Physics-guided Graph Diffusion Network for Combining Heterogeneous Simulated Data: An Application in Predicting Stream Water Temperature

Xiaowei Jia[1], Shengyu Chen[1], Can Zheng[1], Yiqun Xie[2], Zhe Jiang[3], Nasrin Kalanat[1]
[1] University of Pittsburgh, [2] University of Maryland, [3] University of Florida
[1]{xiaowei,shc160,caz51,nak168}@pitt.edu,[2]xie@umd.edu,[3]zhe.jiang@ufl.edu

Abstract

This paper introduces a new method for combining simulated data over different types of nodes in heterogeneous graphs to facilitate predictive learning. Simulation has been widely used in scientific domains to mitigate the need for a large number of observation samples. However, simulated data are often created separately for each type of physical systems while interactions amongst different types of systems remain unexplored. Our method is developed in the context of predicting water temperature in stream networks, which is critical for decision making in water management. In particular, we first develop a graph diffusion network (GDN) to model the interactions amongst stream segments and reservoirs in a heterogeneous graph. We use the GDN model to combine simulated data for both streams and reservoirs in the graph, and use the obtained composite simulations to train the GDN model in a semi-supervised manner. Then the GDN model is further fine-tuned using true observations. Since observation data are often sparse and localized, we further leverage the information from simulations to build a reweighting strategy so as to migitage the discrepancy between training and testing data. Our evaluations in the Delaware River Basin have shown the superiority of the proposed method over multiple baselines using either sparse or localized training data. The proposed GDN model also creates a better composite simulation dataset for heterogeneous graphs.

1 Introduction

Graph neural networks have been widely used to model the spatial dependencies amongst different objects in a variety of scientific applications, including freshwater science [1, 2], bio-medicine [3], and quantum chemistry [4]. Given their ability to capture interactions amongst physical processes, these models often achieve better performance than other machine learning models without spatial awareness. The graph neural networks were further extended to model physical systems with different types of interacting objects and processes using multiple types of nodes and edges, which is also referred to as heterogeneous graphs [5]. For example, heterogeneous graphs have been used to model interactions amongst multiple streams and reservoirs [6]. The water managers can release cold water from reservoirs to cool down the water temperature for the downstream river segments. The river segments and reservoirs need to be modeled separately as different types of nodes because of their difference in water dynamics caused by the distinction in their geometric structures and stratification patterns. Despite the promise of graph network models, the data annotations in real scientific problems are often very sparse, which makes it challenging to train advanced graph network model.

Physics-based models have been widely used to study scientific and engineering systems in scientific domains [7, 8, 9]. Physics-based models are based on known physical laws that govern relations between input and output variables, but most physics-based models are necessarily approximations of reality due to incomplete knowledge of certain processes or excessive complexity in modeling these processes. For example, existing physics-based approaches for river networks simulate the internal distribution of target variables (e.g., water temperature and streamflow) based on general physical relations such as energy and mass conservation. However, given that some processes are not fully understood and many physical variables cannot be easily measured, the model predictions still rely on parameterizations of land surface and subsurface processes based on soil and surficial geologic classification along with topography, land cover, and climate input.

Prior work has shown that machine learning models can achieve better performance using a small number of observation samples after they are pre-trained using simulated data [10, 11]. The idea is that the pre-trained model can extract general physical relationships from simulated data and stay closer to the optimal solution so it requires fewer data samples to fine-tune itself to a quality model. Similar ideas have also been pursued in other domains, such as autonomous vehicle [12]. However, simulations are often less accurate for graph data because calibrating physics-based models is challenging

with the interactions amongst different nodes. Moreover, existing physics-based models are often designed for specific physical systems (i.e., each type of node) without considering the interactions amongst different types of nodes in the heterogeneous graph. For example, the PRMS-SNTemp model [7] has been used to simulate water temperature for streams but it does not consider the cold water release from reservoirs. Hence, the simulation produced by PRMS-SNTemp can often overestimate the stream temperature for the region downstream from reservoirs. Recently, the GLM model [8] was used to simulate water temperature for different depth layers of reservoirs by treating them as artificial lakes [13]. However, the combination of reservoir modeling and stream modeling remains largely unexplored.

To address this issue, we propose Physics-guided Graph Diffusion Network, to combine simulated data for multiple types of nodes in a graph (e.g., streams and reservoirs) and leverage the knowledge embodied in simulations to enhance the learning process. In particular, we create a heterogeneous graph that contain multiple types of nodes and edges, and develop a diffusion graph network for propagating the information of data input, temporal patterns, and simulated data to the spatial neighborhood. Compared to other standard graph neural networks models, the diffusion network allows flexible setting of the neighborhood radius for information propagation in a continuous manner. We further extend the diffusion structure to be adaptive for each type of node in the graph due to the distinct influence of water released from reservoirs and advected from streams.

Specifically, we train the model in two stages. In the first stage, we create new composite simulated labels by propagating simulated data over the graph using the diffusion network. Then we use the obtained composite data to tune the diffusion network in a self-supervised manner. This approach helps fully leverage the knowledge embodied in the simulated data, which can be generated for all the nodes and all the time steps. In the second stage, we fine-tune the graph diffusion network using the available true observations. To address the distribution shift between localized training observations and testing data, we adopt a reweighting strategy to amplify the importance of training samples that are closer to the distribution of testing data.

Our evaluations on real stream data from the Delaware River Basin in a 14-year period show the superiority of the proposed method over multiple baselines, especially when the available observed labels are sparse and localized. Besides, we show that our method can produce a better simulation dataset combining existing simulations on different types of nodes.

