sensors

MDPI

Article

A Probabilistic Digital Twin for Leak Localization in Water Distribution Networks Using Generative Deep Learning

Nikolaj T. Mücke [1,2,*], Prerna Pandey [3], Shashi Jain [3], Sander M. Bohté [1,4,5] and Cornelis W. Oosterlee [2]

[1] Centrum Wiskunde & Informatica, Science Park 123, 1098 XG Amsterdam, The Netherlands; sbohte@cwi.nl
[2] Mathematical Institute, Utrecht University, 3584 CS Utrecht, The Netherlands; c.w.oosterlee@uu.nl
[3] Department of Management Studies, Indian Institute of Science, Bangalore 560012, India; preranap@iisc.ac.in (P.P.); shashijain@iisc.ac.in (S.J.)
[4] Swammerdam Institute of Life Sciences (SILS), University of Amsterdam, 1098 XH Amsterdam, The Netherlands
[5] Bernoulli Institute, Rijksuniversiteit Groningen, 9747 AG Groningen, The Netherlands
* Correspondence: nikolaj.mucke@cwi.nl

Abstract: Localizing leakages in large water distribution systems is an important and ever-present problem. Due to the complexity originating from water pipeline networks, too few sensors, and noisy measurements, this is a highly challenging problem to solve. In this work, we present a methodology based on generative deep learning and Bayesian inference for leak localization with uncertainty quantification. A generative model, utilizing deep neural networks, serves as a probabilistic surrogate model that replaces the full equations, while at the same time also incorporating the uncertainty inherent in such models. By embedding this surrogate model into a Bayesian inference scheme, leaks are located by combining sensor observations with a model output approximating the true posterior distribution for possible leak locations. We show that our methodology enables producing fast, accurate, and trustworthy results. It showed a convincing performance on three problems with increasing complexity. For a simple test case, the Hanoi network, the average topological distance (ATD) between the predicted and true leak location ranged from 0.3 to 3 with a varying number of sensors and level of measurement noise. For two more complex test cases, the ATD ranged from 0.75 to 4 and from 1.5 to 10, respectively. Furthermore, accuracies upwards of 83%, 72%, and 42% were achieved for the three test cases, respectively. The computation times ranged from 0.1 to 13 s, depending on the size of the neural network employed. This work serves as an example of a digital twin for a sophisticated application of advanced mathematical and deep learning techniques in the area of leak detection.

Keywords: leak localization; water distribution network; Bayesian inverse problems; generative deep learning; digital twin

Citation: Mücke, N.T.; Pandey, P.; Jain, S.; Bohté, S.M.; Oosterlee, C.W. A Probabilistic Digital Twin for Leak Localization in Water Distribution Networks Using Generative Deep Learning. *Sensors* **2023**, *23*, 6179. https://doi.org/10.3390/s23136179

Academic Editors: Laura Belli, Luca Davoli, Marco Martalò and Gianluigi Ferrari

Received: 14 June 2023
Revised: 29 June 2023
Accepted: 30 June 2023
Published: 5 July 2023

1. Introduction

Water distribution systems make up a large and important part of our civil infrastructure. They need to be safe, efficient, and reliable. Ensuring a constant supply of clean water is, however, not an easy task. Distribution is typically carried out using networks of pipes, which can be highly intricate and involve multiple components, such as several kilometers of pipe segments and numerous junctions, valves, pumps, reservoirs, and tanks. Such networks are difficult to manage and they are prone to failure due to leakages and blockages, which may result in economic losses and environmental damage. Therefore, it is important to have a quick and trust-worthy monitoring method in place. Monitoring is typically done by recording information from a number of sensors installed at critical locations within the network. However, detecting the occurrence and location of leaks can still be a challenging task, even with sensor data being available, because the information captured by sensors will be incomplete in both time and space, making it difficult to pinpoint the exact location

and timing of the leak. In this paper, we address the problem of leak localization in real time by means of machine learning techniques.

For a leak localization framework to be considered useful in practice, it has to satisfy certain requirements. First, it should be sufficiently accurate. Second, it must be computationally fast, so that any leakage can be identified quickly. Third, it must be reliable. That is, the framework should not only provide an estimated leak location but also a measure of the level of confidence of the estimate. Fourth, it should be general and work under different circumstances. The framework should be sufficiently generic to work with different water distribution networks with widely varying complexities. Last, it must be flexible. In this context, flexibility refers to various aspects, such as whether it is possible to include prior knowledge when such becomes available, and whether it can handle different kinds of sensor readings or varying numbers of sensors, etc. There are not many approaches that satisfy all these requirements, since many of them are difficult to satisfy simultaneously. For example, computing uncertainty often comes at the cost of computation time, and generalized models are typically less accurate than models tailored to specific cases, and they struggle with incorporating prior knowledge, as they then lose their general nature. The framework presented here satisfies all these requirements, at the cost of a computationally intensive training stage.

1.1. Leak Localization Literature

There has already been a significant amount of research in the area of leak localization. However, not many approaches satisfy all the above-mentioned requirements. The authors in [1,2] made use of a model to generate synthetic sensor observations. The residuals between the model output and the observed values were then used to predict the leak location using a trained classifier. Similarly, in [3] residuals were employed for leak detection, after which the characteristics of the residuals were used to localize the leak. In [4], encoding of sensor observations was used, together with a trained classifier and a graph theory-based clustering method. While these approaches are computationally fast and were shown to be accurate for the selected test cases, they mainly work for the sensor configurations they were trained on and are limited regarding uncertainty quantification, as they do not model the inherent input uncertainty in, e.g., the demand. Furthermore, even if it is available, prior information cannot be embedded in these approaches. On the other hand, in [5], a combination of model- and graph theory-driven techniques was used, together with online training of a neural network classifier, to predict the cluster of nodes in which the leak was present. This is a flexible setup that allows for changes in sensor configurations. However, the uncertainty quantification is still limited, as only the size of the predicted cluster is used as a proxy for the reliability of the prediction.

1.2. Literature on Modern Machine Learning Techniques

In a rather different setting, the authors of [6] proposed a GAN-based automatic property generation approach, to generate verification properties for model checking. The verification properties, encoded in computational tree logic, were used as input to the GAN, whereas [7] presented a novel memory-augmented autoencoder approach for unsupervised anomaly detection in IoT data, which mitigated over-generalization by incorporating a memory mechanism in a time-series autoencoder (TSMAE). A methodology based on semi-supervised learning was introduced in [8], using an opposition-based novel updating spotted hyena optimization (ONU-SHO)-based recurrent neural network (RNN) for handling continuous or streaming data. The authors in [9] proposed a new record linkage (with the task of identifying and linking records from multiple sources) model for unstructured data, wherein a deep learning approach is used to improve the generalization of the Siamese multilayer preceptron model, to make it less sensitive to parameter selection.

1.3. Bayesian Inference

As an alternative to the above-mentioned approaches, one can make use of Bayesian inference. This allows one to accurately solve the leak localization problem using uncertainty quantification. The general technique is to compute the data likelihood and combine this with a prior. The output from a Bayesian inference approach is then a distribution of the leak location conditioned on sensor observations, i.e., the posterior distribution. This approach also provides flexibility, as new sensor configurations can be incorporated by simply modifying the likelihood. Both the sensor noise and model uncertainty are modeled within the likelihood. That is, the uncertainty associated with imperfect sensors and, e.g., stochastic nodal demand in the water network, can be included in the computation of the posterior directly. Furthermore, one can incorporate prior information in a straight-forward manner. However, the Bayesian approach has the significant drawback of being computationally expensive. Approximating the likelihood accurately requires solving the model many times, due to the nonlinearity and high-dimensionlity. Hence, it is often infeasible to run a Bayesian inference procedure in real time. In [10], the authors overcame this problem by assuming normally distributed demands and inputs, to be able to use gradient-based optimization together with kernel methods. However, this assumption may be restrictive. Instead, we propose an alternative solution to this problem that does not require such a restriction.

1.4. Computational Bottleneck

To overcome the computational bottleneck associated with Bayesian inference, one can make use of a surrogate model. A surrogate model is trained in an offline stage, before being used in an online stage for leak localization. The usage of surrogate modeling has become widespread nowadays, due to the potential for computational speed-up without an essential loss of accuracy. Conventionally, a linear dimensionality reduction, such as proper orthogonal decomposition (POD), is used together with a regression method in the reduced space; for example, in [11], a combination of POD and radial basis functions with neural networks was used in an inverse analysis for structural diagnosis. In [12], POD was used with stochastic spectral methods to relate the input to the output within a Bayesian inverse problem to speed up the process. In [13], POD was combined with Gaussian processes, to speed up nonlinear structural analysis. In recent years, deep neural networks have become a popular choice for surrogate models in scientific computing, due to their performance in dimensionality reduction and their predictive power [14–16]. For the purpose of speeding up Bayesian inference, we decided to model a high-dimensional stochastic problem. We made use of the concept of generative modeling. While applications of generative modeling in various scientific fields are already widespread [17–19], it has not yet been tailored to the area of water management. The general concept is to train a neural network to learn the underlying distribution of a data set, in order to be able to sample from it after a training phase. This enables fast sampling from complicated and highly dimensional distributions. There are several kinds of generative neural network, such as generative adversarial networks [20], variational autoencoders [21], and diffusion models [22]. Each of these models has its advantages and drawbacks. Specifically, there are three factors to consider when using generative models: sampling quality, speed, and diversity [23]. In this work, we adopt the Wasserstein autoencoder [24], as it performs highly satisfactory in these three criteria.

1.5. Overview of the Paper

In this paper, we present a novel leak localization framework based on Bayesian inference and generative deep learning. By formulating the leak localization problem as a Bayesian inverse problem, the uncertainty in model parameters and sensor observations is included in the leak location estimation, which enables accurate uncertainty quantification of the predicted leak location. Furthermore, prior information can be included in the computations. To overcome the computational drawbacks, we make use of neural networks

as the stochastic surrogate model. A neural network is trained as a generative model to estimate the distribution of pressure heads and flow rates given a certain leak location.This enables the fast evaluation of the likelihood function, while retaining a high accuracy.

In Section 2, we present the theory behind our framework. The problem setting and the underlying equations are described; the leak localization problem is presented as a Bayesian inverse problem; and the generative neural networks, WAEs, and neural network architectures are presented. In Section 3, we combine the components from Section 2 in the presented framework. In Section 4, we showcase the framework on three test cases. The paper is then concluded in Section 5.

2. Problem Setting and Preliminaries

In this section, we introduce the problem setting and briefly cover the preliminaries for the proposed framework. We start by introducing the mathematical model for the water distribution network, which allows us to simulate the pressure heads and flow rates given a set of parameters, such as the demand, pipe roughness, and leak location. Then, we describe leak localization as a Bayesian inverse problem and explain how the Bayesian setting allows us to model the uncertainty and incorporate prior information. Furthermore, we state the assumptions and the drawbacks of the proposed framework. Lastly, we present the relevant machine learning framework and deep learning architecture. Specifically, we discuss Wasserstein autoencoders, which are crucial for speeding up the Bayesian inference, as well as residual and transformer neural networks, which enable us to replace the conventional mathematical model, without losing significant accuracy.

