university of groningen

faculty of science and engineering

DeepFlow: A deep learning pipeline for leakage detection in water distribution networks

Research internship

July 2023

Student: C.M. van Riemsdijk

First supervisor: D. Düştegör

Secondary supervisors: H. Truong & A. Tello

Contents

List of Figures

List of Tables

Table 1: List of Abbreviations

BattLeDIM The Battle of the Leakage Detection and Isolation Methods. 7, 13

CNN convolutional neural network. 5, 7

DL deep learning. 5–7, 10, 11

FP false positives. 12

GNN graph neural network. 4–7, 14

ML machine learning. 5

MLP Multi-layer perceptron. 7

SCADA supervisory control and data acquisition. 5, 7

SILU Sigmoid linear unit. 11

TPR true positive rate. 12–14

WDN water distribution networks. 4–8, 10, 11, 14

Abstract

In the literature, many efforts have been made to detect and localize leakages in water distribution networks (WDN). Leakage detection has been done by model-based, data-based, and model-transient-based methods. Many show promising results but are limited by needing a lot of historical data or that they only work on parts of WDNs, this is mostly because the models are not topologically aware. Therefore, efforts have been made to use graph neural network (GNN)s to create topological awareness within the models. As reproducibility is important, an effort is made to create a modular pipeline based on the literature to create a playground for researchers to easily train, and evaluate different GNN architectures. Moreover, we conduct our own experiments on this modular pipeline to compare with the literature.

1 Introduction

Leakage detection in water networks is paramount since water is a crucial resource for all life on Earth. As we are growing as a population as a whole, it is necessary to have efficient management of water resources. Distributing water is done by water distribution networks (WDN). These networks, consisting of junctions, water pipes, pumps, reservoirs, and tanks, are mostly located underground and are therefore difficult to maintain. Due to, for example, the ageing of the pipes or corrosion, leaks can form in these pipes. This can cause huge losses of water, resulting in economic problems. The percentage of the loss of water is between 15% to 50% (Shukla and Piratla, 2020). Preventing water leakage is therefore a state-of-the-art research topic.

In the literature, different approaches are used for leakage detection. These approaches can be categorized into three categories: *model-based* approaches, *data-driven* approaches (Teck Kai et al., 2018), and *model-transient* approaches (Kang et al., 2017). Model-based approaches use hydraulic mathematical equations and systems to simulate water distribution networks. With these systems, the WDN's can be modeled by historical *demand* data. Multiple studies show that model-based approaches have strong leakage detection capabilities. However, they require historical demand data and that is hard to come by (Vrachimis et al., 2021; Soldevila et al., 2016). Data-driven approaches use historical pressure data. These measurements are recorded by supervisory control and data acquisition (SCADA) systems. The downside of this approach is that there are rarely any data available from the WDN. In the cases where it is available, most of the data is limited. Data-driven methods need a lot of historical data, and because leaks are in the minority in the data, we need to search for a better option. One of these options is deep learning (DL).

Deep learning has shown in multiple areas – computer vision, audio, natural language processing – that it can be a great estimator for many tasks. Where in classic machine learning (ML) we use a feature extractor, in DL feature extraction is done by the model itself. Due to this, a DL model is able to extract the important features from a WDN itself. Moreover, DL models contain the capability to model non-linear relationships for input vectors, which is the case for WDNs. Over the last few years, a lot of research is done in deep learning methods. One method to detect and localize leakages is done by transforming the pressure time series to a pressure map and feeding that to a convolutional neural network (CNN) (Javadiha et al., 2019). These approaches work fine, however, a lot of data is needed for deep learning (Goodfellow et al., 2016). However, the data that you feed DL models is important for their generalizability. Many methods still encounter problems with the under-representation of leakages in the data or the presence of noise (Ben et al., 2022).

Newer research suggests using a combination of *model-based* and *data-driven* approaches, so-called *model-transient* approaches. In many cases, data-driven approaches are used to predict the entire state of the WDN, and the model-based approach is used for leakage detection (Örn Gararsson et al., 2022; Ben et al., 2022). Where Ben et al. (2022) use linear interpolation to recreate the entire pressure state of the WDN, Örn Gararsson et al. (2022) use graph neural network (GNN) to predict the state. The advantage of using GNNs is that it can use the topological structure of the WDN to have better generalizability and to reduce the impact of data-hungry models (Örn Gararsson et al., 2022). This report adds to the work of Örn Gararsson et al. (2022) where the pipeline that they proposed is built into a modular pipeline. This pipeline creates a playground for researchers to experiment, tune, and evaluate models. To make the pipeline proposed by Örn Gararsson et al. (2022) modular, we need to handle the following:

1. GNN models need to be exchangeable, this means that different internal architectures can be different provided that the input and output sizes are the expected sizes in the pipeline.
2. Different datasets, data formats, and WDNs should be easy to use.
3. Leakage formats should be standardized such that ground truth values only need to be parsed once, and therefore a standard algorithm can be used for calculating performance metrics.
4. Configuration should be easy and extensive for reproducibility and ease of use.

In section 2 we will explain GNNs, dataset, and the pipeline. Section 3 will highlight the implementation details of the pipeline and its modularity. Section 4 will show the results of the GNNs followed by a discussion.