2 Related Work

Graph neural networks (GNN) have been widely adopted in a variety of scientific problems due to their capacity in modeling interacting processes [14, 15, 1]. This can be especially beneficial for complex physical systems because modeling interacting processes and calibrating parameters in these processes often require substantial efforts in traditional physics-based models. The GNN models have been further extended to represent heterogeneous interactions amongst different types of objects and processes [16, 17, 6, 18, 19]. Compared to convolutional neural networks (CNN), the GNN-based models are more flexible in representing spatial dependencies amongst irregularly distributed locations, which are common in scientific applications. The GNN-based model can also be used as a building block and combined with other temporal models, e.g., Long-short Term Memory (LSTM) and temporal convolution network (TCN), to capture both the spatial and temporal dependencies [20, 2].

Despite its capabilities, GNN has several limitations when applied to scientific problems. For example, it often prioritizes the nodes with more training samples over other nodes in node-level prediction tasks [21, 22]. Moreover, most existing GNN models were not designed for representing complex physical processes. The nature of scientific studies requires adaptation of these neural network models based on scientific knowledge to better represent the influence amongst processes. Finally, training these models often require a large number of observation samples, which can be difficult to obtain in many real scientific problems.

One promising direction to mitigate these issues is to combine physical simulations with machine learning models. Simulated data can be generated by physics-based models that represent underlying physical processes using a series of mathematical equations. For example, prior work has shown that ML models pre-trained using simulated data can perform much better under data-scarce scenarios for predicting water temperature in lake and stream systems [10, 2, 23, 24]. Similar idea has been pursued in other disciplines [11, 25]. Other works have also explored using simulated target variables to augment input data and achieved improved predictive performance [26, 19]. These methods often require high-quality simulations that are close to reality, otherwise, the improvement can be limited. However, it can be challenging to calibrate physics-based models for complex interacting processes. Moreover, many physics-based models do not consider the interactions amongst different types of processes and objects. Hence, new mechanism is needed to create better simulations to facilitate machine learning in heterogeneous graphs.

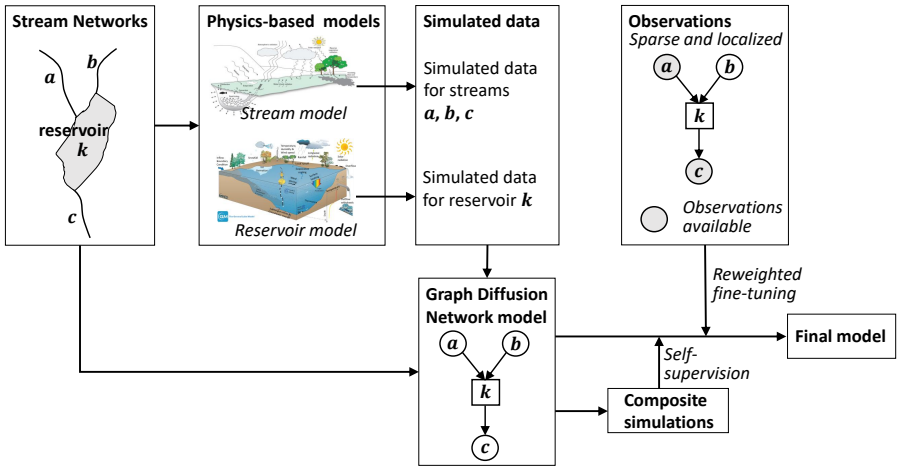


Figure 1: The overall flow of the proposed method.

3 Problem definition

We consider N stream segments and M reservoirs in a stream network. Each stream segment i has input features over multiple daily time steps $\mathbf{X}_i = \{\mathbf{x}_i^1, \mathbf{x}_i^2, ..., \mathbf{x}_i^T\}$. The input features $\mathbf{x}_i^t$ include climate drivers and geometric parameters of the segment (more details can be found in Section 5.1). Similarly, for each reservoir j, we are provided with its input features $\mathbf{XR}_j = \{\mathbf{xr}_j^1, \mathbf{xr}_j^2, ..., \mathbf{xr}_j^T\}$, and the features $\mathbf{xr}_j^t$ include climate drivers and its meta-features, e.g., the height and width of the dam. We also have observed water temperature $\mathbf{Y} = \{y_i^t\}$ for certain segments and on certain dates. This work is focused on predicting water temperature only for N stream segments. The proposed method can be easily extended to model target variables for different types of nodes in heterogeneous graphs.

We use a graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \mathbf{A}\}$ to represent dependencies amongst stream segments and reservoirs. Here the node set $\mathcal{V} = \{\mathcal{V}_s, \mathcal{V}_r\}$ contains the set of river segments $\mathcal{V}_s$ and reservoirs $\mathcal{V}_r$. The edge set $\mathcal{E} = \{\mathcal{E}_{ss}, \mathcal{E}_{sr}, \mathcal{E}_{rs}\}$ contains three types of edges among river segments and reservoirs. Specifically, $\mathcal{E}_{ss}$ represents the edges between pairs of segments (i, j) where the segment i is upstream from the segment j, $\mathcal{E}_{sr}$ represents the edges between river segments and their downstream reservoirs, and $\mathcal{E}_{rs}$ represents the edges between reservoirs and their downstream river segments. We create three adjacency matrices $\mathbf{A}^{ss} \in \mathbb{R}^{N \times N}$, $\mathbf{A}^{sr} \in \mathbb{R}^{N \times M}$, and $\mathbf{A}^{rs} \in \mathbb{R}^{M \times M}$ based on the inverse of stream distance between streams to streams, streams to reservoirs, and reservoirs to streams. All the three matrices are row-normalized.

4 Method

This section provides the details of the proposed algorithm (Fig. 1). We first introduce the structure of the graph diffusion network. Then we discuss the adaptive diffusion structure and the self-supervised learning using the composite simulations. Finally, we describe the fine-tuning process with the reweighting strategy.