2.1. Problem Setting

We consider a water distribution network that has N_p pipes, N_j variable head nodes, and N_f fixed head nodes. The head losses in all pipes in the network are assumed to be modeled by the Hazen–Williams formula, so the relation between the heads at two ends (nodes i and k) of a pipe j and the flow is given by

$$h_i - h_k = r_j Q_j^n, \tag{1}$$

where Q_j is the flow rate in pipe p_j; h_i is the head at node i, $n = 1.852$, and r_j is the pipe resistance factor, which depends on the length, diameter, and the material of the pipe. Define $\boldsymbol{q} = (Q_1, \ldots, Q_{N_p})^\top$ as a vector of unknown fluid flow rates in the pipes.

The network topology is modeled using the incidence matrices $A_1 \in \mathbb{R}^{N_p \times N_j}$ and $A_2 \in \mathbb{R}^{N_p \times N_f}$ for the unknown head nodes and the fixed head nodes, respectively. These incidence matrices are defined as

$$A_b = \begin{cases} -1 & \text{if the flow in pipe j enters the node i,} \\ 0 & \text{if the j does not connect to the node i,} \\ 1 & \text{if the flow in pipe j leaves the node i,} \end{cases} \tag{2}$$

where $b = 1, 2$. The unknown heads at different nodes are defined as $\boldsymbol{h} = (h_1, \ldots, h_{N_j})^\top$, the known nodal demands as $\boldsymbol{d}_m \in \mathbb{R}^{N_j}$, and $\boldsymbol{e}_l \in \mathbb{R}^{N_f}$ are the fixed head elevations. Additionally, we define the following matrices: O, as the $N_j \times N_j$ zero matrix, $\boldsymbol{o}$, as an $N_p \cdot N_j$ zero vector, and an $\mathbb{R}^{N_p \times N_p}$ diagonal matrix $G(\boldsymbol{q})$, with diagonal entries,

$$G_{jj}(\boldsymbol{q}) = r_j |Q_j|^{n-1}. \tag{3}$$

The hydraulic problem entails solving the unknown flow rates in the N_p pipes, $\boldsymbol{q}$, and the unknown heads at the N_j nodes, $\boldsymbol{h}$, given the network topology A_b, the demand at the nodes $\boldsymbol{d}_m$, and a fixed head elevation, $\boldsymbol{e}_l$, such that the mass and energy for the flow

are balanced. The continuity equation to be solved in matrix form is written as (see [25] for details):

$$f(\boldsymbol{x}) = \begin{pmatrix} G(\boldsymbol{q}) & -A_1 \\ -A_1^\top & \boldsymbol{O} \end{pmatrix} \begin{pmatrix} \boldsymbol{q} \\ \boldsymbol{h} \end{pmatrix} - \begin{pmatrix} A_2 \boldsymbol{e}_l \\ \boldsymbol{d}_m \end{pmatrix} = \boldsymbol{o}, \tag{4}$$

where we solve for the unknown vector $\boldsymbol{x} := (\boldsymbol{q}^T, \boldsymbol{h}^T)^T$. The above set of equations is typically solved using Rossman's popular program EPANET ([26]), to obtain a steady state solution.

A leak is modeled by adding a leak demand at a specific node. The leak demand is given by

$$d_{\text{leak}} = C_d A p^\alpha \sqrt{\frac{2}{\rho}}. \tag{5}$$

Typically, $\alpha = 0.5$ is chosen [26]; C_d is the dimension-less discharge coefficient, A [m^2] is the leak area, g [m/s^2] is acceleration by gravity, p [Pa] is the gauge pressure, and ρ [kg/m^3] is the density. Note that a leak is modeled as a nodal demand dependent on the pressure head. However, in the vast majority of cases, a leak will be located in a pipe section and not at a node. Therefore, we will introduce an extra node into the pipe section in which the leak occurs. The demand on that node is the leak demand.

2.2. Leak Localization as a Bayesian Inverse Problem

Detecting water leaks can be formulated as an inverse problem. Solving inverse problems is the process of computing the causal factors that give rise to a set of observations. These causal factors are typically either model parameters, the model itself, or the model output. There are, in general, two approaches for solving inverse problems: The variational approach, where a functional is minimized; and the Bayesian approach, where a posterior distribution is computed. We focus on the Bayesian approach, as we aim to resolve, in addition to a point estimate, the uncertainty associated with the estimate.

We use the notation $P_{\boldsymbol{x}}$ for the probability distribution of the stochastic variable, $\boldsymbol{x}$, and $p_{\boldsymbol{x}}$ for the associated density function. $P_{\boldsymbol{x}}(\boldsymbol{x}_i)$ then denotes the probability of a specific value $\boldsymbol{x}_i$.

The leak location is denoted by $c \in \{1, \ldots, N_p\}$, the sensor observations by $\boldsymbol{y} \in \mathbb{R}^{N_y}$, the state is $\boldsymbol{x} = (\boldsymbol{q}^T, \boldsymbol{h}^T)^T \in \mathbb{R}^{N_j+N_p}$, the uncertain parameters are $\boldsymbol{\omega} \in \mathbb{R}^{N_\omega}$, $\boldsymbol{\omega} \sim P_{\boldsymbol{\omega}}$, the forward model is given by $F : \mathbb{R} \times N_{\boldsymbol{\omega}} \to \mathbb{R}^{N_j+N_p}$, $F(c, \boldsymbol{\omega}) = \boldsymbol{x}$, the observation operator by $H : \mathbb{R}^{N_j+N_p} \to \mathbb{R}^{N_y}$, $H(\boldsymbol{x}) = \boldsymbol{y}$, and the observation noise is $\boldsymbol{\eta} \in \mathbb{R}^{N_y}$, $\boldsymbol{\eta} \sim P_{\boldsymbol{\eta}}$. These quantities are related in the following way:

$$\boldsymbol{y} = H(F(c, \boldsymbol{\omega})) + \boldsymbol{\eta}. \tag{6}$$

The forward model, F, is closely related to Equation (4). The output, $F(c, \boldsymbol{\omega}) = \boldsymbol{x}$, is the solution to Equation (4) for a given c and $\boldsymbol{\omega}$. The uncertain parameters, $\boldsymbol{\omega}$, depend on the problem at hand, and can, for example, be the demand at each node or the pipe roughness in each pipe section. One would typically have some knowledge of the distribution, $P_{\boldsymbol{\omega}}$, based on previous observations and calibration. The forward model is the function that maps the leak location and parameters to the solution of Equation (4). Hence, for larger WDNs, this can potentially be expensive to evaluate.

The distribution of interest is the posterior distribution of the leak location, i.e., the distribution over the leak location given the observations, $P_{c|\boldsymbol{y}}$. Since c is a discrete variable, we have to compute the posterior probability of all possible values of c, $P_{c|\boldsymbol{y}}(c_k|\boldsymbol{y})$, $c_k = 1, \ldots, N_p$. Using Bayes' theorem for density functions, we obtain:

$$p_{c|\boldsymbol{y}}(c_k|\boldsymbol{y}) = \frac{p_{\boldsymbol{y}|c}(\boldsymbol{y}|c_k) p_c(c_k)}{p_{\boldsymbol{y}}(\boldsymbol{y})}, \tag{7}$$

where $p_{\boldsymbol{y}|c}$ is referred to as the likelihood, p_c is the prior, and $p_{\boldsymbol{y}}$ the evidence. For leak detection, we often choose a uniform prior distribution. However, there are cases where prior information is known and incorporating it is highly beneficial. The evidence serves as a normalizing constant, which ensures that $p_{c|\boldsymbol{y}}$ sums to one over all possible values of c_k. This is given by

$$p_{\boldsymbol{y}}(\boldsymbol{y}) = \sum_{k=1}^{N_p} p_{\boldsymbol{y}|\boldsymbol{c}}(\boldsymbol{y}|c_k). \tag{8}$$

The likelihood is not directly available; however, from Equation (6) we obtain:

$$p_{\boldsymbol{y}|c,\boldsymbol{\omega}}(\boldsymbol{y}|c_k,\boldsymbol{\omega}) = p_{\boldsymbol{\eta}}(\boldsymbol{y} - H(F(c_k,\boldsymbol{\omega}))), \tag{9}$$

which implies that we can compute the likelihood by marginalizing over $\boldsymbol{\omega}$:

$$\begin{aligned} p_{\boldsymbol{y}|c}(\boldsymbol{y}|c_k) &= \int_{-\infty}^{-\infty} p_{\boldsymbol{y}|c,\boldsymbol{\omega}}(\boldsymbol{y}|c_k,\boldsymbol{\omega}) p_{\boldsymbol{\omega}|c}(\boldsymbol{\omega}|c_k) \mathrm{d}\boldsymbol{\omega} \\ &= \int_{-\infty}^{-\infty} p_{\boldsymbol{y}|c,\boldsymbol{\omega}}(\boldsymbol{y}|c_k,\boldsymbol{\omega}) p_{\boldsymbol{\omega}}(\boldsymbol{\omega}) \mathrm{d}\boldsymbol{\omega} \\ &= \mathbb{E}_{\boldsymbol{\omega}\sim P_\omega}\left[p_{\boldsymbol{y}|c,\boldsymbol{\omega}}(\boldsymbol{y}|c_k,\boldsymbol{\omega})\right]. \end{aligned} \tag{10}$$

We assume that $\boldsymbol{\omega}$ is independent of c_k.

As observations will arrive with time, we need to update the posterior when new observations become available, using the posterior from the previous observation time as the prior. We denote observations at time t_i by $\boldsymbol{y}_i$, which gives us the following posterior:

$$\begin{aligned} p_{c|\boldsymbol{y}}(c_k|\boldsymbol{y}_i) &= \frac{p_{\boldsymbol{y}|c}(\boldsymbol{y}_i|c_k) p_{c|\boldsymbol{y}}(c_k|\boldsymbol{y}_{i-1})}{p_{\boldsymbol{y}}(\boldsymbol{y}_i)} \\ &= \frac{\mathbb{E}_{\boldsymbol{\omega}\sim P_\omega}\left[p_{\boldsymbol{y}|c,\boldsymbol{\omega}}(\boldsymbol{y}_i|c_k,\boldsymbol{\omega})\right] p_{c|\boldsymbol{y}}(c_k|\boldsymbol{y}_{i-1})}{\sum_{j=1}^{N_p} \mathbb{E}_{\boldsymbol{\omega}\sim P_\omega}\left[p_{\boldsymbol{y}|c,\boldsymbol{\omega}}(\boldsymbol{y}_i|c_j,\boldsymbol{\omega})\right]}. \end{aligned} \tag{11}$$

The posterior distribution after observations at times $t_0, t_1, \ldots, t_{N_t}$ is given by

$$p_{c|\boldsymbol{y}}(c_k|\boldsymbol{y}_{0:N_t}) = p_c(c_k) \prod_{i=0}^{N_t} \frac{\mathbb{E}_{\boldsymbol{\omega}\sim P_\omega}\left[p_{\boldsymbol{y}|c,\boldsymbol{\omega}}(\boldsymbol{y}_i|c_k,\boldsymbol{\omega})\right]}{\sum_{j=1}^{N_p} \mathbb{E}_{\boldsymbol{\omega}\sim P_\omega}\left[p_{\boldsymbol{y}|c,\boldsymbol{\omega}}(\boldsymbol{y}_i|c_j,\boldsymbol{\omega})\right]}. \tag{12}$$

We can write down the expression $p_{c|\boldsymbol{y}}(c_k|\boldsymbol{y}_{0:N_t})$ for all k, but it is not feasible to analytically solve it. Therefore, we need to make use of numerical approximations, such as Monte Carlo approaches. However, there are several computational challenges associated with this:

- $P_{\boldsymbol{\omega}}$ is not necessarily known or it could be difficult to sample from;
- $\boldsymbol{\omega}$ is, in general, high-dimensional. For example, when $\boldsymbol{\omega}$ represents the stochastic demand in each node, then $\boldsymbol{\omega}$ is N_j-dimensional. This makes the integral to be computed in Equation (10) high-dimensional, and it is thereby not feasible to compute the likelihood, $\mathbb{E}_{\boldsymbol{\omega}\sim P_\omega}\left[p_{c|\boldsymbol{y},\boldsymbol{\omega}}(\boldsymbol{y}_i|c_k,\boldsymbol{\omega})\right]$, for all k in real time;
- $p_{c|\boldsymbol{y},\boldsymbol{\omega}}(\boldsymbol{y}_i|\boldsymbol{c}_k,\boldsymbol{\omega})$ can be expensive to evaluate as it requires solving Equation (4).

