Deep Learning for Real-Time Inverse Problems and Data Assimilation with Uncertainty Quantification for Digital Twins

Machinaal leren voor real-time inverse problemen en data-assimilatie met onzekerheidskwantificatie voor digitale tweelingen

(met een samenvatting in het Nederlands)

Proefschrift

ter verkrijging van de graad van doctor aan de
Universiteit Utrecht
op gezag van de
rector magnificus, prof. dr. H.R.B.M. Kummeling,
ingevolge het besluit van het college voor promoties
in het openbaar te verdedigen op

donderdag 27 juni 2024 des morgens te 10:15 uur

door

Nikolaj Takata Mücke

geboren op 18 Maart 1993
te Greve, Denmark

Promotoren:

Prof. dr. ir. C.W. Oosterlee
Prof. dr. S.M. Bohté

Beoordelingscommissie:

Prof. dr. D. Crommelin
Prof. dr. ir. J.E. Frank
Prof. dr. T. van Leeuwen
Prof. dr. C. Pain PhD
Dr. ir. F.C. Vossepoel

SUMMARY

We develop novel approaches to enhance the functionality and efficiency of digital twins through deep learning techniques. Digital twins, sophisticated virtual models of physical systems, serve as dynamic counterparts, mirroring real-world entities' behavior and performance. Their primary role includes real-time monitoring, flaw detection, automated control, and proactive maintenance. To perform such tasks it is necessary to use various methods from scientific computing to combine data with physics models. Furthermore, to assess the reliability of a digital twin, it is important to not only make predictions, but also quantify the uncertainty of estimations and predictions. In this regard, Bayesian methods serve as a natural framework to pose the problems. However, the complexity and real-time operation requirements pose significant computational challenges. The thesis explores the integration of deep learning in scientific computing for enabling digital twins. The focus is on overcoming the computational hurdles in real-time data assimilation and inverse problem-solving through surrogate modeling. Deep learning, with its capability to handle high-dimensional functions in both deterministic and stochastic settings is shown to be a viable solution to these challenges.

This work investigates several key areas, beginning with a brief introduction to digital twins, data assimilation, and inverse problems. It proceeds to elaborate on the role of Bayesian inversion problems in static and dynamic contexts, highlighting the importance of uncertainty quantification. The focus is on the challenges of real-time computation and the potential of Bayesian methods, despite their computational intensity. The thesis investigates deep learning architectures like Dense and Residual Neural Networks, Convolutional Neural Networks, Recurrent Neural Networks and Transformers.

The core contributions are presented in the four main chapters of the thesis:

1) Reduced Order Modeling for Parameterized Time-Dependent Partial Differential Equations: Here, we focus on the forward problem. We introduce a novel deep learning framework for the numerical solution of parameterized partial differential equations (PDEs), focusing on spatially and memory-aware neural networks. We discuss strategies to achieve long term predictions of the time dependent solution. The approach is demonstrated on a linear advection equation and incompressible Navier-Stokes equations.

2) Markov Chain Generative Adversarial Neural Networks: Here, we aim to enhance Bayesian inversion for static problems with sparse observations. We present a unique algorithm using Markov Chain Monte Carlo methods and Generative Adverserial Networks (GANs) for solving Bayesian inverse problems in physics applications. A GAN is trained to sample from the prior distribution and is then embedded into a Hamiltonian Monte Carlo procedure for sampling from the posterior. By using a GAN to generate samples from the the prior distribution, the posterior can be formulated in a latent space, significantly lowering the dimension of the problem. Furthermore, sampling then becomes computationally cheap as it only requires evaluating the GAN generator. We prove that sampling in the latent space is equivalent to sampling directly in the high-fidelity space in a weak sense, as well as convergence of the posterior in the Wasserstein-1 metric. The method is demonstrated on a Darcy flow problem and leakage detection in pipe flow.

3) Probabilistic Digital Twin for Leak Localization: In this chapter, we look into the specific problem of leak localization in water distribution networks. A probabilistic framework for leak localization is developed, utilizing generative deep learning in form of supervised Wasserstein Autoencoders. The autoencoder consists of transformer neural networks that are trained to generate the flow rates and head losses conditional on a leak location and size. In the online stage, the autoencoder is used to speed up a Bayesian inference procedure, resulting in a posterior over all possible discrete leak locations. The approach is tested on several network models that are commonly employed in the literature.

4) The Deep Latent Space Particle Filter: The final contribution involves a real-time nonlinear data assimilation method for dynamic PDEs with uncertainty quantification. We utilize the findings from the previous chapter to develop a method that assimilates data online, by introducing a transformer-based dimensionality reduction and time stepping in the context of particle filters. We present various regularization techniques to ensure that the latent space has desirable properties. The resulting methodology, the Deep Latent Space Particle Filter (D-LSPF), is showcased on the viscous Burgers' equation, harmonic wave generation over a submerged bar, and leak localization in multi-phase pipe flow. In all cases observations are sparse in both time and space.

The thesis offers comprehensive and novel contributions to the field of digital twins, focusing on solving, real-time, inverse problems and data assimilation through advanced deep learning methods. The methodologies proposed not only address the existing computational limitations but also open new avenues for the practical application of digital twins in various industries.

SAMENVATTING

We ontwikkelen nieuwe benaderingen om de functionaliteit en efficiëntie van zogenaamde ‘digital twins’, of ‘digitale tweelingen’ te verbeteren door middel van deep learning-technieken. Digitale tweelingen zijn geavanceerde virtuele modellen van fysieke systemen. Ze fungeren als dynamische tegenhangers die het gedrag en de prestaties van echte systemen reflecteren, en kunnen worden gebruikt voor bijvoorbeeld realtime monitoring, foutdetectie, geautomatiseerde besturing en proactief onderhoud. Om dergelijke digitale tweelingen mogelijk te maken, is het noodzakelijk om verschillende methoden uit de wetenschappelijke computing te gebruiken om gegevens met fysica-modellen te combineren; om de betrouwbaarheid van een digitale tweeling te beoordelen is het verder belangrijk om niet alleen voorspellingen te doen, maar ook de onzekerheid van schattingen en voorspellingen te kwantificeren. Hier zijn Bayesiaanse methoden een natuurlijk kader. Echter, de complexiteit en de eisen aan realtime operaties vormen een aanzienlijke computationele uitdaging. Deze thesis verkent de integratie van deep learning in wetenschappelijke computing om digitale tweelingen mogelijk te maken. De focus ligt op het oplossen en beheersbaar maken van de computationele hindernissen bij realtime data-assimilatie en het oplossen van inverse problemen via surrogaatmodellering. Deep learning, met het vermogen om hoog-dimensionale functies in zowel deterministische als stochastische settings te hanteren, wordt ontwikkeld tot werkbare oplossingen voor deze uitdagingen.

Dit werk onderzoekt verschillende sleutelgebieden, beginnend met een korte introductie tot digitale tweelingen, data-assimilatie en inverse problemen. Vervolgens wordt uitgebreid ingegaan op de rol van Bayesiaanse inversieproblemen in statische en dynamische contexten, met nadruk op het belang van onzekerheidskwantificering. De focus ligt op de uitdagingen van realtime berekening en het potentieel van Bayesiaanse methoden, ondanks hun computationele intensiteit. De thesis onderzoekt deep learning-architecturen zoals Dense en Residual Neural Networks, Convolutional Neural Networks, Recurrent Neural Networks en Transformers.

De voornaamste bijdragen worden gepresenteerd in de vier primaire hoofdstukken van de thesis:

1) Gereduceerde Orde Modellering voor Geparametriseerde Tijdafhankelijke Partiële Differentiaalvergelijkingen: Hier richten we ons op het forward problem. We introduceren een nieuw deep learning-framework voor de numerieke oplossing van geparametriseerde partiële differentiaalvergelijkingen (PDE's), met focus op ruimtelijk en geheugenbewuste neurale netwerken. We bespreken strategieën om langetermijnvoorspellingen van de tijdafhankelijke oplossing te bereiken. De aanpak wordt gedemonstreerd aan de hand van een lineair advectie probleem.

2) Markov Chain Generatieve Adversariale Neurale Netwerken: Hier is ons doel om Bayesiaanse inversie voor statische problemen met spaarzame observaties te verbeteren. We presenteren een algoritme met behulp van Markov Chain Monte Carlo-methoden en Generatieve Adversariale Netwerken (GAN's) voor het oplossen van Bayesiaanse inverse problemen in fysica-toepassingen. Een GAN wordt getraind om te sampelen uit de prior-verdeling en vervolgens ingebed in een Hamiltonian Monte Carlo-procedure voor het sampelen uit de posterior. Door een GAN te gebruiken om samples uit de prior-verdeling te genereren, kan de posterior worden geformuleerd in een latente ruimte, wat de dimensie van het probleem aanzienlijk verlaagt. Bovendien wordt het sampelen computationeel goedkoop omdat het alleen het evalueren van de GAN-generator vereist. We bewijzen dat sampelen in de latente ruimte equivalent is aan direct sampelen in de hoog-dimensionele ruimte in een zwakke zin, evenals convergentie van de posterior in de Wasserstein-1-metriek. De methode wordt gedemonstreerd op een Darcy-stromingsprobleem en lekdetectie in pijpstroom.

3) Probabilistische Digitale Tweeling voor Leklokalisatie: In dit hoofdstuk bestuderen we het specifieke probleem van leklokalisatie in waterdistributienetwerken. Een probabilistisch kader voor leklokalisatie wordt ontwikkeld, gebruikmakend van generatief diep leren in de vorm van Wasserstein Autoencoders. De autoencoder bestaat uit transformer neurale netwerken die getraind zijn om de debieten en drukverliezen te genereren, conditioneel op een leklocatie en -grootte. In de online fase wordt de autoencoder gebruikt om een Bayesiaanse inferentieprocedure te versnellen, resulterend in een posterior over alle mogelijke discrete leklocaties. De aanpak wordt getest op verschillende netwerkmodellen die vaak worden gebruikt in de literatuur.

4) De Diepe Latente Ruimte Partikelfilter: De laatste bijdrage betreft een real-time niet-lineaire data-assimilatiemethode voor dynamische PDE's met onzekerheidskwantificering. We gebruiken de bevindingen uit het vorige hoofdstuk om een methode te ontwikkelen die gegevens online assimileert, door een op transformers gebaseerde dimensionaliteitsreductie en tijdstappen in de context van particle filters te introduceren. We presenteren verschillende regularisatietechnieken om ervoor te zorgen dat de latente ruimte gewenste eigenschappen

heeft. De resulterende methodologie, de Diepe Latent Space Particle filter (D-LSPF), wordt gedemonstreerd op de viskeuze Burgers-vergelijking, harmonische golfgeneratie over een ondergedompelde bar, en leklokalisatie in meerfasige pijpstroom. In alle gevallen zijn de waarnemingen spaarzaam, zowel in tijd als ruimte.

Deze thesis biedt uitgebreide en vernieuwende bijdragen aan het veld van digitale tweelingen, gericht op het oplossen van realtime inverse problemen en data-assimilatie door middel van geavanceerde deep learning-methoden. De voorgestelde methodologieën pakken niet alleen de bestaande computationele beperkingen aan, maar openen ook nieuwe mogelijkheden voor de praktische toepassing van digitale tweelingen in verschillende industrieën.

CONTENTS

Summary iii

Samenvatting v

1 Introduction 1

1.1 Digital Twins . . . 1

1.2 Data Assimilation and Inverse Problems . . . 2

1.2.1 Bayesian Inverse Problems . . . 3

1.2.2 Static and Dynamic Inverse problems . . . 4

1.3 Deep Learning in Scientific Computing . . . 7

1.3.1 Neural Network Architectures . . . 7

1.3.2 Deep Learning-Based Surrogate Models . . . 12

1.4 Structure of Thesis . . . 15

2 Reduced order modeling for parameterized time-dependent PDEs using spatially- and memory-aware deep learning 17

2.1 Introduction . . . 18

2.2 Parameterized Time-Dependent PDEs . . . 20

2.3 Dimensionality Reduction . . . 21

2.3.1 Linear Dimensionality Reduction . . . 23

2.3.2 Nonlinear Dimensionality Reduction . . . 24

2.4 Approximating Parameterized Time Evolution using Neural Networks . . . 27

2.4.1 The Complete Scheme . . . 32

2.5 Results . . . 32

2.5.1 Linear Advection Equation . . . 37

2.5.2 2D Nonlinear Equation - Flow Past Cylinder . . . 41

2.5.3 Computation Time and Accuracy . . . 43

2.6 Conclusion . . . 47

3 Markov chain generative adversarial neural networks for solving Bayesian inverse problems in physics applications 49
3.1 Introduction 50
3.2 Notation, Problem Setting, and Preliminaries 52
  3.2.1 Problem Setting 52
  3.2.2 Markov Chain Monte Carlo Methods 54
  3.2.3 Generative Adversarial Neural Networks 56
3.3 Markov Chain GAN 57
  3.3.1 Proposed Algorithm 58
  3.3.2 Latent Space Sampling 61
  3.3.3 Convergence of Generated Posterior 63
  3.3.4 Convergence of the Generated Prior 68
3.4 Alternative Methods 68
  3.4.1 Ensemble Kalman Filter 68
  3.4.2 Surrogate Models 69
  3.4.3 Likelihood-Free methods 70
3.5 Results 71
  3.5.1 Darcy Flow 74
  3.5.2 Leakage Detection in Pipe Flow 78
  3.5.3 Summary of Results 83
3.6 Conclusion 86

4 A Probabilistic Digital Twin for Leak Localization in Water Distribution Networks Using Generative Deep Learning 89
4.1 Introduction 90
  4.1.1 Leak localization literature 91
  4.1.2 Bayesian inference 91
  4.1.3 Computational bottleneck 92
  4.1.4 Overview chapter 92
4.2 Problem Setting and Preliminaries 93
  4.2.1 Problem Setting 93
  4.2.2 Leak Localization as a Bayesian Inverse Problems 95
  4.2.3 Supervised Wasserstein Autoencoder 97
4.3 Proposed Framework 100
  4.3.1 Stopping Criterion in the Online Stage 102
  4.3.2 Estimating Uncertainty 103
  4.3.3 Model Architectures 103
4.4 Results 104

4.4.1 Classification Model . . . 105
4.4.2 Test Cases . . . 106
4.4.3 Test WDN1: Hanoi Network . . . 110
4.4.4 Test WDN2 . . . 111
4.4.5 Test WDN3: Modena . . . 112
4.4.6 Comparison with Baseline . . . 112
4.5 Conclusion . . . 114

5 The Deep Latent Space Particle Filter for Real-Time Nonlinear Data Assimilation with Uncertainty Quantification 117
5.1 Introduction . . . 118
5.2 Problem Setting . . . 120
5.2.1 Data Assimilation . . . 121
5.2.2 Particle Filter . . . 121
5.3 Deep Latent Space Particle Filter . . . 122
5.3.1 Latent Space Regularized Autoencoder . . . 124
5.3.2 Transformer-Based Dimensionality Reduction . . . 125
5.3.3 Time Stepping . . . 126
5.4 Results . . . 128
5.4.1 Viscous Burgers Equation . . . 129
5.4.2 Harmonic wave generation over a submerged bar . . . 130
5.4.3 Multi-phase leak localization . . . 134
5.5 Conclusion . . . 140

6 Conclusion 143
6.1 Chapter Conclusions . . . 143
6.2 Outlook and Future Work . . . 144

Bibliography 147

Curriculum Vitae 161

List of Publications 163

List of Presentations 165

Acknowledgements 167

1

INTRODUCTION

1.1 Digital Twins

A digital twin is a sophisticated virtual model of a physical object or system [127]. It acts as a dynamic digital counterpart, reflecting the real-world entity's behavior, characteristics, and performance. In the realm of fluid dynamics, for example, a digital twin must adeptly handle intricate phenomena like turbulence and fluid-structure interactions. This is achieved by integrating fluid flow equation solutions with data from diverse sources, including IoT devices and sensors, leading to a multifaceted interplay of multiple mathematical and computational models. These models span various physical phenomena and scales, contributing to the digital twin's complexity.

The practical applications of a fully functional digital twin are manifold. Its primary role is to facilitate a deeper understanding and enhanced utilization of the physical asset. By mirroring a physical entity digitally, one can engage in real-time monitoring to detect flaws, implement automated control through closed-loop optimization, and explore 'what-if' scenarios for proactive maintenance. Essentially, the digital twin operates as a versatile simulation platform or a 'sandbox.'

Digital twins stand out from traditional high-fidelity models by being accurate representations of the physical asset throughout its entire lifecycle. Unlike a conventional simulation focused on a specific process, a digital twin comprises various interacting models, enabling the simulation of multiple processes simultaneously. Moreover, whereas a conventional simulation is designed for a specific problem, a digital twin continuously adapts and calibrates itself based on incoming data streams [6, 79]. The aim of a digital twin is comprehensive real-time mirroring, as opposed to a conventional simulation's focus on particular processes of interest.

Nevertheless, the complexity and the need for real-time operation make digital twins challenging to implement and use. The substantial computational requirements for real-time calibration, prediction, and control present a significant hurdle. Additionally, the need for reliable output necessitates uncertainty quantification, making the computational burden even larger. This thesis proposes to address these challenges by leveraging deep learning for efficient real-time data assimilation, enhancing the practicality and performance of digital twins.

1.2 Data Assimilation and Inverse Problems

This section offers a concise introduction to data assimilation and inverse problems, focusing on delivering an intuitive grasp of these concepts without going into the deep theoretical aspects such as function spaces and advanced theoretical results. These more complex considerations are reserved for later chapters where they are discussed in detail as required.

At its core, data assimilation and inverse problems are concerned with deducing the underlying parameters and/or states that have generated a particular set of observations [33, 78]. This process is commonly divided into two categories: state estimation, which is the deduction of the state from a limited set of observations, and parameter estimation or calibration, which is the inference of parameters.

In defining an inverse problem, we begin by identifying three essential elements: the state, denoted as $q(\mu)$, the parameters, represented by μ, and the observations, symbolized as y. It's important to recognize that the state typically depends on the parameters. For example, in fluid dynamics, the state might comprise velocity and pressure fields, the parameters could include the Reynolds number, but also boundary condition values or the location of a source, and the observations might be pressure readings from a limited number of sensors.

Throughout this thesis, we adopt specific notation for clarity: Capital letters represent stochastic variables, $P(X = x)$ denotes the probability of X equaling x, and ρ symbolizes the probability density function (PDF) associated with the distribution P. We assume the existence of a PDF for all discussed distributions.

The relationship between the state and the observations is established through the observation operator, H, expressed as follows:

$$y = H(q(\mu)) + \eta, \quad \eta \sim P_\eta(\eta) \tag{1.1}$$

Here, η represents the noise in the observations. The observation operator essentially maps the complete state to specific observation points, such as sensor locations. The inverse problem, therefore, involves calculating q and/or μ based solely on the available observations y. In this context, $q(\mu)$ represents an idealized true state, which, in reality, is unattainable. However, it is usually assumed that q adheres to a physical process that can be described by a partial differential equation (PDE), referred to as the forward problem:

$$F(q(\mu), \mu) = 0. \tag{1.2}$$

It is important to note that this formulation pertains to static problems. In the context of dynamic phenomena, the forward problem is given by:

$$\partial_t q(t, \mu) = F(q(t, \mu), \mu, t), \tag{1.3}$$

with the corresponding observations being:

$$y(t) = H(q(t, \mu)) + \eta(t). \tag{1.4}$$

Practically, one often considers the discrete-time version of this problem:

$$q_n(\mu) = F_n(q_{n-1}(\mu), \mu), \tag{1.5}$$

$$y_n = H(q_n(\mu)) + \eta_n. \tag{1.6}$$

Where $q_n(\mu) = q(t_n, \mu)$ for a time discretization, $t_0, \ldots, t_N$. When F represents a PDE, space discretization is further applied to transform this into a set of algebraic equations.

1.2.1 Bayesian Inverse Problems

There are two primary methods for solving an inverse problem: the variational approach and the Bayesian approach [33, 147]. The variational approach involves formulating an optimization problem that includes an objective function and the forward problem. This objective function is designed to quantify the discrepancy between actual and simulated observations. Solving this optimization problem directly yields the state and/or parameters, using the forward map as either a regularization term or, alternatively, as a hard constraint. Conversely, the Bayesian approach treats the problem as an estimation of the posterior distribution. It utilizes the forward problem and observations to calculate a likelihood, which is then combined with a prior distribution over the states and/or parameters.

The variational approach is typically faster and simpler to implement but lacks uncertainty quantification. It also necessitates a differentiable solver for the forward problem or the derivation of adjoint equations. The Bayesian approach, however, provides a comprehensive view of the state and parameters through the posterior distribution, inherently quantifying uncertainty. This method incurs higher computational costs; it either requires pre-selecting a family of distributions or employing sampling techniques like Monte Carlo sampling. While sampling methods are powerful in approximating any distribution, making them suitable for nonlinear, complex problems, they are computationally intensive. Each sample necessitates solving the forward problem, often requiring numerous samples for accuracy. This thesis focuses exclusively on Bayesian methods, prioritizing the quantification of uncertainty in solutions.

1.2.2 Static and Dynamic Inverse problems

Static and dynamic inverse problems differ in their temporal aspects. Static inverse problems seek to determine the posterior distribution of a system's state and/or parameters when there is no temporal evolution. In contrast, dynamic inverse problems involve identifying the state and/or parameters at various time points as new data becomes available.

Static Inverse Problems

We first introduce the concept of the augmented state, denoted as $u = (q, \mu)$. This augmented state contains both the state and the parameters, simplifying our notation.

The posterior PDF of the augmented state, $\rho_{u|y}(u|y)$, is derived from Bayes' theorem and is expressed as:

$$\rho_{u|y}(u|y) = \frac{\rho_{y|u}(y|u)\rho_0(u)}{\int \rho_{y|u}(y|u)\rho_0(u)\mathrm{d}u}, \tag{1.7}$$

where $\rho_{u|y}$ represents the posterior, $\rho_{y|u}(y|u)$ the likelihood, $\rho_0(u)$ the prior, and the denominator signifies the evidence. The likelihood, specifically, models the probability of observing y given a particular state, u. As derived from the earlier general inverse problem formulation, it can be reformulated as,

$$\rho_{y|u}(y|u) = \rho_\eta(y - H(u)). \tag{1.8}$$

Here, the likelihood essentially quantifies the residual between the forward model's output and the observations, considering the observation noise. When given observations y, the likelihood is often considered a function of u, namely $\Phi(u) = \rho_\eta(y - H(u))$, at which point it ceases to be a PDF.

The prior PDF encapsulates our initial beliefs about the system, generally informed by previous experiments or knowledge about the system. Selecting an appropriate prior is

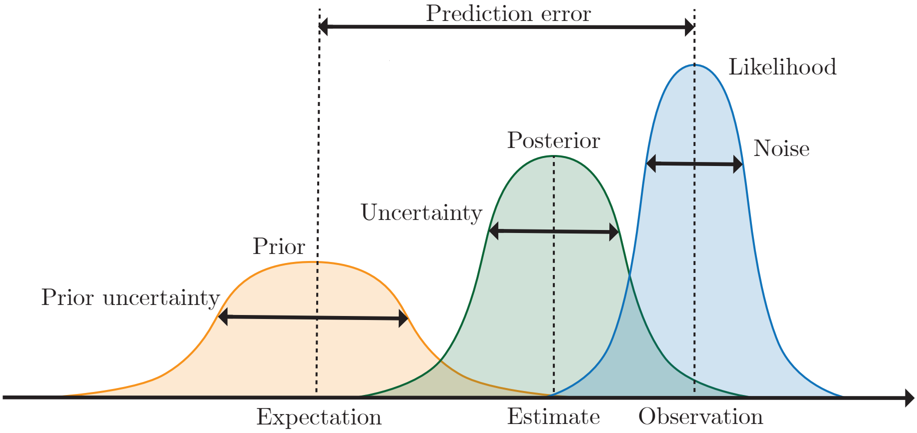


Figure 1.1: Illustration of the distributions in Bayesian inference.

critical, as an inadequate choice can severely affect the posterior.

The evidence is the unconditional probability of the observed data and serves the role of normalizing the posterior to ensure it integrates to 1. However, it is often overlooked in many algorithms since it doesn't directly influence the parameter of interest.

The interplay between the posterior, prior, and likelihood is visualized in Figure 1.1.

Computing the posterior as per (1.7) is a complex task. As previously noted, the true posterior is typically approximated through sampling methods. These methods aim to sample from the posterior without calculating it directly. To accurately represent the entire distribution, a large number of samples is needed. The computation of the posterior (which is the product of the compute-intensive prior and the likelihood) is particularly resource-intensive, as it necessitates solving the forward problem, making real-time computation challenging. Significant research efforts focus on enhancing the efficiency of sampling, by reducing the number of samples required. Techniques such as Markov Chain Monte Carlo (MCMC) and its variations, including Hamiltonian Monte Carlo and Langevin Monte Carlo, are developed to minimize wastage of resources on unlikely sampling regions in the augmented state space (refer to Section 3.2 for more on these methods). Even with these advanced methods, it's common to require between $10^3 - 10^6$ samples to accurately depict the posterior.

In Chapters 3 and 4, we explore and demonstrate techniques to expedite this process using generative deep learning approaches.

Dynamic Inverse Problems

Dynamic inverse problems, unlike their static counterparts, incorporate the added dimension of time, often being synonymous with data assimilation in many contexts.

Similar to static Bayesian inverse problems, our goal is to compute a distribution of the augmented state based on the observations. In the dynamic scenario, however, the posterior distribution is over full trajectories of states in time. To express this, we introduce the notation $u_{0:n} = u_0, \dots, u_n$. Applying Bayes' theorem, the posterior is then given by:

$$\rho(u_{0:n}|y_{0:n}) = \frac{\rho(y_{0:n}|u_{0:n})\rho(u_{0:n})}{\int \rho(y_{0:n}|u_{0:n})\rho(u_{0:n})\mathrm{d}u_{0:n}}, \tag{1.9}$$

where

$$\rho(y_{0:n}|u_{0:n}) = \prod_{i=0}^{n} \rho(y_i|u_i), \qquad \rho(u_{0:n}) = \rho(u_0)\prod_{i=1}^{n} \rho(u_i|u_{i-1}). \tag{1.10}$$

Rather than calculating the posterior over the entire trajectory, we focus on estimating the posterior filtering distribution at each time step:

$$\rho(u_n|y_{0:n}) = \frac{\rho(y_n|u_n)\rho(u_n|y_{0:n-1})}{\int \rho(y_n|u_n)\rho(u_n|y_{0:n-1})\mathrm{d}u_n}, \tag{1.11}$$

where

$$\rho(u_n|y_{0:n-1}) = \int \rho(u_n|u_{n-1})\rho(u_{n-1}|y_{0:n-1})\mathrm{d}u_{n-1}. \tag{1.12}$$

This implies that the posterior distribution at a given time step, n, can be calculated using the posterior from the previous time step to establish the current prior. Repeatedly applying this process yields a series of posterior distributions.

The computation of $\rho(u_n|u_{n-1})$ involves the forward problem, F_n, as shown in Eq. (1.5). This forward problem is often modified with stochastic perturbations to account for model errors (see Section 5.2 for details).

As with static cases, the filtering distribution, $\rho(u_n|y_{0:n})$, is typically not possible to compute analytically, necessitating numerical methods. Challenges arise particularly when dealing with high-dimensional states and/or parameters and complex forward problems characterized by nonlinearities or discontinuities. In such instances, accurately representing $\rho(u_n|u_{n-1})$ becomes computationally prohibitive, especially for real-time applications. Certain methods, like the Kalman filter, simplify the complexity by assuming Gaussian distributions, focusing on calculating the mean and variance of the posterior. However, the Kalman filter is limited to linear forward problems. The ensemble Kalman filter extends this approach to weakly nonlinear problems, still under the assumption of Gaussian distributions. While computationally efficient, it struggles with highly nonlinear and non-Gaussian problems. This is where more general methods, such as particle filters, become necessary. Particle filters do not impose constraints on the forward problem or the distributions, but require a significantly higher computational effort due to the increased number of samples needed for propagation over time (more details in Section 5.2).

In Chapter 5, we introduce the Deep Latent Space Particle Filter, an innovative approach that leverages neural networks to enhance the efficiency of particle filtering.

1.3 Deep Learning in Scientific Computing

For digital twins to operate effectively and reliably, they must be capable of executing Bayesian inversion in real-time, as Bayesian inversion is the preferred method for estimating an unknown distribution of any type. However, as previously mentioned, this can be highly challenging due to the extensive computations required, especially in solving the forward problem multiple times for high-dimensional, nonlinear problems. Consequently, the integration of deep learning into scientific computing has become an increasingly common practice [17, 43, 87].

Deep learning focuses on modeling with deep Artificial Neural Networks (ANNs), commonly referred to simply as neural networks (NNs). These networks map inputs to outputs through a sequence of affine transformations and nonlinear activation functions, drawing inspiration from the human brain's complex neural network [50].

NNs offer several beneficial properties for scientific computing. They are universal approximators [71], implying that, given sufficient training data and network depth, they can approximate theoretically any function. They excel at handling high-dimensional functions and, due to their architecture, calculating gradients is computationally efficient through a process known as backpropagation [133].

As data-driven models, neural networks are trained on a collection of examples, known as training data. During training, the network adjusts its weights to minimize a loss function, gradually improving its performance [50].

It's crucial to recognize that neural networks form a vast and varied class of functions, with their primary unifying feature being the multi-layered processing of data. Various types of layers exist within neural networks, including dense, convolutional, recurrent, and transformer layers, each with its own set of strengths and weaknesses [50]. The subsequent section will introduce and detail the kinds of layers relevant to this thesis.

1.3.1 Neural Network Architectures

Dense Neural Networks

The arguably most common ANN architecture is the dense neural network (DNN). A DNN can be considered a function, $G : \mathbb{R}^{N_i} \to \mathbb{R}^{N_o}$, consisting of a series of affine transformations,

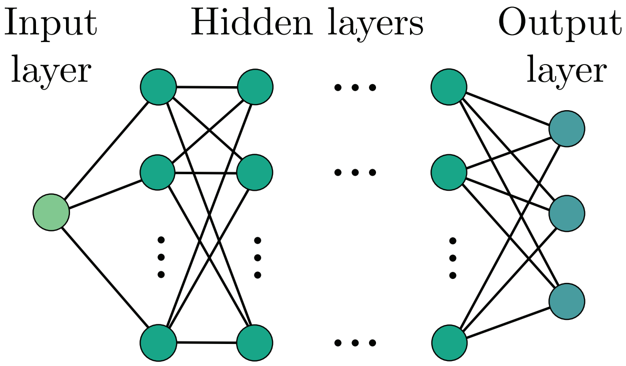


Figure 1.2: Visualization of a feedforward densely connected neural network.

T_i, followed by an element-wise (nonlinear) activation function, σ_i:

$$G(x;\theta) = \sigma_L \circ T_L \circ \ldots \circ \sigma_1 \circ T_1(x). \tag{1.13}$$

The combination of an afine transformation followed by the activation is called a neuron. The afine transformation can be written as $T_i(x) = W_i x + b_i$, where $W \in \mathbb{R}^{M_i \times M_{i-1}}$ and $b \in \mathbb{R}^{M_i}$. We call W_i the weight matrix, b_i the bias vector, and M_i the number of neurons in layer i, and L the number of layers. (1.13) is conveniently visualized as a network of neurons. We will refer to the set of parameters as $\theta = \{W_1, b_1, \ldots, W_L, b_L\}$.

Residual Neural Networks

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

$$x^{i+1} = \mathcal{F}^i(x^i) + x^i, \tag{1.14}$$

where 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.

ResNets were developed to deal with the so-called vanishing gradient problem. The vanishing gradient problem occurs when the gradients of the NNs weights become very small as they are propagated back through the network. This is particularly problematic in networks with many layers (deep networks). The advantage of using ResNets is that the problem of vanishing gradients is not very apparent, which makes them easier to train [63].

Convolutional Neural Networks

Convolutional neural networks (CNNs) gained attention due to their great performance in image recognition. The general principle is to utilize local properties of the data instead of only considering global properties. This is done by having local connections and shared weights in the neural networks. These properties are not only important for detecting patterns in data but it also enable computations on very high-dimensional data.

A convolutional layer is effectively a feature map where each unit in the layer is connected to a local patch of the previous layer through a filter bank and an activation function. A feature map at layer l is a tensor, $H^l \in \mathbb{R}^{N^l_{chan} \times N^l_1 \times N^l_2}$, where $H^l_{i,j,k}$ is a unit at channel i, row j, and column k. The filter bank at layer l is a 4-dimensional tensor, $F^l \in \mathbb{R}^{N^l_{filter} \times N^{l-1}_{chan} \times k_1 \times k_2}$, where $F^l_{i,j,m,n}$ connects a unit in channel i of the output and channel j of the input with m and n being the offset of rows and columns respectively. N^l_{filter} denotes the number of filters in the feature bank in layer l and k_1 and k_2 denotes the kernel size. The convolution operation between a feature map and a filter bank is given by

$$H^l_{i,j,k} = \sigma_l \left(\sum_{r=1}^{N^{l-1}_{chan}} \sum_{m=1}^{N^{l-1}_1} \sum_{n=1}^{N^{l-1}_2} H^{l-1}_{r,(j-1)s+m,(k-1)s+n} F^l_{i,r,m,k} + B^l_{i,j,k} \right), \tag{1.15}$$

where $B^l_{i,j,k}$ is a bias term and σ_l is an activation function applied element-wise. s denotes the stride, effectively downsamples the feature map between layers. The filters, $F^l_{i,r,m,k}$, and biases, $B^l_{i,j,k}$, are the learnable parameters while the kernel sizes, k_1, k_2, the stride, s, and the number of filters, N^l_{filter}, are chosen. Often these are subject case specific objectives or hyperparameter optimization.

Causal Convolutional Neural Networks

As the name suggests, causal convolutional neural networks (CCNNs) are related to convolutional neural networks.

CCNNs are used for encoding time series data with the purpose of forecasting or classification [14, 110]. The approach is to use 1-dimensional convolutions on time series data. In the multivariate case, the multiple dimensions are interpreted as channels. The term causal refers to the fact that the filter banks are only convolved with the current and previous time steps, thus establishing a causal relationship between the past and the future.

Recurrent Neural Networks and Long Short-Term Memory

A recurrent neural network (RNN) is an alternative to a CCNN for interpreting time series data. In this thesis, we solely focus on a specific RNN called long short-term memory (LSTM) [68]. For an input consisting of several previous time steps, x^n, an LSTM layer consists of four components [49]: An input gate:

$$i^{n+1} = \sigma\left(W_i x^n + b_i\right), \tag{1.16}$$

a forget gate:

$$f^{n+1} = \sigma\left(W_f x^n + b_f\right), \tag{1.17}$$

an output gate:

$$o^{n+1} = \sigma\left(W_o x^n + b_o\right), \tag{1.18}$$

and a cell state

$$c^{n+1} = i \odot c^n + i^n \odot \tanh\left(W_c x^n + b_c\right). \tag{1.19}$$

The prediction is then given by

$$x^{n+1} = o^n \odot \tanh\left(c^n\right). \tag{1.20}$$

$W_i, b_i, W_f, b_f, W_o, b_o, W_c, b_c$ are the trainable weight matrices and bias vectors, and $\odot$ is the Hadamard product. Ideally, the input gate identifies what information to be passed to the cell state, the forget what to be dropped, and the output gate decides what to be passed to the final prediction.

Attention and Transformers

A transformer model, a sophisticated neural network architecture, excels in understanding context and semantics within sequential data, such as the arrangement of words in a sentence. By employing advanced techniques like attention mechanisms, these models analyze the interplay between different elements of the sequence, perceiving subtle dependencies and influences even across distances. This quality to capture relationships within sequential data makes transformer models particularly adept at unraveling the complexities of temporal dynamics, making them well-suited for tasks requiring a deep understanding of sequential events within time stepping for PDE-based physical models .

Transformers were introduced in 2017 [156] and rapidly 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 [29], protein structure prediction [76], and time series forecasting [29].

We will highlight the relevant features of the transformer architecture used in this thesis.

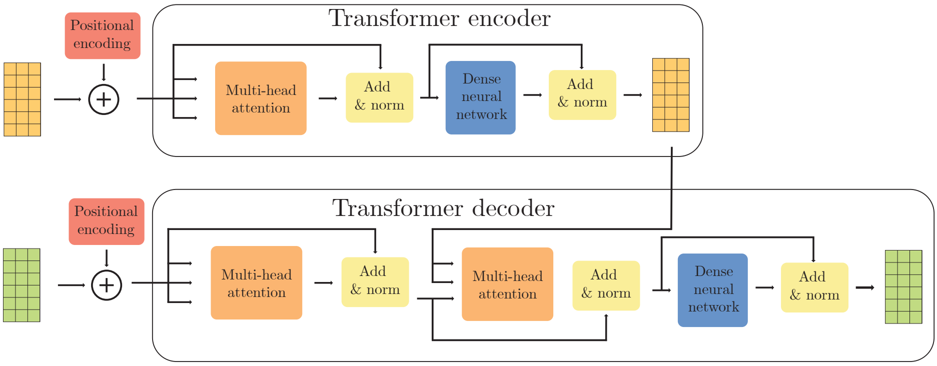


Figure 1.3: Transformer encoder and decoder architecture.

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:

$$K = \mathcal{F}_k(X) \in \mathbb{R}^{k\times d}, \quad Q = \mathcal{F}_q(X) \in \mathbb{R}^{k\times d}, \quad V = \mathcal{F}_v(X) \in \mathbb{R}^{k\times d}, \tag{1.21a}$$

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d}}\right)V, \tag{1.21b}$$

where K is referred to as 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 1.3 for a visualization.

In Equation (1.21), 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 by 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 1.3 for a visualization.

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

1.3.2 Deep Learning-Based Surrogate Models

With an understanding of relevant neural network (NN) layers established, we now turn our attention to the applications of NNs in scientific computing, particularly focusing on surrogate models. A surrogate model is essentially a replacement for a high-fidelity model. While it offers the advantage of faster evaluation, this comes at the cost of reduced accuracy and potentially extensive offline training.

The implementation of surrogate models typically involves two distinct stages: the offline and the online stages. During the offline stage, the surrogate model is defined and trained using relevant data. In scientific settings where actual data is scarce, this training data is often generated through simulation using high-fidelity models, making the offline stage resource-intensive. The online stage involves utilizing the trained surrogate model for real-time predictions or for integration into other computational tasks, such as control systems or data assimilation processes.

Surrogate models can be built through various approaches, primarily differentiated by whether they are intrusive or non-intrusive. Historically, intrusive methods were more prevalent, with techniques like proper orthogonal decomposition (POD) being widely employed. POD involves generating a set of high-fidelity data, denoted snapshots, which are then used to create an optimal low-dimensional basis via singular value decomposition. This basis is used to project the high-fidelity model onto a lower-dimensional space, reducing computational demands (refer to Section 2.3.1 for detailed discussion on POD). Although effective in many cases, POD struggles with highly nonlinear or hyperbolic problems due to the slow decay of the Kolmogorov N-width, making significant speed improvements difficult. Additionally, intrusive methods require access to the underlying PDE models, which can be restrictive if the source code is not available.

In contrast, non-intrusive surrogate models offer greater flexibility and potentially larger speed-ups during the online stage, as they operate purely on inputs and outputs without solving equations. However, they typically demand more training data and time. These challenges are mitigated to some extent by the fact that training occurs during the less time-sensitive offline stage. The learning task is more complex in non-intrusive models, as they must learn the entire input-output relationship without a model prior. This complexity is where NNs excel, capable of modeling intricate relationships far beyond traditional methods like polynomial chaos expansion.

Numerous examples of deep learning-based surrogate models in scientific computing exist. In [40, 160], neural networks are trained non-intrusively to map parameters and to solve a PDE at specific times. For time stepping, conventional architectures such as LSTMs and CNNs have been developed to non-intrusively approximate parametric time stepping [105, 166]. Here, the neural network surrogate advances the state in time, conditioned on the provided parameters. Recent work has also explored alternative neural network architectures for surrogate models, such as transformers, graph neural networks, and Fourier neural operators [43, 58, 91, 111].

Within this thesis, we explore NN applications in surrogate modeling across three key categories:

- **Dimensionality Reduction** – focusing on reducing the state dimensionality;
- **Time Stepping** – using NNs to advance the state in time;
- **Generative Modeling** – learning a distribution to enable efficient sampling.

Each category is outlined below and detailed in subsequent chapters.

Dimensionality Reduction

Dimensionality reduction is the process of reducing high-dimensional data into a low-dimensional representation. Traditional methods like Principal Component Analysis (PCA) [157] and Proper Orthogonal Decomposition (POD) [67, 123] have been standard approaches, but they often fall short with highly nonlinear and hyperbolic problems. To address these challenges, nonlinear methods, particularly those employing neural networks (NNs), have gained traction due to their ability to achieve more substantial dimensionality reductions.