4.1 Diffusion networks on the heterogeneous graph

The proposed graph diffusion networks (GDN) captures both the spatial dependencies amongst different nodes in the heterogeneous graph and the temporal dynamics for each node. It maintains two sets of states for the two types of nodes, i.e., stream segment states $\{\mathbf{cs}_i^t\}_{i=1:N}^{t=1:T}$ and reservoir state $\{\mathbf{cr}_j^t\}_{j=1:M}^{t=1:T}$. Each state variable is updated over time using a temporal structure that is similar to the LSTM model. Specifically, LSTM creates the model state at each time step by combining the current input data and the previous state. For example, for each stream segment i, its state can be updated as $\mathbf{cs}_i^t = \text{LSTM}(\mathbf{x}_i^t, \mathbf{cs}_i^{t-1})$, where the function LSTM involves a series of gating functions to filter the information from the previous time steps and the current time step, and combine them to compute the current state. The obtained state $\mathbf{cs}_i^t$ can then be used to create hidden representation $\mathbf{h}_i^t$ and prediction $\hat{y}_i^t$.

To further incorporate the spatial dependencies in updating the stream state, the GDN model updates the stream state as follows:

$$\mathbf{cs}_i^t = f_s(\mathbf{x}_i^t, \mathbf{cs}_i^{t-1}, \mathbf{p}_i^{t-1}), \tag{4.1}$$

where $\mathbf{p}_i^{t-1}$ captures the influence from node i's neighborhood, which is described later. We use the neighborhood influence from the previous time step $t-1$ due to the time delay of influence propagation, e.g., the travel time for water flows advected from upstream to downstream segments. The function f_s is defined in a similar way with the LSTM model. Specifically, it uses three gating variables, input gate $\mathbf{ig}_i^t$, forget gate $\mathbf{fg}_i^t$, and spatial gate $\mathbf{sg}_i^t$, to filter the information of current input data at time t , the previous state $\mathbf{cs}_i^{t-1}$, and the neigh-

borhood influence $\mathbf{p}_i^{t-1}$, respectively. The input gate and forget gate are created using the current input data $\mathbf{x}_i^t$ and the previous hidden representation $\mathbf{h}_i^{t-1}$ through a fully connected layer. The spatial gate is created using the current input $\mathbf{x}_i^t$ for the node i and its spatial influence $\mathbf{p}_i^{t-1}$ through a fully connected layer. All the gating variables are computed with a sigmoid function after the fully connected layer so their values are within (0,1).The filtered information from different sources are then combined to compute the current stream state as

$$\mathbf{cs}_i^t = \mathbf{ig}_i^t \odot \bar{\mathbf{cs}}_i^t + \mathbf{fg}_i^t \odot \mathbf{c}_i^{t-1} + \mathbf{sg}_i^t \odot \mathbf{p}_i^{t-1}, \tag{4.2}$$

where $\odot$ denotes entry-wise product, and $\bar{\mathbf{cs}}_i^t$ represents the candidate stream state, which encodes the information from the current step, and is computed using $\mathbf{x}_i^t$ and $\mathbf{h}_i^t$ using a fully connected layer, following the same process as in the standard LSTM.

To neighborhood influence $\mathbf{p}_i^{t-1}$ aggregates the information from the neighbors of node i in the graph, including both streams ($\mathcal{N}(i)$) and reservoirs ($\mathcal{M}(i)$) in the neighborhood. The aggregation is conducted through a graph diffusion process, as

$$\mathbf{p}_i^{t-1} = \tanh\Big(\sum_{i' \in \mathcal{N}(i)} \mathbf{T}_{i'i}^{ss} \phi_s(\mathbf{cs}_{i'}^{t-1}) + \sum_{j \in \mathcal{M}(i)} \mathbf{T}_{ji}^{rs} \phi_r(\mathbf{cr}_j^{t-1})\Big), \tag{4.3}$$

where the functions ϕ_s and ϕ_r are transformations implemented by fully connected networks, and $\mathbf{T}^{ss}$ and $\mathbf{T}^{rs}$ represent the diffusion matrices for stream-stream interactions and reservoir-stream interactions, respectively. The diffusion matrices are essentially generalized adjacency matrices that capture multi-hop spatial dependencies. This is especially important for modeling stream networks because the water flows can affect multiple downstream segments and the range of such influence can depend on catchment characteristics such as soil properties, groundwater, and land covers.

In particular, the diffusion matrices can be computed from the initial adjacency matrix. For example, the most common way of building a diffusion matrix is through weighted sum of a power series of the adjacency matrix , i.e., $\mathbf{T}^{ss} = \sum_k \theta_k [\mathbf{A}^{ss}]^k$, where θ_k is the weight for k-hop dependencies. Our proposed GDN model uses an adaptive diffusion strategy that distinguishes between different types of nodes (i.e., streams and reservoirs) as well as different reservoirs. The details about how to create diffusion matrices in the proposed GDN model will be introduced in Section. 4.2.

We further augment the GDN model with the simulated target variables for all the nodes. Given the simulated water temperature $\tilde{y}_i^t$ for each stream segment i at time t, we augment the transformed stream state $\phi_s(\mathbf{cs}_i^t)$ in Eq. 4.3 as follows:

$$\tilde{\phi}_s(\mathbf{cs}_i^t) = \phi_s(\mathbf{cs}_i^t) + \psi_s(\tilde{y}_i^t), \tag{4.4}$$

where ψ_s is a fully connected network to transform the simulated data.

