection Agency (EPA) for estimating hydraulic heads. The results indicate that the proposed metamodel requires a smaller number of examples to reach high accuracy and speed-up, in comparison to fully connected neural networks. Furthermore, the metamodel shows transferability as it can be used to predict hydraulic heads with high accuracy on unseen parts of the network. This work presents a novel approach that benefits both urban drainage practitioners and water network modeling researchers. The proposed metamodel can help practitioners on the planning, operation, and maintenance of their systems by offering an efficient metamodel of SWMM for computationally intensive tasks like optimization and Monte Carlo analyses. Researchers can leverage the current metamodel's structure for developing new surrogate model architectures tailored to their specific needs or start paving the way for more general foundation metamodels of urban drainage systems.

1. Introduction

Storm Water Systems (SWSs) are vital infrastructures that provide environmental protection and flood prevention (Larsen and Gujer, 1997; Chocat et al., 2007). Climate change is intensifying the water cycle, affecting rainfall patterns, bringing more intense rainfall and associated flooding in many regions (IPCC, 2021). Thus, proper design, operation, and management of these infrastructures are of utmost importance. To achieve this, water utilities often rely on computational models that abstract the physical functioning of the system, enabling exploration of potential scenarios and adaptation alternatives. However, some critical applications, such as (re)design optimizations, real-time forecasts, and uncertainty analyses require computationally fast models. To address this challenge, researchers have developed surrogate models, which provide an alternative approach for efficient analysis and decision-making in SWSs.

Surrogate models are methods that approximate the response of a computationally demanding model with a cheaper-to-run, yet sufficiently accurate, version (Razavi et al., 2012). These surrogate models can be categorized in two groups: physically-based lower fidelity (PBLF) models and response surface (RS) metamodels. PBLB models involve employing coarser versions of the expensive model, exchanging fidelity for computational speed. Examples in the context of SWSs include 1D simplifications (Bermúdez et al., 2018; Leitão et al., 2010) and lumped systems (Dempsey et al., 1997; van der Werf et al., 2023). On the other hand, RS metamodels are approximators that map the input

* 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.2024.122396
Received 22 February 2024; Received in revised form 26 July 2024; Accepted 3 September 2024
Available online 11 September 2024

variables to the outputs generated by the original model. Examples of response surface metamodels include function fitting (Schultz et al., 2004; Mahmoodian et al., 2018) and supervised machine learning (ML) algorithms (Palmitessa et al., 2022; Luo et al., 2023). Supervised ML algorithms use input–output pairs for training, mirroring the operation of response surface metamodels. This resemblance, combined with the recent advancements in Deep Learning (DL), has made artificial neural networks (ANNs) the most popular ML-based metamodels for water networks (Garzón et al., 2022). In particular, multi-layer perceptrons (MLPs) are the most widely used ANNs. Nevertheless, due to their fully connected architecture, they require considerable amounts of training data which exponentially grows as the complexity of the problem increases. This is known as the curse of dimensionality (Keogh and Mueen, 2017). This may defeat the primary purpose of metamodelling, which aims to circumvent resource-intensive computations, especially for complex case studies.

To address these issues, Garzón et al. (2022) proposed the introduction of inductive biases in machine learning-based metamodels for water networks. Inductive biases refer to the built-in assumptions within a ML model that guide its learning process even before seeing any data. For example, convolutional and recurrent layers are designed to leverage the spatial and sequential nature of image and text data, respectively (Lecun et al., 2015). These assumptions streamline the resulting model architectures, allowing them to use fewer parameters more efficiently and learn from fewer training examples. This property, known as data efficiency (Adadi, 2021), is the quality of maximizing performance while reducing data requirements. In metamodelling terms, this means that the ML metamodel requires less training examples to achieve the same (or higher) level of performance if the inductive biases had not been considered. In turn, this translates to less time needed for the development of the metamodel.

Furthermore, inductive biases nudge models towards leveraging the underlying, often generalizable, structures in data. This promotes adaptability and robustness across diverse input configurations, and enables transfer learning to leverage knowledge across domains and case studies (Vilalta et al., 2017). Transfer learning is a technique in ML that uses training from one domain to improve performance in a different domain. By pre-training the ML model on data from the first domain, it requires less data from the second domain for fine-tuning, i.e., re-training all or part of the ML model for the new case. This means that a metamodel of a SWS can be re-used for a different SWS without creating a new extensive training dataset. Transferability enables DL models to address data scarcity (Wad et al., 2022), and even achieving zero- or few-shot generalization—making accurate predictions in unfamiliar domains based primarily on inherent knowledge and assumptions (Pourpanah et al., 2023). A higher degree of transferability would allow the metamodel to be used for different downstream tasks or applications, in a similar fashion to a foundation model (Jung, 2023).

For SWS, we identified three relevant inductive biases: temporal, physical, and graph-relational. The introduction of a temporal bias can be achieved by incorporating inputs and outputs from preceding time steps using an auto-regressive approach. For example, non-linear auto-regressive with exogenous inputs (NARX) networks have proven effective in forecasting stormwater surcharge dynamics (Schmid and Leandro, 2023). To incorporate a physical bias, respecting pre-defined physical laws is essential. One illustrative application involves constraining hydraulic head predictions within a range defined by minimum and maximum water levels, as demonstrated by Palmitessa et al. (2022). The relational bias introduces explicit relationships between entities. Graphs provide a mathematical framework that represents pairwise relations, such as nodes connected by pipes in a SWS. Graph Neural Networks (GNNs) are a recently developed DL architecture related to this inductive bias. GNNs can process data over graphs by using an intrinsic structure of relations of a system as computational graph. GNNs have shown permutation equivariance and stability to changes in topology, which has made them powerful tools for learning in physical networks (Gama et al., 2020). Instead of connecting all the available features like fully connected ANNs, the inherit network acts as a selector that focuses the parameters in the meaningful relationships indicated by the graph. For a more comprehensive understanding of GNNs and their applications, readers are referred to recent reviews and seminal papers in the field (Bronstein et al., 2017; Zhou et al., 2018; Gama et al., 2020).

However, while GNNs have gained popularity in various applications in the water sector (e.g. Kerimov et al., 2023; Xing and Sela, 2022; Tsiami and Makropoulos, 2021), their utilization in urban drainage systems is underdeveloped. The only existing application of this architecture is focused on database reconstruction (Belghaddar et al., 2021). Despite the substantial potential of GNNs' inductive bias in facilitating the development, training, and out-of-the-domain transferability of metamodels for SWSs, their development in this context is still not yet investigated.

In this research, we address this gap by presenting the first metamodel that uses a GNN to account for the natural relational bias of the SWSs for the widely-used Storm Water Management Model (SWMM) (Rossman, 2015) created by the Environmental Protection Agency (EPA). The proposed metamodel reproduces the evolution of the hydraulic heads across the entire network due to rainfall. Using GNNs' inductive bias, the metamodel can efficiently learn shared representations from the training data, and generalize to unseen parts of the SWS. Furthermore, the proposed metamodel strengthens the temporal and physical biases by considering multiple time-steps in the input and physical features such as node elevation, pipe diameter, and length. We demonstrate the capabilities of our approach for a SWS model based on Tuindorp, part of the city of Utrecht, The Netherlands. We compare our method against an alternative metamodel based on a state-of-the-art approach (Palmitessa et al., 2022), analyzing overall performance, data efficiency, and transferability.

2. Methods

2.1. Data preparation

Data representation We represent the key attributes of the SWS modeled by SWMM based on three inductive biases: topological (planar graph $\mathcal{G}$ consisting of a set $\mathcal{V}$ of N nodes and a set $\mathcal{E}$ of M pipes), physical (node elevations and pipe dimensions), and temporal (time series of runoff $\mathbf{R} \in \mathbb{R}^{N\times T}$ and hydraulic heads $\mathbf{H} \in \mathbb{R}^{N\times T}$). We use the runoff calculated by SWMM instead of direct rainfall since the performance bottleneck lies on the hydraulic solver and not on the hydrological calculation (Palmitessa et al., 2022).

We use an approach known as sequential supervised learning (Dietterich, 2002) in which the results of each simulation are divided into multiple consecutive windows. This way, the time series of the simulations can be transformed into multiple pairs of input–output training examples. In this setting, a window of runoff and hydraulic heads can be used as input for predicting the next window of hydraulic heads. A window is defined as a fixed-size segment of the time series. A window considers p timesteps in the past. Fig. 1(a) illustrates the concept of window.

Data balancing The simulations can have periods in which the SWS is dry, e.g. start, end, and long periods between rain peaks; thus, the time series present two distinct regimes: *flow* and *no-flow* periods. We define *flow* conditions as having at least one node with a water level higher than 1 mm throughout the window duration. The *no-flow* periods are defined as the complement. As there could be more windows from one regime than the other, this inherent imbalance could significantly impact the metamodel's learning process, biasing it towards predicting the more prevalent behavior for all time steps. This is, predicting a constant value of zero depth or failing to learn the draining behavior.