While several methods exist that address the computation of stochastic integrals, most of them have undesirable issues. For example, Gaussian processes (GPs) can be trained to compute the posterior distribution directly. However, with GPs, one is restricted to modeling multivariate Gaussian distributions. Furthermore, it is well known that kernel methods are, in general, not suitable for very high-dimensional cases [27]. An alternative to GPs is the polynomial chaos expansion (PCE), which allows for more complicated

distributions than GPs. However, with PCE, one is typically even more restricted in dimensionality, as these methods suffer from the curse of dimensionality [28]. We will make use of deep learning, as it allows us to deal with arbitrary distributions and high-dimensional problems.

2.3. Supervised Wasserstein Autoencoder

The methodology chosen here to address the above-mentioned challenges is generative modeling. Particularly, we will focus our discussion on the use of the autoencoder set-up, whose details are explained in this subsection. A supervised Wasserstein autoencoder (SupWAE) represents a type of neural network that simultaneously achieves a problem dimensionality reduction and an approximation of the relevant distribution [24]. Before explaining the SupWAE, we briefly introduce the regular autoencoder (AE) and add the necessary components for leak detection.

2.3.1. Autoencoders

A regular autoencoder (AE) is often used to identify an accurate low-dimensional representation of data [29]. This low-dimensional representation is then referred to as the latent state, which is an element of the latent space, while the original data are referred to as the high-fidelity state, belonging to the high-fidelity space.

An AE consists of two neural networks: An encoder, ϕ_{enc}, that sparsifies (i.e., reduces the dimensionality of) the data, and a decoder, ϕ_{dec}, that reconstructs the data:

$$\phi_{\text{enc}}(\boldsymbol{x}) = \boldsymbol{z}, \quad \phi_{\text{dec}}(\boldsymbol{z}) = \tilde{\boldsymbol{x}}, \quad \phi_{\text{dec}}(\phi_{\text{enc}}(\boldsymbol{x})) = \tilde{\boldsymbol{x}} \approx \boldsymbol{x}. \tag{13}$$

Considering a training set, $\{\boldsymbol{x}_i\}_{i=1}^N \sim P_{\boldsymbol{x}}(\boldsymbol{x})$, AEs are trained by minimizing the mean squared error (MSE) with a weight regularization:

$$L_{\text{AE}}(\phi_{\text{enc}}, \phi_{\text{dec}}) = \frac{1}{N} \sum_{zi=1}^{N} (\boldsymbol{x}_i - \phi_{\text{dec}}(\phi_{\text{enc}}(\boldsymbol{x}_i)))^2 + \alpha R(\phi_{\text{enc}}, \phi_{\text{dec}}) \tag{14}$$

L_{AE} is minimized with respect to the weights of ϕ_{enc} and ϕ_{dec}, typically using stochastic gradient-descent-type algorithms. The term $R(\phi_{\text{enc}}, \phi_{\text{dec}})$ is chosen to be the l^2 norm of the weights. This is referred to as the weight decay within machine learning; α determines how much we regularize the weights. The purpose of this regularization is to avoid overfitting to the training data.

2.3.2. Wasserstein Autoencoders

While AEs provide a framework for computing latent representations of data, they lack certain properties that are of interest when computations in the latent space are necessary. Specifically, small perturbations in the latent space should also result in small perturbations in the high-fidelity space. Moreover, it should be possible to sample from the latent space. By using Wasserstein AEs (WAEs) [24], we can also obtain these properties.

We introduce a prior distribution to the latent space, $P_z(\boldsymbol{z})$. While the encoder and decoder remain deterministic mappings, they define the conditional distributions $P_{\text{enc}}(\boldsymbol{z}|\boldsymbol{x})$ and $P_{\text{dec}}(\boldsymbol{x}|\boldsymbol{z})$, respectively, using the densities:

$$p_z(\boldsymbol{z}) = \int_{\boldsymbol{x}} p(\boldsymbol{z}|\boldsymbol{x}) p_{\boldsymbol{x}}(\boldsymbol{x}) \mathrm{d}\boldsymbol{x} \approx \int_{\boldsymbol{x}} p_{\text{enc}}(\boldsymbol{z}|\boldsymbol{x}) p_{\boldsymbol{x}}(\boldsymbol{x}) \mathrm{d}\boldsymbol{x} = p_{\text{enc}}(\boldsymbol{z}), \tag{15}$$

$$p_{\boldsymbol{x}}(\boldsymbol{x}) = \int_{\boldsymbol{z}} p(\boldsymbol{x}|\boldsymbol{z}) p_z(\boldsymbol{z}) \mathrm{d}\boldsymbol{z} \approx \int_{\boldsymbol{z}} p_{\text{dec}}(\boldsymbol{x}|\boldsymbol{z}) p_z(\boldsymbol{z}) \mathrm{d}\boldsymbol{z} = p_{\text{dec}}(\boldsymbol{x}). \tag{16}$$

Here, a sample from $P_{\text{enc}}(\boldsymbol{z}|\boldsymbol{x})$ is computed using the encoder, $\boldsymbol{z} = \phi_{\text{enc}}(\boldsymbol{x})$, and similarly a sample from $P_{\text{dec}}(\boldsymbol{x}|\boldsymbol{z})$ is obtained using the decoder, $\boldsymbol{x} = \phi_{\text{dec}}(\boldsymbol{z})$. The goal is to ensure that the encoder approximates the latent prior distribution, $P_z(\boldsymbol{z}) \approx P_{\text{enc}}(\boldsymbol{z})$, and the decoder

approximates the high-fidelity prior distribution, $P_{\boldsymbol{x}}(\boldsymbol{x}) \approx P_{\text{dec}}(\boldsymbol{x})$. This is achieved by simultaneously minimizing the reconstruction error and a divergence, D, between $P_{\boldsymbol{z}}(\boldsymbol{z})$ and $P_{\text{enc}}(\boldsymbol{z})$, which measures the similarity of the two distributions:

$$L_{\text{WAE}}(\phi_{\text{enc}}, \phi_{\text{dec}}) = \underbrace{\frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{x}_i - \phi_{\text{dec}}(\phi_{\text{enc}}(\boldsymbol{x}_i)))^2}_{\text{reconstruction}} + \lambda \underbrace{D(P_{\boldsymbol{z}}(\boldsymbol{z}), P_{\text{enc}}(\boldsymbol{z}))}_{\text{divergence}} + \alpha \underbrace{R(\phi_{\text{enc}}, \phi_{\text{dec}})}_{\text{regularization}}. \tag{17}$$

Here, λ is another regularization parameter to be determined through hyperparameter tuning. In this paper, we choose the maximum mean discrepancy (MMD) with a multi-quadratics kernel for the divergence, which gives us the WAE-MMD. In [30], it was argued that this is an accurate choice when $P_{\boldsymbol{z}}(\boldsymbol{z})$ is the normal distribution, see [24]. In summary, we obtain:

$$\begin{aligned} L_{\text{WAE}}(\phi_{\text{enc}}, \phi_{\text{dec}}) &= \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{x}_i - \phi_{\text{dec}}(\phi_{\text{enc}}(\boldsymbol{x}_i)))^2 + \lambda \text{MMD}\Big(\phi_{\text{enc}}, \{\boldsymbol{z}_i\}_{i=1}^{N}\Big) \\ &+ \alpha R(\phi_{\text{enc}}, \phi_{\text{dec}}), \end{aligned}$$

where $\boldsymbol{z}_i \sim P_{\boldsymbol{z}}(\boldsymbol{z})$,

$$\begin{aligned} \text{MMD}\Big(\phi_{\text{enc}}, \{\boldsymbol{z}_i\}_{i=1}^{N}\Big) &= \frac{\lambda}{N(N-1)}\sum_{l\neq j}^{N}\big[k(\boldsymbol{z}_l, \boldsymbol{z}_j) + k(\phi_{\text{enc}}(\boldsymbol{x}_l), \phi_{\text{enc}}(\boldsymbol{x}_j))\big] \\ &+ \frac{2\lambda}{N^2}\sum_{l,j}^{N}k(\boldsymbol{z}_l, \phi_{\text{enc}}(\boldsymbol{x}_j), \end{aligned}$$

and

$$k(\boldsymbol{z}_l, \boldsymbol{z}_j) = \frac{C}{C + ||\boldsymbol{z}_l - \boldsymbol{z}_j||_2^2}. \tag{18}$$

After training, it is possible to sample from the chosen latent prior distribution, $p_{\boldsymbol{z}}(\boldsymbol{z})$, pass it through the decoder, ϕ_{dec}, and obtain a high-fidelity sample, $\phi_{\text{dec}}(\boldsymbol{z}) = \boldsymbol{x}$.

To obtain the supervised version of the WAE, we introduce c as an extra input to the decoder, so $\phi_{\text{enc}}(\cdot) := \phi_{\text{enc}}(\boldsymbol{z}, c)$. In this way, the decoder models the conditional probability density function, $p_{\boldsymbol{x}|c}(\boldsymbol{x}|c)$,

$$\begin{aligned} p_{\boldsymbol{x}|c}(\boldsymbol{x}|c) &= \int_{\boldsymbol{z}} p_{\boldsymbol{x}|c,\boldsymbol{z}}(\boldsymbol{x}|c, \boldsymbol{z}) p_{\boldsymbol{z}}(\boldsymbol{z}) \mathrm{d}\boldsymbol{z} \\ &\approx \int_{\boldsymbol{z}} p_{\text{dec}}(\boldsymbol{x}|c, \boldsymbol{z}) p_{\boldsymbol{z}}(\boldsymbol{z}) \mathrm{d}\boldsymbol{z} = \int_{\boldsymbol{z}} \phi_{\text{dec}}(\boldsymbol{z}, c) p_{\boldsymbol{z}}(\boldsymbol{z}) \mathrm{d}\boldsymbol{z} = p_{\text{dec}}(\boldsymbol{x}|c), \end{aligned} \tag{19}$$

so that c is considered a known condition. During training, the WAE sees pairs $(\boldsymbol{x}, c)$ and uses this information to learn the conditional distribution. The condition c determines the class, while the latent variable, $\boldsymbol{z}$, determines the style. This separation will be crucial for the proposed framework. A visualization of the supervised WAE-MMD is shown in Figure 1.