Similarly, we augment the transformed reservoir state $\phi_r(\mathbf{cr})$ using the simulated reservoir temperature $\tilde{\mathbf{r}}$. When aggregating the influence from reservoirs to streams, we have to consider not only the simulated water temperature for each reservoir, but also the amount of water released from the reservoir. Combining these data can help estimate the water temperature for the water flows released from each reservoir. The augmentation of reservoir state can be expressed as:

$$\tilde{\phi}_r(\mathbf{cr}_j^t) = \phi_r(\mathbf{cr}_j^t) + \psi_r(\mathbf{fl}_j^t, \tilde{\mathbf{r}}_j^t), \tag{4.5}$$

where the function ψ_r transforms the released flow $\mathbf{fl}_j^t$ and the simulated water temperature $\tilde{\mathbf{r}}_j^t$, both of which are available for multiple depth layers in the reservoir. In contrast, the simulated stream temperature $\tilde{y}_i^t$ is a scalar because streams are much shallower and are often assumed to be well-mixed.

Following the same process as Eqs. 4.1 and 4.2, we update reservoir states using another function f_r, as

$$\mathbf{cr}_j^t = f_r(\mathbf{xr}_j^t, \mathbf{cr}_j^{t-1}, \mathbf{pr}_j^{t-1}), \tag{4.6}$$

where the neighborhood influence $\mathbf{pr}_j^{t-1}$ for each reservoir j aggregates the information from its upstream river segments $\mathcal{U}(j)$. Here we assume that reservoirs do not interfere with each other because reservoirs are often built distant from each other. Specifically, the neighborhood influence $\mathbf{pr}_j^{t-1}$ is computed as follows:

$$\mathbf{pr}_j^{t-1} = \tanh\Big(\sum_{i \in \mathcal{U}(j)} \mathbf{T}_{ij}^{sr} \phi_p(\tilde{\mathbf{cs}}_i^{t-1})\Big), \tag{4.7}$$

where ϕ_p is implemented by fully connected networks.

Since this work is focused on predicting stream water temperature, we create hidden representation and predicted output from the stream states, as follows:

$$\begin{aligned} \mathbf{h}_i^t &= \mathbf{og}_i^t \odot \tanh(\mathbf{cs}_i^t), \\ \hat{y}_i^t &= \mathbf{w}_y \mathbf{h}_i^t + b_y, \end{aligned} \tag{4.8}$$

where $\mathbf{og}_i^t$ represents the output gate variable used to filter the stream state. The output gate $\mathbf{og}_i^t$ is computed following the standard LSTM model, i.e., by transforming the current input and the previous hidden representation via a fully connected layer. The overall model structure is shown in Fig. 2.

4.2 Adaptive graph diffusion and self-supervised learning In a standard homogeneous graph, the diffusion matrix $\mathbf{T}$ used for graph convolution is often computed as the aggregated power series of the adjacency matrix $\mathbf{A}$, as follows:

$$\mathbf{T} = \sum_k \theta_k \mathbf{A}^k, \tag{4.9}$$

where θ_k represents the weight for the k-hop transition relations, i.e., $\mathbf{A}^k$. The weight coefficients $\{\theta_k\}$ satisfy $\sum_k \theta_k = 1$. When the weight θ_k is higher for larger k values, the model will focus on long-distance information propagation. Intuitively, the radius of the information propagation can be defined as $\sum_k k\,\theta_k$.

This work adopts the heat kernel for defining the weights $\{\theta_k\}$. The heat kernel describes the heat diffusion process [27], and the weight is estimated as:

$$\theta_k = e^{-\eta}\frac{\eta^k}{k!}, \tag{4.10}$$

where η is the diffusion parameter. It can be proved that the radius of the diffusion matrix in Eq. 4.9 using the heat kernel has the radius of η [28]. Hence, the use of heat kernel with $\eta \in [0, \infty)$ enables defining the spatial neighborhood in a continuous manner.

We now extend this diffusion structure to the heterogeneous graph. Different types of nodes in the graph often have different properties for diffusion so they need to be modeled using separate diffusion matrices and parameters. We first create the diffusion matrix amongst stream segments, as follows:

$$\mathbf{T}^{ss} = \sum_{k=0}^{\infty} e^{-\eta}\frac{\eta^k}{k!}[\mathbf{A}^{ss}]^k = e^{-\eta\mathbf{L}}, \tag{4.11}$$

where L is the Lapacian matrix $\mathbf{I} - \mathbf{A}^{ss}$. The last equation is obtained using the matrix exponential function.

Similarly, we create the diffusion matrix between reservoirs and streams. Here we use γ to represent the diffusion parameter for the reservoirs, as follows:

$$\begin{aligned}
\mathbf{T}^{rs} &= \theta_0\mathbf{A}^{rs} + \theta_1\mathbf{A}^{rs}\mathbf{A}^{ss} + ... + \theta_k\mathbf{A}^{rs}[\mathbf{A}^{ss}]^k + ... \\
&= \mathbf{A}^{rs}\sum_{k=0}^{\infty} e^{-\gamma}\frac{\gamma^k}{k!}[\mathbf{A}^{ss}]^k = \mathbf{A}^{rs}e^{-\gamma\mathbf{L}}, \\
\mathbf{T}^{sr} &= \theta_0\mathbf{A}^{sr} + \theta_1\mathbf{A}^{ss}\mathbf{A}^{sr} + ... + \theta_k[\mathbf{A}^{ss}]^k\mathbf{A}^{sr} + ... \\
&= \sum_{k=0}^{\infty} e^{-\gamma}\frac{\gamma^k}{k!}[\mathbf{A}^{ss}]^k\mathbf{A}^{sr} = e^{-\gamma\mathbf{L}}\mathbf{A}^{sr}.
\end{aligned} \tag{4.12}$$

Here each entry $\mathbf{T}^{rs}_{ji}$ represents the connection strength between a reservoir j and a multi-hop downstream river segment i, and each entry $\mathbf{T}^{sr}_{ij}$ represents the connection strength between a river segment i and a multi-hop downstream reservoir j. Different from $\eta \in [0, \infty)$ in Eq. 4.11, the value of γ is in $[-1, \infty)$ because the term $e^{-\gamma\mathbf{L}}$ captures the connection between one river segment and the outlet/inlet segment of a reservoir, and $\gamma = -1$ indicates that we consider only the connection between the outlet/inlet river segment and the reservoir captured by $\mathbf{A}^{rs}/\mathbf{A}^{sr}$.