2 Methodology

2.1 Graph neural networks

As mentioned in section 1, many deep learning approaches have already been used for leak localization. The results are promising, but, these results have flaws. In a few cases, a lot of data is needed, in others, the test conditions are constricted to the paper's dataset and/or problem. We would like to generalize this for WDNs such that we can use certain models for multiple datasets and compare them. If we abstract a WDN it is essentially a graph of the form: $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. $\mathcal{V}$ denotes the set of nodes, $v \in \mathcal{V}$ could be a junction, a tank, or a reservoir, but in this particular case we only consider the junctions in the WDN. $\mathcal{E}$ is the set of edges, where each $e_{ij} \in \mathcal{E}$ is a pipe connecting two junctions. An example of a WDN can be seen in Figure 1.

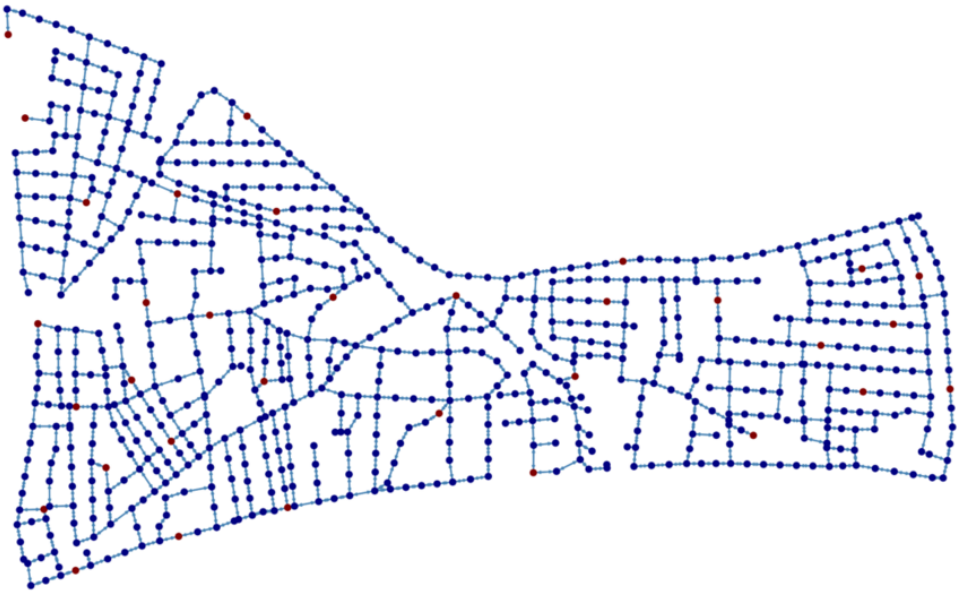

Figure 1: WDN graph of L-town (Vrachimis et al., 2022)

Figure 1 clearly shows the graphical nature of a WDN. The nodes (V) and edges (E) are clearly visible. Graphs can have directed and undirected edges. For this report, we will focus on undirected graphs, as the difference with directed does not influence the performance of the models (Hajgató et al., 2021). It is clear that graphs do not use standard Euclidean

data. GNNs are neural networks that can cope with non-Euclidian/graph formatted data. Their goal is to learn node and edge embeddings from data to do certain tasks, e.g. node- and edge-classification. In the literature, there are 2 sorts of GNNs: spatial- and spectral-based. The former lends its idea from CNNs, where message-passing is the core principle. The general form of message-passing GNNs is:

$$x_i^{(k)} = \gamma^{(k)} \left(x_i^{(k-1)}, \bigoplus_{j \in \mathcal{N}(\mathcal{I})} \phi^{(k)}(x_i^{(k-1)}, x_j^{(k-1)}, e_{j,i}) \right)$$

where $\bigoplus$ is a differentiable operator, e.g., sum, max. $x_i^{(k-1)}$ is the node's representation of the node X_i in the latent space $\mathcal{Z}$ for the layer $(k-1)$. With the exception, that in layer 0 X_i represents the input (original) features of the node. The same holds for the optional edge features: $e_{j,i}^{(k-1)}$. At last, the γ and ϕ are differentiable functions like a Multi-layer perceptron (MLP). The advantages of spatial-based GNNs are that they are very simple and easy to adapt, however, issues start existing when we try to make the network deeper, thus scaling it. The issue that arises is the *oversmoothing* problem (Li et al., 2018). As neighbouring nodes communicate a lot, over time node embeddings will start to assimilate. This is not desired, as different nodes will have similar node embeddings when they are not similar. This eventually will hurt the final task of the end-to-end DL pipeline. Therefore, the literature proposed spectral-based GNNs. By using graph Fourier transforms, we get a signal X that can be used in the general form of the spectral-based approach (Wang and Zhang, 2022):

$$Z = \phi(g(\hat{L})\gamma(X))$$

Where $\hat{L}$ is the normalized Laplacian, Z is the prediction ϕ and γ are differentiable functions like MLPs and g is polynomial. The advantage of this approach is that it helps alleviate the *oversmoothing* problem, however, this comes with a disadvantage, namely that the computational costs increase due to the Fourier graph transforms. There are more methods to alleviate oversmoothing, e.g., normalization, regularization, and residual connections (Rusch et al., 2023).