A key technique in this area is the autoencoder (AE), a specialized NN framework designed for data-driven dimensionality reduction (refer to [81, 151] for foundational work). An AE is composed of two parts: an encoder that compresses the data into a lower-dimensional 'latent' state, and a decoder that reconstructs the original data from this latent representation. Detailed discussions on AEs are found in Section 2.3.2.

The construction of both the encoder and decoder in an AE can incorporate various types of NN layers, each chosen based on the specific nature of the data. For instance, convolutional layers are prevalently used in scientific computing due to their efficacy in processing spatially structured data, as demonstrated in Chapter 2. However, certain applications demand higher accuracy than convolutional layers can provide. To address this, we also explore the use of transformer layers, known for their powerful sequence modeling capabilities, in Chapters 4 and 5. These layers offer enhanced performance in applications where convolutional layers are insufficient, showcasing the versatility and adaptability of NNs in scientific computing.

Time Stepping

Time stepping is a critical task in computational modeling, involving the progression of a system's state over time. The objective here is to leverage a neural network (NN) to approximate the time advancement function, F_n, as shown in Eq. (1.5). Essentially, this means using an NN to predict the subsequent state, q_n, based on the current state, q_{n-1}. Adopting an NN for this purpose offers several potential advantages that can significantly speed up the process of time advancement.

One key benefit of using an NN for time stepping is the ability to take larger time steps. Unlike traditional methods that are often constrained by factors like the Courant–Friedrichs–Lewy (CFL) condition [89], NNs are not bound by such limitations, allowing for more flexibility in time incrementation. Additionally, NNs generally require a single evaluation to progress in time, contrasting with conventional time stepping approaches that necessitate multiple evaluations, whether through several stages in Runge-Kutta methods or solving a set of nonlinear equations in implicit methods.

In this thesis, our focus is on training NNs for time stepping within a reduced-dimensional space, achieved through dimensionality reduction via an AE. In such scenarios, non-intrusive methods become essential since deriving explicit equations for the latent state is not possible due to the nonlinear nature of the encoder. Consequently, the NN dedicated to time stepping learns this function solely from data.

Significant speed improvements are realized as the time stepping occurs in a substantially lower-dimensional space. This reduction not only accelerates the time stepping process itself but also facilitates efficiency in subsequent tasks, such as data assimilation. Operating these tasks within the latent space benefits from the reduced dimensionality, thereby enhancing overall computational efficiency and effectiveness.

Deep Generative Models

Deep generative models are a type of neural network that learns how to sample from complex distributions, which are typically difficult to sample from directly. These models achieve this by learning a transformation (known as a 'pushforward' map) to convert simple distribution samples (like those from a standard normal distribution) into samples resembling the target data.

Key frameworks for deep generative models include, Generative Adversarial Networks (GANs), Variational Autoencoders (VAEs), Wasserstein Autoencoders (WAEs), and Denoising Diffusion Models. Each model comes with it's advantages as well as disadvantages. To this end we talk about three qualities a generative model can have: Speed, quality, and mode coverage. GANs excel at speed and quality, but it is difficult to achieve full mode coverage. VAEs and WAEs offer speed and mode coverage, but generally produce samples of lower quality. Denoising diffusion models are slow but provides high-quality samples and mode coverage.

As speed is of utmost importance for our real-time applications, we are not discussing denoising diffusion models further, but are focusing on GANs and WAEs in the following chapters instead.

1.4 Structure of Thesis

In this thesis, we introduce a range of deep learning-based methodologies aimed at enhancing digital twins. Our primary focus is on accelerating the forward problem and optimizing data assimilation processes for both static and dynamic problems. Throughout the thesis, each chapter deals with distinct aspects, ultimately building towards a suite of tools and techniques.

Chapter 2

In this chapter, we develop a methodology that speeds up simulations of dynamic PDEs. We present a framework that makes use of AEs for nonlinear dimensionality reduction and time stepping NNs for time stepping in the latent space. The AE is made up of convolutional NNs, which ensures that the dimensionality reduction utilizes the spatial structure of the states. Furthermore, the time stepping is performed using memory-aware NNs such as LSTMs and CCNNs. We also discuss considerations regarding long-term stability of the time stepping in shape of residual learning and regularizing the Jacobian of the NNs.

Chapter 3

In the third chapter of the thesis, we focus on solving static Bayesian inverse problems. We train a GAN to sample from the prior of the states and parameters and embed it into a Hamiltonian Monte Carlo sampling scheme. By using a GAN, we can perform the sampling in the low-dimensional latent space, resulting in a method we refer to as the Markov Chain GAN (MCGAN). Furthermore, by using an NN as a probabilistic surrogate model, we can utilize gradient computations to make the sampling more efficient in shape of the Hamiltonian Monte Carlo Method. The theoretical convergence of the method in the Wasserstein-1 distance is proved, which provides a theoretical foundation.

Chapter 4

In this chapter, we focus on the case of localizing leaks in large water distribution networks. We train a supervised Wasserstein AE to generate the full state of flow rates and pressure heads and embed it into a Bayesian inference framework. By using a generative NN, we speed up the problem significantly and thereby we can efficiently compute the posterior distribution over possible leak locations. To get the optimal performance of the framework we develop a novel transformer-based Wasserstein AE.

Chapter 5

Lastly, in Chapter 5, we present the methodology that enables real-time dynamic data assimilation. A novel transformer-based dimensionality reduction layer is used in a Wasserstein AE to reduce the dimension of the state. Then a time stepping transformer is used to perform parameterized time stepping which allows us to perform the state estimation in the latent space as well as parameter estimation. The NNs are embedded into a particle filter which not only gives estimates, but the full posterior filtering distribution. The novel method is referred to as the Deep Latent Space Particle Filter (D-LSPF).

2

REDUCED ORDER MODELING FOR PARAMETERIZED TIME-DEPENDENT PDES USING SPATIALLY- AND MEMORY-AWARE DEEP LEARNING

We present a novel reduced order model (ROM) approach for parameterized time-dependent PDEs based on modern learning. The ROM is suitable for multi-query problems and is nonintrusive. It is divided into two distinct stages: A nonlinear dimensionality reduction stage that handles the spatially distributed degrees of freedom based on convolutional autoencoders, and a parameterized time stepping stage, based on memory aware neural networks (NNs), specifically causal convolutional and long short-term memory NNs. Strategies to ensure generalization and stability are discussed. To show the variety of problems the ROM can handle, the methodology is demonstrated on the advection equation, and the flow past a cylinder problem modeled by the incompressible Navier-Stokes equations.

This chapter is based on the article "Reduced order modeling for parameterized time-dependent PDEs using spatially- and memory-aware deep learning", *Journal of Computational Science*, 53:101408, 2021 [105].

2.1 Introduction

Simulations based on first-principles models often form an essential element for understanding, designing, and optimizing problems in, for example, physics, engineering, chemistry, and economics. However, with an increasing complexity of the mathematical models under consideration, it is not always possible to achieve the desired fidelity of such simulations in a satisfactory time frame. This is especially the case when dealing with multi-query and/or real time problems, as encountered in uncertainty quantification and model predictive control, where the computational model is typically parameterized.

There are several approaches to reduce the computation time bottleneck. The arguably most common ones include high-performance computations [57], high-order discretizations [83], iterative and/or multigrid methods [136, 152], and reduced order modeling (ROM) [123]. High-performance computing may be costly; the improvements due to high-order discretization strongly depend on the smoothness of solutions at hand, and iterative methods are highly dependent on suitable preconditioners. Furthermore, these approaches may suffer from the curse of dimensionality. ROM, a relatively recent research area, is an interesting alternative to the other approaches.

The ROM solution process is generally divided into two distinct stages [123]: A so-called ”offline stage”, in which the reduced model is derived, and an ”online stage”, where the reduced model is utilized and solved. Popular choices for the two stages are the proper orthogonal decomposition (POD) model definition, combined with a (Galerkin) projection procedure in the online stage [67, 123]. Whereas this combination has shown important successes, it has also been shown that the POD and projection approaches perform worse in specific settings, such as for advection-dominated or nonlinear problems. Furthermore, projection-based methods are intrusive, as they require access to the underlying high-fidelity model. Nowadays, it is a reasonable assumption that an industrial model is not directly accessable, and therefore non-intrusive approaches, i.e. approaches that are only based on a series of snapshots of solutions, are increasingly interesting alternatives.

Machine learning has recently gained the attention from the scientific computing community due to great successes of artificial intelligence in various settings. Specifically Artificial Neural Networks (ANNs), often simply denoted neural networks (NNs), have shown remarkable results in tasks such as image analysis and speech recognition. Much of the success has been boosted further by the availability of open source software frameworks, such as PyTorch [114] and Tensorflow [1], which have made implementation and training possible without expert knowledge and the availability of computation accelerating hardware, such as GPUs, has made training of very large models feasible. These recent advances have accelerated research in especially deep learning, i.e. multilayered NNs, which was not possible few a years ago, resulting in many NN architectures specialized in certain tasks, such as time series forecasting and dimensionality reduction.

NNs have gained traction within the mathematics, numerical analysis, and engineering communities either as a replacement or as a supplement to conventional function approximation methods. For an overview of articles, prospects, and future challenges see e.g. [9, 17, 88]. In this chapter, we will combine ROM and machine learning in both the offline and the online stages to showcase the potential of using these technologies on conventional problems from scientific computing.

Important work has already been done on the topic of NN-based ROM. For example, the authors in [56, 65, 150] have used proper orthogonal decomposition (POD) for dimensionality reduction and data-driven methods to map the parameters to the reduced basis coefficients. However, none of these approaches considers time-dependent problems.

Examples of approaches that utilize POD and also deal with time are found in [8, 108, 160]. A difference with our method is that these approaches do not compute the unsteady states by means of time stepping, but rather consider time an extra parameter. Hence, it is not possible to advance a solution in time from an arbitrary point on the trajectory. The above mentioned approaches are based on a linear dimensionality reduction scheme in the form of the POD.

In [87], a convolutional autoencoder (CAE) is utilized for a nonlinear dimensionality reduction, while the time stepping is done, intrusively, using multistep methods on the reduced model, derived from a Galerkin projection procedure. Due to the Galerkin projection of the high-fidelity model, this approach requires access to the underlying model. In [49], a CAE is also used for model reduction and an LSTM is used for time stepping of the reduced state, but with the problem parameters kept fixed. The paper [18] also considers CAEs for dimensionality reduction and a dense feedforward neural network (DFFNN) to map the parameters but without any time stepping procedure. Closest to our work is [165], where a CAE is employed to reduce the dimension and a causal convolutional neural network (CCNN) to encode previous reduced states. The CCNN and the DFFNN are trained independently of each other. Stability of the methodology is not discussed in that paper and neither are comparisons with alternative regression techniques.

In our work, we present a non-intrusive framework, based on deep learning, for computing parameterized spatio-temporal dynamics. The resulting reduced order model is divided into two distinct stages: Firstly, a dimensionality reduction stage based on CAEs, and secondly a memory-aware NN stage for parameterized time stepping. This methodology utilizes the effectiveness of CAEs as nonlinear dimensionality reduction techniques for spatially distributed data. To discuss the advantages of using CAEs, we make a comparison with the widely used linear counterpart, POD. Specifically, we show that POD is a special case of an autoencoder. Furthermore, we present a flexible neural network structure for time stepping, that takes into consideration previous states as well as parameters. The framework is quite general and allows for incorporation of various types of neural network architectures, hence allowing state-of-the-art techniques that fit the problem at hand. We present and compare two modern time series forecasting architectures, Long Short-Term Memory (LSTM) networks [68], and Causal Convolutional Neural Networks (CCNNs) [110]. Furthermore, we present and discuss a series of approaches to ensure stability and generalization of the time stepping network. The scheme presented here is compared to similar, but different, schemes, like Gaussian processes with POD.

To the best of our knowledge, there is no other work on deep learning-based ROM that is non-intrusive, uses CAEs for dimensionality reduction, has memory-aware and parameterized time stepping, and discusses practical approaches to ensure stability and generalization. The result is a flexible offline-online scheme that works for various physical phenomena and can easily be modified according to the specific problem at hand. This makes the presented approach suitable for multi-query problems.

The structure of the present chapter is as follows. In Section 2.2, we present parameterized time-dependent PDEs. In Section 2.3, we discuss dimensionality reduction. Furthermore, we discuss how convolutional autoencoders are used for nonlinear dimensionality reduction. In Section 2.4, we present the parameterized memory-aware time stepping neural network. In Section 2.5, we showcase the performance on two test problems: A linear advection equation and a flow past a cylinder modeled by the incompressible Navier Stokes equations.

2.2 Parameterized Time-Dependent PDEs

The model under consideration is of the form

$$\partial_t u(t,x;\mu) = F(t,x,u;\mu), \quad u(0,x;\mu) = u_0(x;\mu), \tag{2.1}$$

where F is a (nonlinear) differential operator, $\partial_t := \frac{\partial}{\partial t}$, $u : \mathbb{R} \times \mathbb{R}^d \times \mathbb{R}^{N_p} \to \mathbb{R}$ or $u : \mathbb{R} \times \mathbb{R}^d \times \mathbb{R}^{N_p} \to \mathbb{R}^d$, $t \in [0,T]$, and $x \in \mathbb{R}^d$. Equation (2.1) is a very general parameterized PDE. μ is to be considered a vector of parameters on which the solution depends. These parameters could be diffusion rate, Reynolds number, parameterize an initial or boundary condition, etc. For technical reasons, the parameter space $\mathcal{P}$ is chosen to be a compact subspace of $\mathbb{R}^{N_P}$ [123].

Spatially discretizing (2.1), using finite elements, finite volumes, finite differences [122], gives the following finite-dimensional dynamical system,

$$\partial_t u_h(t,\mu) = F_h(t, u_h(t,\mu);\mu), \quad u_h(0,\mu) = u_h^0(\mu), \tag{2.2}$$

h defines the granularity of the discretization, i.e. grid size, number of elements, etc. We will not go into details regarding these discretizations and it should be assumed that the discretized system is stable and converges to the exact solution when granularity is refined. $u_h(t, \mu) \in \mathbb{R}^{N_h}$ will be referred to as the high-fidelity or full-order solution.

The manifold of high-fidelity solutions, parameterized by time and the parameters, is called the spatial discrete solution manifold,

$$M_h = \{u_h(t, \mu) \mid \mu \in \mathcal{P},\ t \in [0, T]\} \subset \mathbb{R}^{N_h}, \tag{2.3}$$

Our goal is to approximate this manifold.

Defining a time discretization, $\{t_0, t_1, \ldots, t_{N_t}\}$, $t_n = n\delta t$, and using a time stepping scheme gives us the time discrete approximation of (2.2):

$$u_h^{n+1}(\mu) = F_{h,\delta t}(u_h^n; \mu), \tag{2.4}$$

where $u^n(\mu) = u(t_n, \mu)$. We will refer to $u_h^n(\mu)$ as the state at time step n. Note that the discrete time evolution map is not necessarily restricted to only depend on the last state, but can take in several previous states, as is done in e.g. multistep methods, or it could depend on the current state as in implicit methods. We can now define the time-discrete high-fidelity solution manifold:

$$M_{h,\delta t} = \{u_h^n(\mu) \mid \mu \in \mathcal{P},\ n = 0, \ldots N_t\} \subset \mathbb{R}^{N_h}. \tag{2.5}$$

The subscripts h and δt refer to the chosen spatial and time discretizations, respectively. $M_{h,\delta t}$ can be seen as the set of discrete state trajectories parameterized by the set of parameters.

In general, N_h will be very large, which makes advancing the state with (2.4) for many time steps time consuming. This is especially the case when dealing with high-dimensional domains and multiphysics problems. It is indeed a problem when dealing with multi-query problems such as uncertainty quantification and data assimilation or when real-time solutions are of importance as in real-time control settings and digital twins.

2.3 Dimensionality Reduction

The fundamental idea of dimensionality reduction is that the minimal number of variables necessary to represent the state, also called the intrinsic dimension, of the dynamical system is low compared to the dimension of the high-fidelity model. However, identifying an optimal low-dimensional representation is, in general, not a trivial task. In this section, we will give a brief overview of linear dimensionality reduction, particularly, the well-known proper orthogonal decomposition (POD). Then, from the linear outset, we will describe the more general case of nonlinear dimensionality reduction.

In general, for both linear and nonlinear dimensionality reduction, we assume that a state, $u_h^n(\mu) \in \mathbb{R}^{N_h}$, can be approximated,

$$u_h^n(\mu) \approx \Phi(u_h^n) = \Phi_{dec} \circ \Phi_{enc}(u_h^n(\mu)), \tag{2.6}$$

where $\Phi_{enc}(u_h) \in \mathbb{R}^{N_l}$ with $N_l \ll N_h$. Φ_{enc} is referred to as the *encoder* and Φ_{dec} the *decoder*. The encoder transforms the high-dimensional input to a *latent space* of low dimension and the decoder transforms the latent variable back to the high-fidelity space. The latent space is often denoted the *reduced trial manifold*. The state at time step n in the latent space is denoted $u_l^n(\mu) = \Phi_{enc}(u_h^n(\mu))$, and will be referred to as the latent state.

Ideally, Φ reconstructs the input perfectly for any given parameters and time step. However, that is, in general, not possible. The precision of the reconstruction is heavily dependent on the dimension of the latent space, as this determines the amount of compression applied. One computes Φ by choosing a latent dimension, N_l, and then solving the minimization problem,

$$\Phi^* = \arg\min_{\Phi} \sqrt{\int_{\mu \in P} \left[\sum_{n=0}^{N_t} ||u_h^n(\mu) - \Phi(u_h^n(\mu))||_2^2 \right] d\mu}, \tag{2.7}$$

where $||\cdot||_2$ denotes the l^2-norm. Theoretically, the reconstruction error should decrease when N_l is increased until the intrinsic dimension of the problem is reached. From thereon, increasing the dimension of the latent space should have very little effect on the reconstruction error.

There are many ways of solving (2.7) [123]. Here, we focus on a data-driven approach, sometimes referred to as the *method of snapshots*. A snapshot is a high-fidelity solution for a given parameter realization at a certain time. The idea of this approach is to make N_{train} samples from the parameter space and then compute a series of $N_T + 1$ snapshots, i.e. trajectories, per parameter sample,

$$M_{N_{train},h,\delta t} = \left\{ u_h^0(\mu_1), \ldots, u_h^{N_t}(\mu_1), u_h^0(\mu_2), \ldots, u_h^{N_t}(\mu_2), \ldots, u_h^0(\mu_{N_{train}}), \ldots, u_h^{N_t}(\mu_{N_{train}}) \right\}. \tag{2.8}$$

Then, (2.7) is rewritten into an empirical minimization problem:

$$\Phi^* = \arg\min_{\Phi} \sqrt{\sum_{i=1}^{N_{train}} \sum_{n=0}^{N_t} ||u_h^n(\mu_i) - \Phi\left(u_h^n(\mu_i)\right)||_2^2}. \tag{2.9}$$

The insight is that sampling a finite number of discrete trajectories a sufficient number of times may yield an accurate enough representation of the time-discrete high-fidelity solution manifold. It should be noted that computing (2.8) is potentially very expensive and even infeasible in some cases.

When a reduction scheme is computed, one can then compute the parameterized trajectories in the latent space, by

$$u_l^{n+1}(\mu) = F_{l,\delta t}(u_l^n;\mu), \quad u_l^0(\mu) = \Phi_{enc}\left(u_h^0(\mu)\right), \tag{2.10}$$

from which the trajectories in the high-fidelity space can be recovered by $u_h^n(\mu) = \Phi_{dec}\left(u_l^n(\mu)\right)$. $F_{l,\delta t}$ can be derived in many ways and much time and effort have been put into deriving optimal latent dynamics.

2.3.1 Linear Dimensionality Reduction

In linear dimensionality reduction, the strategy is to find a reduced linear trial manifold of low dimension. Since the sought manifold is linear, it can be written as the column space, $\text{Col}(V)$ of some matrix, $V \in \mathbb{R}^{N_h \times N_l}$. The column space is the space spanned by the columns of the matrix V. From the orthogonal projection theorem, it can be shown that the optimal projection onto a latent linear space is given by

$$u_h \approx VV^T u_h. \tag{2.11}$$

Hence, this is a special case of (2.6) where

$$\Phi = VV^T, \quad \Phi_{enc} = V^T, \quad \Phi_{dec} = V. \tag{2.12}$$

This simplification reduces (2.9) to

$$V^* = \arg\min_V \sqrt{\sum_{i=1}^{N_{train}} \sum_{n=0}^{N_t} ||u_h^n(\mu_i) - VV^T u_h^n(\mu_i)||_2^2}, \tag{2.13}$$

often accompanied by the constraint that the columns of V are orthogonal, $V^T V = 0$. It can be shown that (2.13) has an exact solution [123]. By collecting the snapshots in a *snapshot matrix*,

$$S = \left[u_h^0(\mu_1) \mid \ldots \mid u_h^{N_t}(\mu_1) \mid \ldots \mid u_h^0(\mu_{N_{train}}) \mid \ldots \mid u_h^{N_t}(\mu_{N_{train}})\right], \tag{2.14}$$

one can show that the optimal $V \in \mathbb{R}^{N_h \times N_l}$ is given by the first N_l *left singular vectors.* The left singular vectors are computed through the singular value decomposition (SVD),

$$S = U\Sigma Z^T, \tag{2.15}$$

where U is a matrix whose columns are the left singular vectors, Z is a matrix whose columns are the right singular vectors, and Σ is a diagonal matrix with the singular values on the diagonal. V is then chosen to be the first N_l columns of U. This method of obtaining V is the *proper orthogonal decomposition* (POD) [123], also denoted *principal component analysis* (PCA) [41].

To obtain $F_{l,\delta t}$, a Petrov Galerkin projection is often performed, which yields

$$F_{l,\delta t}(u_l^n;\mu) = W^T F_{h,\delta t}(V u_l^n;\mu). \tag{2.16}$$

When $W = V$, it is denoted the Galerkin projection. This approach is *intrusive*, which means that direct access to the model, $F_{h,\delta t}$, is required. Furthermore, in the online phase a transformation between the latent space and the high-fidelity space must be performed in each time step in order to be able to evaluate $F_{h,\delta t}(V u_l^n;\mu)$, which slows down the computations. Various methods to circumvent that problem, such as the discrete empirical interpolation methods [123], already exist. In recent years, there are also studies exploring approximating $F_{l,\delta t}$ with neural networks [108, 119].

While there are many advantages of a linear reduction scheme, such as the explicit solution to (2.13), there are, indeed, disadvantages as well. A significant problem is the restriction to a linear trial manifold. The optimal trial manifold, i.e. the trial manifold of the intrinsic dimension, is rarely linear. Especially for advection-dominated and nonlinear problems, it has been shown that a linear reduced approximation does not necessarily lead to significant speed-ups.

2.3.2 Nonlinear Dimensionality Reduction

The extension from linear to nonlinear dimensionality reduction comes naturally and addresses several of the drawbacks of linear dimensionality reduction. The fundamental difference is that we remove the constraint that the latent space has to be a linear manifold. Due to this generalization, we cannot write the projection operator as the matrix product VV^T anymore, but instead we must use the general form in (2.6), where Φ_{enc} and Φ_{dec} can be any type of nonlinear functions. This gives rise to a major difference in solving (2.9), since no general exact solution exists and therefore (2.9) will be solved numerically.

Even though extra approximation steps have to be introduced in the nonlinear case, the potential gains will, in some cases, outweigh this hurdle. This is due to the fact that with a nonlinear reduction scheme, it is theoretically possible to reduce the high-fidelity space down to its intrinsic dimension, $N_P + 1$. However, this depends on the choice of Φ and the minimization scheme.

A common method for nonlinear dimensionality reduction from the machine learning communities is, among others, kernel PCA. Here, the nonlinear manifold is embedded into a linear space, often of higher dimension, using a predefined nonlinear mapping, $\psi : \mathbb{R}^{N_h} \to \mathbb{R}^{N_k}$, $N_k > N_h$. From thereon, a linear PCA is performed on the high-dimensional linear data. In order to speed up computations, the so-called kernel trick is typically invoked. Utilizing that, the nonlinear embedding induces a kernel, $K = k(\psi(x), \psi(y)) = \psi(x)^T \psi(y)$, one can compute the low-dimensional basis without explicitly transforming the data and perform PCA in the high-dimensional space. For more details, see [157].

This approach works well in many cases, but suffers from one crucial downside: Choosing the nonlinear mapping, ψ, or the kernel, K, is far from trivial. There exist no clear guiding principles that work across several cases.

Autoencoders

To overcome the problems of other nonlinear dimensionality reduction methods, such as kernel PCA and DEIM, we present Autoencoders (AEs). AEs are a type of NN. For a brief introduction to NNs and the terminology used here, see Section 1.3.1. In the context of dimensionality reduction, one can interpret an AE as a kernel PCA where the kernel is learned during the training process. Thus, one circumvents the problem of choosing a suitable kernel. Note that this interpretation is merely presented in order to give an intuition of AEs in context of other methods. To further explain the connections between AEs and PCA, it is worth noting that a single hidden layer AE with linear activation functions is equivalent to PCA. A single hidden layered AE without bias terms can be written as

$$\Phi(u_h^n(\mu); \theta) = T_2 \circ T_1(x) = W_2 W_1 u_h^n(\mu), \tag{2.17}$$

where $\theta = \{W_1, W_2\}$, $T_1 : \mathbb{R}^{N_h} \to \mathbb{R}^{N_l}$, and $T_2 : \mathbb{R}^{N_l} \to \mathbb{R}^{N_h}$ are linear maps and W_1 and W_2 are matrices. Typically, the mean squared error is chosen as the loss function for AEs, which gives the following minimization problem for the single hidden layer AE:

$$\arg \min_{W_1, W_2} \frac{1}{N_{train} N_t} \sum_{i=1}^{N_{train}} \sum_{n=0}^{N_t} \left|\left| u_h^n(\mu_i) - W_2 W_1 u_h^n(\mu_i) \right|\right|^2 . \tag{2.18}$$

Hence, training a single layer AE is equivalent to solving the PCA minimization problem, (2.13), without the orthogonality constraint. Conclusively, Eq. (2.11) in the PCA context, can be considered a special case of an AE.

By dividing the AE into the encoder and decoder parts and allowing an arbitrary number of layers and nonlinear activation functions, it is easier to understand the similarities to linear dimensionality reduction and why AEs have the potential to perform significantly better. Consider the encoder part with a linear activation in the final layer,

$$\Phi_{enc}(u_h^n(\mu);\theta) = T_L^{enc} \circ \underbrace{\sigma_{L-1}^{enc} \circ T_{L-1}^{enc} \ldots \circ \sigma_1^{enc} \circ T_1^{enc}(u_h^n(\mu))}_{\psi_{enc}(u_h^n(\mu);\theta)} = W_L^{enc}\psi_{enc}(u_h^n(\mu);\theta) = z, \tag{2.19}$$

where $\psi_{enc}(x;\theta) = (\psi_{enc}^1(x;\theta), \ldots, \psi_{enc}^{N_e}(x;\theta)) \in \mathbb{R}^{N_e}$ and $W_L^{enc} \in \mathbb{R}^{N_e \times N_l}$. For convenience, we ignore bias terms. We see that this corresponds to a nonlinear embedding onto $\mathbb{R}^{N_e}$ and then a projection onto the space spanned by the vectors $\psi_{enc}^1, \ldots, \psi_{enc}^{N_e}$. This is similar to the idea behind kernel PCA. The difference is that in the AE framework we adjust the nonlinear embedding in the training instead of defining it beforehand. The decoder part is similarly written as

$$\Phi_{dec}(z;\theta) = T_L^{dec} \circ \underbrace{\sigma_{L-1}^{dec} \circ T_{L-1}^{dec} \ldots \circ \sigma_1^{dec} \circ T_1^{dec}(z)}_{\psi_{dec}(z)} = W_L^{dec}\psi_{dec}(z;\theta) = \tilde{u}_h^n(\mu), \tag{2.20}$$

where $W_L^{dec} \in \mathbb{R}^{N_d \times N_h}$.

Note that this is merely a brief discussion of the topic of AEs aiming to give an intuitive understanding. For more details see [168].

Convolutional Autoencoders

Convolutional autoencoders (CAEs) are a special type of AEs utilizing convolutional layers instead of dense layers. A brief introduction to convolutional neural networks (CNNs) can also be found in Section 1.3.1. It can be shown that dense and convolutional neural networks are equivalent regarding approximation rates [121], which means that theoretical approximation results for dense NNs translate almost directly to CNNs. For practical purposes, however, convolutional layers are often to be preferred due to especially the following two properties:

- *Local connections*, which utilize that spatial nodes close to each other are highly correlated.
- *Shared weights*, which in practice make the affine transformations very sparse and enables location invariant feature detection.

An additional advantage is that it is straightforward to handle multiple spatially distributed states. These occur in coupled PDEs, such as the Navier Stokes equations, where one is dealing with both the $x-$, $y-$ and $z-$components of the velocity field as well as the pressure field. In the framework of CAEs, these can all be included by interpreting them as different channels. This enables the possibility of including multiple spatial states without increasing

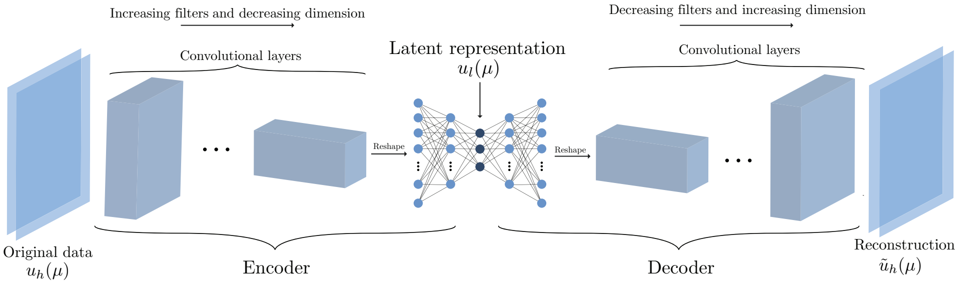


Figure 2.1: Illustration of a convolutional autoencoder

the number of weights in the neural network significantly. The connection between PDEs and CNNs has already been made, see e.g. [134].

In Figure 2.1, one sees an illustration of a CAE. The encoding consists of a series of convolutional layers with an increasing number of filters and decreasing dimension, effectively downsampling the number of degrees of freedom, followed by dense layers. Similarly, the decoding consists of a series of dense layers followed by a series of deconvolutional layers with a decreasing number of filters and increasing dimension, effectively upsampling. The downsampling is often performed by utilizing pooling layers or strides larger than one.

It is worth noting that computing the decoder, Φ_{dec}, of a CAE in the training phase is effectively solving an inverse problem. Inverse problems are, in general, ill-posed and therefore require regularization. L^2-regularization, often referred to as weight decay, is frequently used, and results in the following minimization problem to solve:

$$\arg\min_{\theta} \frac{1}{N_{train} N_t} \sum_{i=1}^{N_{train}} \sum_{n=0}^{N_t} ||u_h^n(\mu_i) - \Phi(u_h^n(\mu_i);\theta)||^2 + \alpha \, ||\theta||_2^2 \, , \tag{2.21}$$

where α is a hyperparameter to be tuned. Besides ensuring well-posedness, the term also ensures generalization.

2.4 Approximating Parameterized Time Evolution using Neural Networks

In the previous section, we presented the general framework for nonlinear dimensionality reduction and showcased how convolutional autoencoders fit into this framework. In this section, we explain how neural networks will be utilized for approximating the dynamics in the latent space. For a brief review of the relevant types of neural networks, see again Section 1.3.1. Neural networks have already shown to be able to approximate dynamical systems [32, 39, 165].

We aim to approximate the dynamics in the latent space non-intrusively by a function, $\Psi \approx F_{l,\delta t}$:

$$u^{n+1} = \Psi(u^n), \tag{2.22}$$

where Ψ is a neural network. The approximated latent states will be denoted, $\tilde{u}_l^n(\mu)$, to distinguish them from the encoded high-fidelity state, $u_l^n(\mu) = \Phi_{enc}(u_h^n(\mu))$. Thereby, we aim to achieve:

$$\begin{aligned} &\left\{\tilde{u}_l^0(\mu_1), \dots, \tilde{u}_l^{N_t}(\mu_1), \dots, \tilde{u}_l^0(\mu_{N_{train}}), \dots, \tilde{u}_l^{N_t}(\mu_{N_{train}})\right\} \\ \approx &\left\{u_l^0(\mu_1), \dots, u_l^{N_t}(\mu_1), \dots, u_l^0(\mu_{N_{train}}), \dots, u_l^{N_t}(\mu_{N_{train}})\right\} \end{aligned} \tag{2.23}$$

Taking Larger Steps

Using high-fidelity methods for time stepping often includes some restrictions on the step size in order for the scheme to be stable. An example is the Courant–Friedrichs–Lewy (CFL) condition for advection-dominated problems [89]. With our strategy, where we aim to learn a neural network representation of the time evolution map, there is no immediate connection between step size and stability. Therefore, in order to speed up online computations, the neural network can be trained to learn to take steps of size $s\delta t$. Hence, $\Psi \approx F_{l,s\delta t}$.

In the offline phase, the high-fidelity trajectories are still computed with step size δt, to ensure stability, but only every s'th step is used for training the NN:

$$\left\{u_h^0(\mu), u_h^1(\mu), u_h^2(\mu) \dots, u_h^{N_t}(\mu)\right\} \mapsto \left\{u_h^0(\mu), u_h^s(\mu), u_h^{2s}(\mu) \dots, u_h^{N_t}(\mu)\right\}. \tag{2.24}$$

In general, $u_h^n(\mu)$ and $u_h^{n+1}(\mu)$, are highly correlated, which means that we gain very little extra information by using both variables in the training of the NN. Therefore, it makes sense to only use every s'th step to save memory and speed up the training. However, the number s must be chosen according to various factors, like the requested detail of the dynamics in the online phase. It should further be kept in mind that larger s-values result in a more complicated map to learn, and thus complicates the training.

For simplicity, we will use the notation $\left\{u_h^0(\mu), u_h^1(\mu), u_h^2(\mu) \dots, u_h^{N_t}(\mu)\right\}$ when referring to the trajectory used for training the neural network.

Approximating the State vs. Residual

At first glance, it makes sense to train a neural network to approximate u_l^{n+1} directly given u^n. However, it is shown in [119] and [45] that learning the residual instead of the next state

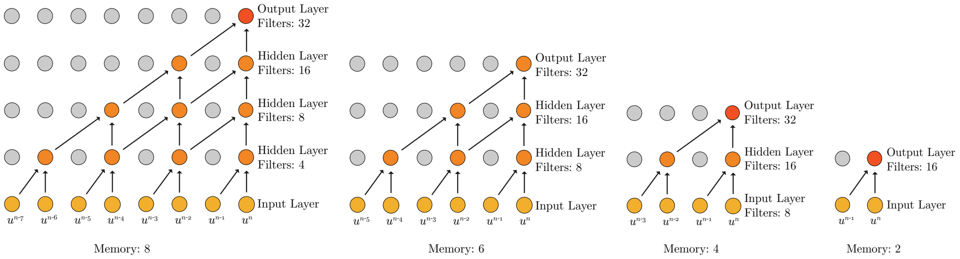


Figure 2.2: CCNN network architectures for varying memory.

often improves the accuracy. Hence, we consider the case

$$u_l^{n+1} = \Psi(u^n) = u_l^n + R(u_l^n), \tag{2.25}$$

where R is approximated by a neural network. This practically makes Ψ what is often referred as a residual neural network.

Incorporating Memory

In [119] and [49], the potential benefits of not only using the present state but also incorporating several previous time steps for the future predictions were shown. Therefore, we now consider

$$u_l^{n+1} = \Psi(u_l^n, u^{n-1}, \dots, u_l^{n-\xi}) = u_l^n + R(u_l^n, u_l^{n-1}, \dots, u_l^{n-\xi}), \tag{2.26}$$

where ξ is the number of previous states included as input into the residual computation by the NN. The principle of incorporating several previous time steps can loosely be compared to linear multistep methods where the order of approximation can be increased by using several previous steps [89]. In contrast to linear multistep methods, NNs incorporate the previous time steps in a nonlinear fashion.

We consider two different types of networks here: LSTM, and the CCNN. The two types of neural networks take varying computational time to train, have varying numbers of parameters, and vary in regards to how they interpret memory. In Section 1.3.1, there is a short description of the two types. Regarding the CCNNs, there are different ways to include memory. We have chosen to include memory in shape of adding more layers, see Figure 2.2, for examples for $\xi = 8$, $\xi = 6$, $\xi = 4$, and $\xi = 2$.

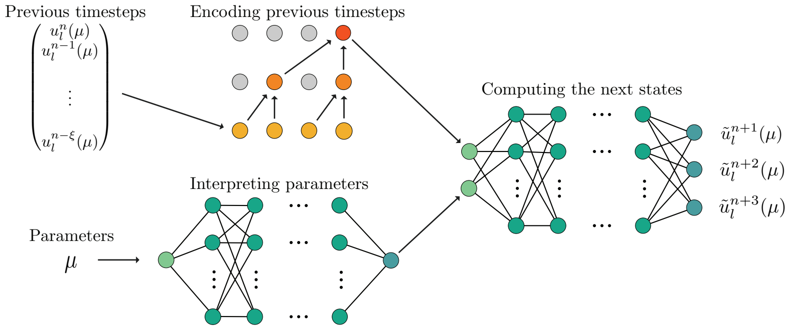


Figure 2.3: Illustration of the parallel neural network structure. The "Encoding previous time steps" part is visualized using the CCNN, but it should be noted that an LSTM network (or any suitable time series encoder) could be put in its place.

Parameterized Dynamics

We aim to simulate parameterized trajectories of the latent dynamics. Hence, we need to incorporate the parameters as input to the residual computation, resulting in a map

$$\Psi : \mathbb{R}^{\xi N_l + N_P} \to \mathbb{R}^{\zeta N_l}, \quad u^{n+1}(\mu) = \Psi(u_l^n(\mu), u^{n-1}(\mu), \ldots, u_l^{n-\xi}(\mu), \mu). \tag{2.27}$$

For now, we consider constant parameters, but it should be possible to incorporate time-dependent parameters. For this reason, the parameters do not need to be part of the memory aware section of the network. We propose a parallel architecture consisting of two branches combining into one: One branch interpreting the last ξ states and one branch processing the parameters. The two branches then connect and provide one final prediction for the residual. Having a single neural network incorporating the previous states and the parameters enables simultaneous training of the two branches. See Figure 2.3 for an illustration of the network structure. This ensures that the learned latent features from both branches are optimal with respect to predicting the next state. This is in contrast to what is done in [165], where the memory and the parameters are incorporated into two completely separate networks.

For the parameter branch, we simply make use of a dense FFNN. There is no immediate reason to believe that more complicated architectures are necessary, since we are neither dealing with time-dependent nor high-dimensional or continuously spatially varying input. Note, however, that there is no reason to believe that this methodology will not work if the FF network in the parameter branch is replaced with a memory aware network in more advanced settings.

The training of the full time-evolution network is done by minimizing the loss function

$$L(u_l,\mu;\theta) = \frac{1}{N_{train}N_t}\sum_{i=1}^{N_{train}}\sum_{n=\xi}^{N_T}\left|\left|u^{n+1}(\mu_i) - \Psi(u_l^n(\mu_i),\dots,u_l^{n-\xi}(\mu_i),\mu_i;\theta)\right|\right|_2^2, \qquad (2.28)$$

with respect to the NN parameters θ.

Whereas the individual techniques described may be well-known, we here show how these techniques can be integrated to achieve high quality performance and accuracy.

Imposing Stability and Generalization

It is well-known that NNs do not necessarily generalize well beyond the training data without some kind of regularization. Combining that with the general risk of having instability in discrete dynamical systems, makes it crucial to address these problems during the training.

The arguably most common technique is to add L^1- or L^2-regularization to the loss function. Furthermore, specifically for dynamical systems, it has been shown in [32] and [112] that regularizing the eigenvalues of the Jacobian of the dynamics with respect to the state variable, $D_u\Psi$, improves long term predictions. In short, this is related to linear and Lyapunov stability analysis of dynamical systems, that are related to sensitivity to initial conditions. Hence, we propose adding the term $||D_u\Psi||_2$, which is the matrix 2-norm, i.e. the spectral radius of the Jacobian of Ψ, to the loss function. In practice, by utilizing the relation

$$||D_u\Psi||_2 \leq ||D_u\Psi||_F\,, \qquad (2.29)$$

we instead add the computationally much cheaper Frobenius norm.