We further make this model to be adaptive for each reservoir node in the graph. This is because different reservoirs are often operated with different priorities in water management, e.g., maintaining water temperature for aquatic life or sufficient water supply, and thus they influence downstream river segments in different ways. Given that we usually have a much smaller number of reservoirs compared to the number of stream segments, we estimate the parameter γ separately for each reservoir in the graph.

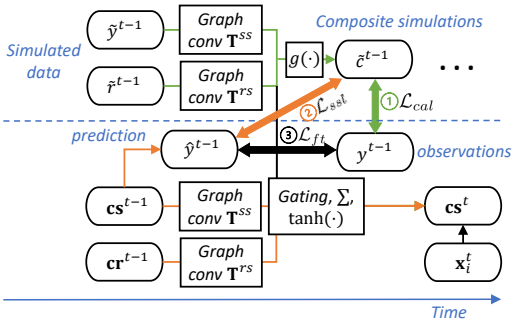


Figure 2: The structure of the proposed GDN model (only for stream state update) and its training objectives. The first step is to generate composite simulations (in orange) by minimizing $\mathcal{L}_{cal}$. The second step is to train the lower part of the GDN model (in orange) by minimizing $\mathcal{L}_{ssl}$. Finally, the entire model is fine-tuned with true observations by minimizing $\mathcal{L}_{ft}$.

In particular, we leverage simulated data from different nodes to calibrate the diffusion parameters η, and γ (separately for each reservoir). The objective is to create composite simulated data using the GDN model by reducing the discrepancy with the true observations. The composite simulated label $\tilde{c}_i^t$ is computed as follows:

$$\tilde{c}_i^t = g\Big(\sum_{i' \in \bar{\mathcal{N}}(i)} \mathbf{T}^{ss}_{i'i}\psi_s(\tilde{y}_{i'}^t) + \sum_{j \in \mathcal{M}(i)} \mathbf{T}^{rs}_{ji}\psi_r(\mathbf{fl}_j^t, \tilde{\mathbf{r}}_j^t)\Big), \tag{4.13}$$

where $\bar{\mathcal{N}}(i)$ denotes the spatial neighbors of node i and the node i itself, and the function g is implemented using fully connected neural networks.

The diffusion parameters η and γ are estimated by minimizing the difference between the composite simulations and true observations, as follows:

$$\min_{\Phi} \mathcal{L}_{cal} = \frac{\sum_{y_i^t \in \mathbf{Y}}(\tilde{c}_i^t - y_i^t)^2}{|\mathbf{Y}|} \tag{4.14}$$

where the parameter set Φ includes η, γ, and parameters in ψ_s, ψ_r, and g.

Then, we conduct a self-supervised training using the generated composite simulations. The objective of the self-supervised learning is to pre-train the GDN model to predict the composite simulations using input features. In this step, GDN creates prediction $\hat{y}_i^t$ using only the transformation of input features (i.e., the components in orange color in Fig. 2) without the

augmentation by Eqs. 4.4 and 4.5. The loss function for the self-supervised learning is shown as follows:

$$\min_{\Psi} \mathcal{L}_{ssl} = \frac{\sum_{i,t}(\hat{y}_i^t - \tilde{c}_i^t)^2}{NT}, \tag{4.15}$$

where the parameter set Ψ includes all the model parameters Θ except parameters in Φ, i.e., $\Psi = \Theta \backslash \Phi$.

4.3 Fine-tuning Next, we fine-tune the entire model using available observations. One major challenge is that the observations are very sparse and localized, resulting in a different distribution with testing data. To address this challenge, we aim to leverage the input features and simulated data that are available over all the nodes to discover the discrepancy between localized observations and the testing data.

Specifically, we propose a reweighting strategy to increase the weights for those training samples that are closer to the distribution of testing samples. We build a classifier $\mathcal{D}$ to distinguish between training and testing data, which is implemented as a two-layer fully-connected network. The classifier takes the hidden representation from the GDN model, which encodes the information of input data, temporal patterns, and physical simulations. Its output is in the range of [0,1], and is closer to 1 if it predicts the data to be more likely from the target years and otherwise is closer to 0. Then the weight of each training sample is $w_i^t = \mathcal{D}(\mathbf{h}_i^t)$.

The obtained weights can then be used in the training loss function to alleviate the temporal data shift in the training process, as follows:

$$\min_{\mathbf{\Theta}} \mathcal{L}_{ft} = \frac{\sum_{y_i^t \in \mathbf{Y}} w_i^t (\hat{y}_i^t - y_i^t)^2}{|\mathbf{Y}|} \tag{4.16}$$