2.2 Dataset

For the dataset, or in this case a water distribution networks (WDN), we used the fictional *L-Town* as seen in Figure 1 created by and for the The Battle of the Leakage Detection and Isolation Methods (BattLeDIM) competition. The purpose of the BattLeDIM dataset (*L-town*) is to provide an objective assessment of different methods for detecting and localizing leakage events (Vrachimis et al., 2022). The WDN consists of 782 junctions, 2 reservoirs, and 1 tank. In total, it has 905 pipes with a length of 42.6 km. Moreover, as for the SCADA readings, we have 33 pressure sensors in *L-Town*. *L-Town* is provided in the EPANET format, which means that we can simulate the WDN with EPANET's simulator. Hence, we can create 5-minute interval pressure readings for each node in the network, whereas normally we would only have 33 pressure sensors. The range of pressure levels for *L-Town* is between $20 - 50m$. We normalize this to a $[0, 1]$ scale. For the BattLeDIM competition, two years of data were created. One year for training, and the other for evaluating. Leakages were artificially created by the same simulator that created the 2 years of data by providing: start- and end times, leak sizes, and the type of leak (abrupt and incipient).

2.3 Pipeline

To detect leakages, we will recreate the pipeline proposed by Örn Gararsson et al. (2022). The pipeline can be seen in Figure 2
As seen in the figure above, the pipeline consists of two GNNs. The *reconstructor* and *predictor*. As said in subsection 2.2, we only have 33 pressure sensors in *L-Town*, but we would

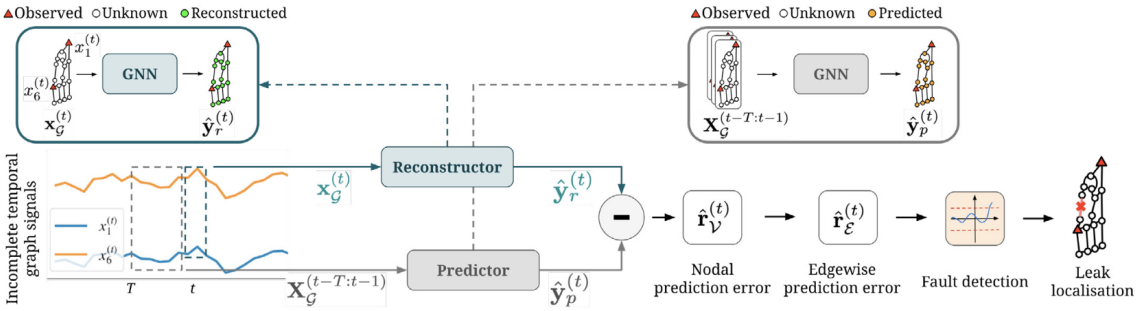

Figure 2: Leakage detection pipeline by Örn Gararsson et al. (2022)

like to infer all pressure levels at each node. Both models have the same aforementioned task, but both do this with different inputs. As we have the ground truth pressure values for each node, we can use this signal of each node to infer the nodes' pressures. The pressure signal of the whole network at time step t is denoted as:

$$X_{\mathcal{G}}^{(t)} = \left[x_1^{(t)}, x_2^{(t)}, \dots x_{\mathcal{N}}^{(t)}\right]$$

Where $\mathcal{N}$ is the total amount of nodes in the WDN. We observe $X_v^{(t)} = 0$ when the node has no pressure sensor. With the 33 pressure sensors in the WDN this means that only $\frac{33}{782} * 100\% = 4.21\%$ of the signal is non-zero. The reconstructor's task is to infer all the pressure levels at the nodes where there are no readings. This signal is defined as $\hat{y}_r^{(t)}$ for time step t. The predictor's task is the same, but instead of reconstructing the signal from the current time step, we use the previous T time steps to predict the current pressure signal. The predictor's input for a given time t and a given window size of T is:

$$X_{\mathcal{G}}^{(t-T:t-1)} = \left[x_{\mathcal{G}}^{(t-T)}, x_{\mathcal{G}}^{(t-T+1)}, \dots x_{\mathcal{G}}^{(t-1)}\right]$$

From this input, we use the predictor to infer the signal and denote it as $\hat{y}_r^{(t)}$. To train both models and to emulate the low percentage of measured pressure nodes, we use a method called *masking*. Masking can be done in two ways, 1) Specifying the nodes that need to be set to 0, 2) Masking nodes randomly with a given percentage. For training, we use the latter option. We set a given percentage, in our case, 95% who are masked. This corresponds roughly to the 4.21% of *L-town*. For each iteration, we reset the mask and generate a new mask corresponding to the 95%.

As seen in Figure 2. A residual signal is created where the output of the predictor is subtracted from the reconstructor

$$r(t) = [r_1(t), r_2(t), \dots, r_n(t)] = \hat{y}_r^{(t)} - \hat{y}_r^{(t)}$$

This results in the nodal residual of both models. Örn Gararsson et al. (2022) state that leaks often occur in pipes instead of the junctions in a WDN. This means that it would be best to transform the nodal residual to the edge residual and do statistical analysis from there. In other words, when two nodes are connected by a pipe p_{uv} where u, v are nodes in the WDN we subtract the residual signals of both nodes resulting in:

$$r_{uv}^{(E)}(t) = r_u(t) - r_v(t)$$

where $p_{uv} \in \mathcal{E}$. Finally, statistical analysis can be used to detect and localize leakages. This is done by calculating both the moving average and moving standard deviation of a given

window size m. Örn Gararsson et al. (2022) state that when pipes are in a faulty-state that there is a significant change and the residual signals means are non-zero. The last step is to filter duplicate alarms that are caused by leakages in different pipes. Therefore, if a leak is detected in a neighbour of a detected edge in a provided hop distance, the leak detection will be suppressed.