To mitigate this issue, we employed a sampling technique, involving selectively training the metamodel on a subset of *no-flow* windows. The

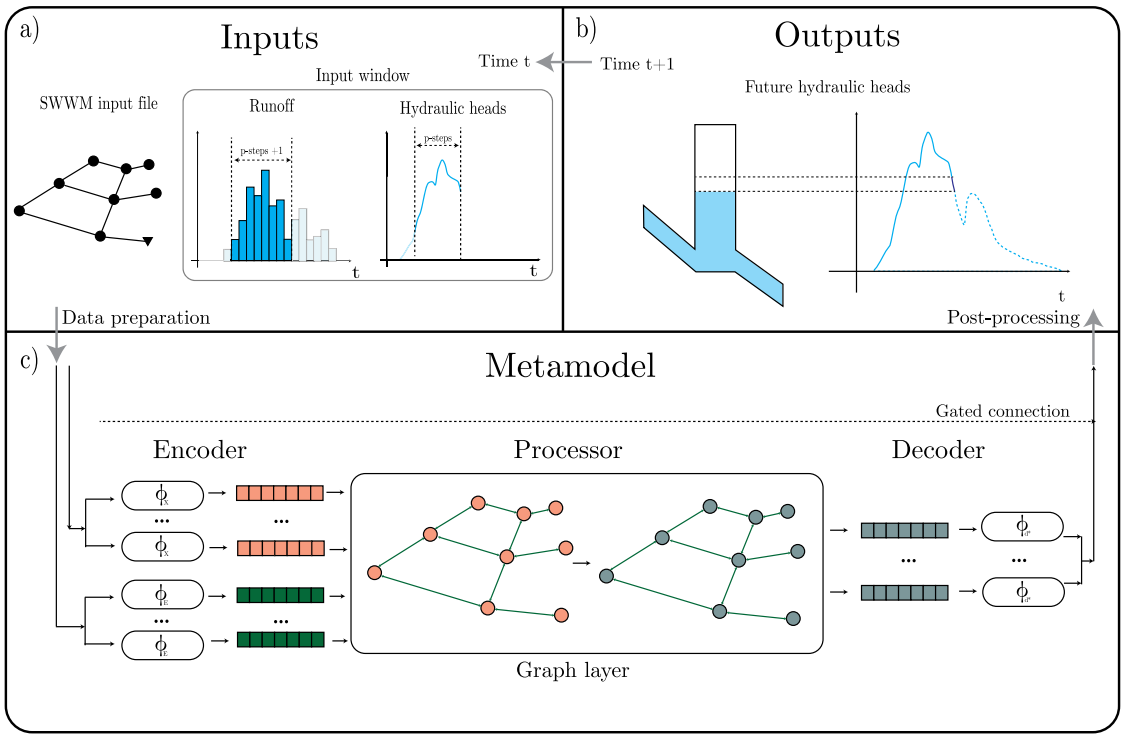


Fig. 1. Summary of the process to generate a prediction for one future time step of hydraulic heads using the metamodel. Subsequent predictions are obtained by iteratively repeating this process. The inputs are partial timeseries of runoff and hydraulic heads, and system information (topology, node elevation, pipe diameters, and lengths) in the SWMM input file. The data is organized in windows and normalized before entering the artificial neural network. Using multi-layer perceptrons, ϕ, the metamodel separately computes the embedding of nodes and pipes which are fed to the graph layer that acts as processor. The output of this phase is then decoded by the processor, which returns the processed embedding into physical values. These values are finally post-processed to obtain a new prediction of hydraulic heads. Having these values, the process repeats to determine the entire time series.

specific ratio of *no-flow* windows to *flow* windows was defined as a hyperparameter called balance ratio. By tuning this ratio, we aimed to achieve a representative distribution of both classes within the training data that improves the metamodel's performance in both regimes. Increasing the value of this hyperparameter causes that the metamodel sees an increased amount of *no-flow* windows during training, which can nudge it to better represent dry conditions but also increases its training time.

Data normalization Normalization is the re-scaling of each input feature. This is a common practice for training ANNs since the non-linear activation functions are more sensitive at intervals close to zero. Also, this prevents the ANN to give more importance to one variable over another based on dimensions (e.g., pipe length vs. diameter). Our approach to normalization was guided by two considerations.

First, we opted for min–max scaling (Mazziotta and Pareto, 2022) as we attempt to retain the physical meaning of the data as much as possible. We scaled all variables to be between zero and one.

Second, vertical distances are normalized based on the maximum possible range in the development system; concretely, the distance between the lowest node invert and the highest superficial street level. This approach, similar to group normalization (Wu and He, 2018) in ML, ensures that hydraulic head, invert elevation, and superficial street levels are scaled using the same reference values. This consideration facilitates physically meaningful algebraic operations and comparisons among these variables. Other variables were normalized independently as the metamodel does not consider any comparison between them.

2.2. Proposed metamodel

Taken as a whole, the metamodel receives normalized node and pipe features (see Fig. 1(a)), and produces normalized hydraulic heads (see Fig. 1(b)). The metamodel operates step-by-step. At each time step t, it takes the inputs and generates a prediction for the subsequent time step $t+1$. This prediction is then used to forecast the output at $t+2$, and this iterative process continues for T time steps determined by the length of the simulation.

The metamodel follows an Encoder–Processor–Decoder structure as seen in Fig. 1(c). This type of architecture is commonly used for approximating the solution of partial differential equations (Brandstetter et al., 2022) like the underlying St. Venant equations in SWMM.

Multiple steps in the metamodel use an MLP, this is a function which we define as ϕ. This function can be described in the following recursive way

$$\phi^{(L)}(\cdot) = \sigma_L(\boldsymbol{\Theta}^{(\mathbf{L})} \cdot \phi^{(L-1)}(\cdot)) \tag{1}$$

where the function $\phi^{(L)}$ at layer L, multiplies the result of the previous layer, $\phi^{(L-1)}$, with a matrix of trainable parameters $\boldsymbol{\Theta}^{(\mathbf{L})}$ followed by the use of a non-linear entry-wise function, $\sigma^{(L)}$ such as hyperbolic tangent, sigmoid, rectified linear unit, or variations of them.

The starting point of this recursive expression can be thought of as an identity function that just receives the raw input information, expressed in mathematical notation:

$$\phi^{(0)}(\cdot) = id(\cdot) \tag{2}$$

For brevity of notation, we omit writing the particular characteristics of the MLPs, such as number of layers, type of non-linearity, and dimensions of the parameter matrices. These are all hyperparameters that vary based on the hyperparameter optimization.

2.2.1. Inputs

In order to process the normalized information of the SWS, we organize the inputs to the metamodel into two matrices: node inputs and pipe inputs.

At the node level, we define these inputs as a single matrix $\mathbf{X} \in \mathbb{R}^{N\times 2(p+1)}$. This matrix concatenates the variables in the following manner:

$$\mathbf{X} = [\mathbf{H}_{t-p:t}|\mathbf{R}_{t-p:t+1}|\mathbf{z}] \tag{3}$$

where $\mathbf{H}_{t-p:t} \in \mathbb{R}^{N\times p}$ is a matrix of p previous time-steps of normalized hydraulic heads. The matrix $\mathbf{R}_{t-p:t+1} \in \mathbb{R}^{N\times(p+1)}$ contains p previous time-steps plus one future time-step of normalized nodal runoff. Lastly, the vector $\mathbf{z} \in \mathbb{R}^{N\times 1}$ represents the normalized node elevation. This last input is static since the elevation of the nodes is a constant feature.

In an analogous way, we define the pipe inputs matrix $\mathbf{E} \in \mathbb{R}^{M\times 2}$ which comprises two static features: normalized diameters $\mathbf{\emptyset} \in \mathbb{R}^{M}$ and normalized lengths $\mathbf{l} \in \mathbb{R}^{M}$ as follows

$$\mathbf{E} = [\mathbf{\emptyset}|\mathbf{l}] \tag{4}$$

We considered only the pipe diameter and pipe length. We consider that other features are not as impactful as diameter and length, such as pipe roughness, or can be derived from the given information, such as pipe slope.

As the pipe features do not vary in time, matrix $\mathbf{E}$ remains constant across the entire simulation.

2.2.2. Encoders

The objective of this step is to create expanded representations of the original information; this encoding step helps capture complex relationships within the data that might not be apparent in the raw features. Here, two distinct MLPs are employed to independently encode the features of each node $i \in \mathcal{V}$ and each pipe $(i,j) \in \mathcal{E}$, producing two separate non-linear representations. This process automatically engineers features, known as embeddings (Makarov et al., 2021). These representations may have more elements and can be thought of as an expansion of the original information into a higher dimension.

The normalized physical data for each node $i \in \mathcal{V}$ is encoded to a node embedding $\mathbf{X}'_i \in \mathbb{R}^F$ of F dimensions, as

$$\mathbf{X}'_i = \phi_{\mathbf{X}}(\mathbf{X}_i) \tag{5}$$

These embeddings can be understood as weighted combinations of the previous hydraulic heads which consider both the previous and current runoff, and the invert elevation.

Similarly, the normalized diameters and lengths in each pipe $(i,j) \in \mathcal{E}$ are encoded into a $F-$dimensional pipe embedding $\mathbf{E}'_{(i,j)} \in \mathbb{R}^F$ using the following equation

$$\mathbf{E}'_{(i,j)} = \phi_{\mathbf{E}}(\mathbf{E}_{(i,j)}) \tag{6}$$

In both cases, the MLPs are shared among all nodes and all pipes, respectively. This means that the same MLP, $\phi_{\mathbf{X}}$, is applied to encode the features of all nodes, and the same MLP, $\phi_{\mathbf{E}}$, is used to encode the features of all pipes.

2.2.3. Processor

The processor receives both node and pipe embeddings and employs a GNN to compute a new node embedding $\mathbf{X}''_i \in \mathbb{R}^{N\times F}$.

The implemented GNN in this study follows the message passing (MP) framework (Bronstein et al., 2021), where each node is characterized by a set of features, known as node embeddings. These embeddings are exchanged with neighboring nodes according to an update equation. This framework allows for the derivation of various architectures, and indeed, numerous variants have been proposed (Zhou et al., 2018). In this application, we utilized the Graph Isomorphism Network with Edge Features (GINEConv) (Hu et al., 2019), which has been found to be highly expressive and significantly improves generalization performance, particularly in transfer learning scenarios. The GINEConv architecture incorporates both node and edge features in its update equation, which is crucial for capturing the fundamental information about the pipes in our application. Specifically, the update equation for GINEConv is given by:

$$\mathbf{X}''_i = \phi_{\mathbf{X}''}\left((1+\epsilon)\cdot\mathbf{X}'_i + \sum_{j\in\mathcal{N}(i)} \mathrm{ReLU}(\mathbf{X}'_j + \mathbf{E}'_{j,i})\right) \tag{7}$$

where $\mathbf{X}''_i$ represents the updated embedding for node i, $\mathbf{X}'_i$ is the current embedding of node i, $\mathbf{X}'_j$ is the current embedding of node j, and $\mathbf{E}'_{j,i}$ are the features of the edge between nodes j and i. For each neighboring node, its embedding is summed with the edge embedding of the pipe connecting the two nodes. This sum is then passed through a Rectified Linear Unit (ReLU), which sets negative values to zero—a common non-linearity used in machine learning models (Agarap, 2018). It is important to note that the sum operation between node and pipe features requires that the hidden dimensions of both encodings to be the same.

The key operation in this graph layer is the summation over the local neighborhood of node i, denoted as $\mathcal{N}(i)$. This set of nodes is determined by the topology of the SWS, making the operation agnostic to any particular layout and only dependent on the known connections of the SWS.