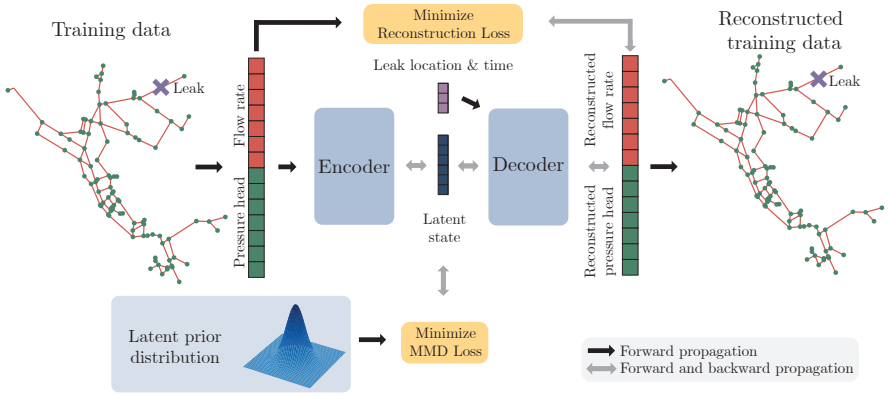


Figure 1. Illustration of a Wasserstein AE for reconstruction of a water distribution network.

2.4. Neural Network Architectures

Different kinds of neural network architecture can be incorporated into the WAEs framework. Therefore, one should choose the architecture that performs optimally for the data type. In this paper, we showcase the performance using two different kinds of architecture—residual neural networks (ResNet) [31] and transformers.

2.4.1. Residual Neural Networks

A residual neural network is a neural network that consists of residual layers. A residual layer is defined as having a skip connection bypassing the normal layer. That is, a residual layer is given by

$$\boldsymbol{x}^{i+1} = \mathcal{F}^i(\boldsymbol{x}^i) + \boldsymbol{x}^i, \tag{20}$$

where $\boldsymbol{x}^i$ is the output of layer i and $\mathcal{F}^i$ is the ith layer. $\mathcal{F}^i$ typically consists of a linear layer, followed by an activation function, and then another linear layer. The linear layers can be either dense or convolutional.

The advantage of using ResNets is that the problem of vanishing gradients is not very apparent, which makes them easier to train [31].

2.4.2. Transformers

Transformers are another type of neural network architecture, introduced in 2017 [32], that became the default choice for natural language processing. Since then, transformers have outperformed the state-of-the-art methodologies in areas such as image recognition [33], protein structure prediction [34], and time series forecasting [35]. While these applications seem very different from leak detection in water distribution networks, they share some features that the transformer architecture addresses well.

Transformers treat the data as if it were a fully connected graph. The multi-head attention models how information from one node or pipe section should be aggregated to another node or pipe section. In this way, the neural networks learn which connections should be strengthened and which should be weakened. Hence, the transformer can efficiently learn long-range relations between nodes and pipe sections. This is in sharp contrast to graph neural networks, where many layers are necessary to capture such long-range relations.

For a more technical presentation of the transformer architecture and the attention mechanism, we refer to Appendix A.

3. Proposed Framework

The aim of the proposed framework is to overcome the challenges related to solving Bayesian inverse problems, as described in Section 2.2, while maintaining a high accuracy

when computing the posterior over the parameters of interest. The framework employed here is an extension of the one presented in [17].

We will use a generative neural network as a stochastic digital twin of the WDN. This can be used to sample pressure heads and flow rates of the entire WDN for a given leak location and time. By modeling the pressure heads and flow rates as distributions conditioned on leak location and time, instead of a deterministic output, the uncertainty due to the stochastic parameters is included in the model output. A generative neural network is a suitable choice for this, as this enables us to sample from the distribution of pressure heads and flow rates.

We will specifically make use of the generative properties of the decoder of the supervised WAE-MMD, whereas the encoder is discarded after training. The decoder is trained to approximate the state, when given the random noise, leak location, and the time of day.It then replaces the forward model, and the latent vector, $\boldsymbol{z}$, replaces the uncertain parameters; in our case, the demand. Using the same approach as in Section 2.2, we can rewrite the posterior for given observations, as follows:

$$p_{c|\boldsymbol{y}}(c_k|\boldsymbol{y}_i) = \frac{p_c(c_k)\int_z p_{\boldsymbol{y}|c,z}(\boldsymbol{y}_i|c_k,\boldsymbol{z})p_z(\boldsymbol{z})\,\mathrm{d}\boldsymbol{z}}{\sum_{j=1}^{N_p}\int_z p_{\boldsymbol{y}|c,z}(\boldsymbol{y}_i|c_j,\boldsymbol{z})p_z(\boldsymbol{z})\,\mathrm{d}\boldsymbol{z}} = \frac{p_c(c_k)\mathbb{E}_{\boldsymbol{z}\sim P_z}\left[p_{\boldsymbol{y}|c,z}(\boldsymbol{y}_i|c_k,\boldsymbol{z})\right]}{\sum_{j=1}^{N_p}\mathbb{E}_{\boldsymbol{z}\sim P_z}\left[p_{\boldsymbol{y}|c,z}(\boldsymbol{y}_i|c_j,\boldsymbol{z})\right]}, \tag{21}$$

where the likelihood is computed by

$$p_{\boldsymbol{y}|c,z}(\boldsymbol{y}_i|c_k,\boldsymbol{z}) = p_{\boldsymbol{\eta}}(\boldsymbol{y}_i - H(\phi_{\mathrm{dec}}(\boldsymbol{z},c_k))). \tag{22}$$

As in Equations (11) and (12), we can use the posterior from time t_{i-1} as the prior for the posterior at time t_i. This gives us the resulting posterior for a series of N_t observations:

$$p_{c|\boldsymbol{y}}(c_k|\boldsymbol{y}_{0:N_t}) = p_c(c_k)\prod_{i=0}^{N_t}\frac{\mathbb{E}_{\boldsymbol{z}\sim P_z}\left[p_{\boldsymbol{y}|c,z}(\boldsymbol{y}_i|c_k,\boldsymbol{z})\right]}{\sum_{j=1}^{N_p}\mathbb{E}_{\boldsymbol{z}\sim P_z}\left[p_{\boldsymbol{y}|c,z}(\boldsymbol{y}_i|c_j,\boldsymbol{z})\right]}. \tag{23}$$

At time t_0, we simply use a uniform prior.

As an addition to the described setup, we add the timestamp as an additional input to the decoder. This gives us the following description:

$$\phi_{\mathrm{dec}}(\boldsymbol{z},c,t_i) = \boldsymbol{x}_i(c,\boldsymbol{\omega}) = (\boldsymbol{q}_i^T(c,\boldsymbol{\omega}),\boldsymbol{h}_i^T(c,\boldsymbol{\omega}))^T. \tag{24}$$

With this formulation the decoder disentangles the temporal information from the rest. Hence, the time dependency of the stochastic demand is explicitly modeled in the decoder.

The proposed framework essentially resolves the challenges described in Section 2.2:

- $P_{\boldsymbol{\omega}}$ is replaced by the latent prior, P_z, which is known, as we used it during training of the WAE-MMD;
- $\boldsymbol{z}$ is of a much lower dimension than $\boldsymbol{\omega}$, which makes the evaluation of the integral in Equation (10) fast and thereby enables real-time computation of the likelihood, $\mathbb{E}_{\boldsymbol{z}\sim P_z}\left[p_{\boldsymbol{y}|c,z}(\boldsymbol{y}_i|c_k,\boldsymbol{z})\right]$;
- The likelihood, $p_{\boldsymbol{y}|c_k,z}$, is computed using a forward propagation of the decoder, instead of solving an expensive forward model.

While the costs are drastically reduced in the leak detection stage, the training stage is now (potentially) expensive to compute. These two stages are referred to as the online stage, in which the trained supervised WAE-MMD is used to solve the leak localization problem for given observations, and the offline stage, in which we generate training data and train the supervised WAE-MMD. These two stages are outlined in Algorithms 1 and 2, respectively. Furthermore, the online stage is visualized in Figure 2.

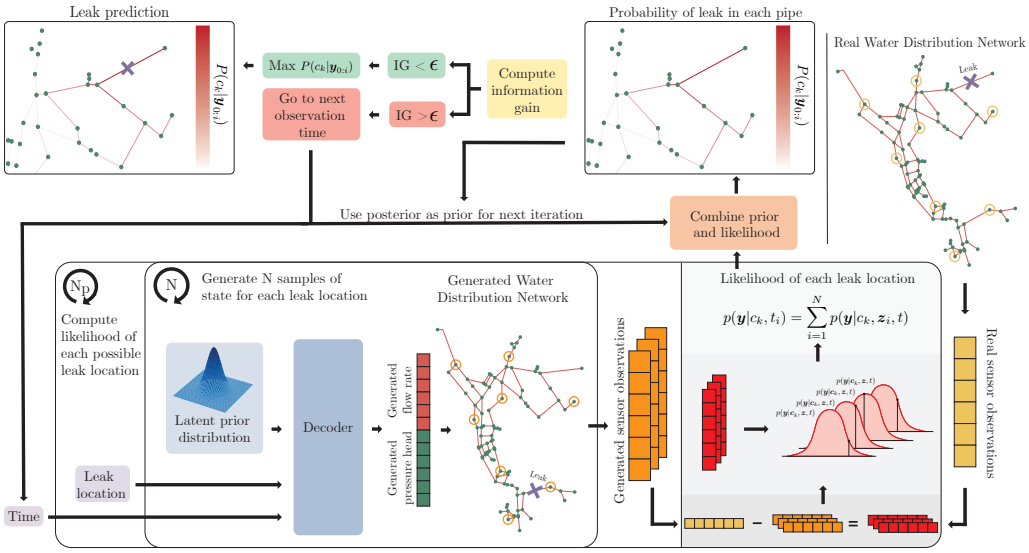


Figure 2. Illustration of the online computation of the posterior distribution, $p(c_k|\mathbf{y}_{0:i})$, $k = 1, \ldots, N_p$.