It can empirically be shown that the long term predictions are significantly better if the network takes several steps at a time instead of a single one. Hence, we modify the output of the NN to

$$R\left(u_l^n(\mu),\dots,u_l^{n-\xi}(\mu),\mu;\theta\right) = \left[R_1,R_2,\dots,R_\zeta\right]^T, \qquad (2.30)$$

which gives future predictions,

$$\begin{bmatrix} u_l^{n+1}(\mu) \\ \vdots \\ u_l^{n+\zeta}(\mu)\end{bmatrix} = \begin{bmatrix}\Psi_1\left(u_l^n(\mu),\dots,u_l^{n-\xi}(\mu),\mu;\theta\right) \\ \vdots \\ \Psi_\zeta\left(u_l^n(\mu),\dots,u_l^{n-\xi}(\mu),\mu;\theta\right)\end{bmatrix} = u_l^n(\mu) + \begin{bmatrix}R_1\left(u_l^n(\mu),\dots,u_l^{n-\xi}(\mu),\mu;\theta\right) \\ \vdots \\ R_\zeta\left(u^n(\mu),\dots,u_l^{n-\xi}(\mu),\mu;\theta\right)\end{bmatrix} \qquad (2.31)$$

Empirically, we see that this modification keeps the prediction from exploding for longer time and it reduces spurious oscillations.

The resulting loss function for the dynamics NN is given by:

$$
\begin{aligned}
L(u,\mu;\theta) = {} & \frac{1}{N_{train}N_t} \sum_{i=1}^{N_{train}} \sum_{n=\xi}^{N_T} \left|\left| \sum_{k=1}^{\zeta} \left[u_l^{n+k}(\mu_i) - \Psi_k \left(u_l^{n}(\mu_i), \dots, u_l^{n-\xi}(\mu_i), \mu_i; \theta \right) \right] \right|\right|_2^2 \\
& + \underbrace{\beta_1 \left|\left|\theta\right|\right|_2^2}_{\text{Weight decay}} + \underbrace{\beta_2 \left|\left|D_u R\right|\right|_F}_{\text{Jacobian regularization}} \quad ,
\end{aligned}
\tag{2.32}
$$

2.4.1 The Complete Scheme

Putting the components together, we have a scheme subdivided into two parts that are trained independently: The CAE, and the time evolution. The whole process is divided into an online phase and an offline phase.

In the offline phase, the CAE is trained on a series of high-fidelity snapshots in order to identify a nonlinear reduced trial manifold. Then, the CAE is used to reduce the high-fidelity snapshots to the latent space. The latent space trajectories are used to train the time evolution NN. The training of the two neural networks is visualized in Figure 2.4 and outlined in Algorithm 1. Note that in Steps 3 and 5, where the autoencoder and the time evolution network, respectively, are being trained, the considerations mentioned in Section 1.3.1 have to be included, like early-stopping, multiple-initialization, choice of optimizer, etc. In Algorithm 2, an algorithm to automatically choose the latent dimension, number of training trajectories, memory, and future steps per iteration is presented. Note that this is a basic approach to tune the network. More advanced methods such as Bayesian optimization or reinforcement learning could be utilized here. Furthermore, it is worth noting that we can, assuming no time constraints, generate as many training samples as necessary.

In the online phase, the first ξ time steps of the state, computed with a high-fidelity scheme for a given parameter realization μ, are projected onto the latent space using the encoder part of the CAE. From there, the time evolution NN computes the parameterized latent space trajectories iteratively. The latent space trajectories are then transformed to the high-fidelity space using the decoder of the CAE. The online stage is visualized in Figure 2.5 and described in pseudo code in Algorithm 3.

2.5 Results

The aim of this section is to showcase how well our frameworks perform for different parameterized PDE problems. Furthermore, we show how the various approaches, regularizations,

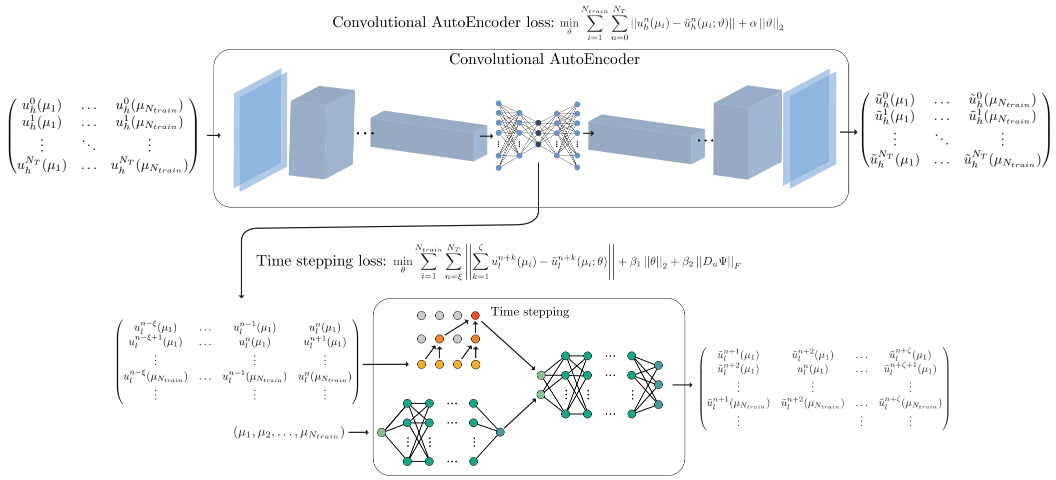

Convolutional AutoEncoder loss: $\min_{\vartheta} \sum_{i=1}^{N_{train}} \sum_{n=0}^{N_T} ||u_h^n(\mu_i) - \tilde{u}_h^n(\mu_i;\vartheta)|| + \alpha ||\vartheta||_2$
Convolutional AutoEncoder
Time stepping loss: $\min_{\theta} \sum_{i=1}^{N_{train}} \sum_{n=\xi}^{N_T} \left|\left| \sum_{k=1}^{\zeta} u_l^{n+k}(\mu_i) - \tilde{u}_l^{n+k}(\mu_i;\theta) \right|\right| + \beta_1 ||\theta||_2 + \beta_2 ||D_u \Psi||_F$
Time stepping
$(\mu_1, \mu_2, \ldots, \mu_{N_{train}})$

Figure 2.4: Illustration of the offline stage.

```
Algorithm 1: Offline Stage - Training
  Input: $N_l$, $\zeta$, $\xi$, $N_{train}$.
1 Sample $N_{train}$ parameter samples from the parameter space.
2 Generate high-fidelity trajectories,
      $\left\{u_h^0(\mu_1), \ldots, u_h^{N_t}(\mu_1), \ldots, u_h^0(\mu_{N_{train}}), \ldots, u_h^{N_t}(\mu_{N_{train}})\right\}.$
3 Train CAE, $\Phi = \Phi_{dec} \circ \Phi_{enc}$, with latent space dimension $N_l$, by minimizing
      $\arg\min_{\theta} \frac{1}{N_{train} N_t} \sum_{i=1}^{N_{train}} \sum_{n=0}^{N_t} ||u_n(\mu_i) - \Phi(u_n(\mu_i);\theta)||^2 .$   (2.33)
4 Encode high-fidelity trajectories to get latent state space trajectories
      $\left\{\Phi_{enc}\left(u_h^0(\mu_1)\right), \ldots, \Phi_{enc}\left(u_h^{N_t}(\mu_1)\right), \ldots, \Phi_{enc}\left(u_h^0(\mu_{N_{train}})\right), \ldots, \Phi_{enc}\left(u_h^{N_t}(\mu_{N_{train}})\right)\right\}.$
5 Train time evolution network, $R$, to take the last $\zeta$ states and output the residuals for
    the next $\xi$ states, by minimizing
      $\frac{1}{N_{train} N_t} \sum_{i=1}^{N_{train}} \sum_{n=\xi}^{N_T} \left|\left| \sum_{k=1}^{\zeta} \left[u_l^{n+k}(\mu_i) - \Psi_k\left(u_l^n(\mu_i), \ldots, u_l^{n-\xi}(\mu_i), \mu_i; \theta\right)\right] \right|\right|_2^2$
      $+ \beta_1 ||\theta||_2^2 + \beta_2 ||D_u R||_F ,$
6 Estimate error on a test set.
  Output: Trained CAE, $\Phi$, and Time Evolution Network, $R$, and test error, E.
```

Algorithm 2: Offline Stage - Tuning

```
Input: Desired test error, $E^*$
Initialize $E = \infty$, $N_l = 1$, $\zeta = 0$, $\xi = 1$, $N_{train}$.
while $E^* < E$ do
    Train $\Phi$ and $R$ and compute test error, $E$, using Algorithm 1.
    Update $N_l$, $\zeta$, $\xi$, $N_{train}$ and according to some update rule.
end while
Output: Optimal latent dimension, $N_l$, $\zeta$, $\xi$, and $N_{train}$.
```

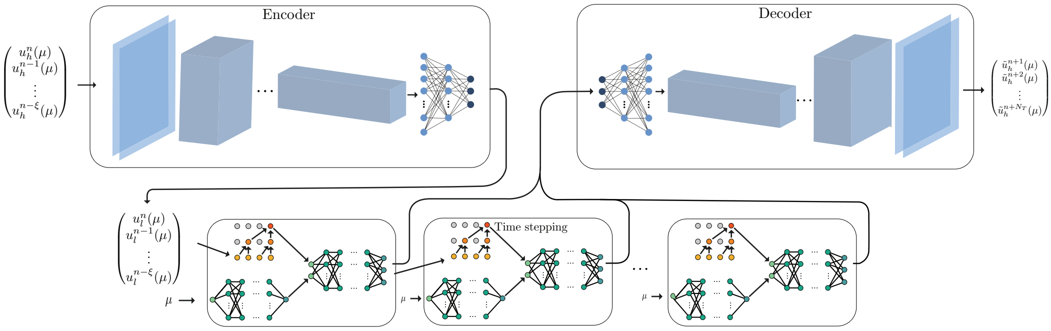

$\begin{pmatrix} u_h^n(\mu) \\ u_h^{n-1}(\mu) \\ \vdots \\ u_h^{n-\xi}(\mu) \end{pmatrix}$
Encoder
Decoder
$\begin{pmatrix} \tilde{u}_h^{n+1}(\mu) \\ \tilde{u}_h^{n+2}(\mu) \\ \vdots \\ \tilde{u}_h^{n+N_T}(\mu) \end{pmatrix}$
$\begin{pmatrix} u_l^n(\mu) \\ u_l^{n-1}(\mu) \\ \vdots \\ u_l^{n-\xi}(\mu) \end{pmatrix}$
μ
Time stepping
μ
μ

Figure 2.5: Illustration of the online stage.

Algorithm 3: Online Stage

```
Input: $\Phi_{dec}$, $R$, $\mu$, $u_h^0(\mu), \dots, u_h^\xi(\mu)$
Output: Approximated trajectory in high-fidelity space.
Encode the initial $\xi$ high-fidelity states,
    $(u_l^0(\mu), \dots, u_l^\xi(\mu)) = \left(\Phi_{enc}\left(u_h^0(\mu)\right), \dots, \Phi_{enc}\left(u_h^\xi(\mu)\right)\right)$
Compute approximated latent trajectory by iterating,
    $\left(\tilde{u}_l^n(\mu), \dots, \tilde{u}_l^{n+\zeta}(\mu)\right) = \tilde{u}_l^n(\mu) + R\left(\tilde{u}^n(\mu), \dots, \tilde{u}^{n-\xi}(\mu), \mu; \theta\right),$
    until desired end time has been reached.
Decode approximated latent space trajectories to high-fidelity space:
    $\left\{\tilde{u}_h^0(\mu), \dots, \tilde{u}_h^{N_t}(\mu)\right\} = \left\{\Phi_{dec}\left(\tilde{u}_l^0(\mu)\right), \dots, \Phi_{dec}\left(\tilde{u}_l^{N_t}(\mu)\right)\right\}.$
```

and parameters affect the performance.

To assess the performance measure, the error on N_{test} test trajectories for parameter values, $\{\mu_1, \dots, \mu_{N_{test}}\}$, that the NNs have not seen in the training phase, is evaluated. We measure the mean relative error (MRE) at every time step and take the mean over multiple runs of the test cases:

$$\text{MRE}(u_h^n(\mu_i), \tilde{u}_h^n(\mu_i)) = \frac{1}{N_{test}} \sum_{i=1}^{N_{test}} \frac{||u_h^n(\mu_i) - \tilde{u}_h^n(\mu_i)||_2^2}{||u_h^n(\mu_i)||_2^2}, \tag{2.34}$$

where

$$||u_h^n(\mu_i)||_2^2 = (u_h^n(\mu_i))^T u_h^n(\mu_i). \tag{2.35}$$

Besides the MRE, we also analyze the standard error:

$$\text{Standard Error} = \frac{\sigma}{\sqrt{N_{test}}}, \tag{2.36}$$

where σ is the variance of the MRE. With this measure, we can assess if the trained NN performs similarly on all the test data, i.e. we empirically show robustness and generalization.

For comparison, we show how the error evolves in time using various regression approaches for time stepping, together with the CAE and POD. We do not compare our method to intrusive approaches, such as POD-Galerkin, as we assume here that the PDE model is not available. The approaches dealing with time as a parameter, instead of performing time stepping, are inherently different and are therefore also not considered here. Instead, we compare another regression technique, Decision Tree Regression (DTR), as in [150], (see [41] for details on DTR). K-Nearest-Neighbour Regression and Gaussian Process Regression (GPR) were also tested. K-Nearest-Neighbour Regression performed very similar to DTR and GPR was infeasible to train due to the many training samples needed in the time stepping training. The implementation of DTR was done with the Python package Scikit-learn [120].

Regarding hyperparameters, we considered the number of layers, number of neurons, regularization parameters, learning rate, batch size, and memory. Due to the high-dimensional hyperparameter space, we used Gaussian process minimization as a quick and approximate way to tune the hyperparameters.

Neural Network Setup

All neural networks are implemented in Tensorflow 2.0 [1] in Python. The training is performed in the Google Colab framework on NVIDIA Tesla P100 GPUs. The neural network architecture configurations for the CAEs can be found in Table 2.1.

	Linear advection equation			Flow Past Cylinder		
Conv	Kernels	Filter Size	Stride	Kernels	Filter Size	Stride
Conv 1	4	5×5	2×2	8	5×5	2×2
BN 1						
Conv 2	8	5×5	2×2	16	5×5	2×2
BN 2						
Conv 3	16	5×5	2×2	32	5×5	2×2
BN 3						
Conv 4	32	5×5	2×2	64	5×5	2×2
BN 4						
Conv 5	-	-	-	128	5×5	2×2
BN 5	-	-	-			
Flatten						
Dense	Neurons			Neurons		
Dense 1	N_l			493		
Dense 2	-			247		
Dense 3	-			N_l		

Table 2.1: Convolutional autoencoder configuration for the advection equation and the Navier Stokes equations. Note that the decoder is the inverse of the encoder. BN = Batchnormalization and Conv = Convolutional layer.

For the CCNN memory encoding, the layers are organized as shown in Figure 2.2. For the LSTM, we work with network architectures of 3-5 layers with 16-64 neurons in each LSTM layer. Furthermore, before the LSTM or CCNN layers every previous state is passed through a dense layer with 16 neurons. In TensorFlow 2.0, this type of layer is denoted `TimeDistributed`.

For the parameter encoding, the neural network is a 3 layer deep network with 16 neurons in each layer. For the final prediction, we utilize a 3-5 layer deep NN with 32 neurons in each layer.

Remark. We only present results on dimensionality reduction using convolutional autoencoders and compare them to POD. It should be noted that dense autoencoders were also tested and showed significantly worse results. Moreover, we only consider LSTMs, and CCNNs for the time stepping. We also studied other architectures, such as neural ODEs [21], gated recurrent units (GRUs), and simple recursive neural networks. However, we chose to not include those results. Neural ODEs performed significantly worse and the training took much longer time. GRUs performed similarly to LSTMs and simple recursive neural networks performed slightly worse.

2.5.1 Linear Advection Equation

We first consider a linear advection equation on the domain $\Omega = [0,1]^2$:

$$\partial_t u(\mu) + b \cdot \nabla u(\mu) = 0, \quad \text{in} \quad \Omega, \tag{2.37a}$$

$$u(\mu) = 0 \quad \text{on} \quad \Gamma, \tag{2.37b}$$

where $\Gamma = \partial\Omega$,

$$b = \mu_1 \begin{pmatrix} -y - \frac{1}{2} \\ x - \frac{1}{2} \end{pmatrix}, \tag{2.37c}$$

with initial condition

$$u_0(\mu) = \exp\left(\frac{1}{2}\left[\frac{(x-x_0)^2}{0.005} + \frac{(y-y_0)^2}{0.005}\right]\right), \tag{2.38}$$

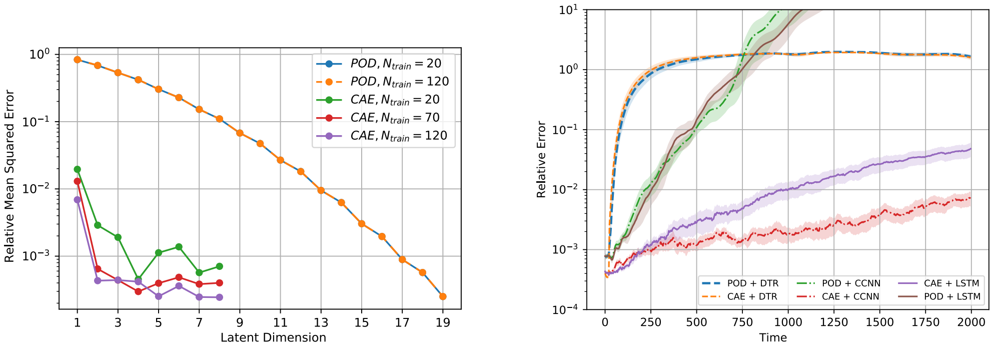
where

$$\begin{pmatrix} x_0 \\ y_0 \end{pmatrix} = \frac{1}{4}\begin{pmatrix} \cos(\mu_2) \\ \sin(\mu_2) \end{pmatrix} + \frac{1}{2}. \tag{2.39}$$

This problem models a Gaussian curve being advected with velocity μ_1 in a circle with origin at $[\frac{1}{2}, \frac{1}{2}]$ and radius $\frac{1}{4}$, starting at the position given by the angle μ_2. This problem is parameterized by two parameters, $\mu = (\mu_1, \mu_2) \in [0.5, 1.5] \times [0, 2\pi]$. The first parameter, the velocity, is directly affecting the phase of the dynamics, while the other, μ_2, is only dictating the initial placement of the Gaussian curve. Hence, we are dealing with a 2-dimensional parameter space, while the dynamics are only parameterized by a single parameter.

The high-fidelity snapshots are computed on a 60×60 grid using the discontinuous Galerkin method with linear Lagrange elements, resulting in a second-order convergence scheme that suits advection dominated problems well. The high-fidelity model consists of 21600 degrees of freedom. For the implementation, we used the FEniCS library in Python [94]. The time stepping is done using the Crank-Nicolson scheme with time steps of size 0.0075 for 2000 steps, resulting in a time interval, $t \in [0, 15]$. The training of the neural networks is done using every 4th time step, $s = 4$, meaning the model is trained to take steps of size 0.03. We are using 15 trajectories for the parameters not included in the training set for testing. The parameter μ is sampled uniformly in the domain $[0.5, 1.5] \times [0, 2\pi]$ for the training data. Furthermore, the training is performed with 120 training trajectories.

In Figure 2.6a, we see a significant improvement by using the CAE compared to the POD approach. Using a latent dimension of 2, which is also the intrinsic dimension of the solution manifold, the CAE reconstructs the high-fidelity solution with an MRE between 10^{-3} and



(a)

(b)

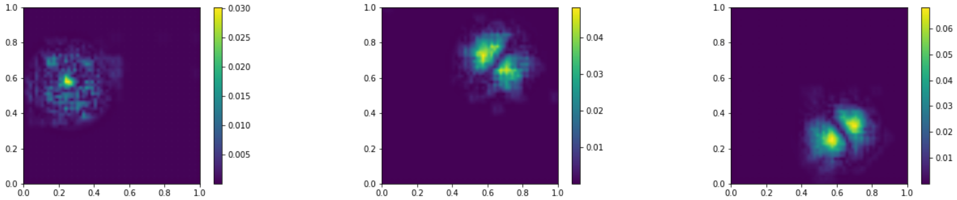
Figure 2.6: (a) Comparison of convergence of the time averaged MRE of the reconstruction using CAE and POD for the advection equation. (b) average test errors computed for the linear advection equation for various combinations of POD, CAE, and regression methods.

10^{-4}. To achieve the same accuracy using the POD method, one needs a latent dimension of at least 17. This supports the previous claim that POD does, in general, not perform well on advection dominated problems. Furthermore, it is shown that the accuracy improves with the amount of training data until the point where more data becomes redundant. Specifically, one sees that using 70 trajectories or 120 trajectories is very similar in performance. Note that the POD method does not improve with the amount of training data.

In Figure 2.6b we compare the mean error at each time step of our method with the alternative approaches. A latent space of dimension 18 is used for the methods using POD to achieve the same accuracy as for the CAE. All regression approaches are trained with 120 trajectories. Clearly, none of the alternative approaches captures the dynamics accurately. Using the POD basis for the NN time stepping performs significantly worse than the proposed methodology. The time stepping map becomes truly high-dimensional and thereby much harder to approximate.

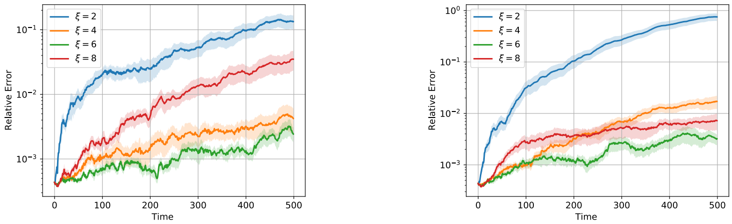
By looking at the pointwise error between the high-fidelity and the NN solutions in Figure 2.7, it is clear that the NN approximation introduces a small phase error. This error could possibly be corrected in a post-processing step.

In Figures 2.8-2.11, various figures showcasing how each parameter affects the accuracy and stability are presented. From these plots, we infer that the NN configuration that performs the best employs $\xi = 6$, $\beta_1 = 10^{-9}$, $\beta_2 = 10^{-6}$, and computes the residual rather than the state directly.



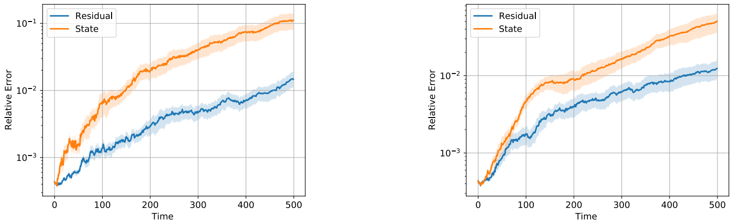
(a) $t = 0$ (b) $t = 7.5$ (c) $t = 15$

Figure 2.7: Pointwise absolute error between the high-fidelity solution and the neural network prediction (CCNN) for the linear advection equation with velocity, $\mu_1 = 1.4161$, and initial angle, $\mu_2 = 2.8744$.



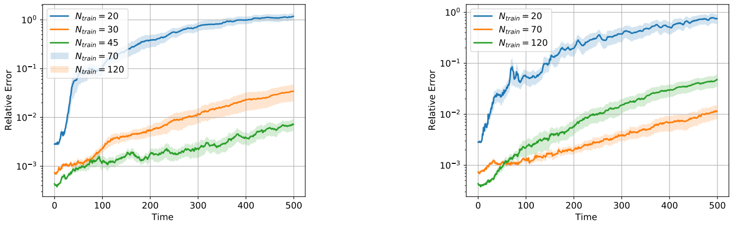
(a) CCNN. (b) LSTM.

Figure 2.8: Comparison of CCNN and LSTM in relative error for each time step in high-fidelity space for the linear advection equation for varying memory, ξ. The error for each time step is a computed average over 15 test cases with the standard error.



(a) CCNN. (b) LSTM.

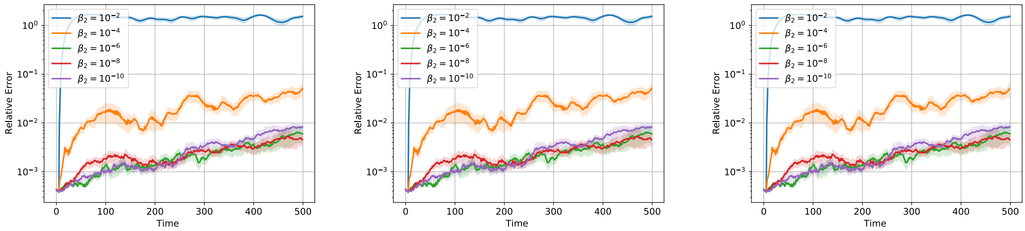
Figure 2.9: Comparison of computing the next step directly and the residual for the CCNN and LSTM. The figures show relative error for each time step in high-fidelity space for the linear advection equation.



(a) CCNN.

(b) LSTM.

Figure 2.10: Comparison of CCNN and LSTM in relative error for each time step in high-fidelity space for the linear advection equation for number of training samples, N_{train}.



(a) $\beta_1 = 10^{-3}$.

(b) $\beta_1 = 10^{-6}$.

(c) $\beta_1 = 10^{-9}$.

Figure 2.11: Impact of the two regularization terms, weight decay, β_1, and Jacobian, β_2 for the linear advection equation. The average relative error in high-fidelity space over 15 test trajectories for each time step is shown. Each figure shows the error for a constant β_1 and varying β_2.

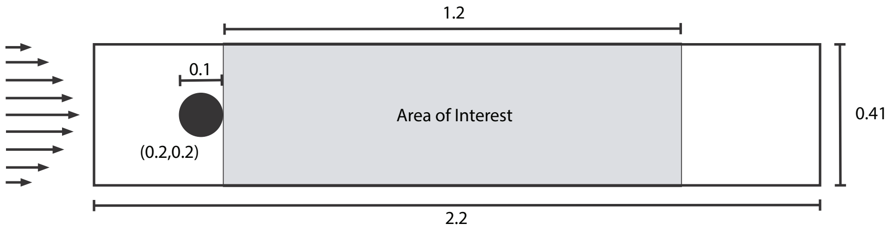


Figure 2.12: The setting for the flow past cylinder problem.

2.5.2 2D Nonlinear Equation - Flow Past Cylinder

We also consider the incompressible Navier-Stokes equations, governing transitional flow, resulting in a complicated flow pattern. The equations are given by:

$$\begin{aligned}
\partial_t u(Re) + (u(Re)\cdot\nabla)u(Re) - \nabla p(Re) &= \frac{1}{Re}\Delta u(Re), && \text{in} \quad \Omega, && (2.40a)\\
\nabla\cdot u(Re) &= 0, && \text{in} \quad \Omega, && (2.40b)\\
u(Re) &= 0, && \text{on} \quad \Gamma, && (2.40c)\\
(u_x(Re), u_y(Re)) &= (1,0) && \text{for} \quad y=0, && (2.40d)
\end{aligned}$$

with initial conditions $u(Re) = 0$ and $p(Re) = 0$. Consider a channel with a cylinder with an inflow at the left side and outflow on the right side, see Figure 2.12 for a visualization of the setting. In the figure, we have also marked the area of interest, as we are interested in the complex flow pattern in the area immediately behind the cylinder. Note that the present methodology can be employed in a specific subdomain of interest. It is not necessary to work on the whole computational domain, as opposed to most high-fidelity methods.

The inflow profile is given by:

$$u(0,y,t) = \left(1.5\frac{4y(0.41-y)}{0.41^2}, 0\right). \tag{2.41}$$

On the walls and the cylinder, no slip conditions are prescribed.

We parameterize the problem by the Reynolds number and consider values in the interval $Re \in [120, 200]$. For Reynolds numbers in this interval, the flow exhibits very interesting nonlinear behavior, such as Karman vortex streets. We compute the velocity as well as the pressure field, meaning we include all relevant physics in the methodology.

The high-fidelity problem is solved here using the finite element method with Taylor-Hood elements and the second-order incremental pressure correction scheme for time stepping [46], implemented in FEniCS. We use 128 elements resulting in 73768 degrees of freedom. The solution is then evaluated at a 300×100 uniform grid.

For the training trajectories, we solve for $t \in [0, 2.5]$, and, for the test trajectories, we compute with $t \in [0, 5]$. This means that we also test how our method performs beyond the training horizon. The high-fidelity model stepsize is 0.0002, resulting in 12500 time steps for the training trajectories and 25000 steps for the test trajectories. For the training of the reduced model, we only use every 5th step, resulting in 2500 time steps, which also means that we use 5000 time steps for the test trajectories.

In Figure 2.15a, we compare the CAE with the POD method for dimensionality reduction for various numbers of training trajectories. Using the CAE, one achieves accuracy of approximately 10^{-4} with a latent dimension of 6. To achieve the same accuracy with POD, one needs a latent space of dimension 76. Furthermore, it is clear that the CAE performs better with more training data until a certain point. However, it is apparent that the error increases when the latent dimension is increased. This phenomena is a result of overfitting or insufficient training, such as convergence to a local minimum.

As for the advection equation, we compare the accuracy with DTR. In Figure 2.15b, we see that CAE+DTR performs well within the interval of the training, but fails to give anything meaningful beyond it. On the other side, CAE+LSTM performs significantly better beyond the training horizon. The CAE+CCNN configuration performs consistently one error of magnitude worse than CAE+LSTM.

In Figure 2.13, we see the velocity magnitude at $t = 25$ in the area of interest. Visually, there is no significant difference between the two, suggesting that the CAE+LSTM approach is able to capture the complicated flow patterns beyond the training time interval. This is further shown in Figure 2.14, where we see the velocity magnitude at two specific points in space. It is clear that there is a small dispersion error as well as small errors in magnitude. However, the general flow pattern is approximated well.

Lastly, we compare how the number of training trajectories affects the accuracy for the CAE+DTR and CAE+LSTM methods in Table 2.2. As expected, the test error decreases with the number of training trajectories. It is further apparent that the CAE+LSTM approach decreases faster, suggesting that this method benefits, to a higher degree, from more data.

Remark - Larger Reynolds Number Intervals. Above, we considered the case with the Reynolds numbers $Re \in [120, 200]$. For larger ranges of Reynolds numbers, the flow in the wake of the cylinder varies more. Specifically, the flow regimes are known to have the following characteristics [39]: $0 < Re < 5$: Steady without a wake; $4 < Re < 40$: Steady symmetric separation; $30 < Re < 90$: Laminar unstable wake; $80 < Re < 300$: Von Karman vortex street; $150 < Re < 1.3 \cdot 10^5$: Vortex street with (turbulent) instabilities. Note that the intervals are overlapping as the exact boundaries between two regimes are unclear.

The neural networks need to have a greater approximation ability in order to capture

N_{train}	20	50	90
CAE + DTR	$1.75 \cdot 10^{-1}$	$1.41 \cdot 10^{-1}$	$6.41 \cdot 10^{-2}$
CAE + LSTM	$4.45 \cdot 10^{-1}$	$1.11 \cdot 10^{-1}$	$3.92 \cdot 10^{-2}$

Table 2.2: Time averaged error for the CAE+DTR andf CAE+LSTM for various number of training trajectories

the dynamics in such different flow regimes. This would require a very large neural network with a large training data set. This can be circumvented by using, instead of a single CAE plus time stepping NN, a conditional approach. By dividing the parameter space into N_I intervals and constructing N_I CAEs with time stepping NNs to be trained on each interval, we made the data fitting an easier task. The offline procedure then involved training N_I CAEs and time stepping NNs and the online procedure included an initial step determining the regime in which the given parameters lie.

The division of the parameter space can be done by utilizing knowledge of the bifurcation diagram. Here, we saw that the NNs are not sensitive to the specific choice of intervals. In this experiment, we divided the parameter space into the following three intervals: $I_1 = [1, 10]$, $I_2 = [10, 65]$, and $I_3 = [65, 120]$. Hence, we tested whether the NNs could approximate the flow across different regimes.

We chose to use the same hyperparameter setting for all N_I networks. The CAEs were trained on 10 trajectories for I_1 and 20 trajectories for I_2 and I_3. The time stepping NNs were trained on 13, 72, and 77 trajectories for I_1, I_2, and I_3, respectively. Testing of the NNs in each interval was done with three trajectories with Reynolds numbers uniformly distributed in the relevant interval.

We compared results for CAE and LSTM with results computed with CAE and DTR. Regarding the convergence of the CAEs for the three Reynolds number intervals, a latent dimension of 4 showed satisfactory results. We found essentially the same results for the three flow regimes, as those presented in the previous subsection for I_4. The CAEs significantly outperfomed POD and LSTMs resulted in a better time stepping scheme that could approximate the flow beyond the training horizon while DTRs failed.

In conclusion, dividing the parameter space into intervals and train NNs in each regime was a feasible solution to the problem of large parameter intervals.

2.5.3 Computation Time and Accuracy

We have showed and discussed performance regarding relative error for the two test cases. The results for the two cases, using the CCNN, LSTM, and DTR, combined with CAE and POD, are summarized in Table 2.3, where the time averaged error is shown. The results for the flow past cylinder case are computed for I_4.

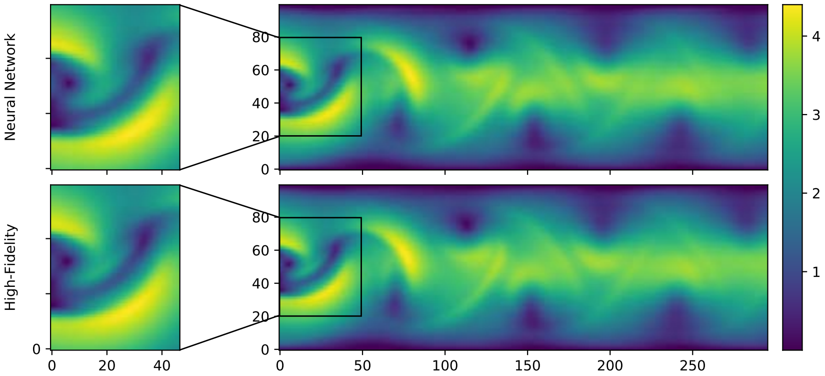
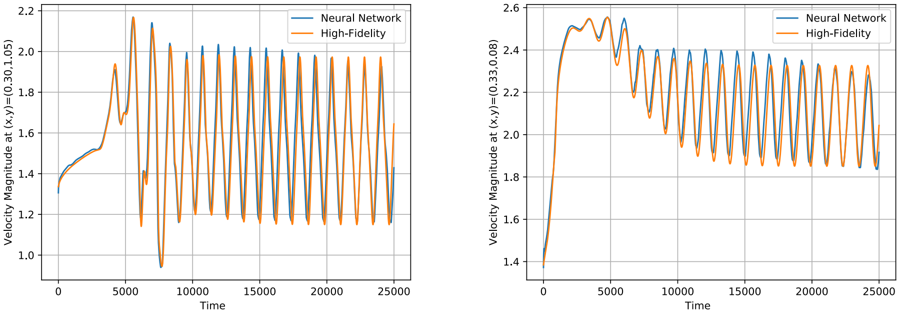
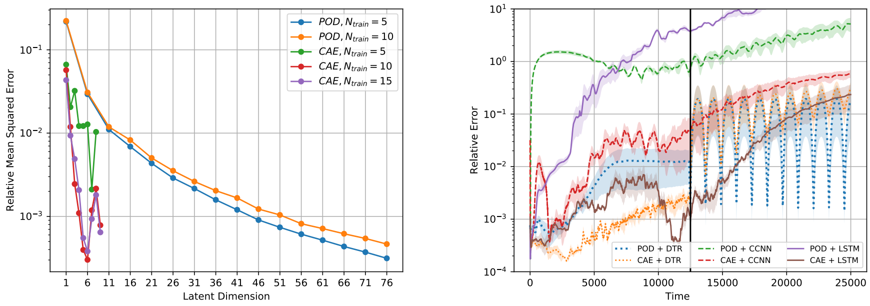


Figure 2.13: Comparison of the velocity at $t = 25$ for $Re = 192$ using CAE+LSTM (top) and the high-fidelity method (bottom).



(a) $(x, y) = (1.5, 0.30)$ (b) $(x, y) = (0.08, 0.33)$

Figure 2.14: A comparison of the velocity magnitude for the flow past cylinder problem at two distinct points in space for $Re = 192$ using CAE+LSTM.



(a) (b)

Figure 2.15: (a) CAE and POD convergence as well as (b) average errors with standard error for the flow past cylinder problem for various Reynolds numbers computed with various combinations of POD, CAE, and regression methods. The vertical black line signifies the end of the training horizon. The CAE based solutions are computed with a latent dimension of 6 and the POD based with a latent dimension of 76.

As mentioned in the introduction, the aim is to be able to compute solutions fast in the online stage. In Table 2.4, the high-fidelity as well as the NN, and DTR online times in seconds are shown. In the online stage, there has not been used any form of parallelization. Therefore, it should be noted that significant speed ups for both the high-fidelity and the regression time stepping approaches could be achieved with a greater effort on this matter. The NN online time and the high-fidelity computation time is computed on an AMD Ryzen 9 3950X CPU. We observe that DTR is computationally faster, due to the much simpler model. However, it was shown above, that DTR was not able to approximate the dynamics well.

Comparing the CAE+LSTM computation with the high-fidelity computation time, we see speed-ups of around 115 times for the advection equation and 1770 times for the flow past cylinder test case. Hence, we see significant speed-ups.

In Table 2.5, the offline time is shown, divided into NN training time and the time it took to generate the training trajectories. In cases where the training trajectories come from collected data, the simulation step is unnecessary, and hence the training time alone is the relevant number. For the training, we used an Nvidia GeForce RTX 3090 GPU. Compared to the online stage, it makes a significant difference to use a GPU instead of a CPU due to the heavy computations associated with backpropagation. We have chosen to only show the GPU training time. It is clear that the most time consuming part is generating the training trajectories.

	CAE			POD		
Test Problem	CCNN	LSTM	DTR	CCNN	LSTM	DTR
Advection Equation	$1.54 \cdot 10^{-3}$	$5.02 \cdot 10^{-3}$	$1.53 \cdot 10^{0}$	$2.02 \cdot 10^{2}$	$2.84 \cdot 10^{1}$	$1.50 \cdot 10^{0}$
Flow Past Cylinder	$1.64 \cdot 10^{-1}$	$3.92 \cdot 10^{-2}$	$6.41 \cdot 10^{-2}$	-	27.389	$5.23 \cdot 10^{-2}$

Table 2.3: Time averaged error for each of the time stepping regression techniques together with CAE and POD respectively for each test problem. The results are for $Re \in [120, 200]$ for the flow past cylinder problem.

		CAE			POD		
Test Problem	High-Fidelity	CCNN	LSTM	DTR	CCNN	LSTM	DTR
Advection Equation	532.22	2.08	4.63	0.03	1.78	3.41	0.02
Flow Past Cylinder	17251.31	7.56	9.74	3.13	5.14	6.57	1.45

Table 2.4: Online computation time in seconds for each of the time stepping regression techniques together with CAE and POD respectively for each test problem.

Test Problem	CCNN	LSTM	Generation of Trajectories
Advection Equation	396.67	622.28	50763.13 (120 trajectories)
Flow Past Cylinder	300.39	985.54	931570.74 (90 trajectories)

Table 2.5: Offline computation time, i.e. NN training time, in seconds for the CCNN and LSTM using GPUs. Furthermore we show the time it took to generate the training trajectories. Note that the generation of training trajectories is not necessary in cases where the data already exists. Furthermore, we have omitted to show the training time for the non-NN regression methods as the training time negligible.

2.6 Conclusion

We presented a novel deep learning approach to non-intrusive reduced order modeling for parameterized time-dependent PDEs using CAEs for dimensionality reduction and CCNNs and LSTMs combined with FFNNs for time evolution. This approach was demonstrated on two test cases and was shown to perform well in the online phase, showcasing the potential of using deep learning based ROMs for different physical phenomena.

Regarding dimensionality reduction, a discussion and comparison of linear and nonlinear methods was presented with POD and CAEs as the focal points. The discussion focused on why a nonlinear approach has the potential to outperform a linear approach.

For time stepping, the general approach was to encode the previous states and the parameters separately in parallel and then combine the encoded data to make a final prediction using an FFNN. The two encoding NNs, as well as the final prediction NN, constitute a single network, meaning all parts are trained simultaneously. This ensures that both the memory and parameters are encoded in relation to one another. Furthermore, various methods to ensure generalization, stability, and precision were discussed and tested.

For the advection equation, the CAE+CCNN approach performed very well with errors below 10^{-2} for all time steps, while the CAE+LSTM performed similarly, but slightly worse. Interestingly, the alternative approaches, using POD instead of CAE and DTR for time stepping, failed to approximate the dynamics in any meaningful way. Furthermore, as expected, the CAE reached much better precision than POD for dimensionality reduction with much fewer dimensions in the latent space.

Secondly, a more involved problem, flow past a cylinder, was also studied. Here, we were dealing with multiple vector fields and complicated nonlinear patterns. Furthermore, we tested how the methodology performs beyond the training horizon. We saw that the CAE+LSTM approach showed errors below 10^{-2} within the training horizon, and a slow increase in error beyond the horizon. However, the increase in error was primarily due to small phase errors, meaning the overall structure of the flow still resembles the high-fidelity flow. When using DTR, either with POD or the CAE, the approximations completely failed beyond the training horizon, suggesting more complicated models are needed to actually learn the time stepping map.