This equation computes a more expressive representation of each node by weighting the sum of the node's own information with the information from its neighboring nodes and pipes. Conceptually, the processed embedding of node i can be interpreted as the cumulative effect of its neighborhood $\mathcal{N}(i)$ on its own state $\mathbf{X}'_i$, analogous to water volume flowing from one node to another with the use of non-linearity aids in accounting for the direction of flow. However, it is important to recognize that this analogy serves more as an interpretative aid than a factual explanation, as such physical interpretations are inherently limited in most machine learning models.

2.2.4. Decoder

The objective of this step is to convert the hidden embedding of the processor into a value with physical dimensions which we consider as a water depth candidate $d_i^{*(t+1)} \in \mathbb{R}$ at each node i. This value is calculated according to the following equation

$$d_i^{*(t+1)} = \phi_{\mathbf{d}^*}(\mathbf{X}''_i) \tag{8}$$

This equation indicates that each processed embedding $\mathbf{X}''_i$ is decoded using a shared MLP. In brief, this step uses the high dimensional values of the GNN embeddings to estimate a value of water depth at the next time step.

2.2.5. Post-processing

The candidate of water depth at each node is then post processed to account for memory of previous solutions and simple physical constraints.

Gated connection

The first post-processing step combines the candidate water depth $d_i^{*(t+1)}$ with the value of normalized water depth at time t. The latter is obtained by subtracting the normalized invert elevation of the nodes, $\mathbf{z}$, from the normalized hydraulic head at time t, $\hat{\mathbf{h}}^t$, as follows

$$\hat{\mathbf{d}}^t = \hat{\mathbf{h}}^t - \mathbf{z} \tag{9}$$

Afterwards, this depth is weighted with the candidate water depth $\mathbf{d}^{*(t+1)}$ according to the following equation

$$\hat{\mathbf{d}}^{*(t+1)} = \alpha\cdot\mathbf{d}^{*(t+1)} + (1-\alpha)\cdot\hat{\mathbf{d}}^t \tag{10}$$

where α is a trainable parameter that weights the contribution of the new candidate over the previous water depth in a similar fashion to gated recurrent units (GRUs) (Chung et al., 2014). This structure allows the metamodel to calibrate the memory of the system, i.e., the relevance of a current state for the future state. Next, the elevation $\mathbf{z} \in \mathbb{R}^{N\times 1}$ is added to obtain a candidate of hydraulic head prediction $\hat{\mathbf{h}}^{*(t+1)} \in \mathbb{R}^{N\times 1}$ using the following equation

$$\hat{\mathbf{h}}^{*(t+1)} = \hat{\mathbf{d}}^{*(t+1)} + \mathbf{z} \tag{11}$$

Physical constraints

Finally, the results are constrained to meet the physical restrictions of minimum and maximum levels as done in Palmitessa et al. (2022). The hydraulic heads are bounded to be higher or equal than the invert elevation of the nodes $\mathbf{z}$ and lower or equal than the superficial street level $\mathbf{g} \in \mathbb{R}^{N\times 1}$. The former constraint avoids unfeasible values of water depth, i.e., negative values. The latter constraint accounts for the water reaching the street level and leaving the system through the manholes. Eq. (12) mathematically represents these constraints as follows

$$\hat{\mathbf{h}}^{t+1} = \min(\mathbf{g}, \max(\hat{\mathbf{h}}^{*(t+1)}, \mathbf{z})) \tag{12}$$

2.2.6. Auto-regression

This final prediction is used as input for the next time step; this way, the metamodel can predict the entire time series for all nodes in an auto-regressive fashion. The metamodel requires an initial state, which is the condition of the system at the beginning of the simulation. For simplicity and computational efficiency, we can assume an initial state of zero water depth and rainfall. Specifically, the matrix of hydraulic heads $\mathbf{H}_{0-p:0}$ contains only the elevation of the nodes (this means, we assume depth = 0 for all nodes) and the matrix of runoff $\mathbf{R}_{0-p:0+1}$ is equal to zero in its first p columns. With the prediction at time $t+1$, it is possible to define a new matrix $\mathbf{H}_{t-p+1:t+1} \in \mathbb{R}^{N\times p}$, by concatenating the prediction as the last column in the new matrix and dropping the first column to maintain the dimensions. This process is then repeated T times, i.e., as many time steps as the simulation has.

2.3. Benchmark metamodel

As benchmark for this study, we used an adapted version of the ML metamodel for SWSs proposed by Palmitessa et al. (2022). We chose this study based on its similarity to the task and approach addressed in this paper. In that study, the metamodel was created as a surrogate of MIKE 1D (DHI, 2017) using TensorFlow. We created a version of that metamodel for hydraulic heads of SWMM using Pytorch, for fairer comparison against our proposed metamodels developed using this framework. The MLP-based metamodel estimates a candidate for the hydraulic heads at the next time step $\hat{\mathbf{h}}^{*(t+1)} \in \mathbb{R}^{N\times 1}$ as

$$\hat{\mathbf{h}}^{*(t+1)} = (\phi_1(vec(\mathbf{H}_{t-p:t})) + \phi_2(vec(\mathbf{H}_{t-p:t}|\mathbf{R}_{t-p:t+1})) \tag{13}$$

where the first MLP ϕ_1 calculates a non-linear combination of the vectorized (or flattened) matrix of normalized hydraulic heads $\mathbf{H}_{t-p:t}$. This preliminary operation is required by the fully connected structure of the MLP, which process all the nodes in the SWS concurrently. This allows ϕ_2, the second MLP, to focus on transforming the vectorized matrix of current hydraulic heads $\mathbf{H}_{t-p:t}$ and runoff $\mathbf{R}_{t-p:t+1}$ into the difference (or residue) between time step t and $t+1$. This is known as a generalized residue connection (Chen and Xiu, 2021).

This candidate solution is then restricted to comply with feasible levels using Eq. (12). Namely, setting all the hydraulic heads between the node invert level and the street surface level. Similar to Section 2.2.6, the process is auto-regressively repeated for each time-step to obtain the output time series of hydraulic heads for all nodes.

2.4. Training strategy

2.4.1. Objective function

For this application, we minimized the sum of mean squared error (MSE) over all training windows. In this case, the metamodel performance for a window is summarized in a single value that considers the quadratic error in hydraulic head over all the nodes for all the considered time steps of each window. The total training loss is the sum of all of the MSEs of the training windows. Since the metamodel produces a time series for each node, Eq. (14) indicates that the loss $\ell \in \mathbb{R}$ per window is an average of quadratic error of hydraulic head over all N nodes and over T time steps.

$$\ell(\mathbf{H}, \hat{\mathbf{H}}) = MSE(\mathbf{H}, \hat{\mathbf{H}}) = \frac{1}{N \cdot T} \sum_{i=1}^{T} \sum_{i=1}^{N} (H_i^{(t)} - \hat{H}_i^{(t)})^2 \tag{14}$$

The MSE penalizes larger errors with more severity in comparison to smaller errors. This characteristic is desired because it nudges the metamodel to accurately predict high water depths.

2.4.2. Performance metrics

Aside from the objective function, we assess the performance of the metamodel based of two main aspects: overall accuracy and execution speed. For quantifying accuracy, we considered two metrics: mean absolute error (MAE), shown in Eq. (15),

$$\text{MAE}(\mathbf{H}, \hat{\mathbf{H}}) = \frac{1}{N \cdot T} \sum_{i=1}^{T} \sum_{i=1}^{N} |H_i^{(t)} - \hat{H}_i^{(t)}| \tag{15}$$

and Nash–Sutcliffe efficiency (NSE), shown in Eq. (16),

$$\text{NSE}(\mathbf{H}, \hat{\mathbf{H}}) = 1 - \frac{\sum_{i=1}^{T} \sum_{i=1}^{n} (H_i^{(t)} - \hat{H}_i^{(t)})}{\sum_{i=1}^{T} \sum_{i=1}^{n} (H_i^{(t)} - \bar{H})^2} \tag{16}$$

Both metrics are defined in terms of a double sum to consider the values for all N nodes and T time steps. Furthermore, apart from overall metrics, we separately calculated the metrics for *flow* and *no-flow* conditions. For the execution speed, we calculated the speed-up with respect to the original model, i.e., the amounts of times that the metamodel time is shorter than the time SWMM takes to run a simulation, shown in Eq. (17).

$$\text{Speed-up}^{(k)} = \frac{T_{\text{SWMM}}^{(k)}}{T_{\text{Metamodel}}^{(k)}} \tag{17}$$

which indicates that the speed-up for event k is the ratio between the execution time from the original model $T_{\text{SWMM}}^{(k)}$ and the time taken by the metamodel $T_{\text{Metamodel}}^{(k)}$.

2.4.3. Parameter optimization

For training the GNN- and MLP-based metamodels, we used the following settings. We chose AdamW (Loshchilov and Hutter, 2019) as adaptative gradient method because of its generalization performance. We used a constant learning rate and trained for a fixed amount of 100 epochs saving the metamodel with the minimum validation loss. In addition to these standard practices, we included a learning strategy named curriculum learning (Soviany et al., 2022; Bentivoglio et al., 2023). In principle, this strategy seeks to train the ML model with progressively harder examples, tasks, or objectives. In our case, the training dataset was divided into two datasets, one with short duration windows and another with long duration windows. In our setting, the metamodel is trained to predict short time series during the first half of the training. This aims to prompt the metamodel to predict the short-term dynamics in the SWS. For the second half, the metamodel is trained with the harder task of predicting longer time series. This task is more difficult because it requires the metamodel to generate solutions that are not only accurate but also stable over time, thereby minimizing error accumulation. During all the training, the validation loss is calculated over longer time series of 50 steps. This length of the validation windows was kept consistent regardless of the length of the training windows. Even though 50 steps is shorter than the mean duration of the SWMM simulations (between 360 and 4320 steps), it was sufficient to promote stable metamodels.

Table 1
Ranges of tested hyperparameters for the metamodels. The bold values are the chosen hyperparameter for the GNN-based metamodel.