Algorithm 1: Offline stage

Input : N_{train}, training hyperparameters, WAE-MMD architecture

Generate training samples, $\{(\mathbf{x}_i, c_i)\}_{i=1}^{N_{\text{train}}}$, by solving the forward problem (see Section 2.1);

Train the supervised WAE-MMD (see Section 2.3);

Output: ϕ_{dec}

Algorithm 2: Online stage

Input : ϕ_{dec} from Algorithm 1, observations $\mathbf{y}$, threshold ϵ

$i = 0$

for $c_k = \{1, \ldots, N_p\}$ **do**

  Compute $p_{c|\mathbf{y}}(c_k|\mathbf{y}_i) = \frac{p_c(c_k)\mathbb{E}_{\mathbf{z}\sim P_{\mathbf{z}}}\left[p_{\mathbf{y}|c,\mathbf{z}}(\mathbf{y}_i|c_k,\mathbf{z})\right]}{\sum_{j=1}^{N_p}\mathbb{E}_{\mathbf{z}\sim P_{\mathbf{z}}}\left[p_{\mathbf{y}|c,\mathbf{z}}(\mathbf{y}_i|c_j,\mathbf{z})\right]}$

end for

while $IG(c, \mathbf{y}_i) > \epsilon$ **do**

  $i = i + 1$

  for $c_k = \{1, \ldots, N_p\}$ **do**

    $p_{c|\mathbf{y}}(c_k|\mathbf{y}_{0:i}) = p_c(c_k)\prod_{i=0}^{i}\frac{\mathbb{E}_{\mathbf{z}\sim P_{\mathbf{z}}}\left[p_{\mathbf{y}|c,\mathbf{z}}(\mathbf{y}_i|c_k,\mathbf{z})\right]}{\sum_{j=1}^{N_p}\mathbb{E}_{\mathbf{z}\sim P_{\mathbf{z}}}\left[p_{\mathbf{y}|c,\mathbf{z}}(\mathbf{y}_i|c_j,\mathbf{z})\right]}$

  end for

end while

Output: $p_{c|\mathbf{y}}(c|\mathbf{y}_{0:i})$

3.1. Stopping Criterion in the Online Stage

With every new set of observations, $\mathbf{y}_i$, we receive additional information about the system at hand. However, at some point, new observations no longer contribute to the accuracy of the posterior. In other words, the posterior density should converge:

$$p_{c|\mathbf{y}}(c_k|\mathbf{y}_{0:N_t}) \to p_{c|\mathbf{y}}(c_k|\mathbf{y}), \quad \text{for} \quad N_t \to \infty. \tag{25}$$

This implies that the algorithm should be stopped when there is no significant change to the posterior. For this reason, we introduce the information gain of c, $IG(c, \mathbf{y}_i)$, obtained from additional observations, as the stopping criterion. The information gain is defined

by the KL divergence between the posterior at time t_i and time t_{i-1}, which measures how much the posterior distribution changed with new observations:

$$\begin{aligned} IG(c, \boldsymbol{y}_i) &= D_{\mathrm{KL}}(P_{c|\boldsymbol{y}}(c_k|\boldsymbol{y}_{0:i})||P_{c|\boldsymbol{y}}(c_k|\boldsymbol{y}_{0:i-1})) \\ &= -\sum_{k=1}^{N_p} p_{c|\boldsymbol{y}}(c_k|\boldsymbol{y}_{0:i}) \log\left(\frac{p_{c|\boldsymbol{y}}(c_k|\boldsymbol{y}_{0:i})}{p_{c|\boldsymbol{y}}(c_k|\boldsymbol{y}_{0:i-1})}\right). \end{aligned} \tag{26}$$

We terminate the computations when the information gain is below a threshold, ϵ.

3.2. Estimating Uncertainty

In order to decide whether a leak location prediction is trustworthy, we can compute the entropy of the posterior:

$$H(C) = -\sum_{i=1}^{N_p} p_c(c_i|\boldsymbol{y}_{0:1}) \log(p_c(c_i|\boldsymbol{y}_{0:1})). \tag{27}$$

This tells us how much information or uncertainty we have in the posterior distribution. A low entropy means a sufficient amount of information, i.e., low uncertainty, and a high entropy signifies the opposite. Therefore, we can use the entropy as a measure of how much we can trust our prediction. That is, when there are insufficient observations to make a trustworthy prediction, the entropy will be higher and thereby indicate that more observations are needed. Hence, when the truth is unknown, we can still assess whether the prediction is accurate or not.

This is a crucial step in applying the methodology in practice, as it provides the necessary information to act on a certain prediction. If the entropy is high, this tells us that we need more observations in order to provide a trustworthy prediction.

3.3. Model Architectures

As mentioned in Section 2, the WAE and the proposed framework do not rely on only one neural network architecture. In this work, we make use of two different types of neural network architecture, to demonstrate that the framework can be based on more than one possible choice. Moreover, we will see that each choice has a superior performance for a certain test case. Specifically, we use transformers and dense ResNets in the experiments that follow. For a detailed breakdown of the model architectures, see Appendix B, and for a visualization of the transformer network, see Figure A2. For the transformers, both the encoder and the decoder make use of a combination of dense neural networks, transformer encoders, and transformer decoders. With this architecture, the network structure of the data is modeled using attention mechanisms. For the dense ResNet, the encoder and decoder consist of a series of residual layers with dimension reduction and increasing layers, respectively, in between.

4. Results

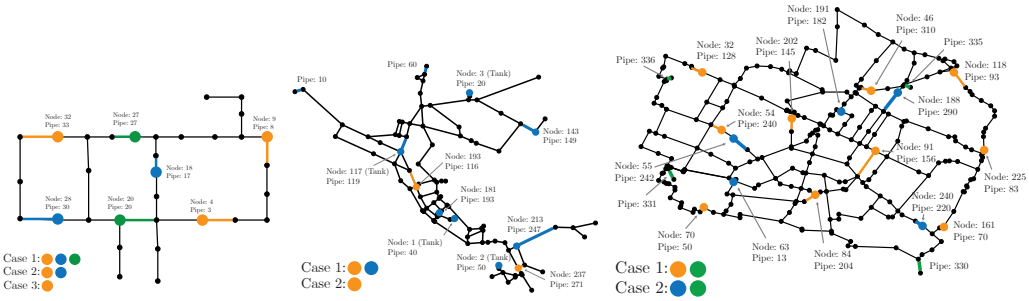
In this section, we show the performance of the proposed framework in three test cases. We assessed the framework's performance with respect to the topological distance and accuracy (The code used for generating the data and the corresponding results can be found on GitHub, see https://github.com/nmucke/DT-for-WDN-leak-localization.git (accessed on 13 June 2023)). We compared the performance of two distinct architectures, a dense ResNet and a transformer architecture. In both cases, the neural networks were similar to the encoder network, but with a softmax activation function at the output.

For comparison, we also computed the leak location using a conventional classification neural network, which was trained to classify a leak location based on sensor observations at a given time. The classification network was trained on the same data as the autoencoders. It is worth noting that, with this method, it is necessary to train a model for each sensor configuration. This is in contrast to the proposed framework, where a single model can

handle all possible sensor configurations. For more details on the classification model, see Appendix D.

4.1. Test Cases

All test cases were defined in a similar manner, however, with some variations. Figure 3 shows the three test WDN topologies, together with the sensor locations. The first one is typically referred to as a Hanoi network; the second is often referred to as Net3 (not to be confused with the numbering of the cases in this paper), and the third is known as Modena.



(**a**) Test WDN1. (**b**) Test WDN2. (**c**) Test WDN3.

Figure 3. Network topologies and the sensor locations for the three test cases.

We placed sensors in various nodes in the WDN. Each sensor measured the pressure head value in the node and the flow rate in a neighboring pipe section. We did not use pressure head sensors at the water sources, as such information does not make sense in many cases, e.g., for lakes and rivers. However, we placed flow rate sensors in pipes connected to those sources, in order to mimic a real-world scenario, where the inflow into the network is measured. For test case 1, we show results for three configurations, each with a varying number of sensors. For test cases 2 and 3, we show results for two sensor configurations. Note that, since the generative model outputs the full state, consisting of pressure heads and flow rate, the different sensor configurations only changed the observation operator in the online stage; that is, only H in (22) was varied with the different sensor configurations.Hence, a single training stage was performed, and the resulting generative model worked under multiple sensors settings.

We also added noise to the sensor readings, to mimic a real-world scenario where the sensors would not be perfect. In all cases and for all pressure head and flow rate observations, the noise was sampled from a normal distribution at each time step and added to the observations. The noise at each sensor was independent of the other sensors. The normal distribution had a mean of zero and a standard deviation corresponding to a percentage of the observed value. We show results for various noise percentages in all test cases, to analyze the performance of the algorithm in various settings. In Table 1, we show the specific parameter and noise settings for the three test cases.

Table 1. Settings for the three test cases.

	WDN 1	**WDN 2**	**WDN 3**
Num. pipes	34	119	317
Num. junctions	31	97	272
Demand noise	10%	5%	10%

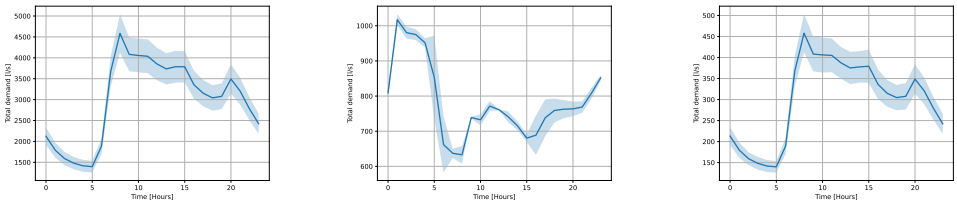
In all test cases, the data were generated by varying the pipe section in which the leak was present. The leak size was also varied by varying the leak area between 0.002 [m^2] (20 [cm^2]) and 0.004 [m^2] (40 [cm^2]). The test cases were simulated for 24 h, with values recorded every hour.

All the demands followed a temporal pattern, i.e., in each node at every time of the day, the demand had a base value. In order to mimic the stochasticity of the demand, we

added noise to the base values. The total demands in each network are shown in Figure 4 and examples of demand patterns for individual nodes are shown in Figure 5.

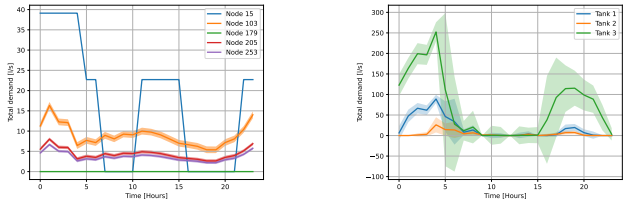
We used two metrics to assess the performance of the framework: the average topological distance (ATD), and the accuracy. The accuracy is the fraction of correctly predicted leak locations. The topological distance is the distance of the shortest route from the predicted leak location to the true leak location. The ATD was then computed by taking the average of the many different solutions of the inverse problems for varying leak locations, sizes, demands, and sensor noises.

Furthermore, it is important to emphasize that the models were trained on a dataset that was distinct from the test dataset used in assessing the performance. The Epanet .inp files can be found in the GitHub repository.



(**a**) Test WDN 1. (**b**) Test WDN 2. (**c**) Test WDN 3.

Figure 4. Total demand with standard deviation.



(**a**) Demands for 5 nodes. (**b**) Demands for tanks.

Figure 5. Demand patterns for the test WDN 2.

4.2. Training of the WAEs

The WAEs were trained on simulated data. Training data were generated by simulating according to the settings described above. The leak locations were sampled from a multinomial distribution, with an equal probability assigned to each pipe section. The leak size was sampled uniformly from the interval $[0.002\ \mathrm{m}^2, 0.004\ \mathrm{m}^2]$. Furthermore, the demand noise was sampled at each time instance from a normal distribution with a mean 0 and a variance equal to 10% of the base value in test cases 1 and 3 and 5% in test case 2.

The hyperparameter settings for the WAEs are described in Appendix C.