3 Implementation

As stated in section 1, in this report we would create the aforementioned pipeline such that it is modular. Therefore, creating a tool for researchers to easily train, evaluate, test, and tune different models and hyperparameters. Meaning that creating and running experiments will be easier. The modular pipeline has a simple architecture, as can be seen in Figure 3

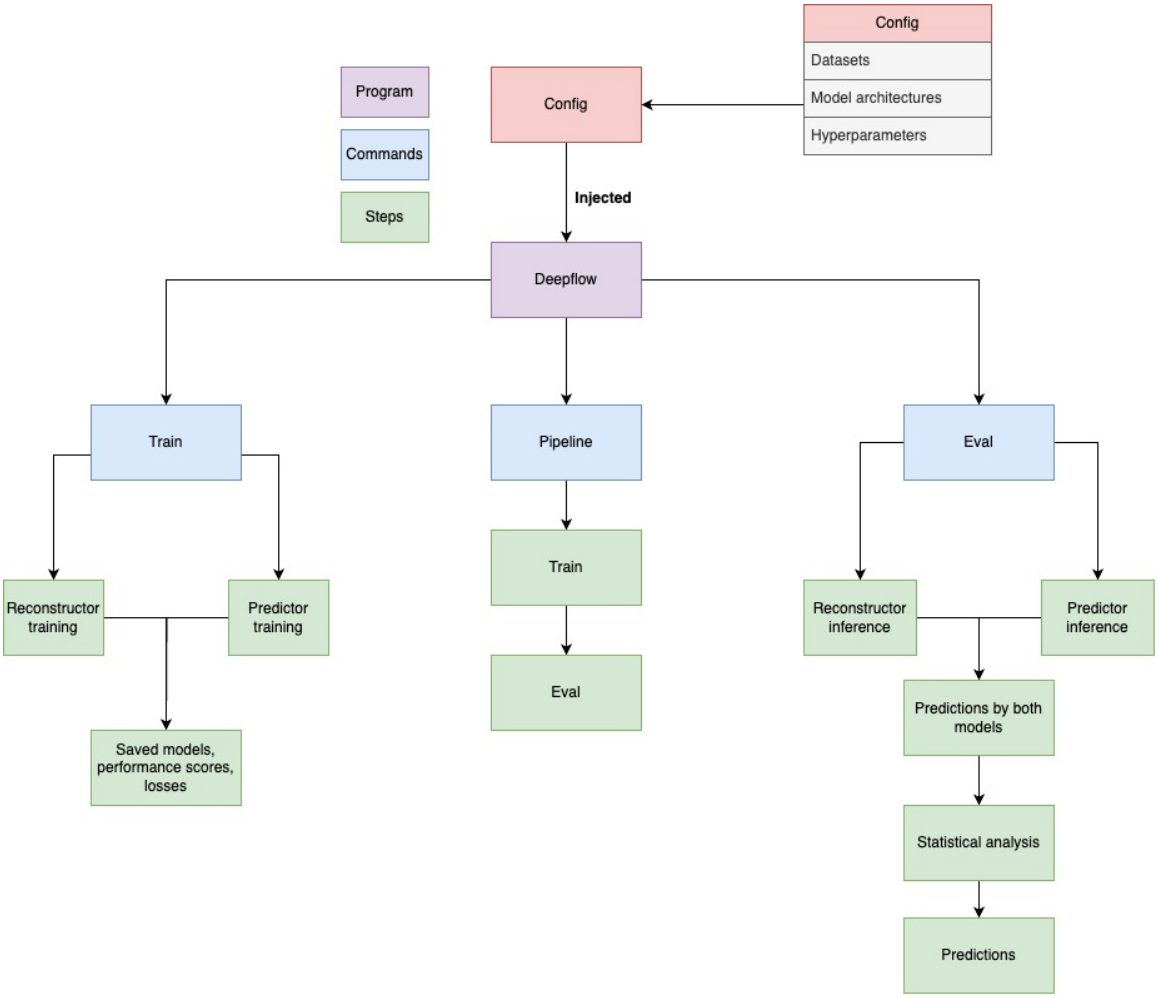

Figure 3: Modular pipeline architecture

The pipeline has 3 options: *train*, *eval*, and *pipeline*. The whole pipeline is written in *Python3.10* with *PyTorch* (Paszke et al., 2019) and *PyTorch geometric* (Fey and Lenssen, 2019) for the DL code. The *train* command is to train the models. This means that we do the training of the reconstructor and predictor sequentially. It is also a possibility to only train one of the models, in the case that a user already has one of the models. The training consists of the following steps:

1. Initialize optimizer and loss function
2. Get corresponding data loaders for the reconstructor or predictor
3. Initialize early stopping
4. Train the model for N amount of epochs

The training loop is a standard *PyTorch* training loop. We get the input, edge indices, and ground truth labels. We mask the inputs and feed them to the model. After which, we mask the output of the model and the labels to calculate the loss. We use the same mask, as we only want to act on the losses of the nodes that are not measured.

The *eval* command is used for evaluating the models with respect to the pipeline. We use a novel/not-seen dataset to evaluate the model's performance. Evaluation is done with the following steps:

1. The residual nodal signal is created from the inference of the reconstructor and predictor, these models can be from the training step, or a provided checkpoint in the config.
2. From the residual nodal signal, a residual edge error is calculated.
3. The residual edge errors are used in statistical analysis to come to a conclusion about whether an edge has a leak.
4. Performance measures are calculated from the statistical analysis.