Group	Name	Description	Ranges
Window	Balance ratio	Ratio between no-flow windows and flow windows.	[0, 1, 2, 3, **4**]
	Steps behind	Number of past time steps that the model considers to make a prediction.	[1, …, **9**, …, 15]
	Steps ahead (Training)	Initial number of time steps for the training examples. [Short, Long] window sizes used in the curriculum learning.	[**[2, 10]**, [2, 25], [5, 25], [10, 50]]
	Steps ahead (Validation)	Number of future time steps for the validation examples.	**50**
Optimizer	Batch size	Number of windows used to train at each training cycle.	[16, **32**, 64]
	Learning rate	Coefficient that mediates the update of the weights in gradient descent.	[0.001, …, **0.00194**, …, 0.006]
	Weight decay	Coefficient that penalizes the magnitude of weights and biases.	[**0.01**, 0.1, 1.0]
Metamodel	Num. hidden layers	Number of intermediate layers for each of the MLPs in the model.	[**0**, 4, 8, 16, 32]
	Non-linearity	Non-linear activation function between layers of the model.	[ReLu, Tanh, **PreLU**]
	Hidden dimensions	Size of the internal hidden dimensions in the model.	[4, 8, 16, **32**]

2.4.4. Hyperparameter optimization

Hyperparameter optimization requires training a multitude of metamodels, each of them with a different combination of hyperparameters. Each combination defines a window size, a unique metamodel, and optimizer settings. Nevertheless, exploring all the possible combinations is practically intractable due to its time consumption, which defeats the purpose of creating a metamodel in the first place. The number of configurations that can be drawn from this subset of possible hyperparameters (shown in Table 1) can easily grow exponentially.

To tackle this issue, we considered three strategies to decrease the number of simulations for finding suitable working metamodels.

1. Grouping the hyperparameters into three groups based on the component they affect, i.e., window, metamodel, and optimizer.
2. Restricting the number of simulations to a manageable yet representative number.
3. Using Bayesian search (Turner et al., 2020) for choosing the hyperparameters that have higher probabilities of decreasing the objective function during the search.

For each configuration of hyperparameters, the metamodel is trained using the same training dataset, and then evaluated over the same validation dataset. For choosing suitable metamodels for the experiments, we assessed the fitness of each simulation based on accuracy and speed. In the cases of the window and metamodel calibration, the speed objective was the average speed-up in a simulation of representative running time (20 s). In the case of the optimizer calibration, the speed indicator is the total training time. The reason for this is that execution time can be regarded as independent from the optimizer settings. Therefore, the objective of this particular calibration is to reduce the metamodel development time.

In the case of the MLP-based metamodel, we included the optimal hyperparameters found in Palmitessa et al. (2022), i.e., six hidden layers, each containing 100 neurons and employing a ReLU activation function.

Table 2
Ranges of system variables. (*)The elevation range refers to the datum of the lowest node invert level and the highest superficial street level. This range is used to normalize the hydraulic heads, the elevations, and the street levels.

Network	Pipe diameters [m]	Pipe lengths [m]	Elevation range* [m]
Development	0.24 to 1.35	2.84 to 84.9	−1.61 to 2.20
Transferability	0.18 to 1.05	3.99 to 84.5	−2.93 to 2.24

For both types of metamodels, GNN- and MLP-based, we trained more than 100 metamodels for each of the groups of hyperparameters: window, optimizer, and metamodel.

3. Case study

3.1. Storm water system description

For SWSs, we adapted part of the Tuindorp network, located in the city of Utrecht, The Netherlands. This is a combined sewer system which has been previously analyzed, and calibrated in Van Bijnen et al. (2017). The Tuindorp sewer network is a looped system, driven by gravity, constructed during the 1970s to drain a residential area of 57 ha. The catchment area includes various contributing zones with diverse roof and pavement types. In this area, the gathered sewage is first transported to the southern part of the catchment. From there, it is pumped to a downstream catchment, which then conveys the water to the Utrecht wastewater treatment plant. For this study, we considered the system as a storm water sewer since the wet weather flow is the relevant regime during urban pluvial flooding. The network layout is shown in Fig. 2. We extracted two sub-networks from the complete system, one for developing the metamodel, shown in blue in Fig. 2, and other for testing its transferability, shown in orange. The criterion for separating the systems was to minimize the number of pipes that connect each sub-network to the rest of the system. The gray area in the figure was not considered in the analysis because it contains multiple components (i.e., pumps, storage tanks, and weirs) whose implementation in the metamodel is beyond the scope of this work. Table 2 shows the ranges of the components in the complete system and its partitions.

3.2. Data collection and generation

We used a SWMM file based on an Infoworks model from Meijer et al. (2018) to generate the database of extended simulations of hydraulic heads. As inputs for the SWMM simulations, we considered real rainfall and synthetic rainfall events. For the real rainfall input time series, we used the radar rainfall dataset of 5-minute precipitation depths at a 1-km grid from the Royal Netherlands Meteorological Institute (KNMI, 2022) for the year 2014, which contained events of extreme rainfall (Eden et al., 2018). We chose each time series such that there was at least one entry higher than 10 mm and a subsequent period of five hours without rain for registering the SWS behavior during emptying. These time series were 24 h long on average but could vary between 6 and 72 h (equivalent to 360 and 4320 min). For augmenting the database, we created the synthetic rainfall time series using the alternating blocks method (Chow et al., 1988) with variable parameters that changed the intensity (up to 52 mm) and duration (between 20 and 180 min) of the events. Individual characteristics of these events such as duration and intensity are summarized in the supplementary material. For all the simulations, we considered rainfalls to be uniformly distributed over the SWS. We used a routing step of one second and a reporting step of one minute. In total, we used 160 rainfall events (125 real, 35 synthetic) to run the simulations. We divided the dataset into three subsets: training, validation, and testing. The training subset, comprising up to 100 events (80 real, 20 synthetic), is used to calibrate the metamodel's internal parameters. In this dataset, the simulated

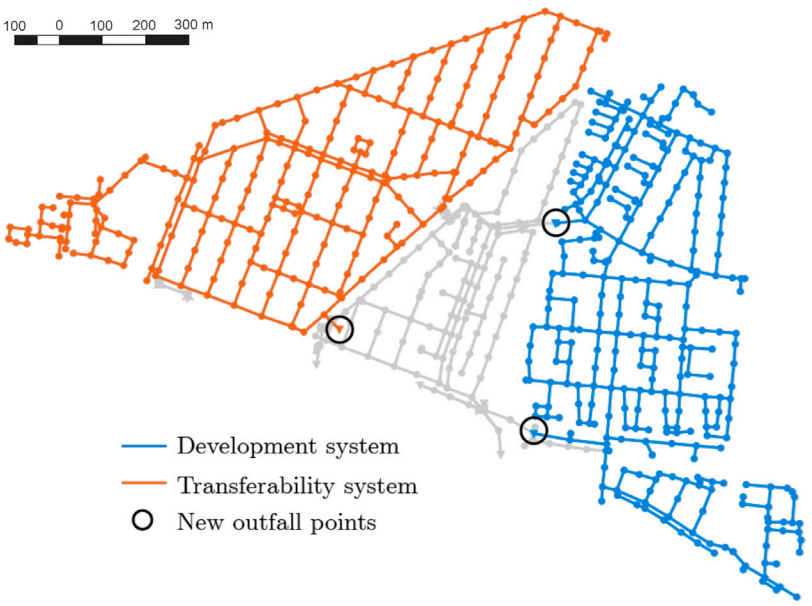


Fig. 2. Layout of the drainage system in Tuindorp, Utrecht, The Netherlands. In blue, the subnetwork used for developing the metamodel. In orange, the subnetwork used for transferability testing. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

time series can reach approximately 70% of *no-flow* windows, which is accounted for with data balancing. The validation subset, consisting of 30 events (15 real, 15 synthetic), assesses metamodel performance without influencing the parameter updates. Lastly, the testing subset, comprising 30 real events, are used to evaluate the metamodel's final performance.

4. Experimental setup

We established three experiments for testing each of the three properties of interest in the GNN-metamodel: in-domain performance, data efficiency, and transferability.

4.1. Experiment 1 — In-domain performance

The first experiment consists on testing the metamodel performance on the same domain it was trained on. We estimated the MAE, NSE, and speed-up metrics for the best GNN-based metamodel found after hyperparameter optimization (see supplementary material). For comparison, we calculated the same metrics for the MLP-based metamodel. These metrics give an indication of the overall performance of the metamodel. Nevertheless, aggregated metrics do not reflect the details of the metamodel fitness in space or time. Thus, we exemplify the simulation of the hydraulic head time series during a rainfall event in both the spatial and temporal domains.

4.2. Experiment 2 — Data efficiency

The experiment consisted on repeatedly training from scratch the GNN-based metamodel and the MLP-based benchmark with gradually increasing training sets in each repetition. We designed the experiment in a progressive manner, in which the ML metamodels were sequentially trained on increasingly larger datasets, from 1 to 100 events, with each subsequent dataset incorporating all the data from the previous ones. We used fixed hyperparameters for both metamodels to ensure the observed effect was due to the change in the size of training dataset.

In this experiment, we do not consider speed-up as the metamodels' speed-up is independent of the size of the training dataset. Given the stochastic nature of the training, we repeated each training with 25 different random seeds.

4.3. Experiment 3 — Transferability

In this experiment, we trained the metamodel on the development subnetwork and tested the performance in the transferability subnetwork, i.e., an unseen part of the SWS. For this experiment, we used the metamodel with the lowest validation loss trained with the maximum number of training events (100) in the development subnetwork. We used the same normalization values from the development subnetwork to normalize the features on the transferability subnetwork. This is because the metamodel's internal parameters were trained on the distribution of input values from the training set. We calculated the same accuracy and speed metrics for the proposed metamodel in the new domain. Since we did not perform additional training or fine-tuning, this is a setting of zero-shot learning. In the case of the MLP-based metamodel, its architecture prevents its use in this setting.

4.4. Technical specifications

For the hydrodynamic simulations, we used SWMM 5.1.015 using dynamic wave as routing model. We used Python 3.10.2 (Van Rossum and Drake, 2009), PyTorch (Paszke et al., 2019) and PyTorch geometric (Fey and Lenssen, 2019) to define and train the metamodels. We used the supercomputer DelftBlue (Delft High Performance Computing Centre, 2022) for training all the metamodels in hyperparameter optimizations and experiments with multiple GPUs (NVIDIA Tesla V100S-PCIE-32 GB). For performance testing in CPU, we employed a PC with Intel(R) Core(TM) i7-8665U at 1.9 GHz CPU. Furthermore, we automated and recorded each simulation for subsequent analysis with the ML platform Weights and Biases (Biewald, 2020).