Lastly, we discussed an approach to deal with large parameter intervals that give rise to highly varying flow regimes. Namely, training a CAE and time stepping NN on subdomains and use the NNs corresponding to the subdomain it was trained on. We conclude that for the flow past cylinder case this approach is successful.

In summary, the contributions in this work include a nonlinear dimensionality reduction scheme using convolutional autoencoders, a novel parallel neural network architecture for parameterized time stepping using CCNNs and LSTMs, and a discussion on different approaches to achieve stability and generalization for neural network-based time stepping. It is furthermore worth mentioning that the framework presented allows for flexibility in shape of replacing certain elements with alternatives. E.g. one could replace the CCNN or LSTM with another choice if needed.

In the future, the methodology should be tested on more advanced PDE problems. By advanced problems, we are both referring to increasing nonlinearity, higher dimensions, and multi-query problems such as uncertainty quantification, model predictive control, and data assimilation. Especially, data assimilation seems a promising direction, since incorporating data could rectify the phase errors.

Besides considering other use cases, one could work on improving the NN architecture and training by, e.g. incorporating the physics in the training [32, 124], and use reinforcement learning [148] to ensure effective snapshot generation. Furthermore, with the amount of hyperparameters (ξ, β_1, β_2, number of layers and neurons, etc.) the task of hyperparameter tuning is not trivial and could potentially be solved more effectively using alternative approaches.

3

Markov chain generative adversarial neural networks for solving Bayesian inverse problems in physics applications

In the context of solving inverse problems for physics applications within a Bayesian framework, we present a new approach, the Markov Chain Generative Adversarial Neural Network (MCGAN), to alleviate the computational costs associated with solving the Bayesian inference problem. GANs pose a very suitable framework to aid in the solution of Bayesian inference problems, as they are designed to generate samples from complicated high-dimensional distributions. By training a GAN to sample from a low-dimensional latent space and then embedding it in a Markov Chain Monte Carlo method, we can highly efficiently sample from the posterior, by replacing both the high-dimensional prior and the expensive forward map. This comes at the cost of a potentially expensive offline stage in which training data must be simulated or gathered and the GAN has to be trained. We prove that the proposed methodology converges to the true posterior in the Wasserstein-1 distance and that sampling from the latent space is equivalent to sampling in the high-dimensional space in a weak sense. The

---

This chapter is based on the article “Markov chain generative adversarial neural networks for solving Bayesian inverse problems in physics applications”, *Computers & Mathematics with Applications*, 147:278-299, 2023 [107].

method is showcased on two test cases where we perform both state and parameter estimation simultaneously and it is compared with two conventional approaches, polynomial chaos expansion and ensemble Kalman filter, and a deep learning-based approach, deep Bayesian inversion. The method is shown to be more accurate than these alternative approaches while also being computationally faster, in multiple test cases, including the important engineering setting of detecting leaks in pipelines.

3.1 Introduction

The Bayesian inference approach is popular for solving inverse problems in various fields including physics and engineering [7, 59, 78, 147], mainly due to the fact that it does not only provide an estimate of the solution but also quantifies the uncertainty of the estimate. Information about the distribution of a computed quantity is important, for example, for digital twins [79].

The general idea of Bayesian inference is to use observations to update a given prior distribution towards a resulting posterior distribution over the parameters of interest. The observations and parameters are linked through a forward map and a noise distribution that make up the likelihood function. The main task in the Bayesian approach is to connect the prior and the likelihood in order to compute the posterior distribution. Since the posterior is typically not analytically tractable, one must use numerical sampling techniques such as Monte Carlo methods to approximate the distribution. However, for each sample, it is necessary to compute the likelihood which in turn requires the evaluation of the forward map. For nontrivial problems, such as high-dimensional or nonlinear partial differential equation (PDE) problems, this becomes a computational bottleneck and often results in unacceptable computation times. In Figure 3.1, the general schematics of an inverse problem are shown.

The two most common approaches for overcoming this problem are to either minimize the required number of samples by making certain assumptions about the posterior or to reduce the computational complexity associated with the forward map by approximating it with a surrogate model. The first approach includes methods such as Kalman filters [59] and Markov Chain Monte Carlo (MCMC) methods [16, 42]. With Kalman filters, one minimizes the number of necessary samples by assuming Gaussian distributions. While this is efficient, it is often quite restrictive when it comes to highly nonlinear problems. MCMC methods, while being quite efficient, are based on fewer assumptions but still require many samples. See Figure 3.1 for a visualization of a common workflow for Bayesian inference using Markov chain Monte Carlo methods (MCMC). The surrogate modeling approach includes methods such as reduced basis methods [123], polynomial chaos expansion (PCE) [164], and Gaussian processes [159]. While a surrogate model enables fast likelihood evaluations, it requires a forward map that can be approximated by a low-order representation. This is however not trivial for problems with a so-called slow Kolmogorov n-width decay, such as very high-dimensional problems and problems involving discontinuities in either the parameters or the state.

In this chapter, we consider an approach that overcomes the above mentioned challenges (high-dimensionality, nonlinearity, discontinuities, expensive sampling) by utilizing machine learning. Specifically, we will make use of neural networks which have already been recognized as promising tools in scientific computing, especially for the case of high-dimensional and nonlinear problems that we wish to address [9, 19, 54, 65, 77, 91, 105]. While there exist several types of neural networks, each aiming at solving specific problems, we focus on generative models here. Generative models aim to learn a distribution from data in order to enable sampling from it at later times [135]. Such models include generative adversarial networks (GANs) [52], variational autoencoders (VAEs) [81], diffusion models [26], and Normalizing Flow models [129].

Examples where generative models have been successfully used for solving Bayesian inverse problems already exist. In [47] and [161], a VAE and Normalizing Flow, respectively, are embedded into a variational Bayesian inference approach and in [118] and [162] a GAN and a VAE, respectively, are used as the prior distribution in MCMC sampling. While [118] and [162] combine generative models for parameter prior approximation with MCMC sampling in order to get samples from the posterior distribution, they do not achieve significant speed-ups. Since the generative models are only used to approximate the parameter prior, they still need the expensive forward problem being solved to match synthetic observations with the real observations. Furthermore, in [2] a GAN has been trained to directly sample from the posterior distribution. This is done by using a conditional GAN that generates samples conditioned on the observations. This approach achieves speed-ups as MCMC sampling is bypassed, but is restricted to the sensor configuration used for training. That is, the observation operator must be chosen when training in order to form the training set and cannot be changed without changing the architecture of the neural network and retraining it.

We will focus on GANs due to their success in learning complicated high-dimensional distributions. When choosing a generative model, there are essentially three aspects to consider [163]: Quality of samples, sampling speed, and mode coverage. VAEs generally generate lower quality samples than GANs, as they tend to blur the samples. Diffusion models [144], on the other hand, generate high quality samples, but they are significantly slower than both VAEs and GANs and are therefore not suitable for solving inverse problems in real-time. While the original GAN was known to suffer from mode collapse, it has been shown that the extension, the Wasserstein GAN (WGAN) [5], overcomes this issue to a large extent. Furthermore, the GAN training is more stable but it comes at a cost of computational time. As the training takes place in the offline stage, this is not a serious problem.

Specifically, GANs learn a target distribution by training a generator to map latent space samples to samples that mimic a nontrivial high-dimensional target distribution. So, GANs provide a way to represent a complicated high-dimensional distribution by means of a low-dimensional latent space distribution.

We here present the novel Markov Chain Generative Adversarial Network (MCGAN) method, visualized in Figure 3.1. In short, we train a GAN to approximate the prior distribution for the states and parameters and thereby obtain a corresponding latent representation. By using an MCMC method, we can then efficiently sample from a latent space posterior instead of the high-dimensional posterior. As a result, we achieve dimensionality reduction, due to the approximation of the desired posterior, and furthermore the forward map is replaced by the generator. In practice, this gives significant computational speed-ups as the computational bottleneck is significantly reduced. The methodology presented draws inspiration from [118], but utilizes the GAN in a different manner. Our extension is hence well-suited for both state and parameter estimation in real-time. Furthermore, we prove that sampling in the latent space is the same as sampling in the high-dimensional space in a weak sense and we provide a proof of convergence of the posterior distribution in the Wasserstein-1 distance.

This chapter's outline is as follows. In Section 3.2, we explain the setting of Bayesian inverse problems as well as the MCMC methods and GANs. Then, in Section 3.3, we present the details of our proposed methodology, the MCGAN methodology, including the theoretical findings. In Section 3.4 we briefly discuss alternative methods. Lastly, in Section 3.5, we show the MCGAN performance on two problems: a stationary Darcy flow and leakage localization in a pipe flow. The results are compared to ensemble Kalman filters, MCMC methods with PCE as the surrogate model and the likelihood-free deep Bayesian inversion.

3.2 Notation, Problem Setting, and Preliminaries

Throughout this chapter, we will make use of the following notation: capital letters will denote random variables, e.g. X and Y. The distribution of X is denoted P_x, where $P_x(A) = P_x(X \in A)$ is the probability of observing $X \in A$. Similarly, the probability of x is denoted $P_x(x) = P_x(X = x)$. We assume that all distributions have a probability density function (PDF), ρ_x. A stochastic variable, X, conditioned on another stochastic variable, Y, is denoted $X|Y$ and is distributed according to $P_{x|y}(X|Y)$ with PDF, $\rho_{x|y}$.

3.2.1 Problem Setting

Let $\mathbf{q} \in \mathbb{R}^{N_q}$ denote the state, $\mathbf{m} \in \mathbb{R}^{N_m}$ the parameters, and $\mathbf{y} \in \mathbb{R}^{N_y}$ the available observations. Note that $\mathbf{q}$ encapsulates the state at all discrete times for time-dependent

---

Note that "real-time" is dependent on the specific problem at hand

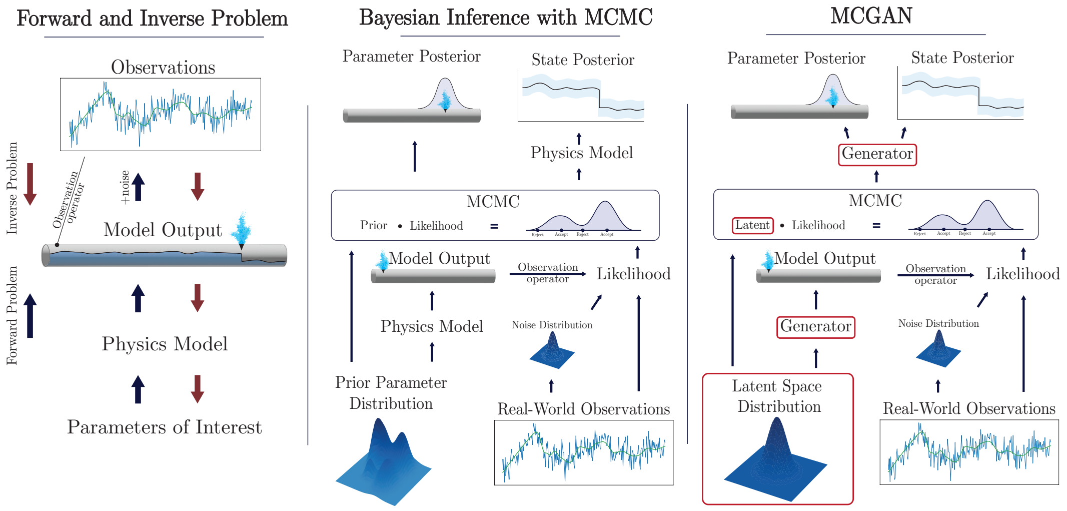


Figure 3.1: Left: overview of the forward problem and the inverse problem. The parameters of interest are typically boundary and/or initial conditions, or physical parameters. The physics model depends on the system at hand and is here a PDE modeling pipe flow. The model output is the result obtained from a numerical simulation, such as pressure or velocity in the case of fluid dynamics. Observations are either observed from a set of sensors or created synthetically from the model output through the observation operator. Middle: a typical approach for doing Bayesian inference with MCMC (see Section 3.2). Right: our proposed method, the MCGAN approach as explained in section 3.3. Note that the complicated prior distribution is replaced with a simple latent distribution. Furthermore, the physical model is replaced with a generator that enables us to evaluate the full forward problem, more or less, instantaneously.

problems. Hence, the state, $\mathbf{q}$, is the full space-time state of the system at hand. $\mathbf{q}$ is computed by solving a forward problem, typically a PDE, depending on the parameters, $\mathbf{m}$. We denote the vector of combined state and parameters, $\mathbf{u} = (\mathbf{q}, \mathbf{m}) \in \mathbb{R}^{N_u}$, $N_u = N_q + N_m$.

The inverse problem deals with the recovery of $\mathbf{u}$ from a vector of observations in space-time, $\mathbf{y} \in O \subset \mathbb{R}^{N_y}$. By setting up the inverse problem in both state and parameters, we allow for the case where the state is not necessarily determined by the parameters of interest alone. The relation between $\mathbf{u}$ and $\mathbf{y}$ is assumed to be of the form

$$\mathbf{y} = \mathbf{h}(\mathbf{u}) + \eta, \quad \eta \sim P_\eta, \quad \eta \in \mathbb{R}^{N_y}, \tag{3.1}$$

where $\mathbf{h} : \mathbb{R}^{N_u} \to O \subset \mathbb{R}^{N_y}$ is referred to as the observation operator and η is a random variable denoting the observation or measurement noise.

From Eq. (3.1), we can write the PDF associated with the probability of observing $\mathbf{y}$

given $\mathbf{u}$, $\rho_{y|u}(\mathbf{y}|\mathbf{u})$, as:

$$\rho_{y|u}(\mathbf{y}|\mathbf{u}) = \rho_\eta(\mathbf{y} - \mathbf{h}(\mathbf{u})). \tag{3.2}$$

When observations are given, one can view this as a function of $\mathbf{u}$, i.e., $\Phi(\mathbf{u}) = \rho_{y|u}(\mathbf{y}|\mathbf{u})$, in which case it is referred to as the *likelihood* since it is not a PDF with respect to $\mathbf{u}$.

We assume that, before observing any data, the probability of $\mathbf{u}$ has the PDF ρ_0, which is referred to as the *prior*. The goal of the Bayesian inverse problem is to identify the PDF, $\rho_{u|y}(\mathbf{u}|\mathbf{y})$, i.e. the PDF of $\mathbf{u}$ given observations, $\mathbf{y}$. Using Bayes theorem, we can write this as:

$$\rho_{u|y}(\mathbf{u}|\mathbf{y}) = \frac{\rho_{y|u}(\mathbf{y}|\mathbf{u})\rho_0(\mathbf{u})}{\int_{\mathbb{R}^{N_u}} \rho_{y|u}(\mathbf{y}|\mathbf{u})\rho_0(\mathbf{u})\,\mathrm{d}\mathbf{u}} = \frac{\rho_\eta(\mathbf{y} - \mathbf{h}(\mathbf{u}))\rho_0(\mathbf{u})}{\int_{\mathbb{R}^{N_u}} \rho_{y|u}(\mathbf{y}|\mathbf{u})\rho_0(\mathbf{u})\,\mathrm{d}\mathbf{u}}. \tag{3.3}$$

The denominator is called the *evidence* and serves as a normalization constant; the lefthand side is the *posterior*.

However, in order to compute the likelihood in Eq. (3.3), a PDE must be solved for a given set of parameters. Moreover, choosing a suitable prior is not always an easy task, and the evidence can be restrictive to compute in high dimensions.

It should be noted that the last problem is alleviated in many methods such as maximum likelihood estimation and MCMC methods, as we will describe below. The other two complications will be minimized using our proposed methodology.

3.2.2 Markov Chain Monte Carlo Methods

MCMC methods [16, 42] form a class of algorithms for sampling from probability distributions. Stated in terms of the posterior PDF, we have:

$$\rho_{u|y}(\mathbf{u}|\mathbf{y}) \propto \rho_{y|u}(\mathbf{y}|\mathbf{u})\rho_0(\mathbf{u}), \tag{3.4}$$

and we aim to generate a set of points distributed according to the PDF $\rho_{u|y}$.The general approach is to construct a Markov chain, $\{\mathbf{u}_1, \dots, \mathbf{u}_N\}$, with a stationary PDF, $\tilde{\rho}_{u|y}$, that approximates $\rho_{u|y}$. We then sample according to $\tilde{\rho}_{u|y}$ by computing the next element in the chain. For MCMC algorithms, we have the following result, under some reasonable assumptions [16]:

$$\lim_{N_{mcmc}\to\infty} \frac{1}{N_{mcmc}} \sum_{i=1}^{N_{mcmc}} f(\mathbf{u}_i) = \mathbb{E}_{\mathbf{u}\sim P_{u|y}}[f(\mathbf{u})], \quad \mathbf{u}_i \sim \tilde{P}_{u|y} \tag{3.5}$$

where $\tilde{P}_{u|y}$ is the probability distribution associated with the density $\tilde{\rho}_{u|y}$. Eq. (3.5) indicates that with enough samples from the chain, we can approximate some statistics of the true posterior arbitrarily well, i.e. the distribution, $\tilde{P}_{u|y}$, converges weakly to $P_{u|y}$.

The arguably most common MCMC sampler is the Metropolis-Hasting (MH) algorithm [61, 102]. However, it is well-known that the MH algorithm converges very slowly in high-dimensional settings. Therefore, we make use here of the Hamiltonian Monte Carlo Method (HMC), which can be considered a special case of the MH algorithm. Instead of computing new proposals by a random walk, the HMC algorithm computes a new sample by moving in a state space defined by a Hamiltonian ODE system.

Starting with a 'momentum' vector, $\mathbf{p}$, of the same size as $\mathbf{u}$, and a joint PDF $\rho_{u,p|y}(\mathbf{u}, \mathbf{p}|\mathbf{y})$, we define the Hamiltonian as:

$$\begin{aligned}\mathcal{H}(\mathbf{u}, \mathbf{p}) &= -\log \rho_{u,p|y}(\mathbf{u}, \mathbf{p}|\mathbf{y}) = -\log \rho_{p|u}(\mathbf{p}|\mathbf{u}) - \log \rho_{u|y}(\mathbf{u}|\mathbf{y}) \\ &\propto \underbrace{\frac{1}{2}\mathbf{p}^T M^{-1}\mathbf{p}}_{=K(\mathbf{p})} - \underbrace{\log\left[\rho_{y|u}(\mathbf{y}|\mathbf{u})\rho_0(\mathbf{u})\right]}_{=U(\mathbf{u})},\end{aligned} \tag{3.6}$$

where we choose the conditional distribution of the momentum given $\mathbf{u}$ to be normally distributed, $P_{p|u}(\mathbf{p}|\mathbf{u}) \sim \mathcal{N}(0, M)$. $K(\mathbf{p})$ is referred to as the kinetic energy and $U(\mathbf{u})$ the potential energy. One can compute trajectories on level sets of the Hamiltonian by solving the Hamiltonian dynamical system. A new sample is then computed by perturbing the current sample, integrating the Hamiltonian system in time and using the final state as the new sample with acceptance probability:

$$\alpha = \min\left\{1, \frac{\exp\left(-\mathcal{H}(\mathbf{u}', \mathbf{p}')\right)}{\exp\left(-\mathcal{H}(\mathbf{u}_i, \mathbf{p}(0))\right)}\right\}, \tag{3.7}$$

where $(\mathbf{u}', \mathbf{p}')$ is the terminal state of the trajectory. Intuitively, this procedure will be biased towards sampling from level sets in the phase space that maximize the likelihood $U(\mathbf{u})$. Furthermore, $K(\mathbf{p})$ ensures that the algorithm explores other areas of the phase space to a degree decided by M and the integration horizon, T. Compared to the standard MH algorithm, this reduces the correlation between elements in the chain by traversing long distances in the phase space while maintaining a high acceptance probability due to the energy preserving properties of Hamiltonian dynamics.

When sampling using the HMC algorithm, a series of choices have to be made, like the number of time steps in the integration and the end time, T. If T is too small, the sampling will resemble a random walk, while T too large may result in trajectories making a 'U-turn' and return to their initial condition. To avoid this, we utilize the No U-Turn Sampler (NUTS) [69].

The approach is to integrate backward and forward in time until a U-turn condition is satisfied. Then, a random point from the computed trajectory is chosen, and the algorithm continues from there.

It is important to emphasize that the HMC algorithm can only be utilized when the likelihood and the prior are differentiable. Furthermore, the derivative should be cheap to compute to get the desired speed-up.

Even though HMC with NUTS is efficient, one still needs many samples to converge. With a good initial sample, the method converges significantly faster. There are several ways of computing a suitable initial guess, one of which is the maximum a posteriori (MAP) estimate [78], which we will use in this work. This is typically computed using the log PDFs:

$$\mathbf{u}_{\text{MAP}} = \arg\max_{\mathbf{u}} \log(\rho_{y|u}(\mathbf{y}|\mathbf{u})) + \log(\rho_0(\mathbf{u})). \tag{3.8}$$

$\mathbf{u}_{\text{MAP}}$ can be computed using standard optimization methods such as gradient descent methods. In our case, both $\rho_{y|u}$ and ρ_u are known PDFs so it is easy to compute derivatives using standard software libraries such as PyTorch.

3.2.3 Generative Adversarial Neural Networks

In this section, we will give a brief overview of generative adversarial networks (GANs), see [52, 74] for more details. We will focus on a version of GANs called Wasserstein GAN (WGAN) [5].

GANs deal with the problem of learning an unknown distribution from samples. Consider a probability distribution, P_u^r, on a data space which is a subset of $\mathbb{R}^m$. We aim to approximate P_u^r with another distribution, P_u^g. We will refer to P_u^r as the real data probability distribution or the target distribution, and P_u^g the generated distribution. In order to compute P_u^g, we define a stochastic latent variable, $Z \in \mathbb{R}^{N_z}$, with prior distribution P_z^g, typically chosen to be a Gaussian. Then, we define a generator, $G_\theta : \mathbb{R}^{N_z} \to \mathbb{R}^{N_u}$, which is a neural network parameterized by its weights, θ. G_θ takes in the latent variable and outputs $G_\theta(Z) \sim P_u^g$. Hence, $P_u^g = G_{\theta\#} P_z^g$ is the *pushforward* of the latent space distribution with PDF $\rho_u^g = \rho_z^g \circ G^{-1}$ [15]. By choosing $N_z \ll N_u$, we effectively get a low-dimensional representation of the N_u-dimensional distribution. Therefore, the variable z can be considered a latent/low-dimensional representation of samples from P_u^r.

Next, we introduce the discriminator, $D_\omega : \mathbb{R}^{N_u} \to \mathbb{R}$. The discriminator takes in samples from either the real data probability distribution or the generated probability distribution, and returns a real number called the *score*. A large score means that the discriminator believes the sample comes from the real data distribution. D_ω is a neural network parameterized by its weights, ω.

In order to learn the target distribution, a zero-sum game between the generator and the discriminator is set up. The generator aims to maximize the discriminator output, while the discriminator tries to minimize the score of generated samples while simultaneously trying to maximize the score of the real samples. For the WGAN, this game is mathematically formulated as [5]:

$$\inf_{\theta} \sup_{\omega} \quad \mathbb{E}_{X\sim P_u^r}\left[D_\omega(X)\right] - \mathbb{E}_{Z\sim P_z^g}\left[D_\omega(G_\theta(Z))\right]. \tag{3.9}$$

It can be shown that this inf-sup problem is equivalent to minimizing the Wasserstein-1 distance between P_u^r and P_u^g due to the Kantorovich-Rubinstein duality [5]. The WGAN framework requires the discriminator to be Lipschitz continuous with respect to the input. Therefore, we introduce a gradient penalty term to constrain the gradient of the discriminator [55]:

$$\inf_{\theta} \sup_{\omega} \quad \mathbb{E}_{X\sim P_u^r}\left[D_\omega(X)\right] - \mathbb{E}_{Z\sim P_z^g}\left[D_\omega(G_\theta(Z))\right] - \lambda \mathbb{E}_{\hat{\mathbf{x}}\sim P_{\hat{X}}}\left[\left(||\nabla_{\hat{X}} D_\omega(\hat{X})|| - 1\right)^2\right], \tag{3.10}$$

where λ is a regularization parameter to be tuned, $\hat{X} = \epsilon X + (1-\epsilon)G_\theta(Z)$, and ϵ is a small positive number.

In practice, we do not update the weights of the generator and the discriminator at the same time. Instead, we split (3.10) into two subproblems: a generator loss, L_G, that aims to minimize (3.10) and a discriminator loss, L_D, that aims to maximize (3.10) by minimizing the negative value:

$$L_G = -\mathbb{E}_{Z\sim P_z^g}\left[D_\omega(G_\theta(Z))\right], \tag{3.11}$$

$$L_D = -\mathbb{E}_{X\sim P_u^r}\left[D_\omega(X)\right] + \mathbb{E}_{Z\sim P_z^g}\left[D_\omega(G_\theta(Z))\right] + \lambda \mathbb{E}_{\hat{X}\sim P_{\hat{X}}}\left[\left(||\nabla_{\hat{X}} D_\omega(\hat{X})|| - 1\right)^2\right]. \tag{3.12}$$

In the WGAN framework, it is important to properly train the discriminator. Therefore, it is common practice to update the discriminator parameters more frequently than the generator parameters. The WGAN is visualized in Figure 3.2.

It has been shown that if the generator and the discriminator have sufficient capacity, the generated distribution converges to the real data probability distribution in the Wasserstein-1 distance [93].

3.3 Markov Chain GAN

In this section, we will outline our proposed method, the Markov Chain GAN (MCGAN) method. The general purpose is to combine MCMC methods with GANs in order to perform state and parameter estimation in a computationally fast and accurate way. As mentioned in the previous section, similar approaches exist using polynomial surrogate models [96, 137]

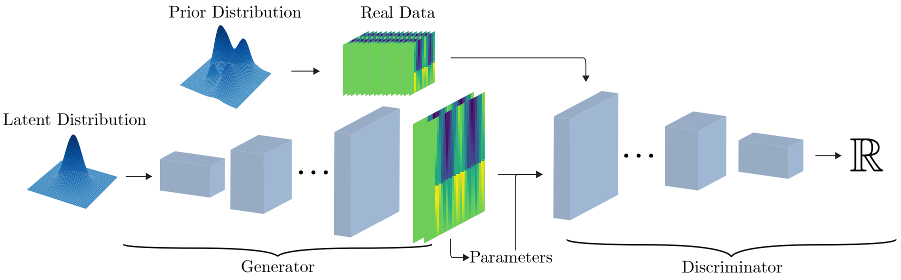


Figure 3.2: GAN architecture.

and Gaussian processes [159]. However, as will be discussed, using GANs gives significant advantages over these alternatives.

3.3.1 Proposed Algorithm

In short, the proposed algorithm aims to speed up posterior sampling without compromising too much on accuracy. The general methodology is to replace the forward model in the likelihood computation with the generator and replace the data prior with the GAN latent distribution (see Figure 3.2).

Firstly, in an offline stage, we train the GAN to generate discrete solutions to the PDE for the desired time span and corresponding parameters. The GAN is trained on samples from the real prior distribution, P_0^r, i.e. solutions computed through conventional numerical methods (finite elements, finite volumes, etc.) and aims to learn a generated prior distribution, P_0^g. Samples from P_0^r are typically computed by sampling the parameters and then solving the physical model to get the state. This can be a lengthy process, but the training data can be simulated completely in parallel on several computer cores. After training, the (single) generator may generate pairs of states and parameters from a latent sample, $\mathbf{z}$:

$$G_\theta(\mathbf{z}) = (G_\theta^q(\mathbf{z}), G_\theta^m(\mathbf{z})) = (\mathbf{q}^g, \mathbf{m}^g) = \mathbf{u}^g \sim P_0^g \tag{3.13}$$

Hence, the generated distribution is approximating the real prior distribution, $P_0^g \approx P_0^r$. In the training, the discriminator will receive pairs of states and parameters, $(\mathbf{q}^r, \mathbf{m}^r)$, sampled from the real data prior and generated pairs of states and parameters, $(\mathbf{q}^g, \mathbf{m}^g)$, sampled from the generated distribution, in order to ensure that the generator learns to generate states and parameters that match.

Remark It should be noted that since the GAN is trained on discrete solutions on a specific grid, it will generate samples on the same grid. Therefore, it is important to train on discrete solutions that have all the necessary properties for the online phase (sufficient resolution, etc.)

In order to take advantage of the low-dimensional latent space, we need to be able to sample solutions and parameters from a posterior on the latent space, instead of the generated distribution in the full data space. To this end, we have the expression for the latent space posterior density:

$$\rho^g_{z|y}(\mathbf{z}|\mathbf{y}) = \frac{\rho^g_{y|u}(\mathbf{y}|G_\theta(\mathbf{z}))\rho^g_z(\mathbf{z})}{\int_{\mathbb{R}^{N_z}} \rho^g_{y|u}(\mathbf{y}|G_\theta(\mathbf{z}))\rho^g_z(\mathbf{z})\,\mathrm{d}\mathbf{z}} = \frac{\rho_\eta(\mathbf{y}-\mathbf{h}(G_\theta(\mathbf{z})))\rho^g_z(\mathbf{z})}{\int_{\mathbb{R}^{N_z}} \rho^g_{y|u}(\mathbf{y}|G_\theta(\mathbf{z}))\rho^g_z(\mathbf{z})\,\mathrm{d}\mathbf{z}} \propto \rho_\eta(\mathbf{y}-\mathbf{h}(G_\theta(\mathbf{z})))\rho^g_z(\mathbf{z}). \tag{3.14}$$

By training the generator to generate pairs of states and parameters, there is no need for the expensive forward model, as it is replaced by an evaluation of the generator at the sampled $\mathbf{z}$. This means that in the online stage, we only have to evaluate the GAN once in order to get both the state and parameters. This is in contrast to "conventional" approaches where a surrogate model only approximates the forward map or the parameters. In that case, one would have to make use of two surrogates – one for the parameters and one for the forward map. Furthermore, it also takes into account the slightly more general case where there is not necessarily a deterministic relationship between the parameters and the state.

This approach yields a significant speed-up in online computation time since evaluating the generator is, more or less, instantaneous and we obtain the posterior over both the state and parameters at no extra cost. The derivation of (3.14) is given in Section 3.3.2 in Theorem 1.

In the online stage, we then use the MCMC method, as discussed in Section 3.2.2, to sample from the latent space posterior. We can make use of the highly efficient HMC algorithm, since derivatives of the likelihood are easily obtained through backpropagation of the neural network generator. For conventional numerical methods, this is rarely the case as computing derivatives of the forward model typically requires expensive numerical approximations or adjoint methods. Furthermore, the MAP estimate is also cheap to compute for the same reasons, which provides an excellent starting point for the HMC algorithm. Since the dimension is lowered significantly, and derivatives and an appropriate starting point are cheaply available, the Markov chain will converge significantly faster and we can get by with much fewer samples. In conclusion, with the MCGAN methodology each sample is cheap and we need fewer samples than for conventional methods.

Given the MCMC samples, we can compute derived quantities, e.g. for a given quantity of interest f, we can compute the expected value by:

$$\begin{aligned}\mathbb{E}_{\mathbf{q}\sim P^r_{q|y}}[f(\mathbf{q})] &\approx \mathbb{E}_{\mathbf{q}\sim P^g_{q|y}}[f(\mathbf{q})] \\ &= \mathbb{E}_{\mathbf{z}\sim P^g_{z|y}}[f(G^q_\theta(\mathbf{z}))]\end{aligned}$$

$$
\begin{aligned}
&\approx \frac{1}{N_{\text{MCMC}}} \sum_{i=1}^{N_{\text{MCMC}}} f(G_\theta^q(\mathbf{z}_i)), \quad \mathbf{z}_i \sim \tilde{P}_{z|y}^g, \\
\mathbb{E}_{\mathbf{m}\sim P_{m|y}^r}[f(\mathbf{m})] &\approx \mathbb{E}_{\mathbf{m}\sim P_{m|y}^g}[f(\mathbf{m})] \\
&= \mathbb{E}_{\mathbf{z}\sim P_{z|y}^g}[f(G_\theta^m(\mathbf{z}))] \\
&\approx \frac{1}{N_{\text{MCMC}}} \sum_{i=1}^{N_{\text{MCMC}}} f(G_\theta^m(\mathbf{z}_i)), \quad \mathbf{z}_i \sim \tilde{P}_{z|y}^g,
\end{aligned}
$$

By choosing an appropriate f, we can thereby compute various quantities of interest by sampling the latent space posterior. See Theorem 1 in Section 3.3.2 for details.

For an overview of the methodology, see Algorithm 4 for the offline stage and Algorithm 5 for the online stage. Here is a summary of the distinct advantages of the proposed method compared to the alternatives discussed in Section 3.4:

- The latent vector $\mathbf{z}$ is, in general, of significantly lower dimension than the state and parameters, effectively reducing the dimension of the stochastic space, resulting in significantly faster convergence of MCMC methods;
- The computationally expensive forward problem is replaced by the generator, whose cost is computationally negligible to evaluate once it has been trained;
- Since the forward map is replaced by a neural network, derivatives of the log-likelihood function can be computed efficiently, which enables computationally fast MAP estimation and allows us to utilize the highly efficient HMC method for sampling.

While the advantages are clear, it is worth mentioning the drawbacks as well:

- There is no immediate way of choosing the dimension of the latent space. However, one can consider it a hyperparameter and perform hyperparameter optimization;
- Training a GAN is not always an easy task, since commonly known problems of training neural networks, such as local minima and generalization, also apply here;
- It is necessary to generate much training data in order to ensure accuracy of the GAN.

Note that the drawbacks are not unique to this methodology, but general when dealing with neural networks. The first two points are a matter of hyperparameter tuning, and the last point is a matter of time in the offline stage. Furthermore, with an efficient numerical solver and the fact that the offline stage can be easily parallellized (since the training samples are independent), the generation of data is often feasible within a reasonable timeframe. If the forward solver would be too expensive to allow for this, it is recommended to first obtain a simpler forward model e.g. by model reduction techniques such as reduced order models.

Remark The purpose of the proposed methodology is to solve Bayesian inverse problems computationally fast in an online stage. As mentioned, this comes at a cost of an expensive offline stage in which the GAN is trained on simulated data. However, when the GAN is trained, it can be deployed in several settings. Therefore, the method is highly suitable in settings where computational speed is crucial and offline training time is less important. This is, for example, the case for digital twins and model predictive control where repeated real-time state estimation and parameter calibration are necessities.

```
Algorithm 4: MCGAN offline stage
  Input: $N_{\text{train}}$, GAN hyperparameters, GAN architecture
  Generate training samples, $\{(\mathbf{q}_i, \mathbf{m}_i)\}_{i=1}^{N_{\text{train}}} \sim P_0^r$, by solving the forward problem;
  Train the GAN to approximate the prior, $P_0^g \approx P_0^r$ (see Section 3.2.3);
  Output: Trained generator, $G_\theta$
```

```
Algorithm 5: MCGAN online stage
  Input: Generator from Algorithm 4, MCMC parameters, observations ($\mathbf{y}$),
  Compute the MAP estimate in the latent space, using a gradient descent algorithm,
    as initial sampling point (see Eq. 3.8);
  Use MCMC algorithm (see Section 3.2.2) to sample from the latent space posterior
    (see Eq. (3.14));
  Generate states and parameters from the posterior latent space samples:
    $\{\mathbf{q}_i, \mathbf{m}_i\} = \{G_\theta^q(\mathbf{z}_i), G_\theta^m(\mathbf{z}_i)\} = \{G_\theta(\mathbf{z}_i)\}, \quad i = 1, \dots, N_{\text{samples}} \quad \mathbf{z}_i \sim \tilde{P}_{z|y}^g.$
  Compute the relevant statistics, such as mean and variance;
  Output: $\{\mathbf{q}_i, \mathbf{m}_i\}_{i=1}^{N_{\text{samples}}}$, statistics
```

3.3.2 Latent Space Sampling

Here we prove that sampling from the latent space posterior essentially yields the same results as sampling from the full data space in a weak sense. From [13], we have the following results for push forward distributions:

$$\mathbb{E}_{U \sim P_u}[f(U)] = \int_V f(\mathbf{u}) \rho_u(\mathbf{u}) \mathrm{d}\mathbf{u} = \int_{G^{-1}(V)} f(G(\mathbf{z})) \rho_z(\mathbf{z}) \mathrm{d}\mathbf{z} = \mathbb{E}_{Z \sim P_z}[f(G(Z))], \tag{3.15}$$

where $\mathbf{u} = G(\mathbf{z}) \in V$, P_u and P_z are the distributions of $\mathbf{u}$ and $\mathbf{z}$ with PDFs ρ_u and ρ_z, respectively, and f is a measurable function on a suitable space, V. Here, $P_u = G_{\#} P_z^g$ is the push forward of P_z by G. The derivation of Eq. (3.15) only requires that G is measurable.

Theorem 1. Let G_θ be a generator. Let Z be a latent space variable, distributed according to a latent space distribution, P_z^g, with PDF ρ_z^g, and let $U = G_\theta(Z)$ be distributed according to the push forward distribution of the latent space distribution, $U \sim P_u^g = G_{\theta\#}P_z^g$, with PDF ρ_0^g. Then, the push forward posterior distribution, conditioned on data $\mathbf{y}$, is equal to the latent space posterior distribution conditioned on the same data in a weak sense, i.e. for all measurable functions f, the following holds:

$$\mathbb{E}_{U\sim P^g_{u|y}}[f(U)] = \mathbb{E}_{Z\sim P^g_{z|y}}[f(G(Z))], \tag{3.16}$$

where the associated PDFs are given by:

$$\rho^g_{u|y}(\mathbf{u}|\mathbf{y}) = \frac{\rho^g_{y|u}(\mathbf{y}|\mathbf{u})\rho^g_0(\mathbf{u})}{\int_{\mathbb{R}^{N_u}} \rho^g_{y|u}(\mathbf{y}|\mathbf{u})\rho^g_0(\mathbf{u})\,\mathrm{d}\mathbf{u}}, \quad \rho^g_{z|y}(\mathbf{z}|\mathbf{y}) = \frac{\rho^g_{y|u}(\mathbf{y}|G_\theta(\mathbf{z}))\rho^g_z(\mathbf{z})}{\int_{\mathbb{R}^{N_z}} \rho^g_{y|u}(\mathbf{y}|G_\theta(\mathbf{z}))\rho^g_z(\mathbf{z})\,\mathrm{d}\mathbf{z}}. \tag{3.17}$$

Proof. Firstly, since neural networks with continuous activation functions are continuous, they are also measurable [13]. Therefore, the generator defines a push forward distribution and Eq. (3.15) is applicable.

Secondly, we look at the evidence. Assuming the likelihood is measurable with respect to $\mathbf{u}$, we have from Eq. (3.15):

$$Q_u(\mathbf{y}) = \int_{\mathbb{R}^{N_u}} \rho^g_{y|u}(\mathbf{y}|\mathbf{u})\rho^g_0(\mathbf{u})\,\mathrm{d}\mathbf{u} = \int_{\mathbb{R}^{N_z}} \rho^g_{y|u}(\mathbf{y}|G_\theta(\mathbf{z}))\rho^g_z(\mathbf{z})\,\mathrm{d}\mathbf{z} = Q_z(\mathbf{y}). \tag{3.18}$$

Consider the expected value of the likelihood times some measurable function, f, with respect to the prior:

$$\begin{aligned}
\mathbb{E}_{U\sim P^g_0}[f(U)\rho^g_{y|u}(\mathbf{y}|U)] &= \int_E \underbrace{f(\mathbf{u})\rho^g_{y|u}(\mathbf{y}|\mathbf{u})}_{=\xi(\mathbf{u})}\rho^g_0(\mathbf{u})\,\mathrm{d}\mathbf{u} \\
&= \int_{G_\theta^{-1}(E)} \underbrace{f(G_\theta(\mathbf{z}))\rho^g_{y|u}(\mathbf{y}|G_\theta(\mathbf{z}))}_{=\xi(G_\theta(\mathbf{z}))}\rho^g_z(\mathbf{z})\mathrm{d}\mathbf{z} \\
&= \mathbb{E}_{U\sim P^g_z}[f(G_\theta(Z))\rho^g_{y|u}(\mathbf{y}|G_\theta(Z))].
\end{aligned} \tag{3.19}$$

Note that ξ is the product of two measurable functions and is therefore measurable. Hence, Eq. (3.15) applies. Now using Eq. (3.18) and (3.19), we get:

$$\begin{aligned}
\mathbb{E}_{U\sim P^g_{u|y}}[f(U)] &= \frac{1}{Q_u(\mathbf{y})}\mathbb{E}_{U\sim P^g_0}[f(U)\rho^g_{y|u}(\mathbf{y}|U)] \\
&= \frac{1}{Q_z(\mathbf{y})}\mathbb{E}_{Z\sim P^g_z}[f(G_\theta(Z)\rho^g_{y|u}(\mathbf{y}|G_\theta(Z))] \\
&= \mathbb{E}_{Z\sim P^g_{z|y}}[f(G(Z))].
\end{aligned}$$

□

Using Theorem 1, we can conclude that sampling from the latent space posterior, $P^g_{z|y}$, and pushing forward using the generator, G_θ, yields essentially the same results as sampling directly from the generated posterior, $P^g_{u|y}$, in a weak sense. While small perturbations in the latent domain might result in different output in the full data space, Theorem 1 shows that, in a weak sense, the posterior obtained from pushing forward the latent samples is equal to the full data posterior.