The *pipeline* command combines the two commands, running *train* and *eval* sequentially. In the following subsections, the requirements that are mentioned in section 1 are explained.

3.1 Models

As explained, models should be exchangeable as long as they are reconstructors or predictors. When using the pipeline proposed by Örn Gararsson et al. (2022) it is mentioned that the only difference between the reconstructor and predictor is the input channels. Where the reconstructor has only 1 input channel and the predictor has T input channels. These input channels correspond to the number of graphs that are used as input as seen in Figure 2, where the reconstructor and predictor have 1 and T input graphs respectively. This simplifies loading the models, as the input channels are the only configuration needed to properly load the models and use the pipeline. If a user of the pipeline wants to use a different architecture, it is as easy as changing the parameters in the provided model or creating a new file that overrides the provided architecture. Loading and saving the model is then taken care of by the pipeline.

3.2 Data

Modularizing data and datasets is a bit different compared to the models, as models have the same *format*. Datasets and WDNs have different data sources, and they should be easy to use. For this, we have created an abstract class called `DeepflowDataset` which itself is a `PyTorch Geometric Dataset`. This abstract class tells the user to implement one

mandatory function, namely `load()`. In this function, you have to set the mandatory fields which are instructed to the user. With this, every data format can be loaded. With the `DeepFlowDataset` you can easily create a `DataLoader` which has multiple capabilities e.g., batch and shuffle data.

3.3 Leakage handling

As we are dealing with different data formats for input data, we also have this problem with the ground truth values for leakages. To easily evaluate your models, we created an abstract class called `LeakageHandler` that also only has one mandatory function that needs implementation, named `parse()`. This function has to return a list of parsed leaks, containing the leak's location, start time, and end time. This list of leaks is then used by the pipeline in the evaluation step to create the wanted metrics.

3.4 Configuration

The last requirement of the pipeline is that it should standardize the hyperparameters that we use for the particular problem. This means that experiments are reproducible and easily maintainable. Moreover, running experiments with different hyperparameters is not a hassle anymore as every parameter is condensed into one file. This is done by creating a `Config` data class that is injected into the pipeline at runtime. This config data class consists of the following settings. First and foremost, the user is able to provide a WDN, provided it is in the *EPANET* format. Next, datasets are also provided by a path. Both training and evaluation datasets can be given. Standard DL hyperparameters are also contained in the `Config` class such as epochs, learning rate, weight decay, batch size, early stopping, and whether to shuffle the data loader. Masking settings are also provided in the configuration, a user can use a masking ratio that masks a percentage of the input vector, or provide specific node numbers that correspond to the nodes in the WDN. Loading models is a core part of the pipeline, thus by only specifying the input and output channels, architectures can be loaded provided they are *PyTorch* modules. Finally, settings for leakage detection can be established.

3.5 Experiment

As for our experiment, we wanted to try and see how well our model would perform compared to the results from Örn Gararsson et al. (2022). We used a smaller version of the proposed *ChebNet* architecture (Defferrard et al., 2017). As can be seen in Figure 4

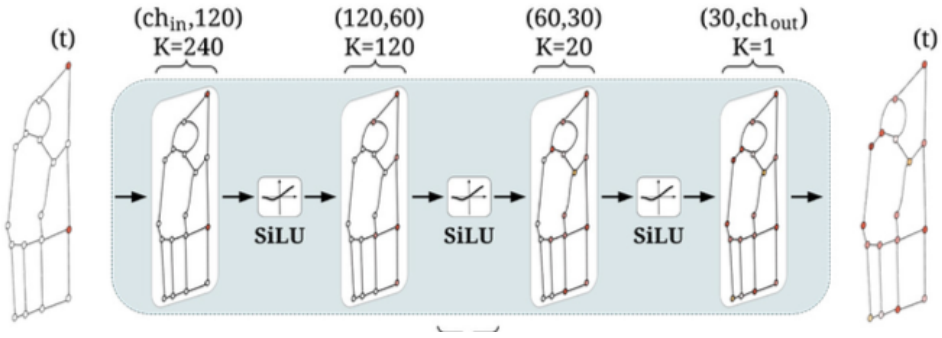

Figure 4: ChebNet architecture

We used a 4-layer structure with the Sigmoid linear unit (SILU) activation function. Hidden layers channels were set to 32 for both input and output. The filters were set to $[F_1, F_2, F_3] =$

[24, 12, 10]. The input channels were set to 1 and 3 respectively for the reconstructor and predictor. Meaning that we used 3 previous time steps for the predictor, which corresponds to 15 minutes of historical data. The training was done on a year of data, split up into 8 months of training and 4 months for evaluation. We used the Adam optimizer with a learning rate of 1×10^{-4} and a weight decay of 5×10^{-4} (Kingma and Ba, 2017). An important hyperparameter of the pipeline is the mask ratio. This constitutes the percentage of the signal that is set to zero to "mask" the input. The mask ratio is set to 0.95, meaning that both models need to reconstruct from only 5% of the readings. The batch size is 64, and early stopping is enabled with patience set to 5. Each time the global validation loss decreases, the pipeline saves the model. As for the performance metrics, we use true positive rate (TPR) and false positives (FP). is calculated by $TPR = \frac{\text{True positives}}{\text{True positives+False Negatives}}$. A true positive constitutes a pipe that is alerted with a potential leak in the given leak start- and end time. Pipes that are in a given K-hop distance from the leak are also seen as a true positive. A false negative constitutes the pipes that were not detected. are the identified leaks that are not real leaks.