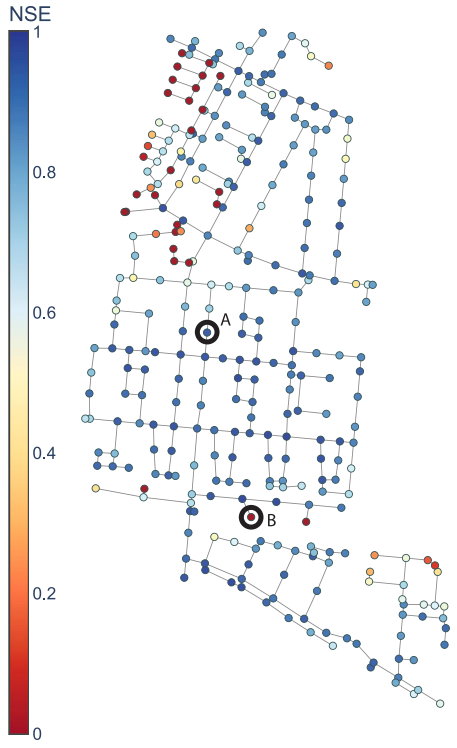
5. Results

5.1. Experiment 1 — In-domain performance

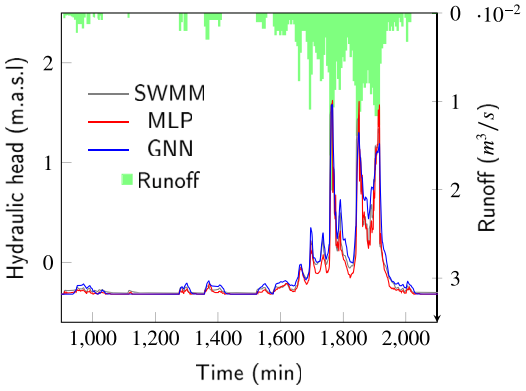
Table 3 summarizes the accuracy metrics over the test dataset. The proposed metamodel has a value of 0.98 NSE overall, being 0.92 and 0.99 for *flow* and *no-flow* periods, respectively. In absolute terms, the proposed metamodel has a mean absolute error of 5.43 cm and 0.17 cm for *flow* and *no-flow* conditions, respectively. The results indicate that the metamodel is overall highly accurate in relative and absolute terms.

Table 3
Descriptive statistics of accuracy metrics for replicating SWMM hydraulic heads using the proposed and benchmark metamodels in the development domain for the test dataset.

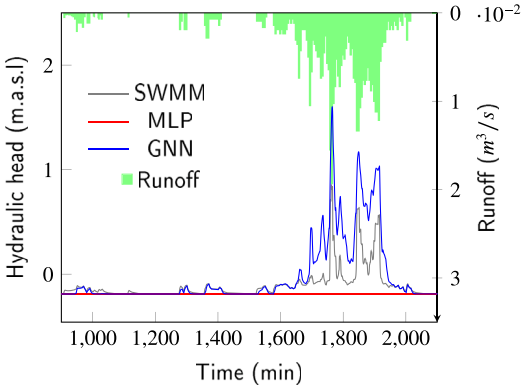
	NSE						MAE [cm]					
	MLP			GNN			MLP			GNN		
	No flow	Flow	Overall	No flow	Flow	Overall	No flow	Flow	Overall	No flow	Flow	Overall
Mean	1.0	0.908	0.976	0.997	0.924	0.981	0.03	4.87	1.15	0.17	5.43	1.36
S.D	0.0	0.076	0.36	0.02	0.30	0.17	0.02	1.81	0.94	0.08	1.47	0.92
Max	1.0	0.979	0.999	1.00	0.977	0.998	0.08	10.24	4.01	0.45	9.78	3.98
Min	1.0	0.709	0.850	0.992	0.855	0.924	0.01	3.22	0.32	0.04	3.34	0.35



(a) Nodal distribution of NSE for a rainfall test event. The high and low performance nodes plotted on the right are marked as A and B, respectively.



(b) Timeseries in node A with high >0.95 NSEs for both metamodels.



(c) Timeseries in node B with low <0.1 NSEs for both metamodels.

Fig. 3. Performance comparison of metamodels against SWMM: Nash–Sutcliffe Efficiency (NSE) as a Color Map (left) and Time Series Comparison (right) for the two metamodels during the same rain event in different nodes of the network. The runoff (shown upside down) is the inflow to the node after the rainfall is transported in the respective subcatchment. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

In comparison, the MLP-based metamodel had a marginally inferior relative performance and a marginally superior absolute performance. These distinctions become negligible when accounting for the standard deviation.

Fig. 3(a) shows the spatial distribution of NSE_i for each node $i \in \mathcal{V}$ of the GNN-based metamodel for a test event. By analyzing the spatial distribution, it can be seen that most of the nodes have a high value of the metric (above 0.85), which indicates a favorable fitting.

By examining the spatial distribution of nodes, we identify two categories of nodes characterized by lower accuracy. The first category consists of various leaf nodes, particularly those experiencing a backwater effect during high flow rates or those connected to pipes that have accumulated water from upstream ones. The second category includes nodes situated near the outflow points of the SWS. In these node types, there are instances where the hydraulic head is predominantly influenced by flow transport.

Figs. 3(b) and 3(c) show the comparison of time series of that event in nodes with high and low NSE, respectively.

Fig. 3(b) presents the time series data from Node A, situated in the center of the SWS. This node is influenced both by the incoming runoff and flow from adjacent pipes. Despite the metamodel encountering challenges in capturing the water transport effects from neighboring nodes, the training process adapts the metamodel's weights and behavior to establish the most accurate relationship between a node's hydraulic head and its adjacent neighborhood for most of the nodes. The high fitness value indicates the metamodel's capability to generally reproduce the entire hydraulic head time series.

Nevertheless, certain nodes exhibit relatively low fitting values, exemplified in Fig. 3(c). In this instance, Node B is linked to a node that has accumulated water from upstream nodes, resulting in a heightened hydraulic head. Consequently, using these values to calculate its own hydraulic heads leads to overprediction. Additional examples of metamodel performance can be found in the supplementary materials.

The MLP-based metamodel displays high accuracy for some nodes; however, for other nodes, it consistently predicted zero water depth

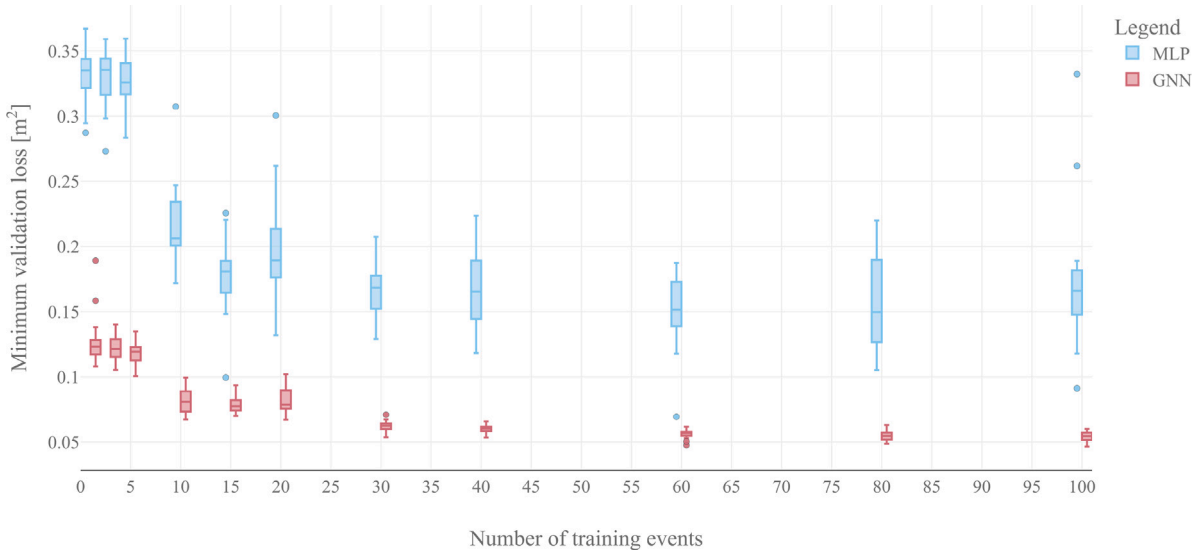


Fig. 4. Minimum validation loss as a function of number of events in training set for GNN and MLP-based metamodels.

Table 4
Descriptive statistics of speed-up of the proposed and the benchmark metamodels in development domain for the test dataset.

	SWMM CPU time [s]	MLP (CPU) Speed-up	MLP (GPU) Speed-up	GNN (CPU) Speed-up	GNN (GPU) Speed-up
Mean	23.16	11.96	35.04	3.69	17.39
S. D	12.93	3.73	9.54	1.22	4.72
Max	69.81	19.37	60.02	6.44	28.36
Min	9.41	4.04	17.71	1.49	10.73

as observed in Fig. 3(c). This behavior can be attributed to the combination of two factors: the inherent tendency of the problem itself towards zero-depth predictions and the lack of structural bias in the MLP architecture.

First, as previously mentioned, the response exhibits two regimes: *flow* and *no-flow*. These regimes may create competing objectives during training, with the generation of a local optimum in the loss function related to zero water depth prediction. While this tendency was accounted for with data balancing (see Section 2.1), it may not be sufficient at a local scale. Second, the fully connected structure of the MLP causes each node to use features from all nodes without accounting for the SWS topology. Without a direct connection, the influence of neighboring nodes must be inferred by the training process. However, this inference can be hindered by low data availability or limited computational resources.

The combination of these two factors leads to a situation where, despite both GNN- and MLP-based metamodels undergoing the same training process, the training process for the MLP-based metamodel can become trapped in a solution with multiple local minima in the loss function. Consequently, this results in low-performing nodes displaying unpredictable spatial patterns. The lack of graph inductive bias would cause all nodes to have the same potential for predicting a constant value. This is illustrated by the distribution of errors in the supplementary materials.

Table 4 summarizes the results of speed-up. The proposed metamodel is up to six times faster than SWMM when using CPU and up to 28 times faster when using GPU. In comparison, the MLP-based metamodel is up to 19 and 60 times faster than the original model.