5 Experimental Results

5.1 Dataset and evaluation details The dataset used in our evaluation is from U.S. Geological Survey (USGS)'s National Water Information System [29] and the Water Quality Portal [30]. The river segments were defined by the national geospatial fabric used for the National Hydrologic Model [31], and the river segments are split up to have roughly a one day water travel time. We study a subset of the Delaware River Basin with 56 river segments at Lordville, New York. We select this subset since we have sufficient observations collected in this area. In particular, we use input features at the daily scale from Jan 01, 1980, to June 22, 2020 (14,784 dates). The input features have 10 dimensions, which include daily average precipitation, daily average air temperature, date of the year, solar radiation, shade fraction, potential evapotranspiration and the geometric features of each segment (e.g., elevation, length, slope and width). Air temperature, precipitation, and solar radiation values were derived from the gridMET gridded dataset [32]. Other input features (e.g., shade fraction, potential evapotranspiration) are difficult to measure frequently, and we use values internally calculated by the physics-based PRMS-SNTemp model [33]. Water temperature observations were available for 29 segments but the temperature was observed only on certain dates (a total of 76,163 observations). In addition, we have the water release data for the two reservoirs in this region, i.e., the Cannonsville Reservoir and Pepacton Reservoir. The release data includes how much water is released under each type of release strategy (conservation-based release, direct water release, and water spill release) at a daily scale. We also have meteorological data for the locations of reservoirs as well as the meta-features of these reservoirs, including dam height, dam length, depth, elevation, and area of catchment.

The following tests use data from Jan 01, 1980, to Dec 25, 2006, for training and then measure the performance from Dec 26, 2006, to June 22, 2020. The evaluation is focused on stream temperature prediction.

The proposed model is implemented using GTX 3080 GPU and AMD Ryzen 9 5950X 16-Core Processor. The training uses the ADAM optimizer [34] with an initial learning rate of 0.002. All the hidden variables and gating variables have 20 dimensions.

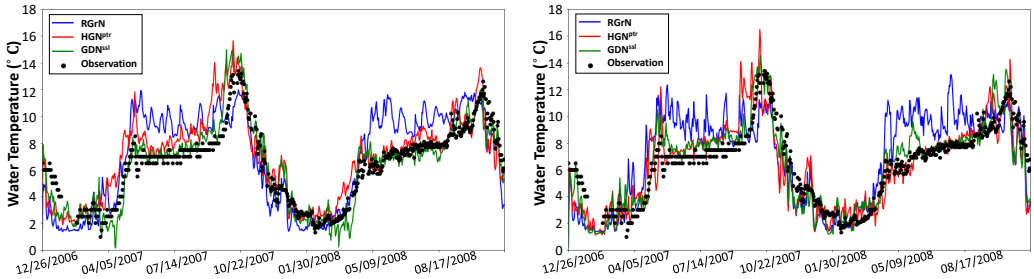
We conduct experiments to answer the following questions: *Q1: Can the proposed outperform existing methods, especially under data-sparse scenarios?*

Q2: What is the effect of the reweighting strategy in fine-tuning the model using localized observations?

Q3: Can GDN create better simulated labels?

Q4: How will the performance change if the model does not use adaptive diffusion structures?

5.2 Predictive performance We measure the predictive root mean squared error (RMSE) on streams for the proposed GDN method against several baselines, including physics-based PRMS-SNTemp (SNTemp) model [7], the RNN model with the LSTM cell (RNN), Recurrent Graph Networks (RGrN) [2], HydroNets [1], and Heterogeneous Graph Networks (HGN) [6]. Here RNN is trained using data from all the streams but it does not consider spatial dependencies. The graph-based methods RGrN and HydroNets only capture stream interactions but do not consider the impact of reservoirs. The HGN model considers both streams and reservoirs using a heterogeneous graph representation, and is implemented in multiple versions (HGN^{ptr} and HGN^{aug}) to further show the effect of incorporating simulated data. The HGN^{ptr} method first pre-trains HGN using simulated stream temperature and then fine-tunes



(a) (b)

Figure 3: The water temperature predictions made by different models using (a)100% and (b) 2% training data.

Table 1: Mean prediction RMSE (± standard deviation) for temperature modeling using 0.1%, 2% and 100% training labels from five runs with random initialization and data sampling. Rows in grey uses simulated data.

Method	0.1%	2%	100%
SNTemp	4.06	4.06	4.06
RNN	3.03±0.09	2.20±0.04	1.95±0.02
RGrN	2.81±0.06	2.13±0.03	1.84±0.03
HydroNets	2.95±0.08	2.15±0.04	1.90±0.03
HGN	2.79±0.05	1.93±0.06	1.48±0.03
HGN^{ptr}	2.54±0.07	1.81±0.06	1.48±0.02
HGN^{aug}	2.62±0.07	1.81±0.06	1.39±0.02
GDN	2.57±0.05	1.89±0.05	1.41±0.02
GDN^{aug}	2.52±0.05	1.80±0.08	1.23±0.03
GDN^{ssl}	2.42±0.08	1.75±0.08	1.20±0.02

HGN using true observations. Here the pre-training uses the composite simulations from a USGS data release that combines SNTemp simulations and reservoir simulations using a distance-based decay function [13]. HGN^{aug} uses the simulations to augment the state variables, in the same way as in the proposed GDN model (described in Section 4.1). We also implement three versions of the proposed GDN model: GDN, GDN^{aug}, and GDN^{ssl}. The first version GDN uses the heterogeneous graph diffusion model, but does not use the simulated data. The GDN^{aug} method uses the simulated data to augment the intermediate model states (Section 4.1), and the GDN^{ssl} method further conducts the self-supervised learning using the obtained composite simulations (Section 4.2). We also randomly select a portion of observation data from the training period (0.1%, 2%) for tuning each model.

Table 1 and Fig. 3 show that our method outperforms all the baselines. We also have several observations: (1) RGrN and HydroNets perform better than RNN because they can capture stream interactions. HydroNets uses both global and node-specific parameters, leading to worse performance for poorly-observed stream segments compared to RGrN. (2) The improvement from RGrN to HGN and GDN confirms the effectiveness of modeling different types of nodes. GDN outperforms HGN because it betters captures long-distance effect of water flows. (3) The methods using simulated data (HGN^{ptr}, HGN^{aug}, GDN^{aug}, GDN^{ssl}) are better than their prototype models without using the simulated data. Also, the augmentation-based method (HGN^{aug} and GDN^{aug}) generally lead to better performance when using a larger number of training samples while the methods with a separate pre-training stage (HGN^{ptr} and GDN^{ssl}) often perform better using sparser observations.