4 Results & Discussion

4.1 Results

The results of the reconstructor and predictor models will now be examined, after which we will discuss the residual edge analysis and the statistical method from Örn Gararsson et al. (2022) to detect leakages. Figure 5 and Figure 6 show the loss curves for the different models: *reconstructor* and *predictor*. The line for training loss indicates how well the model fits to the training data. The other line is validation loss, which measures the ability of the model to generalize to new data. The ideal situation is therefore that both lines approach zero and are close to each other. Both models do this nicely. This means that there is no overfitting, as overfitting can be determined by looking at the generalization gap. Which is the difference between the loss of the test set and the train set. $L_{gap} = l_{\text{test}} - l_{\text{train}}$ (Goodfellow et al., 2016). Generally, overfitting happens when $L_{gap} >> 0$. However, in cases where the generalization gap is smaller, it is a judgment game, and there is no clear rule in the literature. In our case, the generalization gap is minimal, thus we observe that there is no overfitting happening.

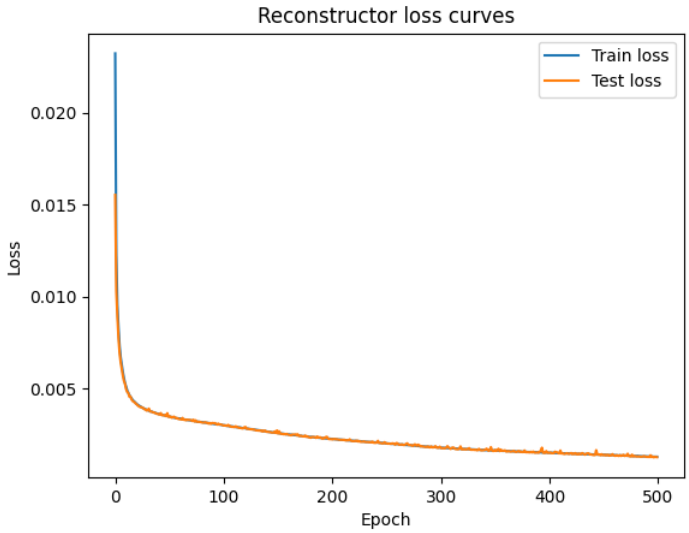

Figure 5: Loss curves reconstructor

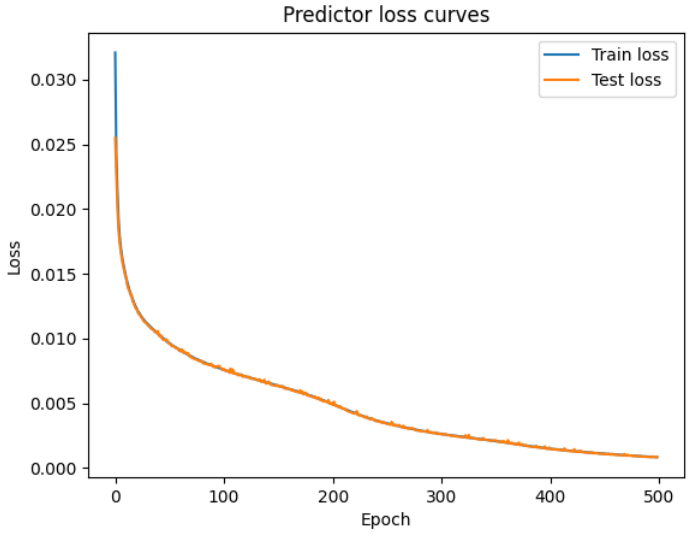

Figure 6: Loss curves predictor

To see how well the models are performing, we will use the other year that is used for validation by the BattLeDIM dataset. We will look at two cases at the beginning of the year. In Figure 7 we can observe the residual signal, moving average over a rolling window of size 7, and the standard deviation of that moving window for pipes 256 and 257. If we compare this with the same features in Figure 8 with pipes 826 and 827, we observe that pipes 256 and 257 are fluctuating a lot more and have variance. This corresponds with the ground truth, as pipe 256 has a leakage and the other pipes do not.

As for the leakage detection algorithm, we had less success compared to Örn Gararsson et al. (2022). We have a TPR of 40.62%. This is a big decrease compared to the TPR of Örn Gararsson et al. (2022) of 82.61%.

4.2 Discussion

We will discuss the results now here. As for training, the models seem to be generalizing well to the data of *L-Town*. The losses are low and there does not seem to be overfitting. We think the residuals from the pipes tell the same story. From the residual of the pipes, we can observe where leakages occur and where not. Örn Gararsson et al. (2022) use a moving window of size M to calculate the moving average and moving standard deviation to define if leaks occur, however, this does not seem to yield good results for us. What we observe in our residual edge errors in Figure 7 and Figure 8 is that our moving standard deviation is more or less around 0. Where our moving average is between 0 and 1. This means that the leakage detection algorithm proposed by Örn Gararsson et al. (2022) did not work for our models. We used the same $\alpha = 1.0$, which is a hyperparameter, to tune the moving standard deviation based on Chebyshev inequalities (Örn Gararsson et al., 2022). As for the reproducibility of the paper, there could be some differences with our approach compared to Örn Gararsson et al. (2022). For example, we do not know a lot of hyperparameters e.g., number of epochs, batch size, learning rate. Moreover, the masking strategy is not clearly stated, we randomly mask the nodes for 95% per epoch. Örn Gararsson et al. (2022) use 33 measurement sensors, it is not clear, but it seems that they fix these pressure sensors for training. Another threat to using the *L-town* dataset is that it is an artificial town and data, with artificial leaks. Generating a new year of data for evaluation may lead to leakage between training and evaluation datasets. All in all, the combination of not being able to reproduce the exact same experiments and not using the same masking strategy may

explain the results obtained by us. We think that there is also future work in the detection algorithm for leakages. An edge-wise approach is indeed a good view as leakages happen in pipes, however using moving average and moving standard deviation might not be sufficient, and this pipeline would benefit from research in other statistical methods to detect leakages.