3.3.3 Convergence of Generated Posterior

In this subsection, we prove that $P^g_{u|y} \approx P^r_{u|y}$ when $P^g_0 \approx P^r_0$ and under some additional reasonable assumptions on the generator. That is, the case where the prior is approximated in the Wasserstein metric and the likelihood is approximated with a surrogate forward map. While [145] proves the cases where either the likelihood or the prior is approximated, we provide a proof where both are being approximated.

Before stating the theorem, we need to define the appropriate spaces and metrics. Let (E, d_E) be a complete metric space. $E \subset \mathbb{R}^d$ is the set containing the state and parameter vectors, $\mathbf{u} \in E$ and $d_E : E \times E \to \mathbb{R}_+$ assigns non-negative distances between two elements of E. Furthermore, in this formulation, the observation operator, $\mathbf{h} : E \to O$ maps elements from E to the observation space.

We can then define the relevant space of probability distributions:

Definition 3.1. *On a metric space, (E, d_E), we define the space of probability distributions as:*

$$\mathcal{W}_q(E) = \left\{P \,:\, |P|_{\mathcal{W}_q} < \infty\right\}, \quad |P|_{\mathcal{W}_q} = \inf_{x_0 \in E} \left(\int_E d_E(x, x_0)^q \rho(x)\, \mathrm{d}x\right)^{1/q}.$$

Then, we define the Wasserstein-1 distance and its dual representation [113]:

Definition 3.2. *For two probability distributions, $P_1, P_2 \in \mathcal{W}_1(E)$, the Wasserstein-1 distance is defined as:*

$$W_1(P_1, P_2) = \inf_{\gamma \in \Gamma(P_1, P_2)} \left| \int_E \int_E d_E(x, y)\gamma(x, y)\mathrm{d}x\mathrm{d}y \right|,$$

where $\Gamma(P_1, P_2)$ is the set of joint PDFs, γ, for combined probability distributions with P_1 and P_2 as marginal distributions, respectively. From the Kantorovich–Rubinstein duality, we can write the Wasserstein-1 distance as:

$$W_1(P_1, P_2) = \sup_{\mathrm{Lip}(f) \leq 1} \left| \int_E f(x)\rho_1(x)\mathrm{d}x - \int_E f(x)\rho_2(x)\mathrm{d}x \right|,$$

where $f : E \to \mathbb{R}$ *is a Lipschitz continuous function,* $\mathrm{Lip}(f)$ *is its corresponding Lipschitz constant, and* ρ_1 *and* ρ_2 *are the PDFs of* P_1 *and* P_2*, respectively.*

Besides the Wasserstein distance, we will also be working with the weighted norms:

$$||f||_{L^1_\rho} = \int |f(x)|\rho(x)\,\mathrm{d}x, \quad ||f||_{L^2_\rho} = \left(\int |f(x)|^2\rho(x)\,\mathrm{d}x\right)^{1/2}.$$

With the proper spaces, norms, and metrics defined, we can state the following theorem, inspired by [145]:

Theorem 2. Let (E, d_E) be a bounded metric space with $\sup_{\mathbf{x}_1,\mathbf{x}_2\in E} d_E(\mathbf{x}_1, \mathbf{x}_2) \leq D < \infty$.

Let $P_0^r \in \mathcal{W}_2(E)$ denote the prior probability distribution of real data, and let $P_0^r \in \mathcal{W}_2(E)$ denote the generated prior probability distribution of generated data.

Let the real data and generated likelihoods satisfy

$$\rho^r_{y|u}(\mathbf{y}|\mathbf{u}) \propto \Phi^r(\mathbf{u}) = e^{-l^r(\mathbf{u})}, \quad \Phi^r : E \to \mathbb{R}_+, \quad l^r : E \to \mathbb{R}_+,$$
$$\rho^g_{y|u}(\mathbf{y}|\mathbf{u}) \propto \Phi^g(\mathbf{u}) = e^{-l^g(\mathbf{u})}, \quad \Phi^g : E \to \mathbb{R}_+, \quad l^g : E \to \mathbb{R}_+,$$

where l^r and l^g are the log-likelihood functions for the real and generated data, respectively, and Φ^r and Φ^g are Lipschitz continuous functions with Lipschitz constants $\mathrm{Lip}(\Phi^r)$ and $\mathrm{Lip}(\Phi^g)$, respectively. Furthermore, let $\Phi^r, \Phi^g \in L^2_{\rho_0^g}$, where $L^2_{\rho_0^g}$ is the weighted L^2 space with ρ_0^g as the weight function.

Assume the GAN has converged, i.e. $W_1(P_0^r, P_0^g) \leq \epsilon_1$. Furthermore, assume that this implies convergence of the log-likelihood, as follows,

$$||l^r(\mathbf{u}) - l^g(\mathbf{u})||_{L^1_{\rho_0^g}} \leq \epsilon_2, \quad ||l^r(\mathbf{u}) - l^g(\mathbf{u})||_{L^2_{\rho_0^g}} \leq \epsilon_3, \tag{3.20}$$

where $||\cdot||_{L^1_{\rho_0^g}}$ is the weighted L^1-norm with ρ_0^g as the weight function. Then, the Wasserstein-1 distance between the real posterior probability distribution given observations and the generated posterior probability distribution given observations satisfies:

$$W_1(P^r_{u|y}, P^g_{u|y}) \leq C_1\epsilon_1 + C_2\epsilon_2 + C_3\epsilon_3, \tag{3.21}$$

where

$$C_1 = \frac{(1 + D\mathrm{Lip}(\Phi^r))}{Q_u^r(\mathbf{y})}, \quad C_2 = \frac{\max(\Phi^r, \Phi^g)}{Q_u^r(\mathbf{y})Q_u^g(\mathbf{y})}(1 + D\mathrm{Lip}(\Phi^r))|P_0^g|_{\mathcal{W}_1}, \quad C_3 = \frac{\max(\Phi^r, \Phi^g)}{Q_u^g(\mathbf{y})}|P_0^g|_{\mathcal{W}_2},$$

where Q_u^r and Q_u^g are the evidence from the real and generated posterior, respectively, and D denotes the maximum distance between two points in the metric space, E.

Proof. We write the Wasserstein-1 distance between the real prior and the generated prior in dual form:

$$W_1(P_0^r, P_0^r) = \sup_{\mathrm{Lip}(f)\leq 1} \left| \int_E f(\mathbf{u})\rho_0^r(\mathbf{u})\mathrm{d}\mathbf{u} - \int_E f(\mathbf{u})\rho_0^g(\mathbf{u})\mathrm{d}\mathbf{u} \right|, \tag{3.22}$$

where $f : E \to \mathbb{R}$ is Lipschitz continuous with Lipschitz constant less or equal 1 and $f(\mathbf{u}_0) = 0$ for some $\mathbf{u}_0$. Note that any function, g, with Lipschitz constant less than or equal 1, is a contraction and therefore admits a fixed point. Now, assuming that $\mathbf{u}_0$ is the fixed point, we can simply define $f = g - \mathbf{u}_0$, which admits $f(\mathbf{u}_0) = \mathbf{u}_0$. Therefore, assuming $f(\mathbf{u}_0) = 0$ for some $\mathbf{u}_0$ is not a restriction. Furthermore, we have:

$$|f(\mathbf{u})| = |f(\mathbf{u}) + f(\mathbf{u}_0) - f(\mathbf{u}_0)| = |f(\mathbf{u}) + f(\mathbf{u}_0)| \leq \mathrm{Lip}(f)d(\mathbf{u}, \mathbf{u}_0) \leq D.$$

The Wasserstein-1 distance between the posteriors is given by:

$$\begin{aligned}
W_1(P_{u|y}^r, P_{u|y}^g) &= \sup_{\mathrm{Lip}(f)\leq 1} \left| \int_E f(\mathbf{u})\rho_{u|y}^r(\mathbf{u}|\mathbf{y})\mathrm{d}\mathbf{u} - \int_E f(\mathbf{u})\rho_{u|y}^g(\mathbf{u}|\mathbf{y})\mathrm{d}\mathbf{u} \right| \\
&= \sup_{\mathrm{Lip}(f)\leq 1} \left| \int_E f(\mathbf{u})(\rho_{u|y}^r(\mathbf{u}|\mathbf{y}) - \rho_{u|y}^g(\mathbf{u}|\mathbf{y}))\mathrm{d}\mathbf{u} \right| \\
&= \sup_{\mathrm{Lip}(f)\leq 1} \left| \int_E f(\mathbf{u}) \left(\frac{\Phi^r(\mathbf{u})\rho_0^r(\mathbf{u})}{Q_u^r(\mathbf{y})} - \frac{\Phi^g(\mathbf{u})\rho_0^g(\mathbf{u})}{Q_u^g(\mathbf{y})} \right) \mathrm{d}\mathbf{u} \right|,
\end{aligned}$$

Adding and subtracting the term $f(\mathbf{u})\frac{\Phi^r(\mathbf{u})\rho_0^g(\mathbf{u})}{Q_u^r(\mathbf{y})}$ gives

$$\begin{aligned}
W_1(P_{u|y}^r, P_{u|y}^g) &= \sup_{\mathrm{Lip}(f)\leq 1} \left| \int_E f(\mathbf{u}) \left(\frac{\Phi^r(\mathbf{u})\rho_0^r(\mathbf{u})}{Q_u^r(\mathbf{y})} - \frac{\Phi^g(\mathbf{u})\rho_0^g(\mathbf{u})}{Q_u^g(\mathbf{y})} + \frac{\Phi^r(\mathbf{u})\rho_0^g(\mathbf{u})}{Q_u^r(\mathbf{y})} - \frac{\Phi^r(\mathbf{u})\rho_0^g(\mathbf{u})}{Q_u^r(\mathbf{y})} \right) \mathrm{d}\mathbf{u} \right| \\
&\leq \sup_{\mathrm{Lip}(f)\leq 1} \left| \int_E f(\mathbf{u}) \frac{\Phi^r(\mathbf{u})}{Q_u^r(\mathbf{y})} \left(\rho_0^r(\mathbf{u}) - \rho_0^g(\mathbf{u})\right) \mathrm{d}\mathbf{u} \right| + \left| \int_E f(\mathbf{u})\rho_0^g(\mathbf{u}) \left(\frac{\Phi^r(\mathbf{u})}{Q_u^r(\mathbf{y})} - \frac{\Phi^g(\mathbf{u})}{Q_u^g(\mathbf{y})} \right) \mathrm{d}\mathbf{u} \right|.
\end{aligned}$$

Subsequently, adding and subtracting the term $f(\mathbf{u})\frac{\Phi^r(\mathbf{u})\rho_0^g(\mathbf{u})}{Q_u^g(\mathbf{y})}$ in the second integral gives

$$\begin{aligned}
W_1(P_{u|y}^r, P_{u|y}^g) \leq& \sup_{\mathrm{Lip}(f)\leq 1} \left| \int_E f(\mathbf{u}) \frac{\Phi^r(\mathbf{u})}{Q_u^r(\mathbf{y})} \left(\rho_0^r(\mathbf{u}) - \rho_0^g(\mathbf{u})\right) \mathrm{d}\mathbf{u} \right| \\
&+ \left| \int_E f(\mathbf{u})\rho_0^g(\mathbf{u}) \left(\frac{\Phi^r(\mathbf{u})}{Q_u^r(\mathbf{y})} - \frac{\Phi^g(\mathbf{u})}{Q_u^g(\mathbf{y})} \right) + f(\mathbf{u})\rho_0^g(\mathbf{u}) \left(\frac{\Phi^r(\mathbf{u})}{Q_u^g(\mathbf{y})} - \frac{\Phi^r(\mathbf{u})}{Q_u^g(\mathbf{y})} \right) \mathrm{d}\mathbf{u} \right| \\
\leq& \sup_{\mathrm{Lip}(f)\leq 1} \underbrace{\left| \int_E f(\mathbf{u}) \frac{\Phi^r(\mathbf{u})}{Q_u^r(\mathbf{y})} \left(\rho_0^r(\mathbf{u}) - \rho_0^g(\mathbf{u})\right) \mathrm{d}\mathbf{u} \right|}_{=I_1}
\end{aligned}$$

$$+\underbrace{\left|\int_E \left(\frac{1}{Q_u^r(\mathbf{y})}-\frac{1}{Q_u^g(\mathbf{y})}\right) f(\mathbf{u})\rho_0^g(\mathbf{u})\Phi^r(\mathbf{u})\mathrm{d}\mathbf{u}\right|}_{=I_2}$$
$$+\underbrace{\left|\int_E \frac{1}{Q_u^g(\mathbf{y})} f(\mathbf{u})\rho_0^g(\mathbf{u})\left(\Phi^r(\mathbf{u})-\Phi^g(\mathbf{u})\right)\mathrm{d}\mathbf{u}\right|}_{=I_3}.$$

We will consider I_1, I_2, and I_3 individually. Starting with I_3, we use that

$$|e^{-x_1}-e^{-x_2}| \leq e^{-\min(x_1,x_2)}|x_1-x_2| \Rightarrow |\Phi^r(\mathbf{u})-\Phi^g(\mathbf{u})| \leq \max_{\mathbf{u}}(\Phi^r,\Phi^g)|l^r(\mathbf{u})-l^g(\mathbf{u})|. \tag{3.23}$$

Using Eq. (3.23) and the fact that $|f(\mathbf{u})| \leq d(\mathbf{u},\mathbf{u}_0)$, together with the Cauchy-Schwartz inequality, we get:

$$\begin{aligned}
\sup_{\mathrm{Lip}(f)\leq 1} I_3 &\leq \sup_{\mathrm{Lip}(f)\leq 1} \frac{\max(\Phi^r,\Phi^g)}{Q_u^g(\mathbf{y})}\left|\int_E f(\mathbf{u})\rho_0^g(\mathbf{u})|l^r(\mathbf{u})-l^g(\mathbf{u})|\mathrm{d}\mathbf{u}\right| \\
&\leq \frac{\max(\Phi^r,\Phi^g)}{Q_u^g(\mathbf{y})}\left|\int_E d(\mathbf{u},\mathbf{u}_0)\rho_0^g(\mathbf{u})|l^r(\mathbf{u})-l^g(\mathbf{u})|\mathrm{d}\mathbf{u}\right| \\
&\leq \frac{\max(\Phi^r,\Phi^g)}{Q_u^g(\mathbf{y})}\left(\int_E d(\mathbf{u},\mathbf{u}_0)^2\rho_0^g(\mathbf{u})\mathrm{d}\mathbf{u}\right)^{1/2}||l^r(\mathbf{u})-l^g(\mathbf{u})||_{L^2_{\rho_0^g}} \\
&\leq \underbrace{\frac{\max(\Phi^r,\Phi^g)}{Q_u^g(\mathbf{y})}|P_0^g|_{\mathcal{W}_2}}_{=C_3}||l^r(\mathbf{u})-l^g(\mathbf{u})||_{L^2_{\rho_0^g}}.
\end{aligned}$$

Considering I_2, we use the following [145]:

$$\left|\frac{1}{Q_u^r(\mathbf{y})}-\frac{1}{Q_u^g(\mathbf{y})}\right| = \frac{|Q_u^g(\mathbf{y})-Q_u^r(\mathbf{y})|}{Q_u^r(\mathbf{y})Q_u^g(\mathbf{y})}, \tag{3.24}$$

and

$$|Q_u^g(\mathbf{y})-Q_u^r(\mathbf{y})| \leq \max(\Phi^r,\Phi^g)||l^r(\mathbf{u})-l^g(\mathbf{u})||_{L^1_{\rho_0^g}}, \tag{3.25}$$

in order to get:

$$\begin{aligned}
I_2 &= \left|\int_E \left(\frac{1}{Q_u^r(\mathbf{y})}-\frac{1}{Q_u^g(\mathbf{y})}\right) f(\mathbf{u})\rho_0^g(\mathbf{u})\Phi^r(\mathbf{u})\mathrm{d}\mathbf{u}\right| \\
&\leq \left|\frac{1}{Q_u^r(\mathbf{y})}-\frac{1}{Q_u^g(\mathbf{y})}\right|\left|\int_E f(\mathbf{u})\rho_0^g(\mathbf{u})\Phi^r(\mathbf{u})\mathrm{d}\mathbf{u}\right| \\
&\leq \frac{|Q_u^g(\mathbf{y})-Q_u^r(\mathbf{y})|}{Q_u^r(\mathbf{y})Q_u^g(\mathbf{y})}\left|\int_E f(\mathbf{u})\rho_0^g(\mathbf{u})\Phi^r(\mathbf{u})\mathrm{d}\mathbf{u}\right|
\end{aligned}$$

$$\leq \frac{\max(\Phi^r, \Phi^g)}{Q_u^r(\mathbf{y})Q_u^g(\mathbf{y})} \left| \int_E f(\mathbf{u})\rho_0^g(\mathbf{u})\Phi^r(\mathbf{u})\mathrm{d}\mathbf{u} \right| ||l^r(\mathbf{u}) - l^g(\mathbf{u})||_{L^1_{\rho_0^g}}.$$

By defining the function, $g : E \to \mathbb{R}$, $g(\mathbf{u}) = f(\mathbf{u})\Phi^r(\mathbf{u})$, one can show that g is Lipschitz continuous with Lipschitz constant $\mathrm{Lip}(g) = 1 + D\mathrm{Lip}(\Phi^r)$ [145]. Furthermore, we have have $|g(\mathbf{u})| \leq d(\mathbf{u}, \mathbf{u}_0)$. This gives:

$$\begin{aligned} \sup_{\mathrm{Lip}(f)\leq 1} I_2 &\leq \frac{\max(\Phi^r, \Phi^g)}{Q_u^r(\mathbf{y})Q_u^g(\mathbf{y})}(1 + D\mathrm{Lip}(\Phi^r)) \left| \int_E d_E(\mathbf{u}, \mathbf{u}_2)\rho_0^g(\mathbf{u})\mathrm{d}\mathbf{u} \right| ||l^r(\mathbf{u}) - l^g(\mathbf{u})||_{L^1_{\rho_0^g}} \\ &\leq \underbrace{\frac{\max(\Phi^r, \Phi^g)}{Q_u^r(\mathbf{y})Q_u^g(\mathbf{y})}(1 + D\mathrm{Lip}(\Phi^r))|P_0^g|_{\mathcal{W}_1}}_{=C_2} ||l^r(\mathbf{u}) - l^g(\mathbf{u})||_{L^1_{\rho_0^g}}. \end{aligned}$$

Finally, we consider I_1. By using the function, $g : E \to \mathbb{R}$, $\mathbf{u} \mapsto f(\mathbf{u})\Phi^r(\mathbf{u})$, as defined above, we get:

$$\begin{aligned} \sup_{\mathrm{Lip}(g)\leq 1} I_1 &\leq \sup_{\mathrm{Lip}(f)\leq 1} \frac{(1 + D\mathrm{Lip}(\Phi^r))}{Q_u^r(\mathbf{y})} \left| \int_E g(\mathbf{u}) \left(\rho_0^r(\mathbf{u}) - \rho_0^g(\mathbf{u})\right) \mathrm{d}\mathbf{u} \right| \\ &= \underbrace{\frac{(1 + D\mathrm{Lip}(\Phi^r))}{Q_u^r(\mathbf{y})}}_{=C_1} W_1(P_0^r, P_0^r). \end{aligned}$$

Combining, I_1, I_2, and I_3 we then get:

$$\begin{aligned} W_1(P_{u|y}^r, P_{u|y}^g) &\leq \sup_{\mathrm{Lip}(f)\leq 1} I_1 + I_2 + I_3 \\ &= C_1 W_1(P_0^r, P_0^r) + C_2||l^r(\mathbf{u}) - l^g(\mathbf{u})||_{L^1_{\rho_0^g}} + C_3||l^r(\mathbf{u}) - l^g(\mathbf{u})||_{L^2_{\rho_0^g}} \\ &\leq C_1\epsilon_1 + C_2\epsilon_2 + C_3\epsilon_3. \end{aligned}$$

□

In short, the proof of Theorem 2 helps us understand when we can expect convergence of the posterior. While the assumptions of the Theorem might seem restrictive, this is actually not the case. Firstly, we assume that the metric space, E, is bounded, which is typically the case in many applications. Secondly, we assume that the likelihood is of the form $e^{-l(\mathbf{u})}$, Lipschitz continuous, and is in the weighted L^1 and L^2 spaces. For simple observation operators (e.g. linear), this is a consequence of the negative log-likelihood function typically being an L^2-norm. This leaves us with the question of the convergence of the prior, which directly determines ϵ_1, ϵ_2, and ϵ_3.

3.3.4 Convergence of the Generated Prior

The convergence of the generated prior is a matter of studying convergence properties of GANs. Such studies are beyond the scope of this work. Instead, we refer to [5, 55, 93], where convergence properties of GANs are discussed. In short, ensuring convergence of GANs is similar to ensuring convergence of other types of neural networks. Hence, it is a matter of having enough data and performing hyperparameter tuning. For the MCGAN, the amount of data is, in general, not a problem, as we simulate the training data.

3.4 Alternative Methods

Here, we will comment on some well-known alternative methods that exist to speed up solving the Bayesian inverse problem, which we will use for comparision with our proposed method. We will also comment on their respective shortcomings.

3.4.1 Ensemble Kalman Filter

Ensemble Kalman filtering (EnKF) is a Kalman filter variant that is suitable for high-dimensional and nonlinear problems [59]. The general idea is to compute the sample mean and sample covariance from an ensemble and then update the prior accordingly. However, as all distributions are assumed to be Gaussian, it means that it is not directly suitable for non-Gaussian problems. In cases with very nonlinear or high-dimensional features, large ensembles are necessary which in turn makes it computationally slow.

We compare the MCGAN framework with two variations of the ensemble Kalman filter (EnKF):

- The (standard) EnKF for dynamic problems, where the state and parameter distributions are computed based on previous time steps along with data availability;

- Ensemble Kalman Inversion (EKI), used for stationary problems, where an artificial time dimension is introduced in order to iteratively update the posterior of the state and parameters.

For the pipe flow equations, the standard EnKF is utilized, while for the Darcy flow, the EKI is used. The EnKF implementation is based on [59] and the EKI implementation is based on [27].

For simultaneously estimating the parameter and state, we make use of disturbance modeling [72]. Here, we define an augmented model:

$$\mathbf{q}_i = F(\mathbf{q}_{i-1}, \mathbf{m}_{i-1}) + \Gamma_q \epsilon_i, \quad \epsilon_i \sim \mathcal{N}(0, Q_q), \tag{3.26a}$$

$$\mathbf{m}_i = \mathbf{m}_{i-1} + \Gamma_m \delta_i, \quad \delta_i \sim \mathcal{N}(0, Q_m), \tag{3.26b}$$

$$\mathbf{y}_i = \mathbf{h}(\mathbf{q}_i) + \eta_i, \quad \eta_i \sim \mathcal{N}(0, R), \tag{3.26c}$$

where F is the discrete one-step time advancement model, ϵ_i is the model noise, Q the model covariance, and R the observation covariance. With this formulation, both the state and parameters are updated in every step of the EnKF algorithm.

Alternative approaches exist, such as particle filters, that do not assume a Gaussian distribution. However, such methods are, in general, computationally very expensive and will not be further discussed.

3.4.2 Surrogate Models

Instead of replacing the sampling method, the forward computations can be done using a surrogate model, such as polynomial chaos expansion (PCE) methods [164], Gaussian processes [146], or reduced basis methods [123]. The idea is to approximate the parameter-to-observations map or the forward map by a low-order model that is computationally fast to evaluate. These approaches have been shown to speed up the sampling significantly. However, in high-dimensional cases the curse of dimensionality hampers the applicability of such methods. Moreover, they usually do not perform well in discontinuous and/or highly nonlinear cases unless the approach is tailored to the problem at hand.

Polynomial Chaos Expansion

In this chapter, we specifically compare with the PCE approach. The basic idea behind PCE is to create a surrogate model that maps the stochastic parameters, $\mathbf{m}$, to a quantity of interest, Q [164]. The surrogate model is defined by a linear expansion of orthogonal polynomials:

$$Q(\mathbf{m}) = \sum_{i=1}^{N} \alpha_i \phi_i(\mathbf{m}), \tag{3.27}$$

where ϕ_i are the polynomials that are chosen based on the distribution of $\mathbf{m}$, and α_i are the generalized Fourier coefficients.

The coefficients, α_i, are typically computed using either spectral projection methods or by least squares minimization. In both cases, the evaluations are carefully chosen according to a quadrature rule.

In our test cases, we choose the quantity of interest to be the observations, i.e. $Q(\mathbf{m}) \approx \mathbf{h}(\mathbf{q}(\mathbf{m}))$ and $\alpha_i \in \mathbb{R}^{N_y}$; $\mathbf{m}$ are the parameters of interest, which are often the model parameters and/or initial and boundary conditions.

When the PCE is computed, the posterior PDF is defined by:

$$\rho_m^y(\mathbf{m}|\mathbf{y}) = \frac{1}{\rho_y(\mathbf{y})}\rho_\eta(\mathbf{y} - Q(\mathbf{m}))\rho_0^m(\mathbf{m}). \tag{3.28}$$

The expected state and parameters are then computed by:

$$\mathbb{E}_{\mathbf{q}}\left[\mathbf{q}\right] \approx \frac{1}{N_{sample}} \sum_{i=1}^{N_{sample}} \mathbf{q}(\mathbf{m}_i), \quad \mathbb{E}_{\mathbf{m}}\left[\mathbf{m}\right] \approx \frac{1}{N_{sample}} \sum_{i=1}^{N_{sample}} \mathbf{m}_i, \quad \mathbf{m}_i \sim P_m^y, \tag{3.29}$$

and the variance is computed in a similar manner.

It is important to notice that the sampling is done in the parameter space and the state is thereafter computed by using the sampled parameters as input for the forward problem. Directly sampling the state is infeasible due to the high-dimensionality of the state.

The implementation of the PCE method in this chapter is done using the Python library Chaospy [37].

3.4.3 Likelihood-Free methods

As mentioned above, the computationally most expensive task is to evaluate the likelihood as this requires the solution of the forward problem. Alternatively, one can compute the posterior distribution without computing the likelihood. Such approaches are termed likelihood-free methods.

There are several ways of making use of likelihood-free methods. One approach that has become increasingly popular is to learn the posterior directly in an offline stage [2]. Here, pairs of state/parameters and corresponding observations, $(\mathbf{u}, \mathbf{h}(\mathbf{u}))$, are required in the training stage. Then, a model is trained to approximate the posterior, $P(\mathbf{u}|\mathbf{h}(\mathbf{u})) \approx P(\mathbf{u}|\mathbf{y})$.

While this approach enables fast computation of the posterior in the online stage, because no sampling is needed, it is less flexible, as $\mathbf{h}$ cannot change between the offline and the online stages, meaning that sensor configurations must be constant. Hence, in cases where the sensor configuration is not known a priori, the methodology is not usable. Furthermore, if the noise levels in the observations change, this cannot be incorporated in the online stage.

Deep Bayesian Inversion

A particular example of a likelihood-free method that makes use of GANs is the Deep Bayesian Inversion (DBI). DBI makes use of conditional GANs (CGANs) to learn the posterior distribution [2]. A CGAN is trained to learn to sample from the posterior distribution, $P_{u|y}$, directly. The general setup is the same as in Section 3.2.3, but with some minor differences. The generator is now a map that takes a latent vector, $\mathbf{z}$, and observations, $\mathbf{y}$, and outputs a sample $\mathbf{u}$:

$$G(Z, \mathbf{y}) = P^g_{u|y}(U|\mathbf{y}) \approx P^r_{u|y}(U|\mathbf{y}), \quad Z \sim P^g_z. \tag{3.30}$$

Similarly, the discriminator takes $\mathbf{y}$ and $\mathbf{u}$ and outputs a real number.

The training is performed by solving the inf-sup problem:

$$\begin{aligned} \inf_\theta \sup_\omega \quad & \mathbb{E}_{X\sim P^r_u}\left[D_\omega(X, \mathbf{h}(X))\right] - \mathbb{E}_{Z\sim P^g_z}\left[D_\omega(G_\theta(Z), \mathbf{h}(G_\theta(Z)))\right] \\ & -\lambda \mathbb{E}_{\hat{\mathbf{x}}\sim P_{\hat{X}}}\left[\left(||\nabla_{\hat{X}} D_\omega(\hat{X}, \mathbf{h}(\hat{X}))|| - 1\right)^2\right]. \end{aligned} \tag{3.31}$$

Hence, the training is performed with the same data as for the MCGAN training, with the difference that the observation operator is used to create observations for training.

In the online stage, when observations become available, one samples several latent vectors for the same observations and uses those to obtain posterior samples:

$$\{G(\mathbf{z}_1, \mathbf{y}), G(\mathbf{z}_2, \mathbf{y}), \ldots, G(\mathbf{z}_N, \mathbf{y})\} = \{\mathbf{u}_1|\mathbf{y}, \mathbf{u}_2|\mathbf{y}, \ldots, \mathbf{u}_N|\mathbf{y}\}, \quad \mathbf{z}_i \sim P^g_z, \quad \forall i. \tag{3.32}$$

The generator is now tied to the observation operator that was used for training.

3.5 Results

In this section, we will present the results on two different problems using the MCGAN methodology. We show two distinct parameter and state estimation cases to highlight various advantages of using the MCGAN methodology. Firstly, we consider a Darcy flow case (stationary flow through a porous medium), with the aim of approximating the horizontal and vertical velocity, the pressure, and the permeability field. The purpose of this case is to emphasize the ability to deal with high-dimensional stochastic problems as the permeability field is spatially distributed and follows a high-dimensional distribution. Secondly, we consider the problem of leakage detection in pipe flow. Here, the challenge lies in dealing with a nonlinear hyperbolic PDE with discontinuities and a non-informative prior.

The results will be assessed using the relative root mean squared error (RRMSE):

$$\text{State RRMSE} = \frac{\sqrt{\sum_{i=1}^{N_q}(\mathbf{q}^*_i - \mathbf{q}_i)^2}}{\sqrt{\sum_{i=1}^{N_q}\mathbf{q}_i^2}}, \quad \text{Parameter RRMSE} = \frac{\sqrt{\sum_{i=1}^{N_m}(\mathbf{m}^*_i - \mathbf{m}_i)^2}}{\sqrt{\sum_{i=1}^{N_m}\mathbf{m}_i^2}}, \tag{3.33}$$

where $\mathbf{q}^*$ and $\mathbf{m}^*$ denote the approximated state and parameters, respectively, and $\mathbf{q}$ and

m are the reference state and parameters, respectively. The reference values are computed using an appropriate numerical solver. This will be discussed for each test case separately.

Furthermore, we will look at the approximated posterior distributions resulting from the MCGAN.

For the details on the hyperparameters for the training in each of the test cases, see Table 3.1. We compared the Adam optimizer and the RMSprop optimizer for training, and found that RMSProp, in general, showed superior results in our test cases. The GAN architectures are shown in Figure 3.3. Furthermore, all the training data for the GANs are generated by sampling the parameter spaces according to the chosen distribution for the test case. The number of training samples is chosen based on the performance of the resulting GAN. Note that it is, in general, a difficult problem to choose the number of necessary training samples.

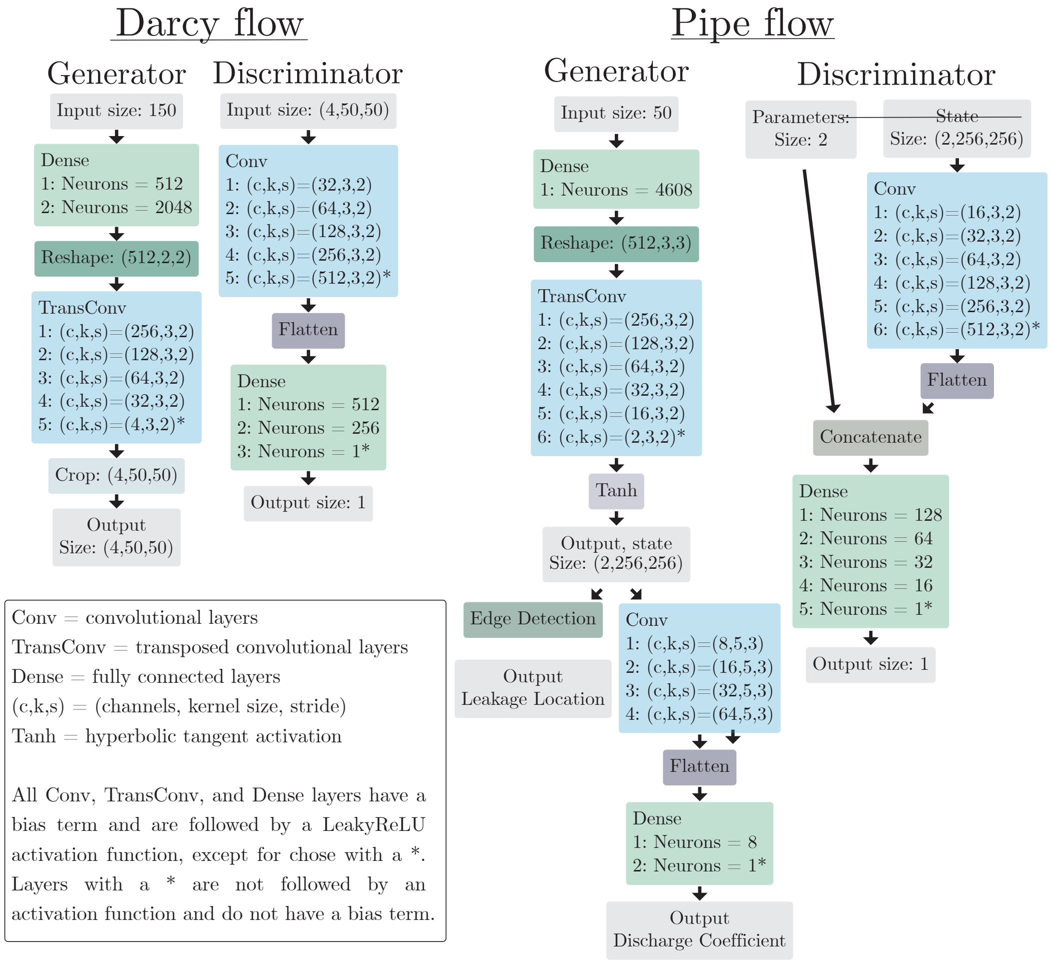
Table 3.1: Hyperparameters for the WGANs for the three test cases. The number of discriminator updates, relative to those of the generator, is denoted by n_{disc}/n_{gen}

Hyperparameters \ Test case	Darcy flow	Pipe flow
Optimizer	RMSProp	RMSProp
Learning rate	10^{-4}	10^{-4}
Batch size	64	64
Gradient penalty	5	5
n_{disc}/n_{gen}	1	2
N_{train}	300,000	100,000
Latent dimension (N_z)	150	50

The specific architectures of the generators and discriminators for each test case can be found in Figure 3.3. It is worth noting that we make use of convolutional neural networks in all cases due to their success in problems dealing with spatially distributed degrees of freedom [19, 105]. It should, however, be noted that convolutional neural networks can essentially only be applied to Cartesian grids. To deal with irregular grids, one could make use of alternative architectures, such as graph neural networks, as in [58], or operator neural networks, such as Fourier neural operators [91].

As mentioned in Section 3.3.4, it is not feasible to compute the Wasserstein distance for very high-dimensional distributions. Therefore, in order to show convergence of the generated prior, we show the convergence of the first two moments, mean and standard deviation, with the training epochs. Here, we have a value for the mean and variance at every grid point and we compute the error as the relative RMSE. The convergence plots are shown in Figure 3.4. While this is a weaker type of convergence than convergence in the Wasserstein-1 distance, it still gives an indication that the GAN error is sufficiently small for the purpose of Bayesian inversion.

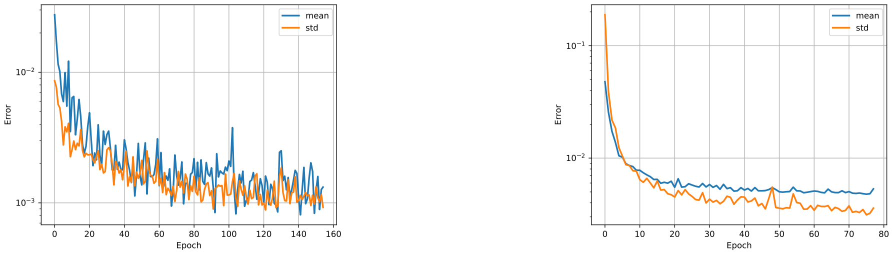
We compare the proposed set-up with three alternatives: Ensemble Kalman filter, poly-



Conv = convolutional layers
TransConv = transposed convolutional layers
Dense = fully connected layers
(c,k,s) = (channels, kernel size, stride)
Tanh = hyperbolic tangent activation

All Conv, TransConv, and Dense layers have a bias term and are followed by a LeakyReLU activation function, except for chose with a *. Layers with a * are not followed by an activation function and do not have a bias term.

Figure 3.3: Generator and discriminator architectures for the two test cases.



(a) Darcy flow.

(b) Pipe flow.

Figure 3.4: Convergence of mean and variance of the generated prior towards the prior computed from simulations. The error is computed using a test dataset.

nomial chaos expansion, and deep Bayesian inversion (DBI) [2]. Brief summaries of each methods can be found in Section 3.4. For the first test case, it is worth noting that we make use of a variation called ensemble Kalman inversion, that is suitable for stationary problems. For the DBI, we make use of the same architectures, except for the input. In the generator, the observations are concatenated with the latent variables and for the discriminator the observations are concatenated with output of the convolutional layers. Furthermore, both the PCE and the DBI approaches are trained to the specific sensor configurations. Hence, they are less flexible than the Kalman filter and MCGAN methods, which allow for varying sensor configurations and a change of likelihood function.

All results are generated using synthetic observations. Therefore, all observations are simulation-based and perturbed with artificial noise. To ensure that we are not subject to inverse crime [25], the synthetic observations are generated with a higher resolution than what is used for the training of the GANs and PCE models, for all experiments. Furthermore, the Kalman filter results are also generated with a lower resolution. Secondly, we will use another distribution for the likelihood function than for the noise in the synthetic observations. The specifics will be discussed in each test case.

The number of necessary MCMC samples was considered a hyperparameter to be tuned and we chose the smallest number of samples that did not sacrifice accuracy in both cases.

3.5.1 Darcy Flow

As a first test case, we consider stationary two-dimensional Darcy flow:

$$\mathbf{v} + k\nabla p = 0, \qquad \mathbf{x} \in [0,1]^2, \tag{3.34a}$$

$$\nabla \cdot \mathbf{v} = 0, \qquad \mathbf{x} \in [0,1]^2, \tag{3.34b}$$

$$p = 1, \qquad \mathbf{x} \in 0 \times [0,1], \tag{3.34c}$$

$$p = 0, \qquad \mathbf{x} \in 1 \times [0,1], \tag{3.34d}$$

$$\mathbf{v} \cdot \mathbf{n} = 0, \quad \mathbf{x} \in [0,1] \times \{0,1\}. \tag{3.34e}$$

$p : [0,1]^2 \to \mathbb{R}$ denotes pressure, $\mathbf{v} : [0,1]^2 \to \mathbb{R}^2$ denotes the velocity, $k : [0,1]^2 \to \mathbb{R}$ is the spatially-dependent permeability field, and $\mathbf{x} = (x_1, x_2)$ are the spatial coordinates in the horizontal and vertical directions, respectively. The permeability field k is modeled as a lognormal field, $\log k = m \sim \mathcal{N}(0, C)$. The problem of state and parameter estimation for Darcy flow is often considered in data assimilation and in uncertainty quantification, see e.g. [28, 132].

The covariance matrix C is derived from the class of Matérn functions [85]:

$$C(\mathbf{x}_i, \mathbf{x}_j) = \sigma^2 \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu}}{l} d(\mathbf{x}_i, \mathbf{x}_j) \right)^{\nu} K_\nu \left(\frac{\sqrt{2\nu}}{l} d(\mathbf{x}_i, \mathbf{x}_j) \right). \tag{3.35}$$

Γ is the gamma function and K_ν is the modified Bessel function of the second kind. $d(\mathbf{x}_i, \mathbf{x}_j)$ denotes the distance between two points, $\mathbf{x}_i$ and $\mathbf{x}_j$, in the domain, ν defines the smoothness, $\sigma^2 > 0$ is the variance, and $l > 0$ is the correlation length.