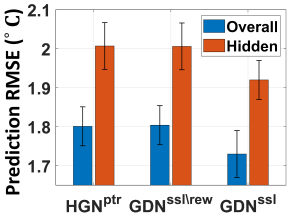


Figure 4: The prediction RMSE over all the segments and only the segments that are not involved in training.

5.3 Localized data To verify the model performance using localized training data, we hide data from a set of four stream segments $\mathcal{S}$ and each stream in $\mathcal{S}$ has over 4,000 training observations. The total training observations in $\mathcal{S}$ covers 56.77% of all the training samples. We measure the performance of HGN^{ptr}, GDN^{ssl} without using the reweighting strategy ($\text{GDN}^{\text{ssl}\backslash\text{rew}}$), and GDN^{ssl} using the reweighting strategy (GDN^{ssl}).

Fig. 4 shows the performance measured over all the testing data and over only the hidden segments $\mathcal{S}$. All the methods have larger errors on the hidden segments compared to the prediction error over all the testing data. This indicates that data patterns from the set of nodes with training data may not apply to other nodes. This can be due to the difference of characteristics across stream segments, e.g., soil properties, groundwater. The GDN^{ssl} method has a smaller error compared to HGN^{ptr} and $\text{GDN}^{\text{ssl}\backslash rew}$ because training samples are reweighted to reduce the discrepancy between local training data and testing data.

5.4 The accuracy of composite simulations Table 2 shows the accuracy of the composite simulated

Table 2: The RMSE of different simulation dataset for the entire dataset, the segments downstream from reservoirs, and other stream segments.

Method	Overall	Down-Reservoirs	Others
SNTemp	4.06	4.06	4.06
Exp-decay	3.19	2.64	3.60
GDN-composite	2.73	2.54	2.90

data created by the proposed GDN model (GDN-composite) as well as the original simulated water temperature on streams (SNTemp) and the composite simulations released by USGS (Exp-decay) [13]. The Exp-decay method uses a linear combination of simulated water temperature for streams and released flows from reservoirs with a decay function. Specifically, for a segment i downstream from the reservoir j, the composite simulation is $c_i^t = w\mathbf{fl}_j^t \cdot \tilde{\mathbf{r}}_j^t + (1-w)\tilde{y}_i^t$, where the weight $w = \exp(-\alpha\, d)$, and d is the stream distance from the stream segment to the reservoir. The parameter α is calibrated using true observations.

We can observe that both GDN-composite and Exponential-decay simulations are better than the SNTemp-based simulations. By comparing GDN-composite and Exponential-decay, we can see that GDN-composite is slightly better for streams downstream from reservoirs while being much better for other segments. This is because GDN also propagates simulated water temperature amongst different stream segments while the Exponential-Decay modifies stream simulations considering only the impact of reservoirs.

Fig. 5 shows simulated data generated by different methods for two streams downstream from the Cannonsville Reservoir. It can be seen that the SNTemp over-estimates the water temperature because it does not consider the cold water released at the beginning of the summer period. The Exp-decay method predicts lower water temperature at the beginning of the summer period because it combines the simulations for the cold water release and the stream water temperature. However, the effect of cold release on downstream streams cannot be fully captured by a simple linear combination. Also, the impact on water temperature may not be reflected immediately. In contrast, GDN-composite better matches true observations.

5.5 Adaptive diffusion model Finally, we test the performance using different numbers of diffusion parameters in the GDN model. We report the performance of three versions of GDN$^{\text{ssl}}$ model in Table 3. First, we use a single diffusion parameter for all the nodes (*Single*), i.e., η=γ in Eqs. 4.11 and 4.12. Second, we use two scaling parameters η and γ for stream segments and reservoirs, respectively (*Node Type*). Third, we also calibrate different diffusion parameters for the Cannonsville Reservoir and the Pepacton Reservoir (*Adaptive*). It can be seen that the simulated data created by all the three models have similar performance. However, the Adaptive approach generates better final predictions.

Table 3: The RMSE of the simulations and final predictions generated by different GDN models.

Method	Simulation	Final prediction
Single	2.75	1.36
Node type	2.73	1.33
Adaptive	2.73	1.20

6 Conclusion

In this paper, we build a GDN model with adaptive diffusion structures for different types of nodes in the heterogeneous graph. The GDN model is applied to transform and combine both input features and simulated data over different nodes. We first use the GDN model to create composite simulations over all the nodes, and then train the GDN model in a self-supervised fashion. Finally, it is fine-tuned using reweighted observed labels. Our evaluations demonstrate that the proposed method can improve the predictive performance even using sparse and localized training samples. The GDN model also creates new composite simulations, which have higher accuracy for streams either affected or not affected by reservoirs. The adaptive diffusion structure is also shown to contribute to better performance. Although this method is developed in the context of modeling stream networks, the method can be generally applicable to a variety of scientific problems with heterogeneous interacting processes. Our future work will also pursue interpreting the GDN combination of simulations to aid in the design physics-based models.

Acknowledgements

This work was supported by NSF awards 2147195, 2105133, and 2126474, NASA award 80NSSC22K1164, the USGS awards G21AC10207, G21AC10564, and G22AC00266, Google's AI for Social Good Impact Scholars program, the DRI award at the University of Maryland, and CRC at the University of Pittsburgh.