5 Conclusion

To conclude the report, a new modular pipeline is introduced that is able to train and evaluate leakage detection in water distribution networks with different GNN architectures, datasets, and leakage standards. Moreover, plenty of hyperparameters are standardized to be used by researchers to create reproducible and comparable experiments. Comparing our results with a different model architecture, we have promising outputs from the reconstructor and predictor with the residual signals for each pipe. However, comparing our TPR with the TPR of Örn Gararsson et al. (2022) we see a decrease. This decrease can be due to multiple reasons as discussed, e.g., reproducibility, masking strategies, and validation datasets. Future work includes researching other statistical methods for detecting leakages from edge-wise residual signals.

6 Acknowledgements

I would like to express my thanks to all supervisors involved: D. Düştegör, H. Truong & A. Tello. They helped me tremendously throughout this whole research internship. Without their guidance, I would not have been able to write this report. This report is performed as a part of the project DiTEC: Digital Twin for Evolutionary Changes in Water Networks (NWO 19454).

References

Ben, L., Alves, D., Blesa, J., Cembrano, G., Puig, V., and Duviella, E. (2022). Leak localization in water distribution networks using data-driven and model-based approaches. *Journal of Water Resources Planning and Management*, 148.

Defferrard, M., Bresson, X., and Vandergheynst, P. (2017). Convolutional neural networks on graphs with fast localized spectral filtering.

Fey, M. and Lenssen, J. E. (2019). Fast graph representation learning with pytorch geometric.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

Hajgató, G., Gyires-Tóth, B., and Paál, G. (2021). Reconstructing nodal pressures in water distribution systems with graph neural networks.

Javadiha, M., Blesa, J., Soldevila, A., and Puig, V. (2019). Leak localization in water distribution networks using deep learning. In *2019 6th International Conference on Control, Decision and Information Technologies (CoDIT)*, pages 1426–1431.

Kang, J., Park, Y.-J., Lee, J., Wang, S.-H., and Eom, D.-S. (2017). Novel leakage detection by ensemble cnn-svm and graph-based localization in water distribution systems. *IEEE Transactions on Industrial Electronics*, PP:1–1.

Kingma, D. P. and Ba, J. (2017). Adam: A method for stochastic optimization.

Li, Q., Han, Z., and Wu, X.-M. (2018). Deeper insights into graph convolutional networks for semi-supervised learning.

Örn Gararsson, G., Boem, F., and Toni, L. (2022). Graph-based learning for leak detection and localisation in water distribution networks∗. $IFAC - PapersOnLine, 55(6) : 661 - -666$.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Rusch, T. K., Bronstein, M. M., and Mishra, S. (2023). A survey on oversmoothing in graph neural networks.

Shukla, H. and Piratla, K. (2020). Leakage detection in water pipelines using supervised classification of acceleration signals. *Automation in Construction*, 117:103256.

Soldevila, A., Blesa, J., Tornil-Sin, S., Duviella, E., Fernandez-Canti, R., and Puig, V. (2016). Leak localization in water distribution networks using a mixed model-based/data-driven approach. *Control Engineering Practice*, 55:162–173.

Teck Kai, C., Chin, C. S., and Zhong, X. (2018). Review of current technologies and proposed intelligent methodologies for water distributed network leakage detection. *IEEE Access*, PP:1–1.

Vrachimis, S., Timotheou, S., Eliades, D., and Polycarpou, M. (2021). Leakage detection and localization in water distribution systems: A model invalidation approach. *Control Engineering Practice*, 110:104755.

Vrachimis, S. G., Eliades, D. G., Taormina, R., Kapelan, Z., Ostfeld, A., Liu, S., Kyriakou, M., Pavlou, P., Qiu, M., and Polycarpou, M. M. (2022). Battle of the leakage detection and isolation methods. *Journal of Water Resources Planning and Management*, 148(12):04022068.

Wang, X. and Zhang, M. (2022). How powerful are spectral graph neural networks.