5.2. Experiment 2 - Data efficiency

Fig. 4 shows the effect of increasing the training dataset on the minimum validation loss for both the MLP-based and GNN-based metamodels. The sizes of the datasets considered for this experiment were 1, 3, 5, 10, 15, 20, 30, 40, 60, 80, and 100 events. This range remained consistent for both metamodels.

Fig. 4 highlights the consistent difference in performances between metamodels. This gap indicates that the GNN-based metamodel is more data efficient than the MLP counterpart. Furthermore, it can be interpreted in two complementary ways. On one side, for a fixed number of training events, the GNN-based metamodel consistently achieves better minimum validation loss. Conversely, the GNN-based metamodel requires less training events to reach a fixed level of performance.

From Fig. 4, we can also observe two additional patterns. First, as expected, the minimum loss decreases with data availability. Second, the variance of the MLP-based metamodel is higher than the GNN-based metamodel. This indicates that the proposed metamodel is more robust to training; that is, the GNN-based metamodel's performance is less influenced by the randomness of seed selection. Nevertheless, there is always a spread of performances based on the random initialization of parameters, this is a known phenomenon for ML models (Picard, 2021). In practice, it is advisable to train multiple versions of a metamodel with different random seeds. However, this can be a burden when having a low computational budget.

5.3. Experiment 3 - Transferability performance

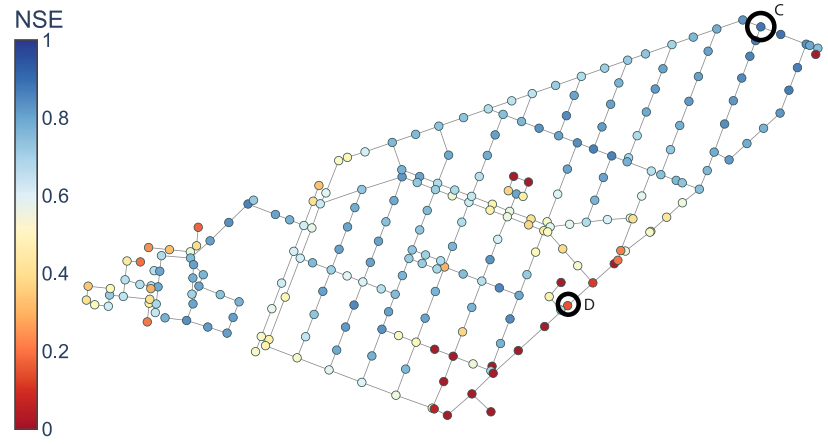
Table 5 summarizes the accuracy and speed metrics over the test dataset in the new domain.

The results indicate that the overall goodness-of-fit remains high with 0.968 NSE and 2.87 cm MAE without need for retraining or fine-tuning. The statistics indicate only a small decrease in performance in comparison to the performance on the development domain. The sustained high performance can be attributed to the inherent similarity between the two domains. The minor decrease in performance is expected, given that the metamodel was not specifically trained on this configuration.

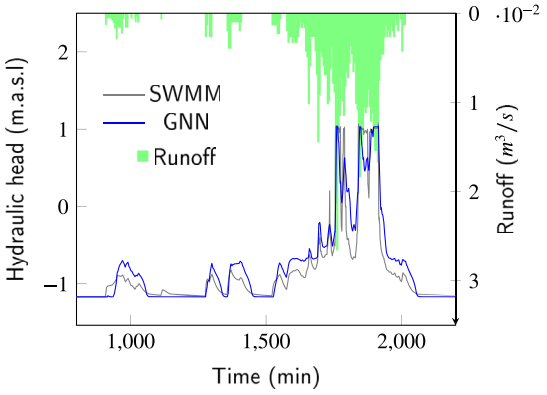
For a better picture of the functioning of the metamodel beyond aggregated metrics, we show the results of the metamodel for the same

Table 5
Descriptive statistics of accuracy metrics for replicating SWMM hydraulic heads using the proposed metamodel in the transfer domain for the test dataset.

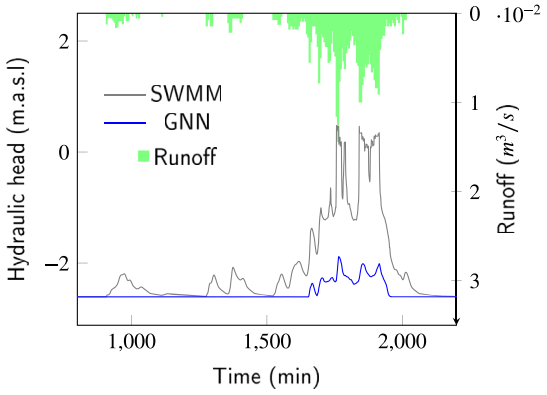
Metric	MAE [cm]			NSE			Speed up	
Condition	No-Flow	Flow	Overall	No-Flow	Flow	Overall	CPU	GPU
Average	0.24	10.31	2.87	0.999	0.900	0.968	3.88	13.15
Stand. deviation	0.17	4.24	2.60	0.01	0.72	0.43	1.20	3.00
Maximum	0.70	22.39	12.05	1.00	0.988	0.999	6.90	20.73
Minimum	0.06	4.38	0.51	0.996	0.725	0.836	1.41	7.98



(a) Nodal distribution of NSE for a rainfall test event for a transferred case.



(b) Timeseries in node C with 0.90 NSE.



(c) Timeseries in node D with 0.16 NSE.

Fig. 5. Performance comparison of metamodels against the original model: Nash–Sutcliffe Efficiency (NSE) as a Color Map (left) and Time Series Comparison (right) for the transferred metamodel during the same rain event in different nodes of the network. The runoff (shown upside down) is the inflow to the node after the rainfall is transported in the respective subcatchment. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

rain event from Section 5.1 in the test dataset. The distribution of NSE values across the network, shown in Fig. 5(a), reveals that a majority of nodes exhibit strong performance, with the exception of some leaf nodes and those residing near the network's outlet (lower right corner). These nodes, which also demonstrated inferior performance in the development network, highlight the transferability of the metamodel, encompassing both its strengths and limitations.

Fig. 5 shows the hydraulic head time series for two different nodes in the transferred system. From Fig. 5(b), it is possible to see a reasonable fit between the two time series. With similar dynamic and orders of magnitude.

While the metamodel generally demonstrates high performance, Fig. 5(c) illustrates a time series representative of a node with suboptimal metamodel accuracy. This node is located in the proximity of the network's outlet. The metamodel exhibits no response throughout the event, except for the peak discharge. These observations restate the metamodel's inability to fully replicate flow transport dynamics, thereby restricting its performance in specific areas of the network.

6. Discussion

6.1. Implications

6.1.1. Accuracy

The proposed metamodel demonstrates sufficient accuracy for diverse applications where similar models have achieved comparable performance. This includes tasks like design optimization (Huang et al., 2015; Zhang et al., 2019), and real-time flood prediction, as exemplified by Chiang et al. (2010). Regardless, it is important to consider that a metamodel is a second-level of abstraction, where each level inherently introduces its own degree of error in representing the actual SWS. For instance, calibration of the SWMM model introduces a similar error in magnitude with respect to the observed data, e.g., 0.94 NSE (Zhang et al., 2021)). Considering the similarity in magnitudes of the model and metamodel's errors, it is advised not to exclusively prioritize the improvement of the metamodel's accuracy when seeking to improve the overall performance of the application in which the metamodel is used.

In optimization applications, it is common to employ evolutionary algorithms. These algorithms involve repeatedly running a hydrodynamic model to assess the effectiveness of progressively improving system proposals. In such scenarios, the metamodel may not require absolute precision; instead, it could focus on comparing relative performance to select one solution over another. Based on the application, different levels of accuracy could be sufficient and it is the modeler's responsibility to indicate its adequacy based on the application at hand.

6.1.2. Speed

The speed performance achieved in this study may not appear substantial compared to other approaches, particularly those designed specifically for real-time applications. One of the main bottlenecks of the current metamodel architectures is their auto-regressive functioning, as the sequence is produced one step at a time. A potential approach to enhance the speed of the metamodels is to predict multiple steps ahead simultaneously by integrating recurrent neural network (RNN) or one-dimensional convolutional neural network (1D CNN) architectures. For even more sophisticated multi-step predictions, combining graph and temporal convolutions, as demonstrated by Wu et al. (2019), can leverage both structural and temporal relationships.

The fact that, despite its larger size, the MLP-based metamodel was faster than the GNN counterpart can be attributed to the recent introduction of graph-based methods in both software and hardware. These components have not undergone equivalent levels of code optimization as more established ANN approaches.

While computational efficiency is an important consideration, the primary focus of this study was on investigating the use of inductive biases to enhance data efficiency and transferability, rather than on achieving maximal speed-ups. This study serves as a proof of concept, and subsequent iterations can further explore more efficient software implementations, other algorithmic structure or hardware enhancements to gain higher speeds.

6.1.3. Spatial distribution of errors

The errors in prediction suggest that the metamodel is less effective in estimating water levels at junctions impacted by flows from other pipes, as opposed to direct flow inputs, indicating challenges in accounting for flow transport. These errors in the spatial distribution can be traced back to the structure of the GNN-metamodel, specifically two key settings. First, the input features do not incorporate flow rates, and as a result, the metamodel lacks training on this crucial information. Consequently, the metamodel cannot adjust its internal parameters to adequately capture all the intricacies of hydraulic dynamics. Second, the GNN layer in the processor only takes into account the first-degree neighbors, limiting the metamodel's ability to learn interactions over longer distances within the SWS's topology.

As a result, the metamodel is most ideally suited for use in SWS with moderate slopes, where backwater effects are minimal, and with low average travel times to prevent substantial flow accumulation. An example scenario is decentralized urban drainage systems where rain and wastewater are managed locally.

For mitigating spatially distributed errors we recommend exploring two promising avenues: targeted intervention and multi-scale processing.

Targeted intervention strategically penalizes the loss function for underperforming nodes. This guides the metamodel to prioritize improvement in specific areas. Multi-scale processing, in contrast, tackles errors on a broader scale. This approach can be implemented in two ways: stacking multiple GNN layers or performing simultaneous computations on diverse graph resolutions (achieved through skeletonization).