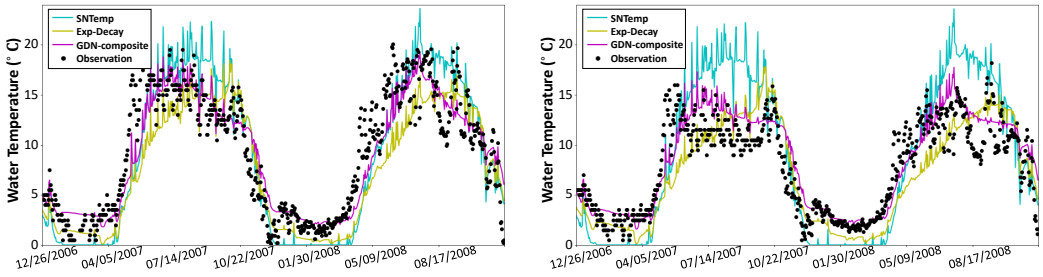
References

[1] Zach Moshe et al. Hydronets: Leveraging river structure for hydrologic modeling. *arXiv preprint arXiv:2007.00595*, 2020.

[2] Xiaowei Jia et al. Physics-guided recurrent graph model for predicting flow and temperature in river networks. In *SDM*, 2021.

[3] Pietro Bongini et al. Molecular generative graph neural networks for drug discovery. *Neurocomputing*, 2021.

[4] Justin Gilmer et al. Neural message passing for quantum chemistry. In *ICML*, 2017.



(a) (b)

Figure 5: The water temperature simulated by different methods in two streams.

[5] Chuxu Zhang et al. Heterogeneous graph neural network. In *SIGKDD*, 2019.

[6] Shengyu Chen et al. Heterogeneous stream-reservoir graph networks with data assimilation. In *ICDM*, 2021.

[7] Steven L Markstrom et al. Prms-iv, the precipitation-runoff modeling system, version 4. *USGS*, 2015.

[8] Matthew R Hipsey et al. A general lake model (glm 3.0) for linking with high-frequency sensor data from the global lake ecological observatory network (gleon). *Geoscientific Model Development*, 2019.

[9] PM Cox et al. The impact of new land surface physics on the gcm simulation of climate and climate sensitivity. *Climate Dynamics*, 1999.

[10] Xiaowei Jia et al. Physics-guided machine learning for scientific discovery: An application in simulating lake temperature profiles. *ACM/IMS TDS*, 2021.

[11] Stephan Rasp and Nils Thuerey. Data-driven medium-range weather prediction with a resnet pretrained on climate simulations: A new model for weatherbench. *JAMES*, 2021.

[12] Shital Shah et al. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and service robotics*, 2018.

[13] S.K. Oliver et al. Predicting water temperature in the delaware river basin: U.s. geological survey data release. 2021.

[14] Tian Xie et al. Crystal graph convolutional neural networks for an accurate and interpretable prediction of material properties. *Physical review letters*, 2018.

[15] Di Zhu et al. Understanding place characteristics in geographic contexts through graph convolutional neural networks. *AAG*, 2020.

[16] Daiguo Deng et al. Describe molecules by a heterogeneous graph neural network with transformer-like attention for supervised property predictions. *ACS omega*, 2022.

[17] Jiajie Peng et al. An end-to-end heterogeneous graph representation learning-based framework for drug–target interaction prediction. *Briefings in Bioinformatics*, 2021.

[18] Shengyu Chen, Jacob A Zwart, and Xiaowei Jia. Physics-guided graph meta learning for predicting water temperature and streamflow in stream networks. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 2752–2761, 2022.

[19] Xiaowei Jia et al. Modeling reservoir release using pseudo-prospective learning and physical simulations to predict water temperature. In *SDM*, 2022.

[20] Zonghan Wu et al. Graph wavenet for deep spatial-temporal graph modeling. *arXiv:1906.00121*, 2019.

[21] Qi Zhu et al. Shift-robust gnns: Overcoming the limitations of localized graph training data. *NeurIPS*, 2021.

[22] Jiaqi Ma et al. Subgroup generalization and fairness of graph neural networks. *NeurIPS*, 2021.

[23] Jordan S Read et al. Process-guided deep learning predictions of lake water temperature. *WRR*, 2019.

[24] Xiaowei Jia, Yiqun Xie, Sheng Li, Shengyu Chen, Jacob Zwart, Jeffrey Sadler, Alison Appling, Samantha Oliver, and Jordan Read. Physics-guided machine learning from simulation data: An application in modeling lake and river systems. In *2021 IEEE International Conference on Data Mining (ICDM)*, pages 270–279. IEEE, 2021.

[25] Herim Han et al. Transfer learning from simulation to experimental data: Nmr chemical shift predictions. *The Journal of Physical Chemistry Letters*, 2021.

[26] Anuj Karpatne et al. Physics-guided neural networks (pgnn): An application in lake temperature modeling. *arXiv preprint arXiv:1710.11431*, 2017.

[27] David Vernon Widder. *The heat equation*. 1976.

[28] Jialin Zhao et al. Adaptive diffusion in graph neural networks. *NeurIPS*, 2021.

[29] US Geological Survey. USGS water data for the Nation: U.S. Geological Survey National Water Information System database, 2016.

[30] Emily K Read et al. Water quality data for national-scale aquatic research: The water quality portal. *Water Resources Research*, 2017.

[31] R Steven Regan et al. Description of the national hydrologic model for use with the precipitation-runoff modeling system. Technical report, USGS, 2018.

[32] gridMET - Climatology Lab. `http://www.climatologylab.org/gridmet.html`.

[33] FD Theurer et al. Instream water temperature model. instream flow information paper 16. us fish wildl serv. *Div. Biol. Serv., Tech. Rep. FWS OBS*, 1984.

[34] Diederik P Kingma et al. Adam: A method for stochastic optimization. *arXiv:1412.6980*, 2014.