A Appendix

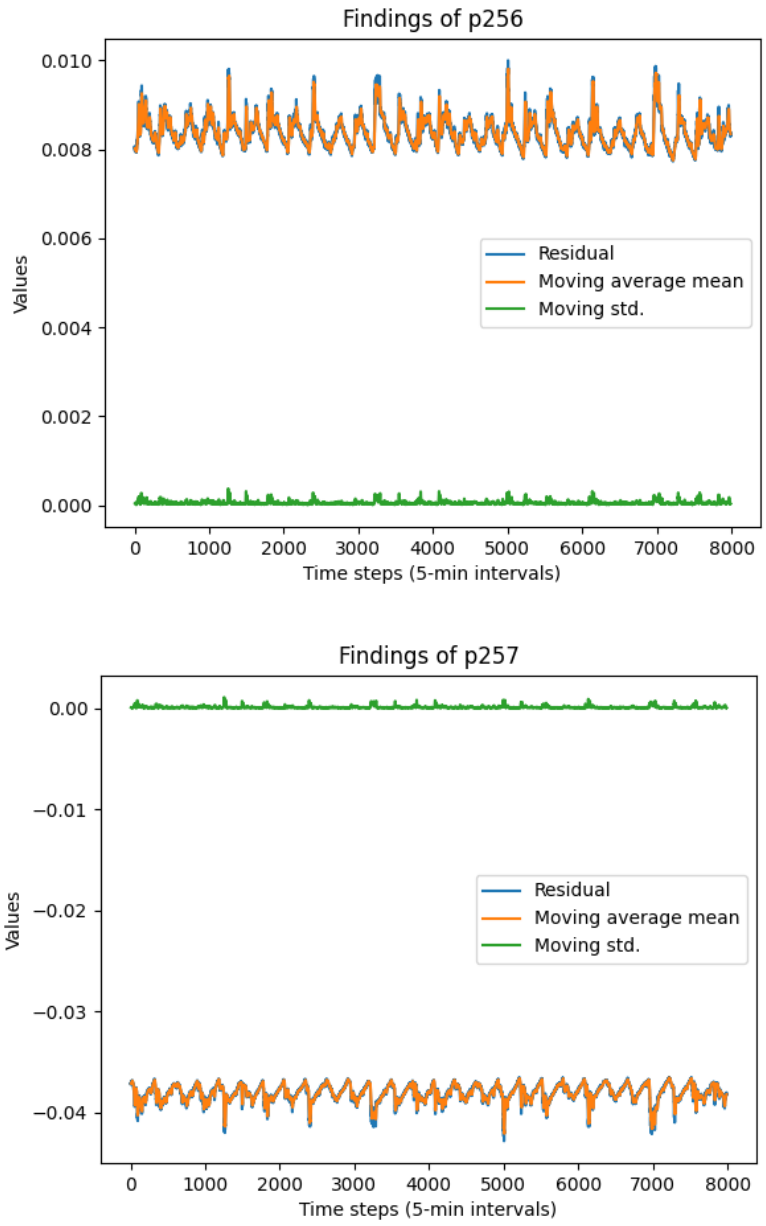

Figure 7: Residual signal, moving average, and moving standard deviation of both pipe 256 and 257

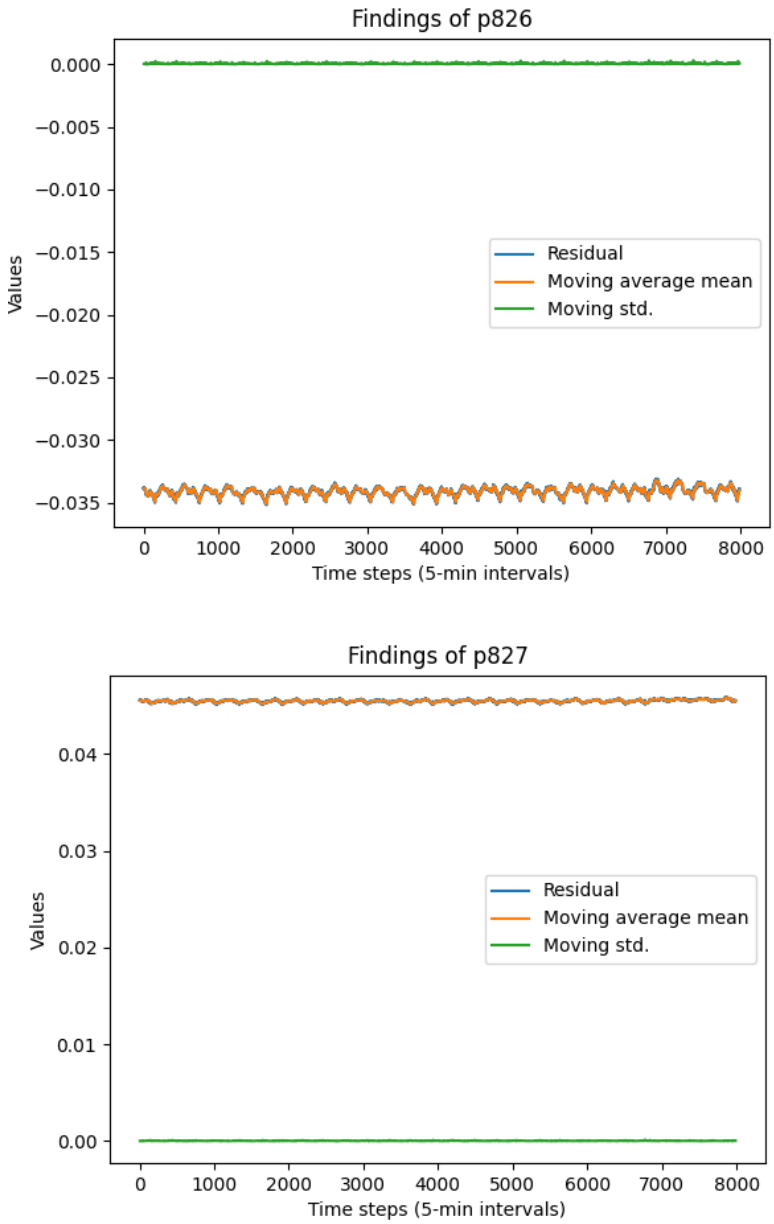

Figure 8: Residual signal, moving average, and moving standard deviation of both pipe 826 and 827