We denote by $\mathbf{m}_N$ the discretized version of m defined on an $N \times N$ grid and the covariance matrix, $C_N \in \mathbb{R}^{N^2} \times \mathbb{R}^{N^2}$, has elements $(C_N)_{i,j} = C(\mathbf{x}_i, \mathbf{x}_j)$. Then, $\mathbf{m}_N$ can be sampled by computing

$$\mathbf{m}_N = \sum_{i=1}^{N^2} \sqrt{\lambda_i} \hat{\mathbf{m}}_i \boldsymbol{\psi}_i, \quad \hat{\mathbf{m}} \in \mathbb{R}^{N^2}, \quad \hat{\mathbf{m}} \sim \mathcal{N}(0, I), \tag{3.36}$$

where $I \in \mathbb{R}^{N^2} \times \mathbb{R}^{N^2}$ is the identity matrix, λ_i are the eigenvalues of C_N in descending order, and $\boldsymbol{\psi}_i$ the corresponding eigenvectors. Hence, the permeability field is determined by $\hat{\mathbf{m}}_i$, $i = 1, \ldots, N$. A reduced representation of the permeability field can then be computed by choosing $n < N^2$:

$$\mathbf{m}_N^{(n)} = \sum_{i=1}^{n} \sqrt{\lambda_i} \hat{\mathbf{m}}_i \boldsymbol{\psi}_i. \tag{3.37}$$

Thereby, the reduced permeability field is determined by n, instead of N^2, parameters.

For generating the training data, Eq. (3.34) is solved using the finite element method. The velocity is discretized by discontinuous Raviart-Thomas elements of polynomial order 3 and the pressure is discretized by Lagrange elements of polynomial order 2. This is known to be a stable pairing of finite element spaces for the stationary Darcy flow [24]. The domain is divided into 32 × 32 squares, each divided into two triangles, resulting in 25793 degrees of freedom in total. The solutions are then evaluated on a 50 × 50 equidistant grid. The implementation is done using the FEniCS library [94].

The specific setting for creating the permeability field here is $n = 1089$, $\nu = 1.5$, $l = 0.2$, and $\sigma = 0.5$.

For the observations, we consider evenly distributed sensors at locations, $(\mathbf{x}_1, \dots, \mathbf{x}_{N_y})$, measuring the horizontal velocity at $N_y = 100$ discrete points, see Figure 3.6a. Thus, $\mathbf{h} : \mathbb{R}^{N \times N} \to \mathbb{R}^{N_y}$, and the measurements are created by:

$$\mathbf{y} = \mathbf{h}(\mathbf{v}) + \eta, \quad \mathbf{h}(\mathbf{v}) = (v_1(\mathbf{x}_1), \dots, v_1(\mathbf{x}_{N_y})) \quad \eta \sim \mathcal{N}(0, 0.01^2 I), \quad \eta \in \mathbb{R}^{N_y}. \tag{3.38}$$

The synthetic observations are generated using 50 × 50 squares divided into two triangles. The velocity is discretized with polynomial order 4 and the pressure with polynomial order 3. The test case is similar to the one presented in [28].

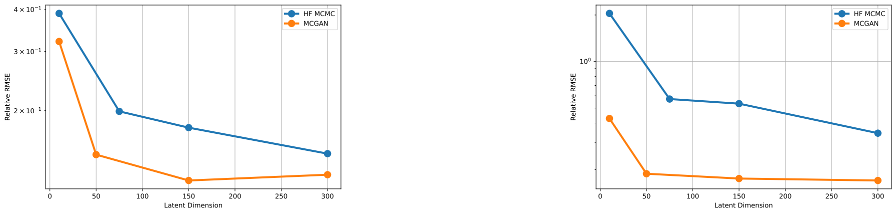
We compare the MCGAN method with the ensemble Kalman inversion (EKI) method [27] and Deep Bayesian Inversion [2]. The DBI is trained to the specific sensor locations. We do not compare with PCE, since it is infeasible to compute a PCE model for a problem of this high dimensionality. For the EKI, we compute ensembles consisting of 4000 forward computations and use 25 iterations. Note that the EKI method is parallel, since each member of the ensemble can be computed independently from the other members. Therefore, we run the EKI using 20 CPU cores. For computing the permeability field, we use $n = 1089$, which is the total number of degrees of freedom. See Figure 3.7 for the results.

GAN setup

The discriminator of the GAN consists of convolutional layers and the generator consists of transposed convolutional layers. The generator is trained to generate the velocity in the horizontal direction, v_1, the velocity in the vertical direction, v_2, the pressure, p, and the log-permeability field, $\log(k)$. Each quantity is considered a channel in the sense of convolutional neural networks. Thereby, the generator outputs tensors of the shape $(4, N, N)$. To avoid boundary artifacts in the generated fields originating from the transposed convolutional layers, the generator is trained to generate fields of the shape $(4, N + l, N + l)$, $l > 0$, which are then cropped to the desired size. For details on the exact architecture specifications, see Figure 3.3.

Results

The MCGAN results are computed with a single chain of 20,000 samples, where the first 12,500 samples are discarded to ensure that we only use samples with a converged chain. The MAP estimate is used as the initial MCMC sample, which reduces the time until convergence for the MCMC method significantly.



(a) State

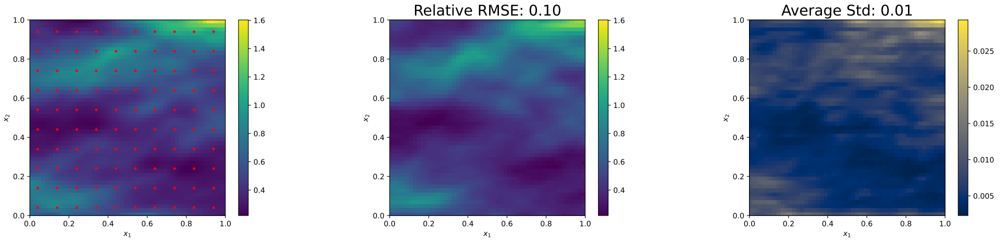
(b) Log-permeability

Figure 3.5: Convergence of the MCGAN and high-fidelity MCMC for the state and log-permeability with respect to the latent dimension.

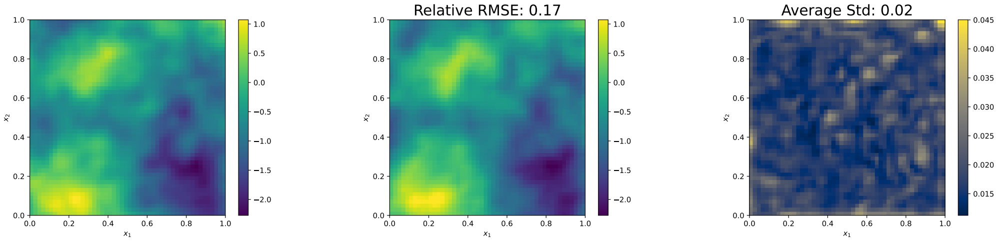
For the likelihood function, we use $\mathcal{N}(0, 0.02^2 I)$, which is different from the distribution used to generate the observation noise.

To assess the convergence of relative RMSE for the MCGAN approximated state and permeability with respect to the latent dimension, we compare the MCGAN approximated state with a high-fidelity MCMC procedure. For each latent dimension, a new GAN is trained with the same architecture, hyperparameters, and training data. For the MCMC procedure, we sample the permeablity and solve the forward problem to get the likelihood. We use the same forward model as for simulating the MCGAN training data and the standard deviation was also the same as for the MCGAN. The mean values of all accepted samples were used for computing the relative RMSE of the permeability. Similarly, the mean values of all corresponding states were used to compute the state relative RMSE. The latent dimension refers to the number of modes used for the permeability, as shown in (3.37). A Metropolis-Hastings algorithm with adaptive proposal standard deviation ensuring that the acceptance rate is between 0.2 and 0.5 was used. We employed 10 uncorrelated chains in parallel, with different initial conditions. The first 200,000 samples from each chain were discarded and then every 5th sample was saved to reduce the correlation between each sample in the chain until 3,000 samples per chain were reached. Hence, a total of 30,000 samples were used for computing the relative RMSE. In Figure 3.5, we see the convergence results. Clearly, with a latent dimension of 10 in the MCGAN, essentially the same accuracy was achieved for the permeability as for the high-fidelity MCMC method with between 150 and 300 modes. Furthermore, for the state, a dimension of 50 in the MCGAN achieves similar accuracy as 300 modes in the high-fidelity MCMC. Hence, the MCGAN approach provides a significant dimensionality reduction, together with a speed-up of the evaluation.

MCMC Setup. In Figure 3.6, the results from using MCGAN for the Darcy flow are shown. We see that the horizontal velocity is estimated accurately with a relative RMSE of 0.10 and a relatively low standard deviation. Not surprisingly, the standard deviation seems



(a) True v_1. (b) Approximated v_1. (c) Standard deviation v_1.



(d) True $\log(k)$. (e) Approximated $\log(k)$. (f) Standard deviation $\log(k)$.

Figure 3.6: MCGAN results for the Darcy flow test case. Top row: v_1. Red dots are points of measurements. Bottom row: $\log(k)$.

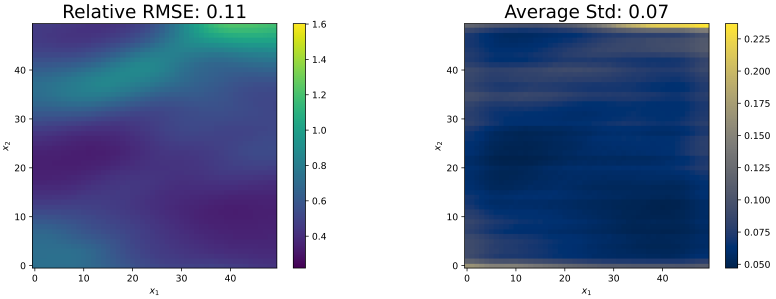
to be largest at the upper boundary where no measurements are available. Furthermore, larger uncertainty is observed in the areas of the domain where the magnitude v_1 is large.

Regarding the log-permeability, the MCGAN captures the structure of the true log-permeability as well as the sharp edges with a relative RMSE of 0.17.

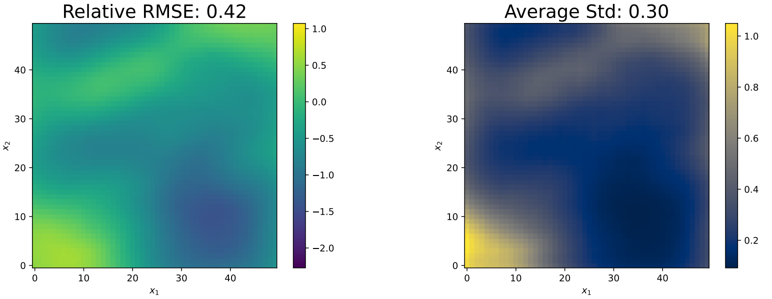
For both the state and log-permeability, the Kalman inversion gives similar, but slightly worse, accuracy and significantly smoother results than the MCGAN approach (see Figure 3.7). Hence, the Kalman inversion is not able to capture the sharper edges. Furthermore, it is an order of magnitude slower (see Table 3.5). The DBI method gives similar results, but a lower accuracy on the parameter estimation is observed (see Figure 3.8).

3.5.2 Leakage Detection in Pipe Flow

To show the method's generality, as a different problem, we consider unsteady single phase flow through a pipeline, until suddenly (at t=10s) a leak occurs. As a consequence, pressure waves start propagating through the pipeline, and the velocity field at the leak becomes discontinuous because of the mass flow leaving through the leak. We have only two measurement locations, one close to the inlet and one close to the outlet of the pipeline measuring pressure, and the goal is to infer the leak location and size based on these measurements and a physical model of the flow in the pipeline. This is a challenging problem because of the very sparse measurement data and the discontinuity in the solution.



(a) Approximated v_1. (b) Standard deviation v_1.



(c) Approximated $\log(k)$. (d) Standard deviation $\log(k)$.

Figure 3.7: EKI results for the Darcy flow test case. In (a)-(b) we see the reconstruction of v_1 and the standard deviation of the reconstruction, respectively. In (c)-(d) we see the reconstruction of $\log(k)$ and the standard deviation of the reconstruction, respectively.

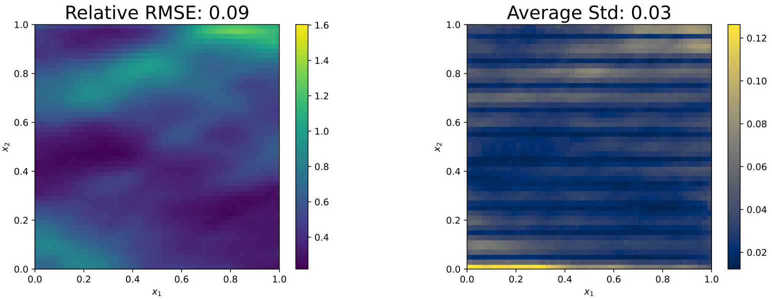
The governing equations are given by the one-dimensional Euler equations for mass and momentum conservation [62, 86]:

$$\partial_t q_1 + \partial_x q_2 = C_d \sqrt{\rho(p(\rho) - p_{\mathrm{amb}})}\delta(x - x_l)H(t - t_l), \tag{3.39a}$$

$$\partial_t q_2 + \partial_x \left(\frac{q_2^2}{q_1} + p(\rho)A \right) = -\frac{1}{2d}\frac{q_2^2}{q_1} f_f(q), \tag{3.39b}$$

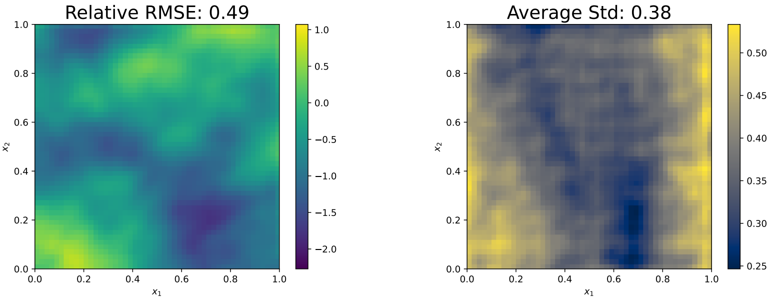
$$v(0,t) = v_0, \quad p(L,t) = p_L. \tag{3.39c}$$

where ρ is the fluid density (not to be confused with the probability density functions in previous sections), $p(\rho) = c^2(\rho - \rho_0) + p_0$ is the pressure, v is the velocity, $q_1 = \rho A$, $q_2 = \rho v A$, δ is the Dirac delta function, and H is the Heaviside function. v_0 represents the boundary conditions prescribed on the velocity at the left end of the pipe and p_L is the prescribed pressure at the right end of the pipe. d, A, p_{amb}, c, ρ_0, are all constants. The physical quantities they represent and the values we will be working with are found in Table 3.2. The righthand side in Eq. (3.39a) is the leakage, modeled as a discharge. t_l is the time at which


Relative RMSE: 0.09

Average Std: 0.03

(a) Spproximated v_1. (b) Standard deviation v_1.


Relative RMSE: 0.49

Average Std: 0.38

(c) Approximated $\log(k)$. (d) Standard deviation $\log(k)$.

Figure 3.8: DBI results for the Darcy flow test case. In (a)-(b) we see the reconstruction of v_1 and the standard deviation of the reconstruction, respectively. In (c)-(d) we see the reconstruction of $\log(k)$ and the standard deviation of the reconstruction, respectively.

the leakage occurs, x_l and C_d are the two parameters of interest. They represent the location and size of the leakage, respectively. The righthand side of Eq. (3.39b) is the friction, where f_f is the Darcy-Weisbach friction coefficient, which is given by the Haaland expression [138]:

$$\frac{1}{\sqrt{f_f}} = -\frac{1}{4}1.8\log_{10}\left[\left(\frac{\varepsilon/D}{3.7}\right)^{1.11} + \frac{6.9}{Re}\right], \tag{3.40}$$

where Re is the Reynolds number, $Re = \frac{\rho v d}{\mu}$, with μ the fluid viscosity and ε the pipe roughness. The values and units of all parameters in the model are in Table 3.2. The initial condition is $(q_1, q_2) = (\rho_0 A, \rho_0 v_0 A)$.

Eq. (3.39) is solved using the nodal discontinuous Galerkin method [66]. We use Legendre polynomials for the local polynomials, and Lagrange polynomials for the nodal representation. The numerical flux is chosen to be the Lax-Friedrichs flux. To ensure stability and non-oscillatory behavior while ensuring high-order accuracy, a TVBM slope-limiter is applied after each time step [66]. The time stepping is performed using the BDF2 method, with an

Table 3.2: Parameters for the pipe flow equations, (3.39). Note that the discharge coefficient and the leakage location have values denoted by intervals, as they are the parameters to determine.

Physical quantity	Constant	Value	Unit
Pipe length	L	2000	m
Diameter	d	0.508	m
Cross-sectional area	A	0.203	m^2
Speed of sound in fluid	c	308	m/s
Ambient pressure	p_{amb}	101325	Pa
Reference pressure	p_{ref}	5016390	Pa
Reference density	ρ_{ref}	52.67	$\mathrm{kg/m}^3$
Inflow velocity	v_0	4.0	m/s
Outflow pressure	p_L	5016390	Pa
Pipe roughness	ε	10^{-8}	m
Fluid viscosity	μ	$1.2 \cdot 10^{-5}$	$\mathrm{N} \cdot \mathrm{s/m}^2$
Leakage start time	t_l	10	s
Discharge coefficient	C_d	$\left[1.0 \cdot 10^{-4}, 9.0 \cdot 10^{-4}\right]$	m
Leakage location	x_l	$[100, 1900]$	m

initial implicit Euler step [89].

For the generation of the training data, we consider 75 elements with a local polynomial order of 3. The resulting solution is then evaluated on an equidistant grid consisting of 256 points. For the time stepping, we consider a horizon of $T = 64$ seconds with 256 time steps. Hence, $(q_1, q_2) \in \mathbb{R}^{256\times256} \times \mathbb{R}^{256\times256}$.

We assume a uniform prior for both the leakage location, $x_l \sim \mathcal{U}(100, 1900)$, and the discharge coefficient, $C_d \sim \mathcal{U}\left(1.0 \cdot 10^{-4}, 9.0 \cdot 10^{-4}\right)$. Other choices of distributions of x_l and C_d are subject to future studies.

For the state and parameter estimation, only measurements of the pressure are observed. We consider the vector, $(x_1, \ldots, x_{N_y})$, of measurement locations, and the vector of measurement times, $(t_1, \ldots, t_{N_y})$. This gives rise to the synthetic observations:

$$\mathbf{y} = \mathbf{h}(p) + \eta, \quad \mathbf{h}(p) = (p(x_1, t_1), \ldots, p(x_{N_y}, t_{N_y})), \quad \eta \sim \mathcal{N}(0, 1500^2 I), \quad \eta \in \mathbb{R}^{N_y}. \tag{3.41}$$

We specifically consider the case where we only observe at $x = 20m$ and at $x = 1980m$ and for all time instances, i.e. $(x_1, \ldots, x_{N_y}) = (20, \ldots, 20, 1980, \ldots, 1980)$ and $(t_1, \ldots, t_{N_y}) = (0.25, \ldots, 64, 0.25, \ldots, 64)$. Hence, $N_y = 2 \cdot 256 = 512$. For simulating the synthetic observations, we used 100 elements with a local polynomial order of 4.

We compare our method with the DBI, PCE and the EnKF approaches. The PCE model is trained to map the leakage location, x_l, and discharge coefficient, C_d, to the observations. We achieved the highest precision with fourth-order polynomials. We performed 50,000 MCMC posterior samples and discarded the first 40,000. The state reconstruction is performed after the sampling by computing the state using the parameters samples from the MCMC sampling. The state reconstructions are computed in parallel using 20 cores. See Figure 3.11 for results.

In the EnKF method, we used an ensemble size of 2000. Γ_q and Γ_m are chosen to be identity matrices and $Q_q = \text{diag}(0.01, 0.001)^2$ and $Q_m = \text{diag}(100, 1 \cdot 10^5)^2$. The ensemble is computed in parallel on 20 CPU cores. See Figure 3.12 for results.

GAN setup

As for the above tests, we use convolutional layers for the discriminator and transposed convolutional layers for the generator. The GAN is trained to generate the velocity, v, and pressure, p, instead of generating the conservative variables q_1 and q_2, since v and p are the quantities of interest. The GAN is trained to generate full space-time solutions in the intervals, $x \in [0, L]$ and $t \in [0, T]$. v and p are considered channels in the sense of convolutional neural networks. Hence, the generator generates tensors of size $(2, 256, 256)$.

At the location of the leakage, there will be a discontinuity in the velocity, due to a drastic drop in the velocity. We use this information to compute the leakage location by identifying the spatial location of the discontinuity, by convolving the state with an appropriate kernel. Furthermore, a dense neural network takes in the generated state and outputs the discharge coefficient. See Figure 3.3 for a visualization of the GAN.

Due to the large differences in orders of magnitude, the velocity, pressure and discharge coefficients are scaled to have values between -1 and 1.

Results

The MCGAN results are computed with a single chain of 15,000 samples, where the first 10,000 samples are discarded to ensure that we only use samples after the chain has converged. The MAP estimate is, again, used as the initial MCMC sample in order to speed up convergence. For the likelihood function, we use $\mathcal{N}(0, 3000^2 I)$, which is different from the distribution used to generate the observation noise.

Figure 3.9 presents the reconstruction of the velocity. It is apparent that the velocity is reconstructed very well with a relative RMSE of 0.01. It is especially worth noting that the uncertainty is largest around the drop in velocity, i.e. at the location of the leakage, as expected. This uncertainty information could further be used to estimate the location of the leakage. While the state estimation is accurate, it is apparent that the velocity estimation is slightly worse in the domain to the right of the leakage ($x > x_l$). The lack of accuracy is accompanied by an increased standard deviation in that part of the domain. Hence, the uncertainty estimates provide useful information.

While the MCGAN is performing well in the interior of the domain, it is noteworthy that the estimation at the boundary at $x = 2000$ is not as accurate.

In Figure 3.10, we see the estimated posterior distributions of the leakage location and discharge coefficient, respectively. In the leakage location posterior, the estimated mean is close to the true mean (see also Table 3.3) and it is, more or less, symmetric. In the discharge coefficient posterior, on the other hand, the estimated value appears to be smaller than the true value. The MCGAN method significantly outperforms the PCE and EnKF methods in this case. The EnKF is initiated with $x_l = 1000$ and $C_d = 5 \cdot 10^{-4}$ and the MCMC with PCE is initiated at the MAP estimate. In Figures 3.11 and 3.12, the results obtained using the EnKF and PCE method are shown. None of the two approaches manages to estimate the state or the parameters in a satisfying manner. Table 3.3 shows that the EnKF approach is unable to update the posterior and the PCE approach only performs marginally better.

The pipe flow equations are highly nonlinear and the solution exhibits a discontinuity at the location of the leakage. Both phenomena are not easy to handle with the PCE nor EnKF approaches, while neural networks have been shown to be well-suited for such tasks.

On the other hand, DBI performs highly satisfactory (see Figure 3.13). For the leak location, DBI is slightly more accurate, while for the leak size, MCGAN performs better. Furthermore, the distribution over the leak size computed by DBI is narrow, suggesting that the approximated value has small uncertainty associated with it, even though the approximation is not accurate. MCGAN, on the other hand, shows larger uncertainty associated with the approximation suggesting more accurate evaluation of the reliability. Lastly, we see that MCGAN approximates the state more accurately than DBI.

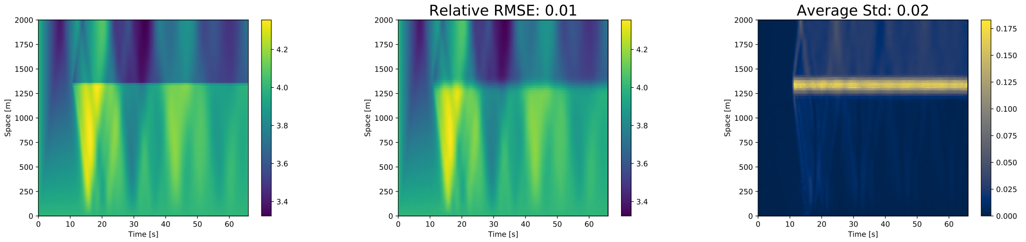
As mentioned, when using DBI, one has to choose the exact location and temporal frequency of incoming observations before training, to create the training set. This is not the case for MCGAN, where the exact sensor configuration and likelihood do not have to be specified before the online stage.

Table 3.3: Estimated parameters for the pipe flow using MCGAN, PCE, and EnKF. The best estimates are highlighted in boldface.

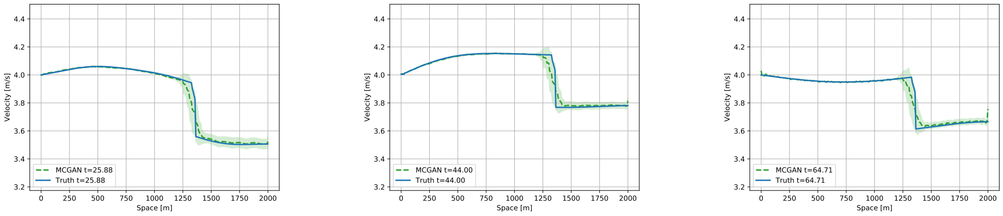
		MCGAN		PCE		EnKF		DBI	
Pars	True val.	Mean	Std	Mean	Std	Mean	Std	Mean	Std
x_l	1354.5	1336.8	44.80	1099.2	69.98	1005.0	10.15	**1357.2**	35.7
C_d	$3.5{\cdot}10^{-4}$	$\mathbf{3.1{\cdot}10^{-4}}$	$5.7{\cdot}10^{-5}$	$2.6{\cdot}10^{-4}$	$2.4{\cdot}10^{-5}$	$7.2{\cdot}10^{-4}$	$3.2{\cdot}10^{-3}$	$2.7{\cdot}10^{-4}$	$1.2{\cdot}10^{-5}$

3.5.3 Summary of Results

To summarize the results obtained using the proposed MCGAN method, we highlight accuracy and computation time. Firstly, in Table 3.4 the relative RMSE for the state and parameters

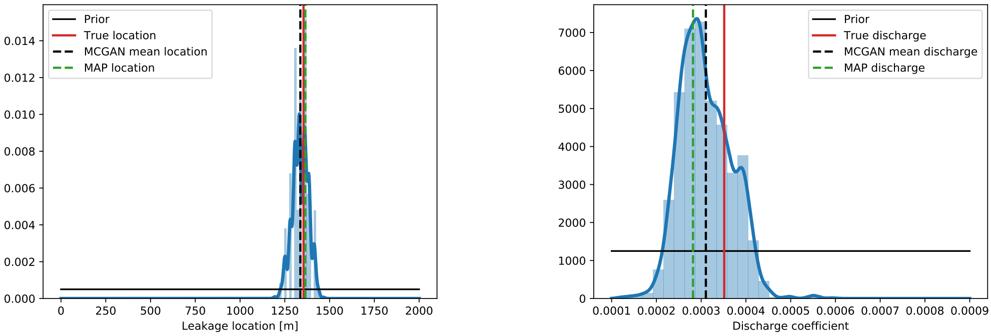


(a) True velocity. (b) Approximated velocity. (c) Velocity standard deviation.



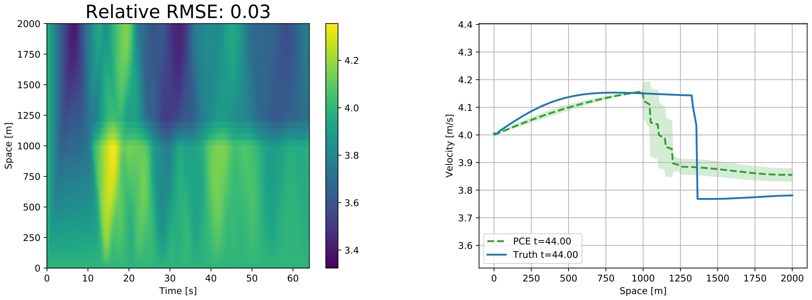
(d) Time 25.88 sec. (e) Time 44.00 sec. (f) Time 64.71 sec.

Figure 3.9: Results for the MCGAN method applied to the pipe flow with a leakage, Eq. (3.39). (a)-(c) are space-time contour plots of the true state, the MCGAN estimated state, and the standard deviation, respectively. (d)-(f) show the state reconstruction at various instances in time with the shaded area denoting one standard deviation away from the reconstruction.



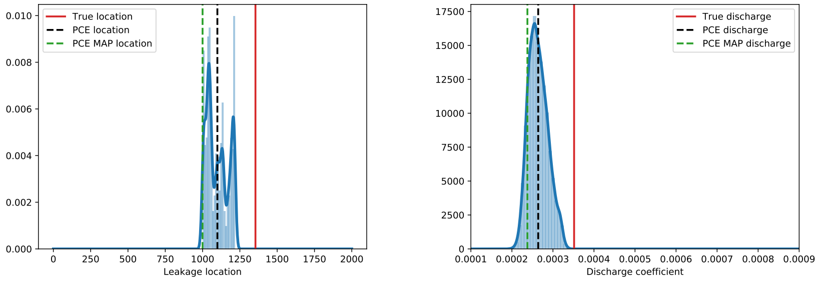
(a) Leakage location. (b) Discharge coefficient.

Figure 3.10: Posterior distributions of the leakage location and discharge coefficient in the pipe flow equation.



(a) Approximated velocity.

(b) Reconstruction at $t = 44$



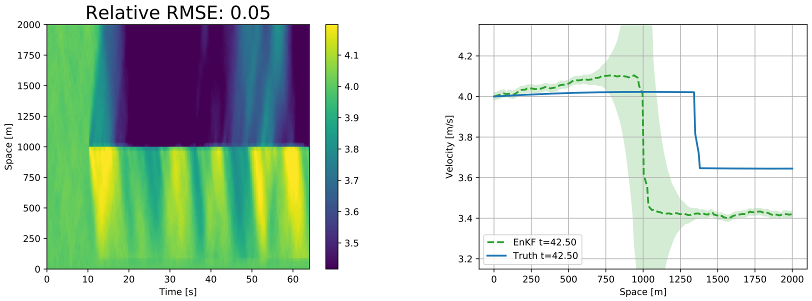
(c) Leakage location.

(d) Discharge coefficient.

Figure 3.11: Results for the MCMC sampling with a PCE surrogate model applied to the pipe flow with a leakage, Eq. (3.39). In (a) we see the space-time contour plots of the reconstructed velocity. In (b) we see the velocity reconstruction at $t = 44$ with the shaded area denoting the standard deviation. In (c)-(d) we see the posterior distributions of the leakage location and discharge coefficient.

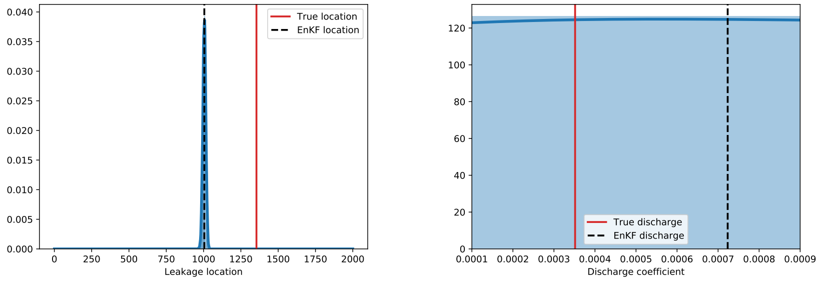
are presented for both test cases. The MCGAN performs better than the three alternative approaches in almost all metrics. For the leakage localization in the pipe flow test case, the MCGAN method outperforms the PCE and the EnKF approaches, while performing similarly to DBI. The MCGAN results are very close to the true values with relative RMSEs that are one order of magnitude better than PCE and ensemble Kalman approaches for the state and parameter estimation. In Table 3.5, the online computation times for the methods applied to the two test cases are shown. It is interesting to note that the computation time does not change much in the two test cases for the MCGAN. This is due to the fact that there are only minor differences in computation time between evaluating a small neural network and a large one. DBI is the fastest approach since the GAN is trained to sample directly from the posterior, in contrast to the MCGAN approach that makes use of MCMC methods.

Lastly, we briefly comment on the offline training time. For the computationally most expensive case, the pipe flow, the most time consuming part is the generation of data. Generating 100,000 training trajectories took about 80 hours on 30 CPU cores (90 seconds



(a) Approximated velocity.

(b) Reconstruction at $t = 44$



(c) Leakage location.

(d) Discharge coefficient.

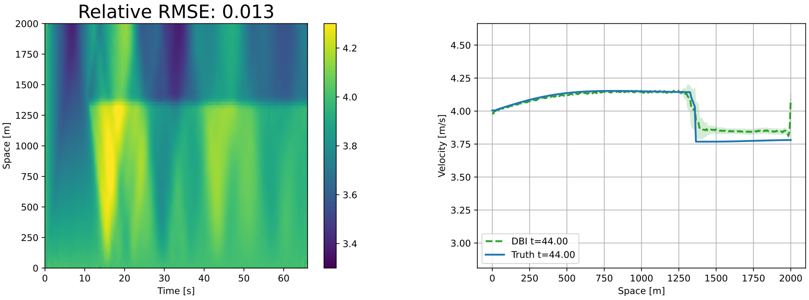
Figure 3.12: Results for EnKF method applied to the pipe flow with a leakage, Eq. (3.39). In (a) we see the space-time contour plots of the reconstructed velocity. In (b) we see the velocity reconstruction at $t = 44$ with the shaded area denoting the standard deviation. In (c)-(d) we see the posterior distributions of the leakage location and discharge coefficient.

per trajectory). The training of the GAN was finished in about 24 hours for both the MCGAN and DBI. In total, the offline stage took approximately 104 hours. The offline time for the Darcy flow was shorter, totaling around 50 hours. We did not experience high sensitivity to the hyperparameters, such as learning rate, batch size, etc. This might be a product of the large number of training samples.

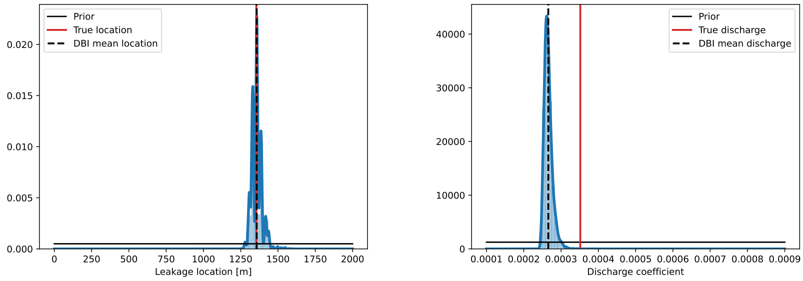
3.6 Conclusion

We have presented a new method, named MCGAN, to efficiently and accurately solve Bayesian inverse problems in physics and engineering applications. The method combines Generative Adversarial Networks and Markov Chain Monte Carlo methods to sample from posterior distributions by utilizing a low-dimensional latent space and a push-forward map defined as a neural network.

The methodology was divided into two distinct stages, an offline stage, in which the GAN



(a) Approximated velocity. (b) Reconstruction at $t = 44$



(c) Leakage location. (d) Discharge coefficient.

Figure 3.13: Results for DBI method applied to the pipe flow with a leakage, Eq. (3.39). In (a) we see the space-time contour plots of the reconstructed velocity. In (b) we see the velocity reconstruction at $t = 44$ with the shaded area denoting the standard deviation. In (c)-(d) we see the posterior distributions of the leakage location and discharge coefficient.

was trained on simulated training data in order to learn the prior distribution, and an online stage, in which the inverse problem was solved for a new set of observations. While the offline stage potentially takes significant computational time, the online stage was computationally very fast and efficient.

We presented a proof of theoretical convergence of the posterior distribution in the Wasserstein-1 distance, in the case where the GAN would be perfectly trained. Furthermore, we provided the insight that sampling from the latent space yielded essentially the same results as sampling from the high-dimensional space, in a weak sense.

To showcase the method's performance, we applied it to two computational engineering test cases with different characteristics and compared it to three alternative approaches. In the high-dimensional problem, the Darcy flow with uncertain permeability field, an improved accuracy was found with MCGAN compared with EKI and DBI, as well as a speed-up of one order of magnitude compared to the EKI method. In the second test case, the leakage localization for flow in a pipe, the MCGAN approach showed increased accuracy for the state

Table 3.4: Relative RMSE for the state and parameter estimation for the various test cases. For the Darcy flow, the Relative RMSE for (v_1, v_2, p) is computed. For the pipe flow, the Relative RMSE for (u, p) is computed. The best performing cases are highlighted in boldface.

	MCGAN		PCE		EnKF/EKI		DBI	
	State	Pars	State	Pars	State	Pars	State	Pars
Darcy flow	$\mathbf{1.3{\cdot}10^{-1}}$	$\mathbf{1.7{\cdot}10^{-1}}$	-	-	$2.0{\cdot}10^{-1}$	$4.2{\cdot}10^{-1}$	$1.9{\cdot}10^{-1}$	$4.9{\cdot}10^{-1}$
Pipe flow	$\mathbf{4.8{\cdot}10^{-3}}$	$1.3{\cdot}10^{-2}$	$1.3{\cdot}10^{-2}$	$1.9{\cdot}10^{-1}$	$3.3{\cdot}10^{-2}$	$2.6{\cdot}10^{-1}$	$7.0{\cdot}10^{-3}$	$\mathbf{2.0{\cdot}10^{-3}}$

Table 3.5: Comparison of online computation time. All simulations are run on CPU cores. Only the number of CPU cores varies.

	MCGAN (1 core)	PCE (20 cores)	EnKF/EKI (20 cores)	DBI (1 core)
Darcy flow	$3.17{\cdot}10^2$s	-	$4.11{\cdot}10^3$s	$3.31{\cdot}10^1$s
Pipe flow	$7.10{\cdot}10^2$s	$1.18{\cdot}10^4$s	$1.25{\cdot}10^4$s	$3.14{\cdot}10^1$s

and leak size detection, while DBI was slightly more accurate regarding the leak location. Out of the four approaches, the MCGAN method was the one that provided accurate results, fast sampling without being trained to only work for a single sensor configuration.

It is worth noting that, similar to any surrogate modeling approach, it is unclear how well the methodology performs on out-of-distribution cases. This is a subject to future study.

While the MCGAN approach performed well on the two test cases, there is still room for future research. We believe the offline stage can be improved by identifying optimal ways of simulating training data and determining hyperparameters for the GAN. This includes determining the optimal size of the latent space. Furthermore, the GAN can be improved by incorporating physics knowledge either in the training or directly in the neural network architecture. This could possibly alleviate the boundary estimation problems. Also, possibilities of using the MCGAN framework in a sequential fashion, as is the case for Kalman filters, would be an interesting direction to explore.

In conclusion, we believe that the MCGAN methodology can form an important piece of the puzzle towards a well performing digital twin framework, in which real-time state and parameter estimation is of crucial importance.

4

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

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, it 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 inherent uncertainty in such models. By embedding this surrogate model into a Bayesian inference scheme, a leak is located by combining sensor observations with model output approximating the true posterior distribution over possible leak locations. We show that our methodology enables fast, accurate and trustworthy results. It shows 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 ranges from 0.3 to 3 for varying numbers of sensors and measurement noise. For two

---

This chapter is based on the article “A Probabilistic Digital Twin for Leak Localization in Water Distribution Networks Using Generative Deep Learning”, *Sensors*, 23.13:6179, 2023 [106]

more complex test cases, the ATD ranges from 0.75 to 4 and from 1.5 to 10, respectively. Furthermore, accuracies upwards of 83%, 72%, and 42% for the three test cases respectively are achieved. The computation times range from 0.1 to 13 seconds, depending on the sizes of the neural networks 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.

4.1 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. The distribution is typically done by using networks of pipes, which can be highly intricate and involve multiple components, such as several kilometres of pipe segments, 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 quick and trust-worthy monitoring in place. Monitoring is typically done by recording information from a number of sensors installed at critical locations within the network. However, detecting the occurrences and locations of leaks can still be a challenging task, even with sensor data 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 chapter, 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. Secondly, it must be computationally fast so that any leakage can be identified quickly. Third, it must be reliable. That is, the framework should not just provide an estimated leak location, but also a measure of the level of confidence in the estimate. Fourth, it should be general and work under different circumstances. The framework should be sufficiently generic to work on different water distribution networks with widely varying complexity. Lastly, 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, whether it can handle different kinds of sensor readings or varying numbers of sensors, etc. There are not many approaches that satisfy all 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 special cases, and they struggle with incorporating prior knowledge as they will then lose their general nature. The framework presented satisfies all requirements, at the cost of a computationally intensive training stage.