Both methods empower the metamodel to capture information from larger node neighborhoods, enabling a more comprehensive understanding of the network's global structure. This becomes particularly crucial for large networks where single-layer GNNs or solely fine-grained analysis struggle to capture the full complexity.

6.1.4. Data efficiency

The GNN-based metamodel consistently demonstrates superior performance over the MLP-based metamodel, primarily attributable to its graph inductive bias. This bias not only enhances performance but also significantly reduces the number of trainable parameters. While the fully connected structure of the MLP-based metamodel requires 755,722 trainable parameters, the GNN-based metamodel requires only 6,090, a 99.2% reduction. In practice, the more parameters an ML model has, the more data and computational resources it requires for training (Al-Jarrah et al., 2015). Consequently, this substantial reduction translates to lower computational costs and data requirements.

6.1.5. Transferability

In terms of applications, the transferability of the metamodel expands the capabilities of previously mentioned applications. For instance, design optimization, flood resilience and uncertainty analyses would be unrestricted to a fixed network layout. Consequently, exploration of alternatives such as expanding the network, re-routing parts of it, or analyzing nearby systems would be possible. The ability of the metamodels to transfer knowledge across different domains and tasks is a crucial stepping stone towards the development of foundation models.

Overall, these results demonstrate the feasibility of creating a functional GNN-based metamodel. The accuracy and speed of the proposed metamodel makes it a candidate for use in applications that require short execution times while maintaining high resolution at node level, i.e., water depth at all the nodes.

6.2. Limitations

6.2.1. Case study

We treated the case study as an SWS when it actually is a combined sewer system. Consequently, we did not account for the dry weather flow. Despite this, this is an acceptable condition because the wet weather flow accounts for the majority of the transported flow during a rainfall event. Moreover, the methodology can be adapted to work with dry weather flow by including a corresponding pattern in the SWMM simulations and as an extra term added to the input runoff.

Another limitation regarding the case study is the consideration of a single case study. This limits the findings as they can be conditioned to particularities of the used SWS. Nevertheless, the selected SWS is a complex looped system from a portion of a real city. In addition, the case study was divided into two separate subnetworks in which the metamodel showed high overall performance. We recommend further testing of the metamodel in other SWSs with different characteristics, e.g. size, slope, topology.

6.2.2. Method

The normalization ranges only considered the values of the development subsystem, which may not represent a completely representative range of values for other potential transferred case study. This difference can reduce the performance of the proposed metamodel in a different case study with different distributions of variables. For this study, both the development and training sub-networks came from the same SWS; therefore, the distributions were similar. Future work can be directed towards development of a metamodel with multiple case studies and large normalization ranges to increase the transferability capability of the metamodel.

In terms of metamodel comparison, we only considered an MLP-based metamodel as benchmark and did not consider other alternatives such as simplified models, e.g., comparison to some lumped or physically-based low fidelity models. Therefore, the results in this paper are pertinent to metamodels for SWSs that use ML. Consequently, the proposed metamodel offers improvement in development and transferability of ML-based metamodels but there are simpler models that exchange accuracy or resolution for speed in different ways. The water

modelers' task is to identify the specific type of metamodel that better suits their needs.

The benchmark metamodel does not consider constant information, i.e., node elevation, pipe length and diameter. This implies that the benchmark metamodel is not receiving the same information as the proposed metamodel, and the higher performance could be attributed to this disparity in inputs. Nevertheless, the original metamodel from which the benchmark is based on does not consider these pipe features while it still achieves remarkable performance.Adequately considering these features (or physically-based combinations of them, e.g. pipe volumes) in the ML metamodels is an area for future development.

6.2.3. Metamodel

The proposed metamodel only predicts hydraulic heads in the nodes and does not provide estimations for flow rates within the pipes. This limitation restricts its utility for applications that specifically require the prediction of flow rates, such as combined sewer overflow estimation, sediment deposition assessment, and water quality evaluations, among others. The inclusion of flow rates would not only extend the scope of variables that the metamodel can predict but also expand the type of available data for its training. Adding the flow rates would potentially enhance the metamodel's performance in predicting hydraulic heads by better capturing the underlying physical principles of the problem, particularly the hydrodynamics of flows in the SWS, which were not considered in the current metamodel.

Similarly, the proposed metamodel does not calculate the flow rate that leaves the system through the nodes. This limitation restricts its utility for urban flooding applications that require the volume of water that leaves the SWS and enters the streets. If the application at hand requires the flow rate through the surcharged nodes as done in Palmitessa et al. (2022), it is advisable to add this variable to the metamodel and the loss function, and increase the training database with more surcharging events.

Another limitation lies on assuming that the SWS starts as a dry network, with zero depth in all nodes and no rain, leading to constant initial conditions for training. While unrealistic for combined sewer systems, it is a practical starting point for simulations. Wet weather conditions' irregularity and intensity outweigh the effect of regular dry weather flow patterns. Extending the metamodel to include dry weather flow could use known initial states or system inflow as starting points.

Finally, this study employed a simplified architecture for both the MLP and GNN-based metamodel. There are multiple points of improvement that can be implemented and explored. The MLP-based metamodel can be extended to use the static features in the SWS, i.e., node elevation, pipe length and diameter. Furthermore, all of the implemented MLPs in this study had a uniform number of hidden units across metamodel. Further exploration of more complex internal MLPs along with variations in the GNN, such as stacking graph layers or using custom layers that combine node and edge embeddings in a more physically-based manner, could lead to improved performance.

In terms of hyperparameter search, these optimal values are difficult to estimate a priori, and are obtained by sampling combinations and testing their performance. Future studies on how to relate these hyperparameters with the physical characteristics of the SWS can offer guidance to metamodel developers.

7. Conclusions and recommendations

This research explored the use of inductive bias for creating a machine learning metamodel of SWMM. The experiments in this study tested the hypothesis that adding domain knowledge of the SWS to the deep learning method would increase the data efficiency of the metamodel; furthermore, it would allow transferability. We summarize key take away messages from this study:

- **Metamodel performance**: Based on the favorable values of fit and speed-up in the test dataset, the results of this investigation show that the proposed metamodel is both accurate and faster than SWMM. Nevertheless, there were some nodes, leaf nodes and close to the network outfalls, with low values of fitness. The errors in these nodes are mainly related to the metamodel architecture which struggles to replicate the internal dynamics of the flow transport. We proposed different approaches for improving both spatial accuracy and execution speed, such as targeted intervention, multi-scale processing, and sequence-to-sequence architecture. Since any metamodel only approximates the response surface of the original model up to certain point, it is the modeler's task to adapt and tune it to the application at hand.
- **Data efficiency:** The proposed metamodel is more data efficient than the MLP-based counterpart. This means that it requires less training examples for reaching certain accuracy; conversely, it can reach a higher accuracy for the same number of training examples in comparison to a metamodel based on a fully connected architecture. Thus, using inductive biases decreases the need for training examples, which in turn decreases the development time of the metamodel.
- **Transferability:** Unlike conventional metamodels that require retraining for each modified configuration, our proposed metamodel exhibits transferability to similar SWS layouts. This allows it to adapt to changes in pipe sizes or network topology while retaining high accuracy. This translates to significant time and resource savings for engineers and urban planners.

Based on the work developed in this study, additional research is needed on speed improvement, metamodel architecture, hyperparameter analysis, and transferability. In addition, we recommend developing alternatives or modifications for adding components such as pumps, and predicting other relevant variables such as flow rates. Finally, we recommend exploring the idea of transferability. This work presents a novel approach that benefits both urban drainage practitioners and water network modeling researchers. Practitioners can efficiently create and transfer SWMM metamodels for computationally intensive tasks like intervention optimization and Monte Carlo simulations. Researchers can leverage the current metamodel's structure for developing new surrogate model architectures tailored to their specific needs or start paving the way for more general foundation metamodels of urban drainage systems.

CRediT authorship contribution statement

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

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.

Data availability

The dataset containing all the SWMM simulations can be found at https://doi.org/10.4121/fec1e3de-9586-4a61-b3a1-02382592e52c. The code repository regarding this study is available at https://doi.org/10.4121/989a0d3d-3b4d-47c7-8677-31c5975f9dec and https://github.com/alextremo0205/SWMM_GNN_Repository_Paper_version.

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

During the preparation of this work the authors used Gemini 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.

Acknowledgments

The authors would like to thank Dr. Job van der Werf for his support on the conversion of the hydraulic model. We also thank Dr. Elvin Isufi and Roberto Bentivoglio for useful discussions. 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.2024.122396.

References

Adadi, A., 2021. A survey on data-efficient algorithms in big data era. J. Big Data 8 (1), http://dx.doi.org/10.1186/s40537-021-00419-9.

Agarap, A.F., 2018. Deep learning using Rectified Linear Units (ReLU). pp. 2–8, CoRR abs/1803.0, (1) URL: http://arxiv.org/abs/1803.08375.

Al-Jarrah, O.Y., Yoo, P.D., Muhaidat, S., Karagiannidis, G.K., Taha, K., 2015. Efficient machine learning for big data: A review. Big Data Res. 2 (3), 87–93. http://dx.doi.org/10.1016/j.bdr.2015.04.001.

Belghaddar, Y., Chahinian, N., Seriai, A., Begdouri, A., Abdou, R., Delenne, C., 2021. Graph convolutional networks: Application to database completion of wastewater networks. Water (Switzerland) 13 (12), 1–19. http://dx.doi.org/10.3390/w13121681.

Bentivoglio, R., Isufi, E., Jonkman, S.N., Taormina, R., 2023. Rapid spatio-Temporal flood modelling via hydraulics-based graph neural networks. Hydrol. Earth Syst. Sci. 27 (23), 4227–4246. http://dx.doi.org/10.5194/hess-27-4227-2023.

Bermúdez, M., Ntegeka, V., Wolfs, V., Willems, P., 2018. Development and comparison of two fast surrogate models for urban pluvial flood simulations. Water Resour. Manag. 32 (8), 2801–2815. http://dx.doi.org/10.1007/s11269-018-1959-8.

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

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

Bronstein, M.M., Bruna, J., Cohen, T., Veli, P., 2021. Geometric deep learning grids, groups, graphs, geodesics, and gauges.

Bronstein, M.M., Bruna, J., Lecun, Y., Szlam, A., Vandergheynst, P., 2017. Geometric deep learning: Going beyond Euclidean data. IEEE Signal Process. Mag. 34 (4), 18–42. http://dx.doi.org/10.1109/MSP.2017.2693418.