4.3. Test WDN1: Hanoi Network

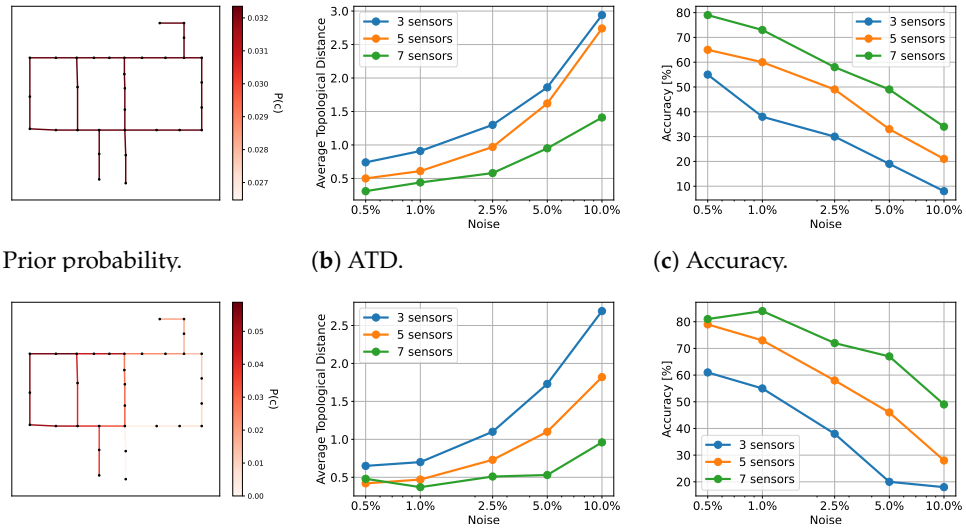
For test WDN1, we considered three different sensor configurations (see Figure 3a), each with five different sensor noise levels. All nodes are given the same demand pattern, but with varying base values in each node. Noise was added to the total demand, which was then distributed to all nodes according to the relative base value. Noise was added to each node independently of the other nodes. See Figure 4a for the total demand time series with the standard deviation.

We tested the framework with both a prior distribution of the leak locations and without any prior knowledge. The prior distribution is shown in Figure 6d.

We made use of a transformer architecture here. The results are shown in Figure 6. The likelihood was computed with 30,000 samples. We tested using 50 different leak locations.

As expected, the ATD increased and the accuracy decreased with increasing noise and a decreasing number of sensors. The ATD went from approximately 0.25 to approximately 3.0 from the best to the worst case, and the accuracy diminished from approximately 79% to

approximately 9% when no prior was available. When a prior was present, the ATD ranged from approximately 0.4 to approximately 2.7, and the accuracy ranged from approximately 81% to approximately 19%, which was a drastic reduction.



(**a**) Prior probability. (**b**) ATD. (**c**) Accuracy.

(**d**) Prior probability. (**e**) ATD. (**f**) Accuracy.

Figure 6. Results for WDN1 for the given priors. The first row shows results for the prior in (**a**) and the second row shows results for the prior in (**d**). The number of sensors counted is the number of flow rate sensors. Note that there are fewer pressure head sensors.

4.4. Test WDN2

For test WDN2, we considered two different sensor configurations (see Figure 3b), each with five different sensor noise levels. The total demand is shown Figure 4b. There was a varying demand pattern for each node. Some nodes mimicked households, other nodes had no demand, and some had demands mimicking factories. Noise was added to each node independently of the other nodes. In Figure 5a, examples of the demand patterns are presented. Furthermore, there were water tanks present, which served as stabilizers for the WDN. Hence, their demand varied according to the total demand. In Figure 5b, examples of three tanks are shown.

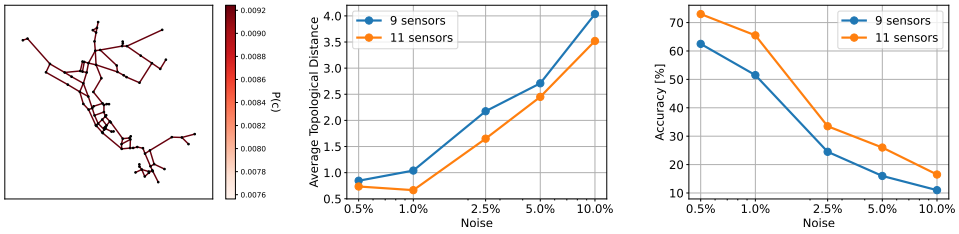
As for test WDN1, we worked with a prior distribution of the leak locations and without any prior knowledge. The prior distribution is shown in Figure 7d. We made use of a transformer architecture. The results are shown in Figure 7. The likelihood was computed with 30,000 samples. We tested with 200 different leak locations.

The results were very similar to the results for WDN1, which suggests that the framework scales well to more complicated settings.

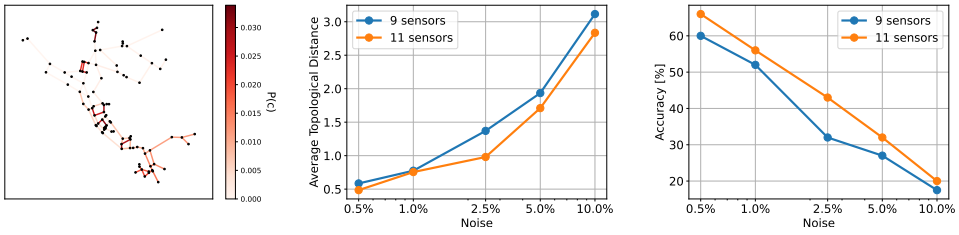
4.5. Test WDN3: Modena

For test WDN3, we again considered two different sensor configurations (see Figure 3c), each with five different sensor noise levels. The demand patterns were modeled in the same way as for WDN1, but with a different total demand. See Figure 4c for the total demand time series.

Uniquely to this test case, we made use of a different standard deviation for the likelihood computation than for the noise added to the observations. Specifically, we used a standard deviation of 5%, no matter the artificial noise added to the observations. This is more reminiscent of a real-world scenario, where the true sensor noise would be unknown.



(**a**) Prior probability. (**b**) ATD. (**c**) Accuracy.

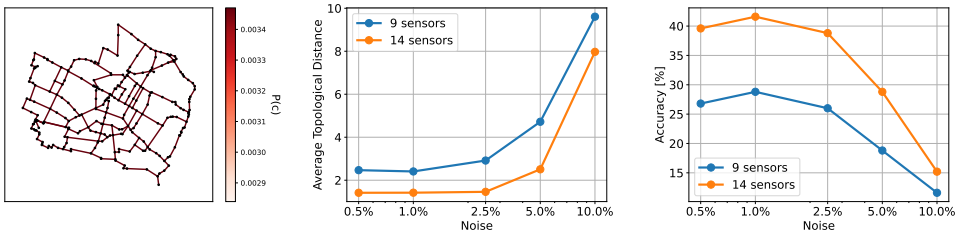


(**d**) Prior probability. (**e**) ATD. (**f**) Accuracy.

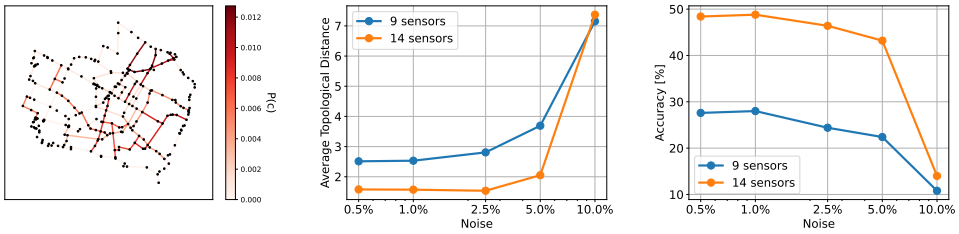
Figure 7. Results for WDN2 for the given priors. The first row shows the results for the prior in (**a**), and the second row shows the results for the prior in (**d**). The number of sensors counted is the number of flow rate sensors. Note that there are fewer pressure head sensors.

As for test cases WDN1 and WDN2, we worked with a prior distribution of the leak locations and without any prior knowledge. The prior distribution is shown in Figure 8d. Here, we made use of the ResNet architecture. The results are shown in Figure 8. The likelihood was computed with 50,000 samples. We tested with 250 different leak locations.

As for the other two test cases, we saw a better performance when we made use of prior information. Furthermore, increasing the number of sensors also gave better results.



(**a**) Prior probability. (**b**) ATD. (**c**) Accuracy.



(**d**) Prior probability. (**e**) ATD. (**f**) Accuracy.

Figure 8. Results for WDN3 for the given priors. The first row shows the results for the prior in (**a**), and the second row shows the results for the prior in (**d**). The number of sensors counted is the number of flow rate sensors. Note that there are fewer pressure head sensors.

We saw a decrease in accuracy and an increase in ATD when the noise level was increased. However, in contrast to the other test cases, the performance was relative constant across noise levels, until 5% and 10%, where the accuracy and ATD worsened. This suggests that our method is stable until a certain threshold, where more information

is needed to say something definite about the leak location. This is not a surprise, as the WDN was significantly larger than in the two other test cases and therefore might require additional observations when there is a lot of noise present. In such cases it is of great value that our method also quantifies the uncertainty associated with the prediction. Hence, the user will know when there is insufficient information to come to a strong conclusion.

4.6. Comparison with the Baseline

In Table 2, we show a comparison of the results computed with our framework and the baseline classifier. Clearly, the proposed framework outperformed the classifier for both WDN1 and WDN2 with both the ResNet and the transformer architectures. Furthermore, the transformer network gave the best results, suggesting that the transformer architecture is a suitable choice for small to medium-sized WDNs.

For WDN3, the results were different. The ResNet architecture with the new framework gave the best results, while the transformer architecture did not perform very well. However, with the classifier, the transformer architecture performed the best, suggesting that the transformer architecture may also be a satisfactory choice in that setting.

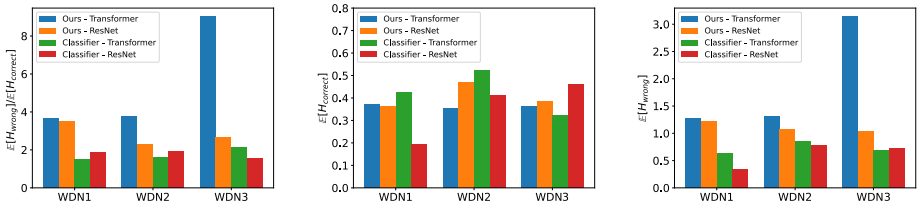
Table 2. Comparison with the baseline. The ATD and accuracy were computed for all noise and sensors cases, both with a prior and without a prior, and then averaged. Our framework is denoted by "Bayes", followed by the neural networks utilized. The vertical arrows denote the desired direction of the metric. The best results are highlighted in boldface.

	WDN1		**WDN2**		**WDN3**	
	ATD ↓	**Acc** ↑	**ATD** ↓	**Acc** ↑	**ATD** ↓	**Acc** ↑
Classifier-ResNet	2.94	26.5	5.81	13.25	10.01	7.85
Classifier-Transformer	2.06	31.67	3.60	21.5	4.88	27.92
Bayes-WAE, ResNet	1.23	45.70	2.91	25.03	**3.49**	**29.50**
Bayes-WAE, Transformer	**1.07**	**50.00**	**1.71**	**39.28**	10.38	4.54

4.7. Posterior Distribution Entropy

To evaluate the entropy computations, we considered three different entropy functions, i.e., the mean entropy of the posterior for all incorrect predictions, $\mathbb{E}[H_{\text{wrong}}(C)]$, the mean entropy of the posterior for all correct predictions $\mathbb{E}[H_{\text{correct}}(C)]$, and the ratio between the two $\mathbb{E}[H_{\text{wrong}}(C)]/\mathbb{E}[H_{\text{correct}}(C)]$. This ratio informs us about the relative size of the entropy of an incorrect prediction compared to a correct prediction. Ideally, this ratio should be large, as this is an indication that the framework provides high uncertainty for incorrect predictions and a low uncertainty for correct predictions.