4.1.1 Leak localization literature

There is already a significant amount of research in the area of leak localization. However, not many approaches satisfy all the above mentioned requirements. The authors in [149] and [75] make use of a model to generate synthetic sensor observations. The residuals between the model output and the observed values are then used to predict the leak location using a trained classifier. Similarly, in [103] residuals are employed for leak detection after which the characteristics of the residuals are used for localizing the leak. In [130], an encoding of the sensor observations is used together with a trained classifier and a graph theory-based clustering method. While these approaches are computationally fast and are shown to be accurate on selected test cases, they mainly work for the sensor configurations they are trained on and are limited in uncertainty quantification as they do not model the inherent input uncertainty in e.g. the demand. Furthermore, even if available, prior information cannot be embedded into these approaches. On the other hand, in [140], a combination of model- and graph theory-driven techniques is used together with online training of a neural network classifier to predict a cluster of nodes in which the leak is present. This is a flexible setup that allows for changes in sensor configurations. However, uncertainty quantification is still limited as only the size of the predicted cluster is used as a proxy for reliability of the prediction.

4.1.2 Bayesian inference

As an alternative to the above mentioned approaches, one can make use of Bayesian inference. This allows one to solve the leak localization problem accurately with uncertainty quantification. The general technique is to compute the data likelihood and combine it 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. Within the likelihood, both the sensor noise and model uncertainty are modeled. That is, the uncertainty associated with imperfect sensors and e.g. stochastic nodal demand in the water network can be included in the computations of the posterior directly. Furthermore, one can incorporate prior information in a straight-forward way. 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 [155], the authors overcome 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 restriction.

4.1.3 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 computational speed-up without essential loss of accuracy. Conventionally, a linear dimensionality reduction, like proper orthogonal decomposition (POD), is used together with some regression method in the reduced space. E.g. in [99], a combination of POD and radial basis functions with neural networks is used in inverse analysis for structural diagnoses. In [38], POD is used with stochastic spectral methods to relate the input to the output within a Bayesian inverse problem to speed it up. In [56], POD is combined with Gaussian processes for speeding up nonlinear structural analysis. In recent years, deep neural networks have become popular choices for surrogate models in scientific computing due to their performance in dimensionality reduction and their predictive power [65, 91, 105]. For the purpose of speeding up Bayesian inference, we want to model a high-dimensional stochastic problem. We will make use of the concept of generative modeling. While applications of generative modeling in various scientific fields are already widespread [30, 107, 117], 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 data in order to be able to sample from it after a training phase. This enables fast sampling from complicated and high-dimensional distributions. There are several kinds of generative neural networks, such as generative adversarial networks [51], variational autoencoders [81], and diffusion models [144]. Each of these models has its advantages and drawbacks. There are specifically three factors to consider when using generative models: sampling quality, speed, and diversity [163]. In this work, we adopt the Wasserstein Autoencoder [151] as it performs highly satisfactory on these three criteria.

4.1.4 Overview chapter

In this chapter, 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 drawback, we make use of neural networks as the stochastic surrogate model. The neural network is trained as a generative model to estimate the distribution of pressure heads and flow rates given a leak location. This enables fast evaluation of the likelihood function while retaining accuracy.

In Section 4.2, we present the theory behind the 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 the neural network architectures, are presented. In Section 4.3, we combine the components from Section 4.2 into the presented framework. In Section 4.4, we showcase the framework on three test cases. The chapter is then concluded in Section 4.5.

4.2 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 demand, pipe roughness, and leak location. Then, we describe leak localization as a Bayesian inverse problem, 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 architectures. Specifically, we discuss Wasserstein autoencoders, which are crucial in 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.

4.2.1 Problem Setting

We consider a water distribution network which 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

$$h_i - h_k = r_j Q_j^n, \tag{4.1}$$

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, that depends on its length, diameter, and the material of the pipe. Define $\boldsymbol{q} = (Q_1, \dots, Q_{N_p})^\top$, as a vector of unknown fluid flow rates in the pipes.

The network topology is modeled by 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

$$A_b = \begin{cases} -1 & \text{if the flow in pipe j enters the node i,} \\ 0 & \text{if the j does not connect to the node i,} \\ 1 & \text{if the flow in pipe j leaves the node i,} \end{cases} \tag{4.2}$$

where $b = 1, 2$. The unknown heads at different nodes are defined as $\boldsymbol{h} = (h_1, \dots, h_{n_j})^\top$, 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,

$$G_{jj}(\boldsymbol{q}) = r_j |Q_j|^{n-1}. \tag{4.3}$$

The hydraulic problem is to solve 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 [143] for details):

$$f(\boldsymbol{x}) = \begin{pmatrix} G(\boldsymbol{q}) & -A_1 \\ -A_1^\top & \boldsymbol{O} \end{pmatrix} \begin{pmatrix} \boldsymbol{q} \\ \boldsymbol{h} \end{pmatrix} - \begin{pmatrix} A_2 \boldsymbol{e}_l \\ \boldsymbol{d}_m \end{pmatrix} = \boldsymbol{o}, \tag{4.4}$$

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 ([131]) 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

$$d_{\text{leak}} = C_d A p^\alpha \sqrt{\frac{2}{\rho}}. \tag{4.5}$$

Typically, $\alpha = 0.5$ is chosen [131]; 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 $\rho[kg/m^3]$ is the density (not to be confused with the probability density functions used later in the chapter). 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 in the pipe section in which the leak will occur. The demand on that node is the leak demand.

4.2.2 Leak Localization as a Bayesian Inverse Problems

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, also the uncertainty associated with the estimate.

We will 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 $c \in \{1, \dots, N_p\}$, 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:

$$\boldsymbol{y} = H(F(c, \boldsymbol{\omega})) + \boldsymbol{\eta}. \tag{4.6}$$

The forward model, F, is closely related to Eq. (4.4). The output, $F(c, \boldsymbol{\omega}) = \boldsymbol{x}$, is the solution to Eq. (4.4) for given c and $\boldsymbol{\omega}$. The uncertain parameters, $\boldsymbol{\omega}$, depend on the problem at hand, and can, for example, be 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 leak location and parameters to the solution of Eq. (4.4). 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 observations, $P_{c|\boldsymbol{y}}$. Since c is a discrete variable, we have to compute the posterior probability of all possible values of c, $P_{c|\boldsymbol{y}}(c_k|\boldsymbol{y})$, $c_k = 1, \dots, N_p$. Using Bayes' theorem for density functions, we get:

$$\rho_{c|\boldsymbol{y}}(c_k|\boldsymbol{y}) = \frac{\rho_{\boldsymbol{y}|c}(\boldsymbol{y}|c_k)\rho_c(c_k)}{\rho_{\boldsymbol{y}}(\boldsymbol{y})}, \tag{4.7}$$

where $\rho_{\boldsymbol{y}|c}$ is referred to as the likelihood, ρ_c is the prior, $\rho_{\boldsymbol{y}}$ 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 $\rho_{c|\boldsymbol{y}}$ sums to one over all possible values of c_k. It is given by:

$$\rho_{\boldsymbol{y}}(\boldsymbol{y}) = \sum_{k=1}^{N_p} \rho_{\boldsymbol{y}|\boldsymbol{c}}(\boldsymbol{y}|c_k). \tag{4.8}$$

The likelihood is not directly available, however, from Eq. (4.6) we get:

$$\rho_{\boldsymbol{y}|c,\boldsymbol{\omega}}(\boldsymbol{y}|c_k, \boldsymbol{\omega}) = \rho_{\boldsymbol{\eta}}(\boldsymbol{y} - H(F(c_k, \boldsymbol{\omega}))), \tag{4.9}$$

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

$$\begin{aligned} \rho_{\boldsymbol{y}|c}(\boldsymbol{y}|c_k) &= \int_{-\infty}^{-\infty} \rho_{\boldsymbol{y}|c,\boldsymbol{\omega}}(\boldsymbol{y}|c_k, \boldsymbol{\omega}) \rho_{\boldsymbol{\omega}|c}(\boldsymbol{\omega}|c_k) \mathrm{d}\boldsymbol{\omega} \\ &= \int_{-\infty}^{-\infty} \rho_{\boldsymbol{y}|c,\boldsymbol{\omega}}(\boldsymbol{y}|c_k, \boldsymbol{\omega}) \rho_{\boldsymbol{\omega}}(\boldsymbol{\omega}) \mathrm{d}\boldsymbol{\omega} \\ &= \mathbb{E}_{\boldsymbol{\omega} \sim P_{\boldsymbol{\omega}}} \left[\rho_{\boldsymbol{y}|c,\boldsymbol{\omega}}(\boldsymbol{y}|c_k, \boldsymbol{\omega}) \right]. \end{aligned} \tag{4.10}$$

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:

$$\begin{aligned} \rho_{c|\boldsymbol{y}}(c_k|\boldsymbol{y}_i) &= \frac{\rho_{\boldsymbol{y}|c}(\boldsymbol{y}_i|c_k) \rho_{c|\boldsymbol{y}}(c_k|\boldsymbol{y}_{i-1})}{\rho_{\boldsymbol{y}}(\boldsymbol{y}_i)} \\ &= \frac{\mathbb{E}_{\boldsymbol{\omega} \sim P_{\boldsymbol{\omega}}} \left[\rho_{\boldsymbol{y}|c,\boldsymbol{\omega}}(\boldsymbol{y}_i|c_k, \boldsymbol{\omega}) \right] \rho_{c|\boldsymbol{y}}(c_k|\boldsymbol{y}_{i-1})}{\sum_{j=1}^{N_p} \mathbb{E}_{\boldsymbol{\omega} \sim P_{\boldsymbol{\omega}}} \left[\rho_{\boldsymbol{y}|c,\boldsymbol{\omega}}(\boldsymbol{y}_i|c_j, \boldsymbol{\omega}) \right]}. \end{aligned} \tag{4.11}$$

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

$$\rho_{c|\boldsymbol{y}}(c_k|\boldsymbol{y}_{0:N_t}) = \rho_c(c_k) \prod_{i=0}^{N_t} \frac{\mathbb{E}_{\boldsymbol{\omega} \sim P_{\boldsymbol{\omega}}} \left[\rho_{\boldsymbol{y}|c,\boldsymbol{\omega}}(\boldsymbol{y}_i|c_k, \boldsymbol{\omega}) \right]}{\sum_{j=1}^{N_p} \mathbb{E}_{\boldsymbol{\omega} \sim P_{\boldsymbol{\omega}}} \left[\rho_{\boldsymbol{y}|c,\boldsymbol{\omega}}(\boldsymbol{y}_i|c_j, \boldsymbol{\omega}) \right]}. \tag{4.12}$$

We can write down the expression $\rho_{c|\boldsymbol{y}}(c_k|\boldsymbol{y}_{0:N_t})$ for all k, but it is infeasible 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 Eq. (4.10), high-dimensional and it is thereby infeasible to compute the likelihood, $\mathbb{E}_{\boldsymbol{\omega} \sim P_{\boldsymbol{\omega}}} \left[\rho_{c|\boldsymbol{y},\boldsymbol{\omega}}(\boldsymbol{y}_i|c_k, \boldsymbol{\omega}) \right]$, for all k in real-time;

- $\rho_{c|\boldsymbol{y},\boldsymbol{\omega}}(\boldsymbol{y}_i|\boldsymbol{c}_k,\boldsymbol{\omega})$ can be expensive to evaluate as it requires solving Eq. (4.4).

While several methods exist that address the computation of stochastic integrals, most of them has 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 [60]. 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 [164]. We will make use of deep learning as it allows us to deal with arbitrary distributions and high-dimensional problems.

4.2.3 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. The Supervised Wasserstein Autoencoder (SupWAE) forms a type of neural network that simultaneously achieves a problem dimensionality reduction as well as an approximation of the relevant distribution [151]. Before explaining the SupWAE, we briefly introduce the regular autoencoder (AE) and add the necessary components for leak detection to it.

Autoencoders

A regular autoencoder (AE) is often used to identify an accurate low-dimensional representation of the data [84]. The low-dimensional representation is then referred to as the latent state which is an element of the latent space, while the original data is 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:

$$\phi_{\text{enc}}(\boldsymbol{x}) = \boldsymbol{z}, \quad \phi_{\text{dec}}(\boldsymbol{z}) = \tilde{\boldsymbol{x}}, \quad \phi_{\text{dec}}(\phi_{\text{enc}}(\boldsymbol{x})) = \tilde{\boldsymbol{x}} \approx \boldsymbol{x}. \tag{4.13}$$

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:

$$L_{\text{AE}}(\phi_{\text{enc}}, \phi_{\text{dec}}) = \frac{1}{N} \sum_{zi=1}^{N} (\boldsymbol{x}_i - \phi_{\text{dec}}(\phi_{\text{enc}}(\boldsymbol{x}_i)))^2 + \alpha R(\phi_{\text{enc}}, \phi_{\text{dec}}) \tag{4.14}$$

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 weight decay within machine learning; α determines how much we regularize the weights. The purpose of the regularization is to avoid overfitting to the training data.

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 result in small perturbations in the high-fidelity space as well. Moreover, it should be possible to sample from the latent space. By using Wasserstein AEs (WAEs) [151], we can also obtain these properties.

We introduce a prior distribution on the latent space, $P_{\boldsymbol{z}}(\boldsymbol{z})$. 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, by the densities:

$$\rho_{\boldsymbol{z}}(\boldsymbol{z}) = \int \rho(\boldsymbol{z}|\boldsymbol{x})\rho_{\boldsymbol{x}}(\boldsymbol{x})\mathrm{d}\boldsymbol{x} \approx \int \rho_{\text{enc}}(\boldsymbol{z}|\boldsymbol{x})\rho_{\boldsymbol{x}}(\boldsymbol{x})\mathrm{d}\boldsymbol{x} = \rho_{\text{enc}}(\boldsymbol{z}), \tag{4.15}$$

$$\rho_{\boldsymbol{x}}(\boldsymbol{x}) = \int \rho(\boldsymbol{x}|\boldsymbol{z})\rho_{\boldsymbol{z}}(\boldsymbol{z})\mathrm{d}\boldsymbol{z} \approx \int \rho_{\text{dec}}(\boldsymbol{x}|\boldsymbol{z})\rho_{\boldsymbol{z}}(\boldsymbol{z})\mathrm{d}\boldsymbol{z} = \rho_{\text{dec}}(\boldsymbol{x}). \tag{4.16}$$

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

$$L_{\text{WAE}}(\phi_{\text{enc}}, \phi_{\text{dec}}) = \underbrace{\frac{1}{N}\sum_{i=1}^{N}\left(\boldsymbol{x}_i - \phi_{\text{dec}}(\phi_{\text{enc}}(\boldsymbol{x}_i))\right)^2}_{\text{reconstruction}} + \lambda \underbrace{D(P_{\boldsymbol{z}}(\boldsymbol{z}), P_{\text{enc}}(\boldsymbol{z}))}_{\text{divergence}} + \alpha \underbrace{R(\phi_{\text{enc}}, \phi_{\text{dec}})}_{\text{regularization}}. \tag{4.17}$$

Here, λ is another regularization parameter to be determined through hyperparameter tuning. In this chapter, we choose the maximum mean discrepancy (MMD) with a multiquadratics kernel for the divergence, which gives us the WAE-MMD. In [53], it was argued that this is an accurate choice when $P_{\boldsymbol{z}}(\boldsymbol{z})$ is the normal distribution, see [151]. Summarizing, we obtain:

$$L_{\text{WAE}}(\phi_{\text{enc}}, \phi_{\text{dec}}) \quad = \quad \frac{1}{N}\sum_{i=1}^{N}\left(\boldsymbol{x}_i - \phi_{\text{dec}}(\phi_{\text{enc}}(\boldsymbol{x}_i))\right)^2 + \lambda \text{MMD}\left(\phi_{\text{enc}}, \{\boldsymbol{z}_i\}_{i=1}^{N}\right)$$

$$+ \quad \alpha R(\phi_{\text{enc}}, \phi_{\text{dec}}),$$

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

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

and

$$k(\boldsymbol{z}_l, \boldsymbol{z}_j) = \frac{C}{C + ||\boldsymbol{z}_l - \boldsymbol{z}_j||_2^2}. \tag{4.18}$$

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

$$\begin{aligned} \rho_{\boldsymbol{x}|c}(\boldsymbol{x}|c) &= \int \rho_{\boldsymbol{x}|c,\boldsymbol{z}}(\boldsymbol{x}|c, \boldsymbol{z}) \rho_{\boldsymbol{z}}(\boldsymbol{z}) \mathrm{d}\boldsymbol{z} \\ &\approx \int \rho_{\text{dec}}(\boldsymbol{x}|c, \boldsymbol{z}) \rho_{\boldsymbol{z}}(\boldsymbol{z}) \mathrm{d}\boldsymbol{z} \\ &= \int \phi_{\text{dec}}(\boldsymbol{z}, c) \rho_{\boldsymbol{z}}(\boldsymbol{z}) \mathrm{d}\boldsymbol{z} \\ &= \rho_{\text{dec}}(\boldsymbol{x}|c), \end{aligned} \tag{4.19}$$

so that c is considered a known condition. During training, the WAE sees pairs $(\boldsymbol{x}, c)$ and uses the 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 4.1.

It is important to note that different kinds of neural network architectures can be incorporated in the WAEs framework. Therefore, one should choose the architecture that performs optimally for the data type. In this chapter, we showcase the performance using two different kinds of architectures – residual neural networks (ResNet) [63] and transformers.

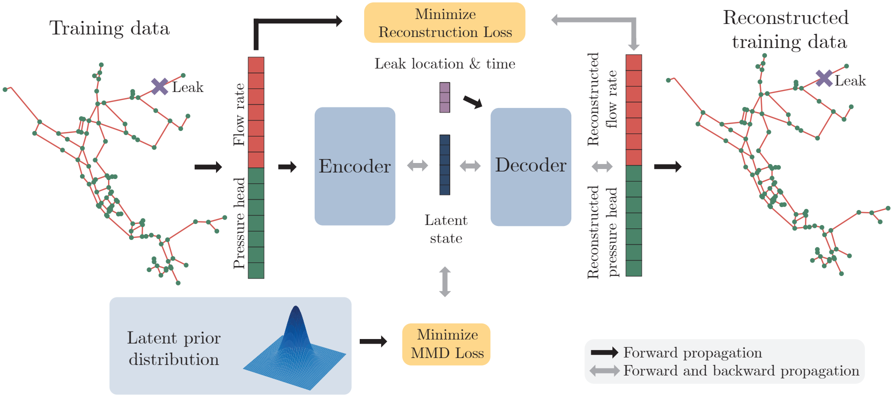


Figure 4.1: Illustration of an adversarial AE for reconstruction of a water distribution network.

4.3 Proposed Framework

The aim of the proposed framework is to overcome the challenges related to solving Bayesian inverse problems, as described in Section 4.2.2, while maintaining high accuracy when computing the posterior over the parameters of interest. The framework employed here is an extension to the one presented in [107].

We will use the generative neural network as a stochastic digital twin of the WDN. It 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 it 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, given random noise, leak location, and time of the 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 4.2.2, we can rewrite the posterior for given observations, as follows,

$$\rho_{c|\boldsymbol{y}}(c_k|\boldsymbol{y}_i) = \frac{\rho_c(c_k)\int_{\boldsymbol{z}} \rho_{\boldsymbol{y}|c,\boldsymbol{z}}(\boldsymbol{y}_i|c_k,\boldsymbol{z})\rho_{\boldsymbol{z}}(\boldsymbol{z})\,\mathrm{d}\boldsymbol{z}}{\sum_{j=1}^{N_p}\int_{\boldsymbol{z}} \rho_{\boldsymbol{y}|c,\boldsymbol{z}}(\boldsymbol{y}_i|c_j,\boldsymbol{z})\rho_{\boldsymbol{z}}(\boldsymbol{z})\,\mathrm{d}\boldsymbol{z}} = \frac{\rho_c(c_k)\mathbb{E}_{\boldsymbol{z}\sim P_{\boldsymbol{z}}}\left[\rho_{\boldsymbol{y}|c,\boldsymbol{z}}(\boldsymbol{y}_i|c_k,\boldsymbol{z})\right]}{\sum_{j=1}^{N_p}\mathbb{E}_{\boldsymbol{z}\sim P_{\boldsymbol{z}}}\left[\rho_{\boldsymbol{y}|c,\boldsymbol{z}}(\boldsymbol{y}_i|c_j,\boldsymbol{z})\right]}, \tag{4.20}$$

where the likelihood is computed by:

$$\rho_{\boldsymbol{y}|c,\boldsymbol{z}}(\boldsymbol{y}_i|c_k,\boldsymbol{z}) = \rho_{\boldsymbol{\eta}}(\boldsymbol{y}_i - H(\phi_{\mathrm{dec}}(\boldsymbol{z},c_k))). \tag{4.21}$$

As in Eqs. (4.11) and (4.12), 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:

$$\rho_{c|\boldsymbol{y}}(c_k|\boldsymbol{y}_{0:N_t}) = \rho_c(c_k) \prod_{i=0}^{N_t} \frac{\mathbb{E}_{\boldsymbol{z}\sim P_{\boldsymbol{z}}}\left[\rho_{\boldsymbol{y}|c,\boldsymbol{z}}(\boldsymbol{y}_i|c_k,\boldsymbol{z})\right]}{\sum_{j=1}^{N_p} \mathbb{E}_{\boldsymbol{z}\sim P_{\boldsymbol{z}}}\left[\rho_{\boldsymbol{y}|c,\boldsymbol{z}}(\boldsymbol{y}_i|c_j,\boldsymbol{z})\right]}. \tag{4.22}$$

At time t_0, we simply use a uniform prior.

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

$$\phi_{\text{dec}}(\boldsymbol{z}, c, t_i) = \boldsymbol{x}_i(c, \boldsymbol{\omega}) = (\boldsymbol{q}_i^T(c, \boldsymbol{\omega}), \boldsymbol{h}_i^T(c, \boldsymbol{\omega}))^T. \tag{4.23}$$

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

The proposed framework essentially resolves the challenges described in Section 4.2.2:

- $P_{\boldsymbol{\omega}}$ is replaced by the latent prior, $P_{\boldsymbol{z}}$, which is known as we used it during training of the WAE-MMD;

- $\boldsymbol{z}$ is of much lower dimension than $\boldsymbol{\omega}$, which makes the evaluation of the integral in Eq. (4.10) fast and thereby enables real-time computations of the likelihood, $\mathbb{E}_{\boldsymbol{z}\sim P_{\boldsymbol{z}}}\left[\rho_{\boldsymbol{y}|c,\boldsymbol{z}}(\boldsymbol{y}_i|c_k,\boldsymbol{z})\right]$;

- The likelihood, $\rho_{\boldsymbol{y}|c_k,\boldsymbol{z}}$, is computed by a forward propagation of the decoder instead of solving the 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. The two stages are outlined in Algorithms 6 and 7, respectively. Furthermore, the online stage is visualized in Figure 4.2.

```
Algorithm 6: Offline stage
  Input: $N_{\text{train}}$, training hyperparameters, WAE-MMD architecture
1 Generate training samples, $\{(\mathbf{x}_i, c_i)\}_{i=1}^{N_{\text{train}}}$, by solving the forward problem (see
   Section 4.2.1);
2 Train the supervised WAE-MMD (see Section 4.2.3);
  Output: $\phi_{\text{dec}}$
```

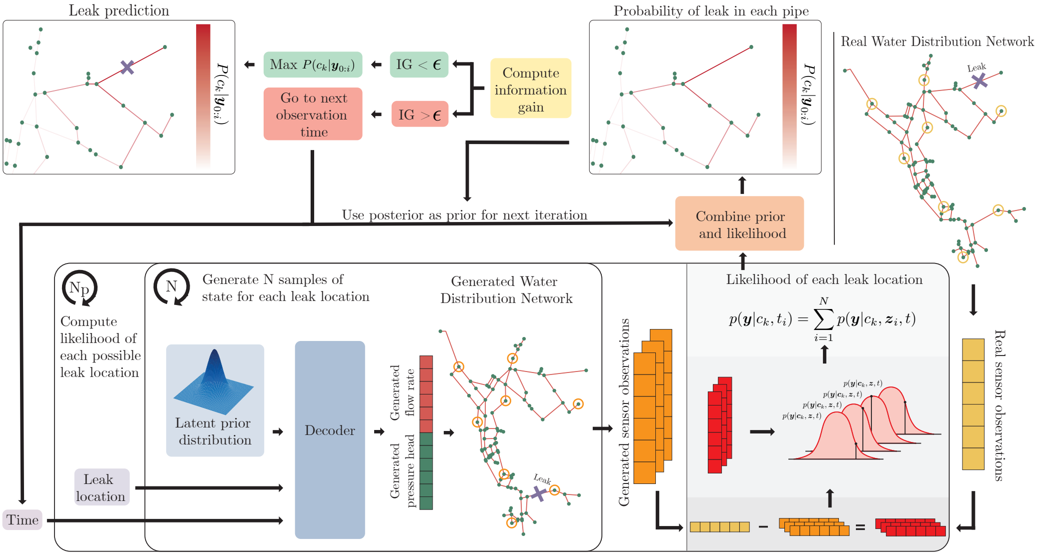

Leak prediction
$P(c_k|\boldsymbol{y}_{0:i})$
Max $P(c_k|\boldsymbol{y}_{0:i})$
IG $< \epsilon$
Go to next observation time
IG $> \epsilon$
Compute information gain
Probability of leak in each pipe
$P(c_k|\boldsymbol{y}_{0:i})$
Real Water Distribution Network
Leak
Use posterior as prior for next iteration
Combine prior and likelihood
Np
Compute likelihood of each possible leak location
N
Generate N samples of state for each leak location
Generated Water Distribution Network
Latent prior distribution
Decoder
Generated flow rate
Generated pressure head
Leak
Leak location
Time
Generated sensor observations
Likelihood of each leak location
$p(\boldsymbol{y}|c_k, t_i) = \sum_{i=1}^{N} p(\boldsymbol{y}|c_k, \boldsymbol{z}_i, t)$
$p(\boldsymbol{y}|c_k, \boldsymbol{z}, t)$
$p(\boldsymbol{y}|c_k, \boldsymbol{z}, t)$
$p(\boldsymbol{y}|c_k, \boldsymbol{z}, t)$
$p(\boldsymbol{y}|c_k, \boldsymbol{z}, t)$
−
=
Real sensor observations

Figure 4.2: Illustration of the online computation of the posterior distribution, $\rho(c_k|\boldsymbol{y}_{0:i})$, $k = 1, \ldots, N_p$.

4.3.1 Stopping Criterion in the Online Stage

With every new set of observations, $\boldsymbol{y}_i$, we receive additional information of the system at hand. However, at some point new observations do not contribute to the accuracy of the posterior anymore. In other words, the posterior density should converge:

$$\rho_{c|\boldsymbol{y}}(c_k|\boldsymbol{y}_{0:N_t}) \rightarrow \rho_{c|\boldsymbol{y}}(c_k|\boldsymbol{y}), \quad \text{for} \quad N_t \rightarrow \infty. \tag{4.24}$$

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, $IG(c, \boldsymbol{y}_i)$, 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:

$$\begin{aligned} IG(c, \boldsymbol{y}_i) &= D_{\text{KL}}(P_{c|\boldsymbol{y}}(c_k|\boldsymbol{y}_{0:i})||P_{c|\boldsymbol{y}}(c_k|\boldsymbol{y}_{0:i-1})) \\ &= -\sum_{k=1}^{N_p} \rho_{c|\boldsymbol{y}}(c_k|\boldsymbol{y}_{0:i}) \log\left(\frac{\rho_{c|\boldsymbol{y}}(c_k|\boldsymbol{y}_{0:i})}{\rho_{c|\boldsymbol{y}}(c_k|\boldsymbol{y}_{0:i-1})}\right). \end{aligned} \tag{4.25}$$

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

Algorithm 7: Online stage

```
Input: $\phi_{\text{dec}}$ from Algorithm 6, observations $\mathbf{y}$, threshold $\epsilon$
$i = 0$
for $c_k = \{1, \dots, N_p\}$ do
    Compute $p_{c|\boldsymbol{y}}(c_k|\boldsymbol{y}_i) = \frac{\rho_c(c_k)\mathbb{E}_{\boldsymbol{z}\sim P_{\boldsymbol{z}}}\left[\rho_{\boldsymbol{y}|c,\boldsymbol{z}}(\boldsymbol{y}_i|c_k,\boldsymbol{z})\right]}{\sum_{j=1}^{N_p}\mathbb{E}_{\boldsymbol{z}\sim P_{\boldsymbol{z}}}\left[\rho_{\boldsymbol{y}|c,\boldsymbol{z}}(\boldsymbol{y}_i|c_j,\boldsymbol{z})\right]}$
end for
while $IG(c, \boldsymbol{y}_i) > \epsilon$ do
    $i = i + 1$
    for $c_k = \{1, \dots, N_p\}$ do
        $\rho_{c|\boldsymbol{y}}(c_k|\boldsymbol{y}_{0:i}) = \rho_c(c_k)\prod_{i=0}^{i}\frac{\mathbb{E}_{\boldsymbol{z}\sim P_{\boldsymbol{z}}}\left[\rho_{\boldsymbol{y}|c,\boldsymbol{z}}(\boldsymbol{y}_i|c_k,\boldsymbol{z})\right]}{\sum_{j=1}^{N_p}\mathbb{E}_{\boldsymbol{z}\sim P_{\boldsymbol{z}}}\left[\rho_{\boldsymbol{y}|c,\boldsymbol{z}}(\boldsymbol{y}_i|c_j,\boldsymbol{z})\right]}$
    end for
end while
Output: $\rho_{c|\boldsymbol{y}}(c|\boldsymbol{y}_{0:i})$
```

4.3.2 Estimating Uncertainty

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

$$H(C) = -\sum_{i=1}^{N_p} \rho_c(c_i|\boldsymbol{y}_{0:1}) \log(\rho_c(c_i|\boldsymbol{y}_{0:1})). \tag{4.26}$$

This tells us how much information or uncertainty we have in the posterior distribution. Low entropy means a sufficient amount of information, i.e. low uncertainty, and high entropy the opposite. Therefore, we can use the entropy as a measure for 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 for applying the methodology in practice as it provides the necessary information to act on a certain prediction. If the entropy is high, it tells us that we need more observations in order to provide a trustworthy prediction.

4.3.3 Model Architectures

As mentioned in Section 4.2, 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 architectures to showcase that the framework can be based on more than one possible choice. Moreover, we will see that each choice shows superior performance for a certain test case. Specifically, we use transformers and dense ResNets in the experiments to follow. For a detailed breakdown of the model architectures, see below and for a visualization of the transformer network, see Figure 4.3. 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 through the 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.

Dense ResNet WAE The encoder consists of a series of ResNet layers, followed by dense layers that reduce the dimension. Similarly, the decoder consists of ResNet layers followed by dense layers that increase the dimension. 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.

Transformer WAE For the encoder, the data is first passed through a dense layer in order to increase the embedding dimension. Then, it is 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 dimension. 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 4.3.

4.4 Results

In this section, we show the performance of the proposed framework on three test cases. We assess the framework's performance with respect to the topological distance and accuracy . We compare the performance of two distinct architectures, a dense ResNet and a transformer architecture. In both cases, the neural networks are similar to the encoder network, but with a softmax activation function at the output.

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

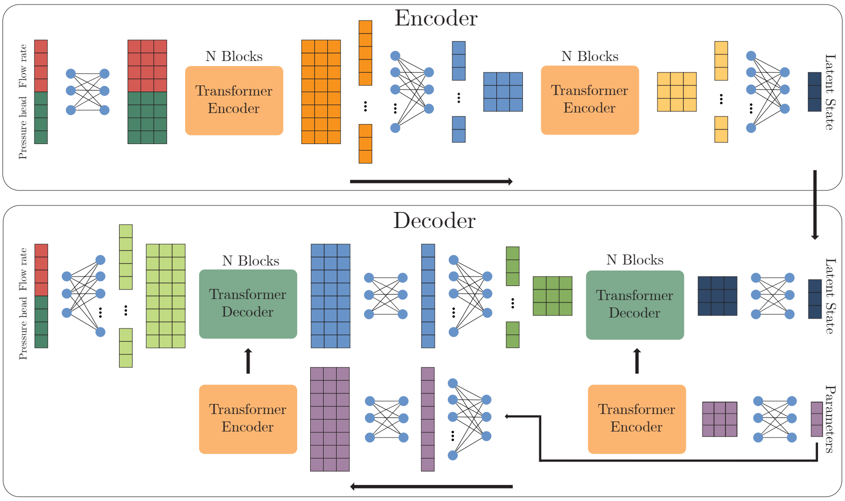


Figure 4.3: Illustration of the encoder and decoder architectures.

4.4.1 Classification Model

For comparison, we also compute the leak location by a conventional classification neural network, which is trained to classify a leak location based on sensor observations at the given time. The classification network is 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.

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 is 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 is one-hot encoded. That is, the target is a zero vector with a one at the entry of leak location index. Hence, the dataset is given by:

$$\{(H(\boldsymbol{x}_1), \boldsymbol{c}_1), \dots, (H(\boldsymbol{x}_N), \boldsymbol{c}_N)\}\,. \tag{4.27}$$

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

$$L(g) = -\sum_{i=1}^{N} \sum_{j=1}^{N_p} \boldsymbol{c}_{i,j} \log(g(H(\boldsymbol{x}_i)_j)), \tag{4.28}$$

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 is trained to output the probability of a leak being present in each possible pipe section given 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:

$$\rho_{c|\boldsymbol{y}}(c_k|\boldsymbol{y}_{0:N_t}) = \rho_c(c_k)\prod_{i=0}^{N_t}\rho_{c|\boldsymbol{y}}(c_k|\boldsymbol{y}_{0:i}) = \rho_c(c_k)\prod_{i=0}^{N_t}g(\boldsymbol{y}_i), \tag{4.29}$$

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

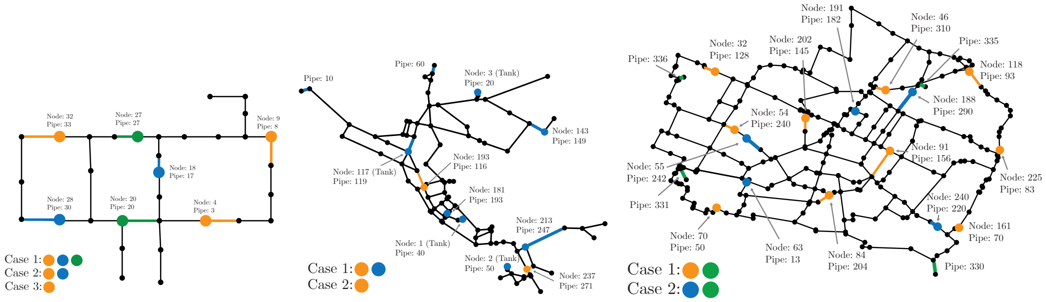
It is important to keep in mind that the classifier only approximate the leak location posterior for a single sensor configuration. This is in contrast to the proposed framework where the full state posterior is approximated together with the leak location. This also means that the classification neural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

4.4.2 Test Cases

All test cases are defined in a similar manner, however, with some variations. Figure 4.4 shows the three test WDN topologies together with the sensor locations. The first one is typically referred to as the Hanoi network; the second one is often referred to as Net3 (not to be confused the numbering of the cases in this chapter), and the third is known as Modena.

We place sensors in various nodes in the WDN. Each sensor measures the pressure head value in the node and the flow rate in a neighboring pipe section. We do not have 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 have located 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. In 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 sensors configurations. Note that since the generative model outputs the full state, consisting of pressure heads and flow rate, the different sensor configurations only change the observation operator in the online stage. That is, only H in (4.21) is varied with sensor configuration. Hence, a single training stage is performed and the resulting generative model works under multiple sensors settings.

We also add noise to the sensor readings to mimic a real-world scenario where sensors are not perfect. In all cases and for all pressure head and flow rate observations, the noise is sampled from a normal distribution at each time step and added to the observations. The noise at each sensor is independent of the other sensors. The normal distribution has mean zero and a standard deviation corresponding to a percentage of the observed value. We



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

Figure 4.4: Network topologies and the sensor locations for the three test cases.

Table 4.1: 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%

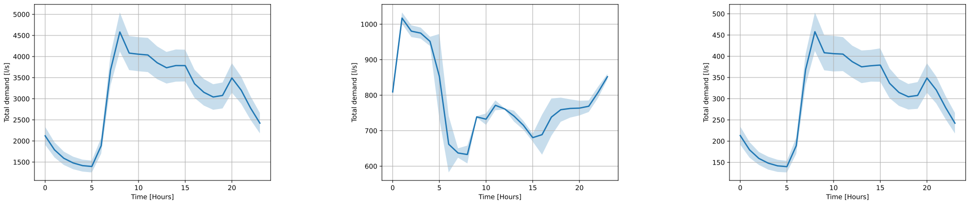
show results for various noise percentages in all test cases to analyze the performance of the algorithm in varying settings. In Table 4.1, we show the specific parameter and noise settings for the three test cases.

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

All the demands follow a temporal pattern, i.e., in each node at every time of the day the demand has a base value. In order to mimic the stochasticity of the demand, we add noise to the base values.

We use two metrics for assessing 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 is then computed by taking the average over many different solutions of the inverse problems for varying leak locations, sizes, demands, and sensor noise.

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

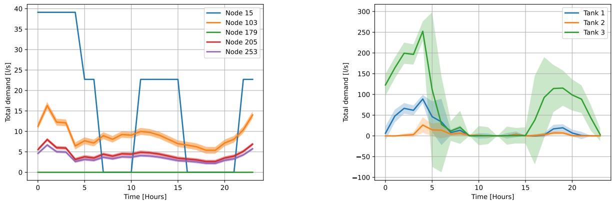


(a) Test WDN 1.

(b) Test WDN 2.

(c) Test WDN 3.

Figure 4.5: Total demand with standard deviation.



(a) Demands for 5 nodes.

(b) Demands for tanks.

Figure 4.6: Demand patterns for test WDN 2.

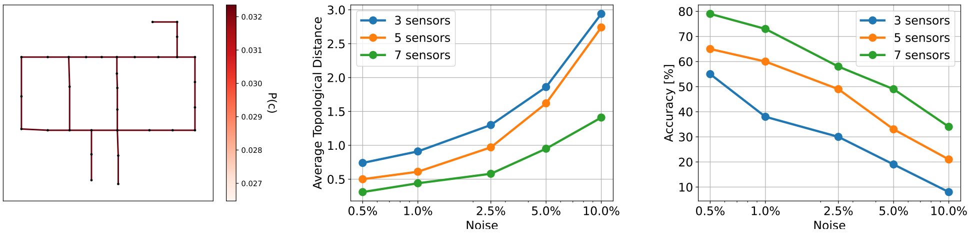
Training of the WAEs

All neural network code is based on PyTorch [115]. Training is performed in a similar manner for all three test cases. We use the Adam optimizer with a cosine warm-up learning rate scheduler with 50 warm-up steps, as described in [95]. That is, the learning rate starts as a small value, and is increased to the chosen value over 50 epochs. Thereafter, it is reduced following a cosine function over the remaining epochs. The gradient norms are clipped to 0.5 to stabilize the training. We make use of early stopping with a patience of 50 to avoid over-fitting. That is, if there are 50 consecutive epochs without improvement on the validation data, we stop the training and make use of the best performing model. In all test cases, we train on 25000 samples and use 5000 samples for validation. For all the hyperparameters, see Table 4.2.

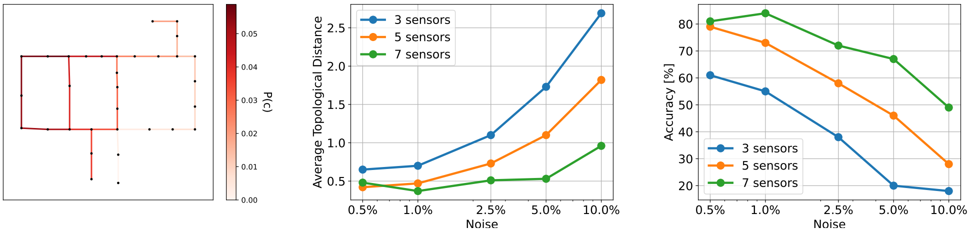
The WAEs are trained on simulated data. Training data is generated by simulating according to the settings described above. The leak locations are sampled from a multinomial distribution with equal probability assigned to each pipe section. The leak size is sampled uniformly from the interval $[0.002\text{m}^2,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

Table 4.2: Hyperparameters for the WAEs

	WDN 1	WDN 2	WDN 3
Training hyperparameters			
Batch size	256	256	256
Learning rate	$5 \cdot 10^{-5}$	$5 \cdot 10^{-5}$	$5 \cdot 10^{-5}$
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



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



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

Figure 4.7: Results for WDN1 for the given priors. First row shows results for the prior in (a) and 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.