Chen, Z., Xiu, D., 2021. On generalized residual network for deep learning of unknown dynamical systems. J. Comput. Phys. 438, 110362. http://dx.doi.org/10.1016/j.jcp.2021.110362.

Chiang, Y.-M., Chang, L.-C., Tsai, M.-J., Wang, Y.-F., Chang, F.-J., 2010. Dynamic neural networks for real-time water level predictions of sewerage systems-covering gauged and ungauged sites. Hydrol. Earth Syst. Sci. 14 (7), 1309–1319. http://dx.doi.org/10.5194/hess-14-1309-2010.

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.

Chow, V.T., Maidment, D.R., Mays, L.W., 1988. Applied hydrology. In: McGraw-Hill Series in Water Resources and Environmental Engineering TA - TT -. McGraw-Hill, New York SE, LK - https://worldcat.org/title/16084171.

Chung, J., Gulcehre, C., Cho, K., Bengio, Y., 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. pp. 1–9, URL: http://arxiv.org/abs/1412.3555.

Delft High Performance Computing Centre, 2022. DelftBlue Supercomputer (Phase 1). URL: https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase1.

Dempsey, P., Eadon, A., Morris, G., 1997. Simpol: A simplified urban pollution modelling tool. Water Sci. Technol. 36 (8–9), 83–88. http://dx.doi.org/10.1016/S0273-1223(97)00615-X.

DHI, M., 2017. 1D-DHI Simulation Engine for 1D River and Urban Modelling-Reference Manual. Danish Hydraulic Institute (DHI), Hørsholm, Denmark.

Dietterich, T.G., 2002. Machine learning for sequential data: A review. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 2396, pp. 15–30. http://dx.doi.org/10.1007/3-540-70659-3_2.

Eden, J.M., Kew, S.F., Bellprat, O., Lenderink, G., Manola, I., Omrani, H., van Oldenborgh, G.J., 2018. Extreme precipitation in the Netherlands: An event attribution case study. Weather Clim. Extremes 21 (November 2017), 90–101. http://dx.doi.org/10.1016/j.wace.2018.07.003.

Fey, M., Lenssen, J.E., 2019. Fast graph representation learning with PyTorch geometric, (1). pp. 1–9, CoRR abs/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.

Hu, W., Liu, B., Gomes, J., Zitnik, M., Liang, P., Pande, V., Leskovec, J., 2019. Strategies for pre-training graph neural networks. pp. 1–22, CoRR abs/1905.1, URL: http://arxiv.org/abs/1905.12265.

Huang, C.-L., Hsu, N.-S., Wei, C.-C., Luo, W.-J., 2015. Optimal spatial design of capacity and quantity of rainwater harvesting systems for urban flood mitigation. Water (Switzerland) 7 (9), 5173–5202. http://dx.doi.org/10.3390/w7095173.

IPCC, 2021. Climate change widespread, rapid, and intensifying. URL: https://www.ipcc.ch/2021/08/09/ar6-wg1-20210809-pr/.

Jung, K.H., 2023. Uncover this tech term: Foundation model. Korean J. Radiol. 24 (10), 1038–1041. http://dx.doi.org/10.3348/kjr.2023.0790.

Keogh, E., Mueen, A., 2017. Curse of dimensionality. In: Encyclopedia of Machine Learning and Data Mining. Springer, Boston, MA, pp. 314–315. http://dx.doi.org/10.1007/978-1-4899-7687-1_192, URL: https://link.springer.com/referenceworkentry/10.1007/978-1-4899-7687-1_192.

Kerimov, B., Bentivoglio, R., Garzón, A., Isufi, E., Tscheikner-Gratl, F., Steffelbauer, D.B., Taormina, R., 2023. Assessing the performances and transferability of graph neural network metamodels for water distribution systems. J. Hydroinform. jh2023031. http://dx.doi.org/10.2166/hydro.2023.031.

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.

Larsen, T., Gujer, W., 1997. The concept of sustainable urban water management. Water Sci. Technol..

Lecun, Y., Bengio, Y., Hinton, G., 2015. Deep learning. Nature 521 (7553), 436–444. http://dx.doi.org/10.1038/nature14539.

Leitão, J.P., Simões, N.E., Maksimović, C., Ferreira, F., Prodanović, D., Matos, J.S., Sá Marques, A., 2010. Real-time forecasting urban drainage models: Full or simplified networks? Water Sci. Technol. 62 (9), 2106–2114. http://dx.doi.org/10.2166/wst.2010.382.

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

Luo, X., Liu, P., Xia, Q., Cheng, Q., Liu, W., Mai, Y., Zhou, C., Zheng, Y., Wang, D., 2023. Machine learning-based surrogate model assisting stochastic model predictive control of urban drainage systems. J. Environ. Manag. 346 (August), 118974. http://dx.doi.org/10.1016/j.jenvman.2023.118974.

Mahmoodian, M., Carbajal, J.P., Bellos, V., Leopold, U., Schutz, G., Clemens, F., 2018. 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.

Makarov, I., Kiselev, D., Nikitinsky, N., Subelj, L., 2021. Survey on graph embeddings and their applications to machine learning problems on graphs. PeerJ Comput. Sci. 7, 1–62. http://dx.doi.org/10.7717/peerj-cs.357.

Mazziotta, M., Pareto, A., 2022. Normalization methods for spatio-temporal analysis of environmental performance: Revisiting the Min–Max method. Environmetrics 33 (5), 1–12. http://dx.doi.org/10.1002/env.2730.

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.

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, URL: https://arxiv.org/abs/1912.01703.

Picard, D., 2021. Torch.manual_seed(3407) is all you need: On the influence of random seeds in deep learning architectures for computer vision, 3407. pp. 1–9, CoRR abs/2109.0, URL: http://arxiv.org/abs/2109.08203.

Pourpanah, F., Abdar, M., Luo, Y., Zhou, X., Wang, R., Lim, C.P., Wang, X.Z., Wu, Q.M., 2023. A review of generalized zero-shot learning methods. IEEE Trans. Pattern Anal. Mach. Intell. 45 (4), 4051–4070. http://dx.doi.org/10.1109/TPAMI.2022.3191696.

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.

Rossman, L.A., 2015. Storm water management model. US EPA, Cincinnati, URL: https://www.epa.gov/water-research/storm-water-management-model-swmm.

Schmid, F., Leandro, J., 2023. An ensemble data-driven approach for incorporating uncertainty in the forecasting of stormwater sewer surcharge. Urban Water J. 00 (00), 1–17. http://dx.doi.org/10.1080/1573062X.2023.2240309.

Schultz, M.T., Small, M.J., Farrow, R.S., Fischbeck, P.S., 2004. State water pollution control policy insights from a reduced-form model. J. Water Resour. Plan. Manag. 130 (2), 150–159. http://dx.doi.org/10.1061/(ASCE)0733-9496(2004)130:2(150), URL: https://www.researchgate.net/publication/248880246_State_Water_Pollution_Control_Policy_Insights_from_a_Reduced-Form_Model.

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.

Tsiami, L., Makropoulos, C., 2021. Cyber—physical attack detection in water distribution systems with temporal graph convolutional neural networks. Water (Switzerland) 13 (9), http://dx.doi.org/10.3390/w13091247.

Turner, R., Eriksson, D., McCourt, M., Kiili, J., Laaksonen, E., Xu, Z., Guyon, I., 2020. Bayesian optimization is superior to random search for machine learning hyperparameter tuning: Analysis of the black-box optimization challenge 2020. Proc. Mach. Learn. Res. 133, 3–26.

Van Bijnen, M., Korving, H., Langeveld, J., Clemens, F., 2017. Calibration of hydrodynamic model-driven sewer maintenance. Struct. Infrastruct. Eng. 13 (9), 1167–1185. http://dx.doi.org/10.1080/15732479.2016.1247287.

van der Werf, J.A., Kapelan, Z., Langeveld, J.G., 2023. Predictive heuristic control: Inferring risks from heterogeneous nowcast accuracy. Water Sci. Technol. 87 (4), 1009–1028. http://dx.doi.org/10.2166/wst.2023.027.

Van Rossum, G., Drake, F.L., 2009. Python 3 Reference Manual. CreateSpace, Scotts Valley, CA.

Vilalta, R., Giraud-Carrier, C., Brazdil, P., Soares, C., 2017. Inductive transfer. In: Sammut, C., Webb, G.I. (Eds.), Encyclopedia of Machine Learning and Data Mining. Springer US, Boston, MA, pp. 666–671. http://dx.doi.org/10.1007/978-1-4899-7687-1_138.

Wad, T., Sun, Q., Pranata, S., Jayashree, K., Zhang, H., 2022. Equivariance and invariance inductive bias for learning from insufficient data. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 13671 LNCS, pp. 241–258. http://dx.doi.org/10.1007/978-3-031-20083-0_15.

Wu, Y., He, K., 2018. Group Normalization. In: European Conference on Computer Vision. ECCV, (no. Figure 1), pp. 3–19, URL: https://research.fb.com/publications/group-normalization/.

Wu, Z., Pan, S., Long, G., Jiang, J., Zhang, C., 2019. Graph wavenet for deep spatial-temporal graph modeling. In: IJCAI International Joint Conference on Artificial Intelligence, vol. 2019-Augus, pp. 1907–1913. http://dx.doi.org/10.24963/ijcai.2019/264.

Xing, L., Sela, L., 2022. Graph Neural Networks for State Estimation in Water Distribution Systems: Application of Supervised and Semisupervised Learning. Journal of Water Resources Planning and Management 148, 1–14. http://dx.doi.org/10.1061/(ASCE)WR.1943-5452.0001550.

Zhang, W., Li, J., Chen, Y., Li, Y., 2019. A surrogate-based optimization design and uncertainty analysis for urban flood mitigation. Water Resour. Manag. 33 (12), 4201–4214. http://dx.doi.org/10.1007/s11269-019-02355-z.

Zhang, Q., Zheng, F., Jia, Y., Savic, D., Kapelan, Z., 2021. Real-time foul sewer hydraulic modelling driven by water consumption data from water distribution systems. Water Res. 188, 116544. http://dx.doi.org/10.1016/j.watres.2020.116544.

Zhou, J., Cui, G., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., Sun, M., 2018. Graph neural networks: a review of methods and applications. pp. 1–22.