In Figure 9, it is clear that, compared with the classifier, the new framework showed consistently higher entropy ratios. Thus, the framework provided more accurate uncertainty estimates compared to the classifier. In particular, for the transformer neural network in the new approach, the entropy ratio was high, because the entropy for incorrect predictions was high. In summary, if the ATDs and accuracy were not high, this would be reflected in the entropy. Therefore, we know a posteriori, and accurately, when predictions computed with the new framework are trustworthy. This is an important feature and one that is not easily attained.



(**a**) Ratio. (**b**) Correct predictions. (**c**) Wrong predictions

Figure 9. Entropy of the estimated posterior distributions. (**a**) Shows the ratio between the mean entropy of the posterior for a wrong prediction and a correct prediction. (**b**) Shows the mean entropy of the posterior distribution for correct predictions. (**c**) Shows the mean entropy of the posterior distribution for wrong predictions

5. Conclusions

We presented a framework for computing leakage locations using Bayesian inference and generative deep learning. A Bayesian approach was used to formulate the leak localization in a probabilistic manner. A generative neural network was trained to approximate the distribution of pressure heads and flow rates given a leak location and time using the WAE framework. To use the generative neural network for leak localization, the Bayesian problem was reformulated to a latent Bayesian inference problem. The approach was showcased in three test cases and showed a superior performance compared to a classification approach.

A Bayesian approach to leak localization offers two distinct advantages compared to non-probabilistic methods. First, it automatically gives the uncertainty of the prediction, in the shape of the posterior distribution. Second, it allows one to incorporate prior knowledge. However, it comes at a cost to the computation time.

The generative deep learning is trained in an offline stage on simulated data. After training, it then serves as a fast to evaluate surrogate model. In other words, the Bayesian inverse problem can be solved efficiently and without sacrificing accuracy when the neural network is trained properly.

We showed that one can obtain good quality performance using different architectures. We specifically showcased results using transformers and dense ResNets. The transformer architecture performed best in the two first test cases, while the ResNet performed best in the last test case. Therefore, one should investigate which neural network architecture to use in advance. However, the fact that the framework works with various architectures means that new state-of-the-art architectures can easily be incorporated.

For all test cases considered, the new framework outperformed the classification approach. Furthermore, the flexibility of the framework was shown through varying the amount of noise in the nodal demand and sensor measurements. Even when the true noise level was unknown, it was shown in test case 3 that the method performed well. Furthermore, the (positive) impact of prior knowledge of the leak location on the accuracy was detailed.

The framework not only provides leak location estimates, but also estimates of the uncertainty in the estimates. This is achieved by means of the entropy of the posterior distribution. We showed that the entropy computed with the new framework was significantly more informative than for the classification approach.

This paper thus showed the potential of using generative deep learning as a stochastic digital twin for water distribution networks, as it was seamlessly integrated with Bayesian inversion schemes.

There are many opportunities for further study. The computation of the posterior could be sped up with more efficient integration methods than Monte Carlo simulation. The physics of the problem could be incorporated into the training of the neural network, to ensure conservation of important quantities such as mass and momentum.

Author Contributions: Conceptualization, N.T.M., S.M.B. and C.W.O.; Methodology, N.T.M.; Software, N.T.M.; Validation, N.T.M. and P.P.; Formal analysis, N.T.M.; Investigation, N.T.M., P.P. and S.J.; Writing—original draft, N.T.M.; Writing—review & editing, N.T.M., S.J., S.M.B. and C.W.O.; Supervision, S.J., S.M.B. and C.W.O.; Project administration, S.J., S.M.B. and C.W.O.; Funding acquisition, S.J., S.M.B. and C.W.O. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Dutch National Science Foundation NWO under the grant number 629.002.213, which was a cooperative project with IISc Bangalore and Shell Research as project partners.

Data Availability Statement: The code for generating the synthetic data used in the paper can be found in the GitHub repository: https://github.com/nmucke/DT-for-WDN-leak-localization.git (accessed on 13 June 2023)).

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A. Attention and Transformers

We will highlight the relevant features of the transformer architecture used in this work. The key to the transformer architecture is the so-called attention mechanism. The scaled dot-product attention is the method of choice here. For a matrix, $X \in \mathbb{R}^{k \times d}$, the scaled dot-product attention is computed by

$$K = \mathcal{F}_k(X) \in \mathbb{R}^{k \times d}, \quad Q = \mathcal{F}_q(X) \in \mathbb{R}^{k \times d}, \quad V = \mathcal{F}_v(X) \in \mathbb{R}^{k \times d}, \tag{A1a}$$

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d}}\right)V, \tag{A1b}$$

where K refers to the keys, Q the queries, and V the values. $\mathcal{F}_k$, $\mathcal{F}_q$, and $\mathcal{F}_v$ are typically shallow neural networks to be fitted during training. k is the context length and d is the embedding dimension. An attention layer typically consists of so-called multiple attention "heads". Each head is of the same structure but consists of different functions, $\mathcal{F}_k$, $\mathcal{F}_q$, and $\mathcal{F}_v$, resulting in multiple attention maps, C_i, that are concatenated along the d^{th} dimension.

By connecting the attention layer to a residual connection, a normalization layer, a dense neural network, another residual connection, and another normalization, we have the transformer encoder module. See Figure A1 for a visualization.

With matrix X, a sequence of k elements, with each element being a vector of size d, the attention mechanism computes how much each element in the sequence should attend to every other element. In the context of natural language processing, the attention mechanism computes how much every word in a sequence is influenced by any other word in the sequence. A slightly more suitable interpretation in the present application is to consider X a graph with k nodes and d features in each node. The attention mechanism then computes how strong the connection should be between each pair of nodes.

In Equation (A1), the attention described is referred to as self-attention, referring to the fact that all elements of X only attend to other elements in X. Similarly, one defines cross-attention using the attention map from another source as keys and values. By combining this with self-attention, normalization, and a dense neural network, we have the decoder transformer module, see also Figure A1 for a visualization.

The transformer encoder and decoder networks are not aware of the relative positions of the individual nodes. Therefore, positional encoding is added to the features before passing them through the transformers.

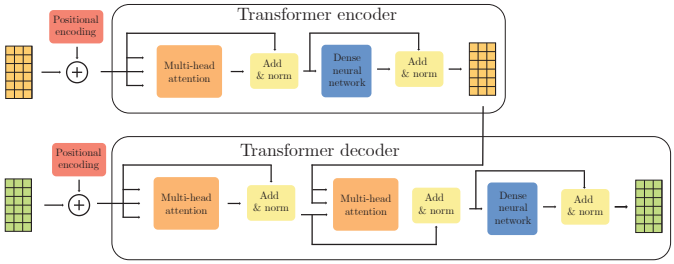


Figure A1. Transformer encoder and decoder architecture.

Appendix B. Model Architecture

Appendix B.1. Dense ResNet

The encoder consists of a series of ResNet layers, followed by dense layers that reduce the dimensionality. Similarly, the decoder consists of ResNet layers, followed by dense layers that increase the dimensionality. The input layer of the encoder takes a vector with the flow rates and pressure heads concatenated and outputs the latent state. The decoder takes the latent state and the parameters (leak location and time) concatenated and outputs the full order state.

Appendix B.2. Transformer

For the encoder, the data are first passed through a dense layer, in order to increase the embedding dimension. Then, they are passed through several transformer encoder layers. The transformer layers model interactions between the nodes and edges of the network. The resulting attention maps are reshaped into a vector and passed through dense layers, in order to decrease the dimensionality. Lastly, the reduced representation of the network is passed through transformer encoder layers, in order to model the relations between the latent features. The decoder follows a similar structure, the difference being the inclusion of the parameters and transformer decoder layers instead of encoder layers. The parameters are passed through two different sets of transformer encoder layers. The output of the first set of transformer encoder layers is used as input to the cross-attention in the latent transformer decoder layers. The output of the second transformer encoder layer is passed to the full state space decoder transformer layers. For a visualization of the architectures, see Figure A2.

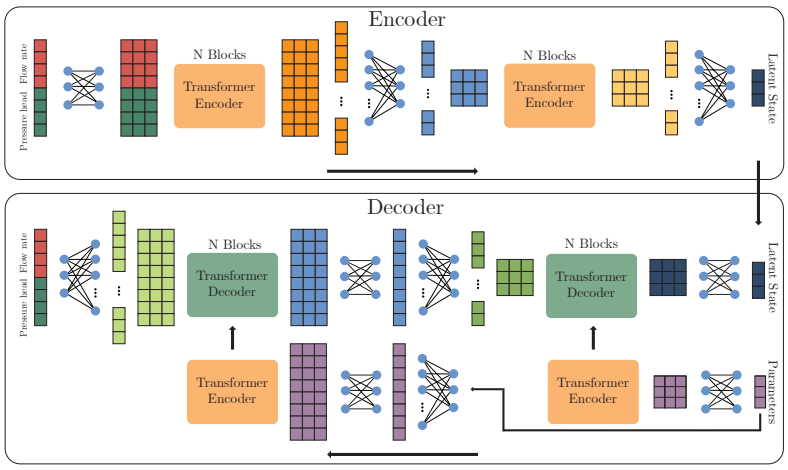


Figure A2. Illustration of the encoder and decoder architectures.

Appendix C. WAE Training

All neural network code was based on PyTorch [36]. Training was performed in a similar manner for all three test cases. We used the Adam optimizer with a cosine warm-up

learning rate scheduler with 50 warm-up steps, as described in [37]. That is, the learning rate started as a small value, and was increased to the chosen value over 50 epochs. Thereafter, it was reduced following a cosine function over the remaining epochs. The gradient norms were clipped to 0.5, to stabilize the training. We made use of early stopping with a patience of 50 to avoid over-fitting; that is, if there were 50 consecutive epochs without improvement on the validation data, we stopped the training and made use of the best performing model. In all test cases, we trained on 25,000 samples and used 5000 samples for validation. For all hyperparameters, see Table A1.

Table A1. Hyperparameters for the WAEs.

	WDN 1	**WDN 2**	**WDN 3**
Training hyperparameters			
Batch size	256	256	256
Learning rate	5×10^{-5}	5×10^{-5}	5×10^{-5}
l^2 regularization	10^{-10}	10^{-10}	10^{-10}
MMD regularization	10^{-2}	10^{-2}	10^{-2}
Scheduler warmup	50	50	50
Early stopping patience	50	50	50
ResNet Architecture			
Latent dimension	8	16	16
# layers	5	5	7
# neurons	64, 48, 32, 24, 16	192, 160, 128, 96, 64, 32	512, 384, 320, 256, 192, 128, 64
Act. func.	Leaky ReLU	Leaky ReLU	Leaky ReLU
Transformer Architecture			
Latent dimension	8	16	16
Num. dense neurons	32	128	256
Embedding dimension	4	4	4
# Attn. heads	2	2	2
# Latent transformer blocks	2	2	2
# Full order transformer blocks	1	1	1
Act. func. transformer layers	GeLU	GeLU	GeLU
Act. func. dense layers	Leaky ReLU	Leaky ReLU	Leaky ReLU

Appendix D. Classification Model

The classification neural network, g, outputs a vector of the same size as the number of pipe sections, N_p. Furthermore, it has a softmax activation after the last layer, to ensure that the output sums to 1. The model was trained on a labeled dataset consisting of noiseless sensor observations, $H(\boldsymbol{x}) = \boldsymbol{y}$ and the corresponding leak location, c, as the target. The leak location was one-hot encoded; that is, the target was a zero vector with a one at the entry of leak location index. Hence, the dataset is given by

$$\{(H(\boldsymbol{x}_1), \boldsymbol{c}_1), \ldots, (H(\boldsymbol{x}_N), \boldsymbol{c}_N)\}. \tag{A2}$$

The neural network is trained using the cross-entropy loss:

$$L(g) = -\sum_{i=1}^{N} \sum_{j=1}^{N_p} \boldsymbol{c}_{i,j} \log(g(H(\boldsymbol{x}_i)_j), \tag{A3}$$

where $\boldsymbol{c}_{i,j}$ is the jth index of the ith training sample and $H(\boldsymbol{x}_i)_j$ is the jth index of the output of the neural network evaluated at the ith training sample. Hence, the neural network was trained to output the probability of a leak being present in each possible pipe section given

the observations, i.e., the posterior distribution, $g(\boldsymbol{y}_i) = p_{c|\boldsymbol{y}}(c_k|\boldsymbol{y}_i)$. In the online stage, the distribution is updated with observations in time, by

$$p_{c|\boldsymbol{y}}(c_k|\boldsymbol{y}_{0:N_t}) = p_c(c_k)\prod_{i=0}^{N_t} p_{c|\boldsymbol{y}}(c_k|\boldsymbol{y}_{0:i}) = p_c(c_k)\prod_{i=0}^{N_t} g(\boldsymbol{y}_i), \tag{A4}$$

until convergence with respect to the KL-divergence. This is the same procedure as for the proposed framework.

It is important to keep in mind that the classifier only approximates the leak location posterior for a single sensor configuration. This is in contrast to the proposed framework, where the full state posterior is approximated together with the leak location. This also means that the classification neural network needs to be retrained every time the sensor configuration is changed.

References

1. Sun, C.; Parellada, B.; Puig, V.; Cembrano, G. Leak localization in water distribution networks using pressure and data-driven classifier approach. *Water* **2019**, *12*, 54. [CrossRef]
2. Javadiha, M.; Blesa, J.; Soldevila, A.; Puig, V. Leak localization in water distribution networks using deep learning. In Proceedings of the 2019 6th International Conference on Control, Decision and Information Technologies (CoDIT), Paris, France, 23–26 April 2019; pp. 1426–1431.
3. Mohammed, E.G.; Zeleke, E.B.; Abebe, S.L. Water leakage detection and localization using hydraulic modeling and classification. *J. Hydroinform.* **2021**, *23*, 782–794. [CrossRef]
4. Romero, L.; Blesa, J.; Puig, V.; Cembrano, G.; Trapiello, C. First results in leak localization in water distribution networks using graph-based clustering and deep learning. *IFAC-PapersOnLine* **2020**, *53*, 16691–16696. [CrossRef]
5. Shekofteh, M.; Jalili Ghazizadeh, M.; Yazdi, J. A methodology for leak detection in water distribution networks using graph theory and artificial neural network. *Urban Water J.* **2020**, *17*, 525–533. [CrossRef]
6. Gao, H.; Dai, B.; Miao, H.; Yang, X.; Barroso, R.J.D.; Walayat, H. A novel gapg approach to automatic property generation for formal verification: The gan perspective. *ACM Trans. Multimed. Comput. Commun. Appl.* **2023**, *19*, 1–22. [CrossRef]
7. Gao, H.; Qiu, B.; Barroso, R.J.D.; Hussain, W.; Xu, Y.; Wang, X. Tsmae: A novel anomaly detection approach for internet of things time series data using memory-augmented autoencoder. *IEEE Trans. Netw. Sci. Eng.* **2022**, 1. [CrossRef]
8. Singh, M.N.; Khaiyum, S. Enhanced data stream classification by optimized weight updated meta-learning: Continuous learning-based on concept-drift. *Int. J. Web Inf. Syst.* **2021**, *17*, 645–668. [CrossRef]
9. Jurek-Loughrey, A. Deep learning based approach to unstructured record linkage. *Int. J. Web Inf. Syst.* **2021**, *17*, 607–621. [CrossRef]
10. van Lagen, G.; Abraham, E.; Esfahani, P.M. A Bayesian Approach for Active Fault Isolation with an Application to Leakage Localization in Water Distribution Networks. *IEEE Trans. Control. Syst. Technol.* **2022**, *31*, 761–771. [CrossRef]
11. Maier, G.; Bolzon, G.; Buljak, V.; Garbowski, T.; Miller, B. Synergic combinations of computational methods and experiments for structural diagnoses. *Comput. Methods Mech.* **2010**, *1*, 453–476.
12. Frangos, M.; Marzouk, Y.; Willcox, K.; van Bloemen Waanders, B. Surrogate and reduced-order modeling: A comparison of approaches for large-scale statistical inverse problems. In *Large-Scale Inverse Problems and Quantification of Uncertainty*; Wiley Online Library: Hoboken, NJ, USA, 2010 ; pp. 123–149.
13. Guo, M.; Hesthaven, J.S. Reduced order modeling for nonlinear structural analysis using Gaussian process regression. *Comput. Methods Appl. Mech. Eng.* **2018**, *341*, 807–826. [CrossRef]
14. Mücke, N.T.; Bohté, S.M.; Oosterlee, C.W. Reduced Order Modeling for Parameterized Time-Dependent PDEs using Spatially and Memory Aware Deep Learning. *J. Comput. Sci.* **2021**, *53* , 101408. [CrossRef]
15. Hesthaven, J.S.; Ubbiali, S. Non-intrusive reduced order modeling of nonlinear problems using neural networks. *J. Comput. Phys.* **2018**, *363*, 55–78. [CrossRef]
16. Li, Z.; Kovachki, N.; Azizzadenesheli, K.; Liu, B.; Bhattacharya, K.; Stuart, A.; Anandkumar, A. Fourier neural operator for parametric partial differential equations. *arXiv* **2020**, arXiv:2010.08895.
17. Mücke, N.T.; Sanderse, B.; Bohté, S.; Oosterlee, C.W. Markov Chain Generative Adversarial Neural Networks for Solving Bayesian Inverse Problems in Physics Applications. *arXiv* **2021**, arXiv:2111.12408.
18. Drygala, C.; Winhart, B.; di Mare, F.; Gottschalk, H. Generative modeling of turbulence. *Phys. Fluids* **2022**, *34*, 035114. [CrossRef]
19. Patel, D.; Oberai, A.A. Bayesian inference with generative adversarial network priors. *arXiv* **2019**, arXiv:1907.09987.
20. Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; Bengio, Y. Generative adversarial networks. *Commun. ACM* **2020**, *63*, 139–144. [CrossRef]
21. Kingma, D.P.; Welling, M. Auto-encoding variational bayes. *arXiv* **2013**, arXiv:1312.6114.
22. Sohl-Dickstein, J.; Weiss, E.; Maheswaranathan, N.; Ganguli, S. Deep unsupervised learning using nonequilibrium thermodynamics. In Proceedings of the International Conference on Machine Learning, Lille, France, 7–9 July 2015 ; pp. 2256–2265.

23. Xiao, Z.; Kreis, K.; Vahdat, A. Tackling the generative learning trilemma with denoising diffusion GANs. *arXiv* **2021**, arXiv:2112.07804.
24. Tolstikhin, I.; Bousquet, O.; Gelly, S.; Schoelkopf, B. Wasserstein auto-encoders. *arXiv* **2017**, arXiv:1711.01558.
25. Simpson, A.; Elhay, S. Jacobian matrix for solving water distribution system equations with the Darcy-Weisbach head-loss model. *J. Hydraul. Eng.* **2011**, *137*, 696–700. [CrossRef]
26. Rossman, L.A. EPANET 2: Users Manual. 2000. Available online: https://nepis.epa.gov/Exe/ZyNET.exe/P1007WWU.TXT?ZyActionD=ZyDocument&Client=EPA&Index=2000+Thru+2005&Docs=&Query=&Time=&EndTime=&SearchMethod=1&TocRestrict=n&Toc=&TocEntry=&QField=&QFieldYear=&QFieldMonth=&QFieldDay=&IntQFieldOp=0&ExtQFieldOp=0&XmlQuery=&File=D%3A%5Czyfiles%5CIndex%20Data%5C00thru05%5CTxt%5C00000024%5CP1007WWU.txt&User=ANONYMOUS&Password=anonymous&SortMethod=h%7C-&MaximumDocuments=1&FuzzyDegree=0&ImageQuality=r75g8/r75g8/x150y150g16/i425&Display=hpfr&DefSeekPage=x&SearchBack=ZyActionL&Back=ZyActionS&BackDesc=Results%20page&MaximumPages=1&ZyEntry=1&SeekPage=x&ZyPURL (accessed on 29 June 2023).
27. Hastie, T.; Tibshirani, R.; Friedman, J.H.; Friedman, J.H. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*; Springer: New York, NY, USA, 2009; Volume 2.
28. Xiu, D. *Numerical Methods for Stochastic Computations: A Spectral Method Approach*; Princeton University Press: Princeton, NJ, USA, 2010 .
29. Kramer, M.A. Nonlinear principal component analysis using autoassociative neural networks. *AIChE J.* **1991**, *37*, 233–243. [CrossRef]
30. Gretton, A.; Borgwardt, K.M.; Rasch, M.J.; Schölkopf, B.; Smola, A. A kernel two-sample test. *J. Mach. Learn. Res.* **2012**, *13*, 723–773.
31. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, USA, 27–30 June 2016; pp. 770–778.
32. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, Ł.; Polosukhin, I. Attention is all you need. *Adv. Neural Inf. Process. Syst.* **2017**, *30* , 6000–6010.
33. Dosovitskiy, A.; Beyer, L.; Kolesnikov, A.; Weissenborn, D.; Zhai, X.; Unterthiner, T.; Dehghani, M.; Minderer, M.; Heigold, G.; Gelly, S.; et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv* **2020**, arXiv:2010.11929.
34. Jumper, J.; Evans, R.; Pritzel, A.; Green, T.; Figurnov, M.; Ronneberger, O.; Tunyasuvunakool, K.; Bates, R.; Žídek, A.; Potapenko, A.; et al. Highly accurate protein structure prediction with AlphaFold. *Nature* **2021**, *596*, 583–589. [CrossRef]
35. Lim, B.; Arık, S.Ö.; Loeff, N.; Pfister, T. Temporal fusion transformers for interpretable multi-horizon time series forecasting. *Int. J. Forecast.* **2021**, *37*, 1748–1764. [CrossRef]
36. Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In Proceedings of the Advances in Neural Information Processing Systems 32, Vancouver, BC, Canada, 8–14 December 2019 ; pp. 8024–8035.
37. Loshchilov, I.; Hutter, F. Sgdr: Stochastic gradient descent with warm restarts. *arXiv* **2016**, arXiv:1608.03983.