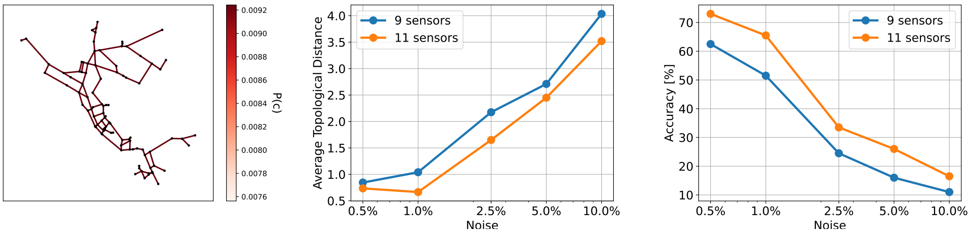
4.4.3 Test WDN1: Hanoi Network

For test WDN1, we consider three different sensor configurations (see Figure 4.4a), each with five different sensor noise levels. All nodes are given the same demand pattern, but with varying base values in each node. Noise is added to the total demand, which is then distributed to all nodes according to the relative base value. Noise is added to each node independent of the other nodes. See Figure 4.5a for the total demand time series with the standard deviation.

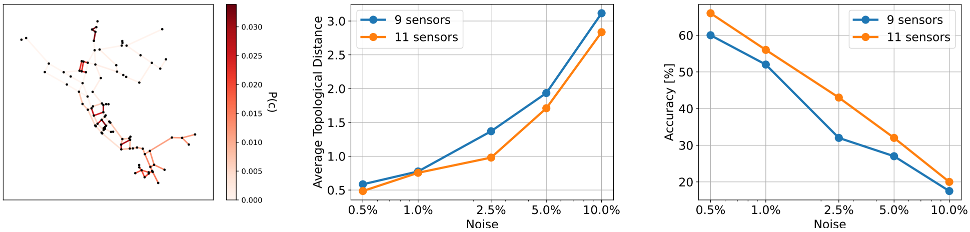
We test the framework with both a prior distribution over leak locations and without any prior knowledge. The prior distribution is shown in Figure 4.7d.

We make use of the transformer architecture here. The results are shown in Figure 4.7. The likelihood is computed with 30000 samples. We test with 50 different leak locations.

As expected, the ATD increases and accuracy decreases with increasing noise and a decreasing number of sensors. The ATD goes from approximately 0.25 to approximately 3.0 from the best to the worst case and the accuracy diminishes from approximately 79% to approximately 9% when no prior is available. When a prior is present, the ATD ranges from approximately 0.4 to approximately 2.7 and the accuracy from approximately 81% to approximately 19%, which is a drastic reduction.



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



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

Figure 4.8: Results for WDN2 for the given priors. First row shows results for the prior in (a) and 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.

4.4.4 Test WDN2

For test WDN2, we consider two different sensor configurations (see Figure 4.4b), each with five different sensor noise levels. The total demand is shown Figure 4.5b. There is a varying demand pattern for each node. Some nodes mimic households, while other nodes have no demand, and some have demands mimicking factories. Noise is added to each node independent of the other nodes. In Figure 4.6a, examples of demand patterns are presented. Furthermore, there are water tanks present, which serve as stabilizers for the WDN. Hence, their demand varies according to the total demand. In Figure 4.6b, examples of three tanks are shown.

As for test WDN1, we will work with a prior distribution over leak locations and without any prior knowledge. The prior distribution is shown in Figure 4.8d. We make use of the transformer architecture. The results are shown in Figure 4.8. The likelihood is computed with 30000 samples. We test with 200 different leak locations.

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

4.4.5 Test WDN3: Modena

For test WDN3, we again consider two different sensor configurations (see Figure 4.4c), each with five different sensor noise levels. The demand patterns are modeled in the same way as for WDN1, but with a different total demand. See Figure 4.5c for the total demand time series.

Unique for this test case, we make use of a different standard deviation for the likelihood computation than for the noise added to the observations. Specifically, use 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 is unknown.

As for test cases WDN1 and WDN2, we will work with a prior distribution over leak locations and without any prior knowledge. The prior distribution is shown in Figure 4.9d. Here, we make use of the ResNet architecture. The results are shown in Figure 4.9. The likelihood is computed with 50000 samples. We test with 250 different leak locations.

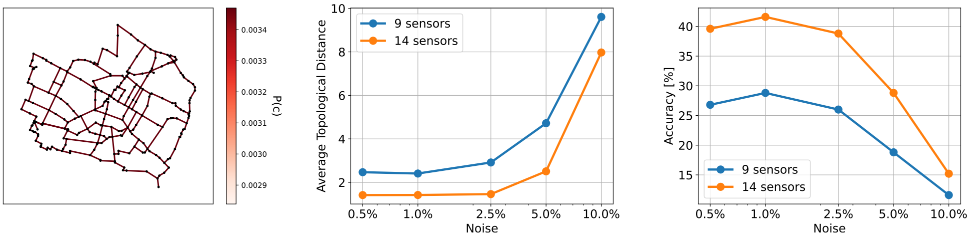
As for the other two test cases, we see better performance when we make use of prior information. Furthermore, increasing the number of sensors also gives better results.

We see a decrease in accuracy and increase in ATD when the noise level is increasing. However, in contrast to the other test cases, the performance is relative constant across noise levels until 5% and 10% where the accuracy and ATD worsen. 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 is significantly larger than the two other test cases and therefore might require additional observations when there is much 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 make a strong conclusion.

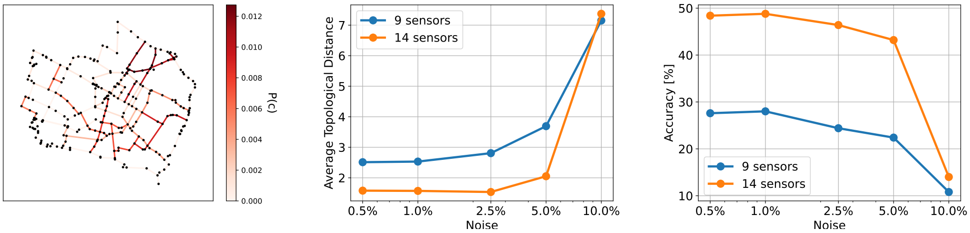
4.4.6 Comparison with Baseline

In Table 4.3, we show a comparison of the results computed with our framework and the baseline classifier. Clearly, the proposed framework outperforms the classifier for both WDN1 and WDN2 with both the ResNet and the transformer architecture. Furthermore, the transformer network gives the best results, suggesting that the transformer architecture is a suitable choice for the small to medium-sized WDNs.

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



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

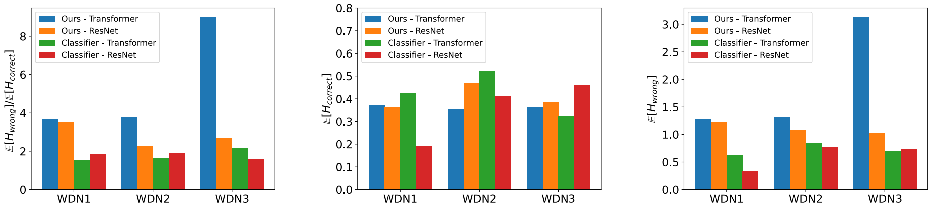


(d) Prior probability. (e) ATD. (f) Accuracy.

Figure 4.9: Results for WDN3 for the given priors. First row shows results for the prior in (a) and 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.

Table 4.3: Comparison with baseline. The ATD and accuracy are computed for all noise and sensors cases with both prior and without 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.

	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



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

Figure 4.10: 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. (b=c) shows the mean entropy of the posterior distribution for wrong predictions

Posterior Distribution Entropy

To evaluate the entropy computations, here we consider 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 $\mathbb{E}[H_{\text{wrong}}(C)]/\mathbb{E}[H_{\text{correct}}(C)]$. The ratio informs us on 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 little uncertainty for correct predictions.

In Figure 4.10, it is clear that, compared with the classifier, the new framework shows consistently higher entropy ratios. So, the framework provides more accurate uncertainty estimates compared to the classifier. Particularly, for the transformer neural network in the new approach, the entropy ratio is high, because the entropy for incorrect predictions is. Summarizing, if the ATDs and accuracy were not high, this would be reflected in the entropy. Therefore, we know a-posteriori, accurately, when predictions computed with the new framework are trustworthy. This is an important feature, that is not easily attained.

4.5 Conclusion

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 way. 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 on three test cases and showed superior performance compared to a classification approach.

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

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

We showed that one can get good quality performance by 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 beforehand. However, the fact that the framework works with various architectures means that a new state-of-the-art architectures can easily be incorporated.

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

The framework does not only provide leak location estimates, but also estimates on the uncertainty in the estimates. This is done 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 chapter thus showed the potential of using generative deep learning as a stochastic digital twin for water distribution networks, as it seamlessly integrated with Bayesian inversion schemes.

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

5

THE DEEP LATENT SPACE PARTICLE FILTER FOR REAL-TIME NONLINEAR DATA ASSIMILATION WITH UNCERTAINTY QUANTIFICATION

In Data Assimilation, observations are fused with simulations to obtain an accurate estimate of the state and parameters for a given physical system. Combining data with a model, however, while accurately estimating uncertainty, is computationally expensive and infeasible to run in real-time for complex systems. Here, we present a novel particle filter methodology, the Deep Latent Space Particle Filter or D-LSPF*, that uses neural network-based surrogate models to overcome this computational challenge. The D-LSPF enables filtering in the low-dimensional latent space obtained using Wasserstein AEs with modified vision transformer layers for dimensionality reduction and transformers for parameterized latent space time stepping. As we demonstrate on three test cases, including leak localization in multi-phase pipe flow and seabed identification for fully nonlinear water waves, the D-LSPF runs orders of magnitude faster than a high-fidelity particle filter and 3-5 times faster than alternative methods while being up to an order of magnitude more accurate. The D-LSPF thus enables real-time data assimilation with uncertainty quantification for physical systems.*

This chapter is based on the article "The Deep Latent Space Particle Filter for Real-Time Nonlinear Data Assimilation with Uncertainty Quantification", currently under review.

5.1 Introduction

Virtual representations of physical systems like digital twins have proven to be invaluable tools for monitoring, predicting, and optimizing the performance of intricate systems, ranging from industrial machinery to biological processes [127]. The efficacy of digital twins relies however on the accurate assimilation of real-time data into the simulations to ensure accurate calibration and state estimation in situations where the state and its dynamics are not known. Importantly, to confidently rely on the information provided, data assimilation should be accompanied by a quantification of the associated uncertainty originating from both measurements and model errors [6, 79].

Performing data assimilation with uncertainty quantification in real time for high-dimensional systems, such as discretized partial differential equations (PDEs), is computationally infeasible due to the need to compute large ensembles of solutions. Approaches such as (ensemble) Kalman filtering [73] aim to overcome this computational bottleneck by assuming Gaussian distributed prior, likelihood, and posterior distributions [33], which is restrictive in practical situations where these assumptions do not hold [4, 153]. Particle filters, on the other hand, can approximate any distribution provided there is a sufficient number of particles in the ensemble [36, 82]. The necessary ensemble size for particle filters however often makes it infeasible to run in real-time for complex, high-dimensional systems. Therefore, there is a need for methods to speed up the computations of ensembles.

To speed up the computation of ensembles, surrogate models are used to approximate the forward problem by replacing the full order model with a computationally cheaper alternative. Surrogate models based on proper orthogonal decomposition and dynamic mode decomposition have been developed with reasonable success [4]. However, for nonlinear, hyperbolic, and/or discontinuous problems, advanced surrogates are necessary to achieve the desired speed-up [87]. Therefore, utilizing deep learning approaches has received increased attention, as significant speed-ups are possible without sacrificing essential accuracy [20, 43, 87, 105].

Deep learning-based surrogate models can be designed in various ways. One very common approach is to make use of latent space representations. High-dimensional states are mapped onto a low-dimensional latent space such that the computations are performed cheaply in this latent space. The autoencoder (AE) [10] is the principal enabling architecture for this. In the AE, an encoder network reduces the original data into a latent representation and a decoder subsequently reconstructs the original data from the latent representation. Since [10], many extensions and improvements have been developed, such as adding probabilistic priors to the latent space [81, 151]. In the context of surrogate modeling for physical systems, the focus has been on ensuring that AEs learn latent representations that are suitable for downstream tasks, such as time stepping, via various regularization techniques [20, 43, 158].

The success of such regularization is important when embedding the surrogate model into a data assimilation framework.

Utilizing neural network surrogate models for speeding up data assimilation has been explored in various studies (see [23] for a review). In [107, 118, 139, 162], generative deep learning has been used for high-dimensional state- and parameter estimation. However, none of these approaches performed sequential assimilation of the data. In [18, 22], deep learning was used for model discovery. In such applications, the model is a-priori unknown and is learned from observations, typically requiring a huge number of observations in space and time to recover the complete, unknown state; to this point it is not clear how these methods perform with real-time data assimilation. Similarly, in [98, 104] a particle filter approach to data assimilation has been used to formulate a variational objective for training a latent space model. Hence, the latent model is trained while data is arriving, which severely limits real-time assimilation for high-dimensional, nonlinear problems with limited observations. In [142], a GAN set-up, combined with proper orthogonal decomposition, was used for sequential data assimilation with an approach similar to randomized maximum likelihood. However, unlike particle filters, convergence of such methods is not ensured for nonlinear cases [33].

Yet, while many deep learning-based surrogate models have been used to speed up data assimilation, there is limited work on such approaches using particle filters [48, 167]. In [48] a back-constrained Gaussian process latent variable model is used to parameterize both the dimensionality reduction and latent space dynamics. In [167], a particle filter using a latent space formulation was presented. The approach evaluated the likelihood by iterative closest point registration fitness scores and the latent time stepping was mainly linear. It is unknown how these methods would perform with highly nonlinear PDE-based problems and only a few observations in space and time.

Here, we propose a deep learning framework for performing particle filtering in real-time using latent-space representations: the Deep Latent Space Particle Filter, or *D-LSPF*, targeting complex nonlinear data assimilation problems modeled by PDEs. For this, we develop a novel extension to the vision transformer layer for dimensionality reduction of the high-dimensional state in an AE setup. Then, a transformer-based network is used for parameterized time stepping, which enables filtering in the latent space as well parameter estimation. To ensure that the latent space has the appropriate desirable properties, we combine several regularization techniques such as divergence and consistency regularization [151, 158]. We showcase the D-LSPF on three distinct test problems with varying characteristics, such as discontinuity, few observations in space and time, parameter estimation, and highly oscillatory real-world data. In all cases, the D-LSPF demonstrates significant speed-ups compared to alternative methods without sacrificing accuracy. This promises to enable true or near real-time data assimilation for new, more complex classes of problems, with direct applications in engineering such as leak localization as well as seabed and wave height estimation.

The chapter is organized as follows. In Section 5.2 we describe the problem setting. This consists of a brief description of the Bayesian filtering problem and an overview of the particle filter. In Section 5.3 we present the D-LSPF. Firstly, we outline the latent filtering problem, followed by a description of the latent space regularized autoencoder using the novel transformer-based dimensionality reduction layers, and parameterized time stepping. Lastly, in Section 5.4 we showcase the performance of the D-LSPF on three test cases, namely the viscous Burgers equation, harmonic wave generation over a submerged bar, and leak localization for multi-phase flow in a pipe. The D-LSPF is compared with a high-fidelity particle filter and the Reduced-Order Autodifferentiable Ensemble Kalman Filter (ROAD-EnKF) [22].

5.2 Problem Setting

We consider problems that are modeled by time-dependent PDEs. Such problems consist of a state, typically made up by several quantities such as velocity and pressure, and parameters, source terms, boundary conditions, and initial conditions. Since the model won't be perfect and true values of the parameters are rarely known, the problem needs to be accompanied by observations coming from a series of sensors. However, sensors deliver noisy data and are often scarcely placed in the domain of interest, so that the data needs to be assimilated into the model to yield an accurate estimate of the state and parameters.

Consider a time and spatially discretized PDE, with accompanying observations:

$$\begin{aligned} \boldsymbol{q}_n &= F(\boldsymbol{q}_{n-1};\boldsymbol{m}_{n-1}) + \boldsymbol{\xi}_{n-1}, & \boldsymbol{\xi}_{n-1} &\sim P_\xi(\boldsymbol{\xi}_{n-1}), \\ \boldsymbol{m}_n &= G(\boldsymbol{m}_{n-1}) + \boldsymbol{\zeta}_{n-1}, & \boldsymbol{\zeta}_{n-1} &\sim P_\zeta(\boldsymbol{\zeta}_{n-1}), \\ \boldsymbol{y}_n &= h(\boldsymbol{q}_n) + \boldsymbol{\eta}_{n-1}, & \boldsymbol{\eta}_{n-1} &\sim P_\eta(\boldsymbol{\eta}_{n-1}), \end{aligned} \tag{5.1}$$

where F is a (nonlinear) operator advancing the state, G is an operator advancing the parameters, $\boldsymbol{q}_n(\boldsymbol{m}_n) \in \mathbb{R}^{N_x}$ is a parameter-dependent state at time step n, $\boldsymbol{m}_n \in \mathbb{R}^{N_m}$ are the parameters, $\boldsymbol{y}_n \in \mathbb{R}^{N_o}$ is an observation vector at time step n, $h : \mathbb{R}^{N_x} \to \mathbb{R}^{N_o}$ is the observation operator, $\boldsymbol{\xi}_n$ is the model error, $\boldsymbol{\zeta}_n$ is the parameter error, and $\boldsymbol{\eta}_n$ is the observation noise. Note that when parameters are constant in time, we use $G(\boldsymbol{m}_n) = \boldsymbol{m}_n$. To simplify notation, we introduce the combined state-parameter variable, $\boldsymbol{u}_n = (\boldsymbol{q}_n, \boldsymbol{m}_n)$. We will refer to $\boldsymbol{u}_n$ as the augmented state and introduce the notation for time series, $\boldsymbol{u}_{0:n} = (\boldsymbol{u}_0, \dots, \boldsymbol{u}_n)$. We further assume that the probability density functions exist and will therefore continue with the derivations using the densities.

5.2.1 Data Assimilation

The goal is to compute the posterior density of the augmented state given observations, i.e., $\rho(\boldsymbol{u}_{0:N_t}|\boldsymbol{y}_{0:N_t})$. Based on the formulation as a filtering problem, it can be solved sequentially as observations become available. This leads to the filtering distribution, $\rho(\boldsymbol{u}_{n+1}|\boldsymbol{y}_{0:n+1})$. Bayes' theorem then gives us:

$$\rho(\boldsymbol{u}_n|\boldsymbol{y}_{0:n}) = \frac{\rho(\boldsymbol{y}_n|\boldsymbol{u}_n)\rho(\boldsymbol{u}_n|\boldsymbol{y}_{0:n-1})}{\rho(\boldsymbol{y}_n|\boldsymbol{y}_{0:n-1})}. \tag{5.2}$$

The problem at hand is to compute Eq. (5.2) as observations become available. The posterior density is not analytically tractable, so we must resort to numerical approximations.

5.2.2 Particle Filter

We will make use of the particle filter, also referred to as sequential Monte Carlo method, where one aims to sample from the posterior instead of computing it [128]. The approximation is performed by creating an ensemble of augmented states (particles) and advancing each augmented state in time using the prior distribution. The posterior is then approximated by the empirical density, made up of N particles,

$$\rho^N(\boldsymbol{u}_n|\boldsymbol{y}_{0:n}) = \sum_{i=1}^{N} w_n^i \delta\left(\boldsymbol{u}_n - \boldsymbol{u}_n^i\right), \tag{5.3}$$

where $\boldsymbol{u}_{n+1}^i$ represents particle i at time step $n+1$ and w_{n+1}^i is its corresponding weight. δ is the Dirac delta function, which gives $w_n^i\delta(x) = w_n^i$ for $x = 0$ and zero otherwise. $\rho^N(\boldsymbol{u}_n|\boldsymbol{y}_{0:n})$ can therefore be considered a discrete approximation of the true posterior. The computation of the weights is done by the specific choice of particle filter. A common choice is the bootstrap filter which makes use of importance sampling, originally described in [82].

The bootstrap filter assimilates data by advancing each particle using the prior distribution and assigning a weight to each. The weights are the normalized likelihoods, computed by evaluating the observation noise density at the residual between the observations and the particles. The weights are then normalized and used to resample the particles using a multinomial distribution with replacement. Note, however, that resampling when a new observation becomes available can lead to poor variablity in the ensemble. Therefore, resampling only occurs when the effective sampling size, $ESS = 1/\sum_{i=0}^{N}(w_n^i)^2$, is below a certain threshold, λ_{ESS}, typically chosen as $N/2$. See Algorithm 8 for an outline of the bootstrap particle filter.

We will refer to the particle filter solution of Eq. (5.1) as the high-fidelity (HF) problem as it is the most accurate solution available.

Algorithm 8: Bootstrap Particle Filter

```
Input: Time stepping operator = $F$, ensemble size = $N$
Compute resample threshold, $\lambda_{ESS} = N/2$;
Initialize $N$ initial conditions, $\{\boldsymbol{u}_0^i\}_{i=1}^N$;
Initialize weights, $\{w_0^i\}_{i=1}^N = \{1/N\}_{i=1}^N$ ;
while new data, $\boldsymbol{y}_n$, arrives do
    Time step states, $\{\boldsymbol{u}_n^i\}_{i=1}^N = \{F(\boldsymbol{z}_{n-1}^i) + \xi_{n-1}^i\}_{i=1}^N$ ;
    Compute weights, $\{\tilde{w}_n^i\}_{i=1}^N = \{\rho_{\eta_n}(\boldsymbol{y}_n - h(\boldsymbol{u}_n^i))w_{n-1}^i\}_{i=1}^N$;
    Normalize weights = $\{w_0^i\}_{i=1}^N = \left\{\tilde{w}_0^i / \sum_{j=0}^N \tilde{w}_n^j\right\}_{i=1}^N$ ;
    if $1/\sum_{i=0}^N (w_n^i)^2 < \lambda_{ESS}$ then
        Resample states, $\{\boldsymbol{u}_n^i\}_{i=1}^N$, with computed weights
    end if
end while
Output: Assimilated ensemble, $\{\boldsymbol{u}_{0:n}^i\}_{i=1}^N$
```

The prior distribution is sampled by time stepping in the underlying discretized PDE and adding random noise. For high-dimensional problems, it is generally not feasible to run a particle filter in real-time as several thousands of particles are needed to accurately approximate the posterior, since the true model error is typically unknown. Therefore, reduced order models are often employed to speed up the computations at the cost of accuracy and training time.

5.3 Deep Latent Space Particle Filter

Here, we present the proposed methodology for real-time data assimilation with particle filters – the D-LSPF.

At its heart, we represent the high-fidelity state in a more compact and cheaper to compute latent space and perform the data assimilation in the latent space. We then compute a posterior distribution over the latent state after which we transform back to the high-fidelity space to obtain the high-fidelity posterior. For this, we employ an autoencoder (AE) to reduce to the latent state, which we combine with a latent time stepping model to advance the latent state. AEs consist of two neural networks: An encoder, $\phi_{\text{enc}} : \boldsymbol{q} \mapsto \boldsymbol{z}$, that reduces the dimension of the data to a latent state, and a decoder, $\phi_{\text{dec}} : (\boldsymbol{z}, \boldsymbol{m}) \mapsto \tilde{\boldsymbol{q}}$, that reconstructs the data. The AE is trained by minimizing the loss (MSE) between the input and the reconstruction – for the parameter dependent cases we include the parameters in the decoder, which increases the reconstruction accuracy [101]. The encoder and the decoder are then used to represent Eq. (5.1) in the latent space:

$$\begin{aligned}
\boldsymbol{z}_n &= f(\boldsymbol{z}_{n-1}; \boldsymbol{m}_{n-1}) + \hat{\boldsymbol{\xi}}_{n-1}, & \hat{\boldsymbol{\xi}}_n &\sim P_{\hat{\xi}}(\hat{\boldsymbol{\xi}}_n), \\
\boldsymbol{m}_n &= G(\boldsymbol{m}_{n-1}) + \boldsymbol{\zeta}_{n-1}, & \boldsymbol{\zeta}_n &\sim P_{\zeta}(\boldsymbol{\zeta}_n), \\
\boldsymbol{q}_n &= \phi_{\text{dec}}(\boldsymbol{z}_n, \boldsymbol{m}_n), & & \\
\boldsymbol{y}_n &= h(\boldsymbol{q}_n) + \boldsymbol{\eta}_n, & \boldsymbol{\eta}_n &\sim P_{\eta}(\boldsymbol{\eta}_n),
\end{aligned} \tag{5.4}$$

Eq. (5.4) differs from Eq. (5.1) in three ways:

- $\boldsymbol{q}_n$ is replaced by $\boldsymbol{z}_n$ – we advance the latent state instead of the high-fidelity state in time;
- $\boldsymbol{\xi}_n$ is replaced by $\hat{\boldsymbol{\xi}}_n$ – the latent time stepping model introduces a model error that differs from the high-fidelity model error;
- $\boldsymbol{q}_{n+1} = \phi_{\text{dec}}(\boldsymbol{z}_{n+1})$ is added – we need to decode the latent state to get synthetic observations in the high-fidelity space.

With the augmented latent state, $\boldsymbol{a}_n = (\boldsymbol{z}_n, \boldsymbol{m}_n)$, the high-fidelity posterior density is replaced by the latent posterior density, $\rho(\boldsymbol{a}_{0:N_t}|\boldsymbol{y}_{0:N_t})$. Formulating the problem as a filtering problem, the sequentially defined posterior density is given by:

$$\rho(\boldsymbol{a}_n|\boldsymbol{y}_{0:n}) = \frac{\rho(\boldsymbol{y}_n|\boldsymbol{a}_n)\rho(\boldsymbol{a}_n|\boldsymbol{y}_{0:n-1})}{\rho(\boldsymbol{y}_n|\boldsymbol{y}_{0:n-1})}. \tag{5.5}$$

The latent prior density is then computed by:

$$\rho(\boldsymbol{a}_n|\boldsymbol{y}_{0:n-1}) = \int \rho(\boldsymbol{a}_n|\boldsymbol{a}_{n-1})\rho(\boldsymbol{a}_{n-1}|\boldsymbol{y}_{0:n-1})\mathrm{d}\boldsymbol{a}_{n-1}, \tag{5.6}$$

which is an integral of much lower dimension than the high-fidelity equivalent. The latent likelihood is computed by:

$$\rho(\boldsymbol{y}_n|\boldsymbol{a}_n) = \rho_{\eta_n}\left(\boldsymbol{y}_n - h(\phi_{\text{dec}}(\boldsymbol{a}_n))\right), \tag{5.7}$$

which is faster to evaluate than the high-fidelity equivalent, since $\phi_{\text{dec}}(\boldsymbol{a}_n))$ is fast to compute once f and ϕ_{dec} have been trained. Eqs. (5.5), (5.6), and (5.7) are approximated using the particle filter, as for the high-fidelity equations. The computationally expensive part of the particle filter, namely the time stepping, is performed efficiently in the latent space.

Here, we make use of the bootstrap particle filter; other types of particle filter algorithms could possibly be used instead. An outline of the D-LSPF algorithm is shown in Algorithm 9 and in Figure 5.1.

Algorithm 9: D-LSPF (based on the Boostrap particle filter)

```
Input: Autoencoder=$(\phi_{\text{enc}}, \phi_{\text{dec}})$, time stepping network=$f$, ensemble size=$N$
Compute resample threshold, $\lambda_{ESS} = N/2$ ;
Encode $N$ initial conditions, $\{\boldsymbol{z}_0^i\}_{i=1}^N = \{\phi_{\text{enc}}(\boldsymbol{u}_0^i)\}_{i=1}^N$ ;
Initialize weights, $\{w_0^i\}_{i=1}^N = \{1/N\}_{i=1}^N$ ;
while new data, $\boldsymbol{y}_n$, arrives do
    Time-step latent states, $\{\boldsymbol{z}_n^i\}_{i=1}^N = \{f(\boldsymbol{z}_{n-1}^i) + \hat{\xi}_{n-1}^i\}_{i=1}^N$ ;
    Decode latent states, $\{\boldsymbol{u}_n^i\}_{i=1}^N = \{\phi_{\text{dec}}(\boldsymbol{u}_n^i)\}_{i=1}^N$ ;
    Compute weights, $\{\tilde{w}_n^i\}_{i=1}^N = \{\rho_{\eta_n}(\boldsymbol{y}_n - h(\boldsymbol{u}_n^i))w_{n-1}^i\}_{i=1}^N$ ;
    Normalize weights = $\{w_n^i\}_{i=1}^N = \left\{\tilde{w}_n^i / \sum_{j=0}^N \tilde{w}_n^j\right\}_{i=1}^N$ ;
    if $1/\sum_{i=0}^N (\tilde{w}_n^i)^2 < \lambda_{ESS}$ then
        Resample latent states, $\{\boldsymbol{z}_n^i\}_{i=1}^N$ with computed weights
    end if
end while
Output: Assimilated latent ensemble: $\{\boldsymbol{z}_{0:n}^i\}_{i=1}^N$, decoded ensemble:
        $\{\boldsymbol{u}_{0:n}^i\}_{i=1}^N = \{\phi_{\text{dec}}(\boldsymbol{z}_{0:n}^i)\}_{i=1}^N$
```

Note that the speed-up in running the particle filter in the latent space comes at a cost of a training stage and the cost of encoding and decoding (which is fast due to parallel computations on a GPU). This is, however, not a significant drawback as the AE and time stepping network can be used numerous times after training.

5.3.1 Latent Space Regularized Autoencoder

For the D-LSPF to function efficiently, the latent space needs to satisfy certain properties. Firstly, the latent space must be smooth enough: to ensure that the latent space perturbations are meaningful, two states that are close to each other in the high-fidelity space must also be close in the latent space. With smoothness thus defined, we enforce this property using a prior distribution on the learned latent space in the form of a Wasserstein autoencoder (WAE) [151] using the maximum mean discrepancy (MMD) loss term – the variational autoencoder (VAE) [81] could serve the same purpose; however, the VAE tends to also smoothen the reconstructions which is undesirable in our settings. Secondly, the autoencoder should ensure that latent space trajectories are simple and easy to learn by a time stepping neural network. We achieve this by adding a consistency regularization term, as in [158], which ensures that the time evolution of the latent state can be modeled by means of an ODE system and thus promotes differentiability, and thereby smoothness, of the time evolution map.

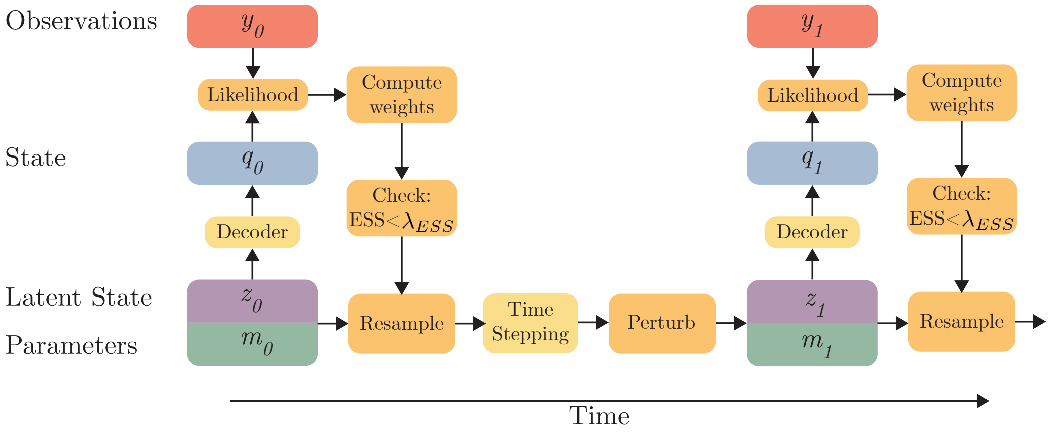


Figure 5.1: Schematic of the D-LSPF.

In summary, for a given training set, $\{\boldsymbol{q}_i\}_{i=0}^{N_{\text{train}}}$, the complete loss function is given by:

$$\begin{aligned} L_{\text{WAE}}(\phi_{\text{enc}}, \phi_{\text{dec}}) &= \frac{1}{N} \sum_{i=1}^{N} \underbrace{\left(\boldsymbol{q}_i - \phi_{\text{dec}}(\phi_{\text{enc}}(\boldsymbol{q}_i))\right)^2}_{\text{Reconstruction}} \\ &+ \underbrace{\alpha R(\phi_{\text{enc}}, \phi_{\text{dec}})}_{\text{Weight regularization}} + \underbrace{\beta \text{MMD}\left(\phi_{\text{enc}}\right)}_{\text{Divergence}} + \underbrace{\lambda C(\phi_{\text{enc}}, \phi_{\text{dec}})}_{\text{Consistency}} . \end{aligned} \tag{5.8}$$

5.3.2 Transformer-Based Dimensionality Reduction

In our approach, the loss function ensures that the autoencoder, and thereby the latent space, has certain desirable properties. To ensure that the autoencoder can learn a low-dimensional representation and reconstruct it accurately, the architecture also has to be able to handle a multitude of possible high-fidelity states.

The arguably most common layer for AEs are convolutional and pooling layers [43, 105, 158]. Convolutional layers however tend to have inherent inductive biases and struggle with discontinuous signals, resulting in spurious oscillations. Transformers, originally developed for text processing, have proven effective for image processing in the form of vision transformers (ViT) [29]. These transformers divide images into patches and apply the attention mechanism between each set of patches. Yet, since there is no natural way of reducing or expanding the dimensionality of the data, the ViT has been used to dimensionality reduction tasks only to a limited degree [64, 90, 111, 126]. Importantly, current ViTs have not been integrated with increasing numbers of channels in convolutional layers to represent increasingly complicated features.

In this section, we extend the vision transformer layer to combine the advantages of

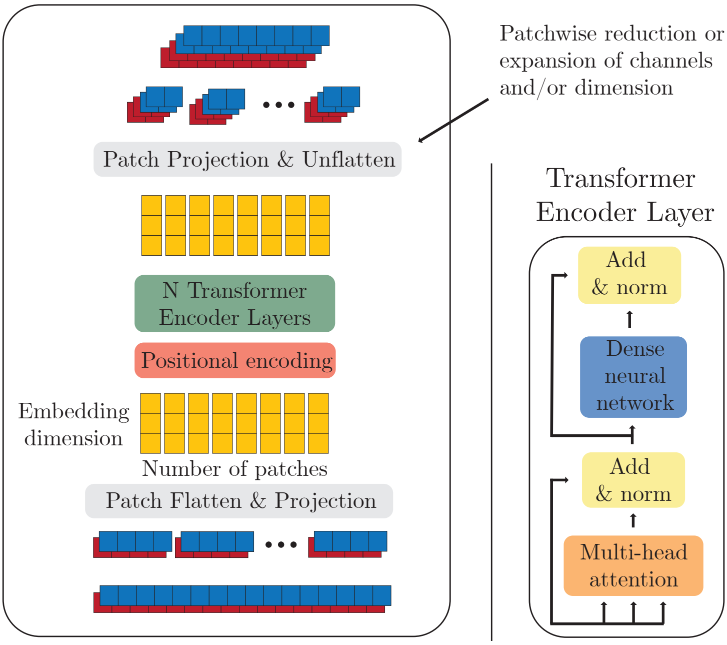

Patchwise reduction or expansion of channels and/or dimension
Patch Projection & Unflatten
N Transformer Encoder Layers
Positional encoding
Embedding dimension
Number of patches
Patch Flatten & Projection
Transformer Encoder Layer
Add & norm
Dense neural network
Add & norm
Multi-head attention

Figure 5.2: Visualization of the ViT dimensionality reduction/expansion layer.

convolutional layers (i.e., dimensionality reduction/expansion and increasing/decreasing number of channels) and ViTs (global information, patch processing).

In the proposed layer, the expansion/reduction of channels and dimensions is done on each individual patch. It can be interpreted as a type of domain decomposition, where the reduction/expansion is performed on each subdomain and the communication between subdomains is handled through the attention mechanism. Formally, let the superscript, l, denote the l'th layer. We divide an input $x^l \in \mathbb{R}^{N_c^l \times N_x^l}$ (N_c^l channels and a spatial dimension of size N_x^l), into p patches, $x_1^l, x_2^l, \dots, x_p^l$ of size N_p^l. That is, $x_i^l \in \mathbb{R}^{N_c^l \times N_p^l}$, for all i. Then, each patch is flattened and projected onto an N_e^l-dimensional embedding space, $e = (e_1, e_2, \dots, e_p) \in \mathbb{R}^{N_e \times N_p}$. Positional encodings are then added after which the embeddings are passed through a standard transformer encoder layer. Each embedded vector, e_i, is projected onto a new dimension of size $N_c^{l+1} N_p^{l+1}$, unflattened, $x_i^{l+1} \in \mathbb{R}^{N_c^{l+1} \times N_p^{l+1}}$ and recombined, $x^{l+1} \in \mathbb{R}^{N_c^{l+1} \times N_x^{l+1}}$. The process is visualized in Figure 5.2.

5.3.3 Time Stepping

Once the AE is trained, we can transform high-fidelity trajectories into latent trajectories. To compute the latent trajectories, we next need to perform time stepping in the latent space. For this, we make use of transformers [156] as they are well suited for modeling physical systems, being able to mimic the structure of multistep time-marching methods [43].

Time stepping in the latent space is done by means of a map, f, that advances a latent state in time. Adopting the concept of multistep time integrators, we use several previous time steps to predict the next latent state:

$$\boldsymbol{z}_{n+1} = f(\boldsymbol{z}_{n-k:n}; \boldsymbol{m}_n), \tag{5.9}$$

where k is referred to as the memory. For multiple time steps, we apply the transformer model recursively. Training is done by minimizing the loss function:

$$L(f) = \sum_{n=k}^{N_t - s} \sum_{i=1}^{s} ||f^i(\boldsymbol{z}_{n-k:n}; \boldsymbol{m}_n) - \boldsymbol{z}_{n+1:n+1+i}||_2^2 + \alpha R(f).$$

Here, R is a regularization, f^i means applying f i times, recursively, on the output, and s is the output sequence length. After trajectories are computed in the latent space, high-fidelity trajectories are recovered through the decoder.

Parameterized Time Stepping

In the high-fidelity space, dynamics are not only dependent on the previous state but also on a set of parameters, and the same applies to the latent dynamics. Including the parameters of interest in the latent space time stepping model can be done in several ways, depending on the specific choice of neural network architecture. We adopt the approach presented in [58], where the parameters are encoded and added to the sequence of states as the first entry:

$$\begin{aligned} &\{g(\boldsymbol{m}_n), \boldsymbol{z}_{n-k}, \boldsymbol{z}_{n-k+1}, \dots, \boldsymbol{z}_n\} \rightarrow \\ &f\left(\{g(\boldsymbol{m}_n), \boldsymbol{z}_{n-k}, \boldsymbol{z}_{n-k+1}, \dots, \boldsymbol{z}_n\}\right) = \boldsymbol{z}_{n+1}, \end{aligned} \tag{5.10}$$

where g is a parameter encoder that lifts a vector of parameters to the same dimension as the latent state. This efficiently allows attention to be computed between the parameters and the sequence of states. Figure 5.3 visualizes the time stepping transformer model.

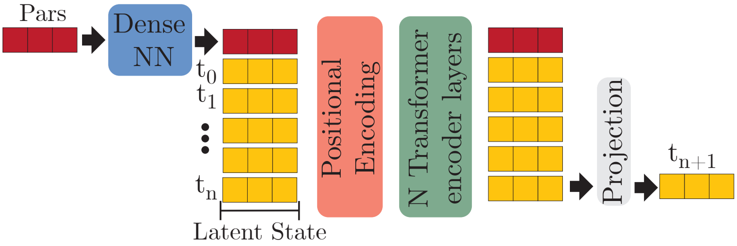


Figure 5.3: Illustration of the transformer model for parameterized time stepping.

5.4 Results

We demonstrate the potential and strength of the D-LSPF for a variety of numerical experiments. The first test case serves as a simple benchmark problem. The second test case uses real-world data from an experimental setting, and shows that the D-LSPF can be applied to real-world situations even when trained on simulation data. The last test case is a realistic engineering setting and is used as an ablation study to emphasize the performance of the architectural choices. An overview of the test cases can be found in Table 5.1. For all neural networks, hyperparameter tuning was performed to find the optimal settings. For the alternative methods we compare with, we adopt hyperparameters as chosen in the respective papers when applicable, and performed hyperparameter tuning when not.

Table 5.1: Overview of test cases.

	Burgers	Multi-phase pipeflow	Waves over submerged bar
Num Train samples	1024	5000	210
Num Test samples	20	8	1
Parametric	No	Yes	Yes
Simulated observations	Yes	Yes	No
Noise variance	0.1	0.003	-
Num Sensors	8	9	8
Num Time steps between obs	30	400	[25, 75]
State DOFs	256	1536	1024
Num States	1	3	2
Num States observed	1	1	1

All neural networks were implemented using PyTorch [116]. The modified ViT layers are implemented by modifying the code from `https://github.com/lucidrains/vit-pytorch`. Training and testing were performed using an Nvidia RTX 3090 GPU and 32 core AMD Ryzen 9 3950X CPU.

All models are trained with the Adam optimizer [80] and a warm-up cosine annealing learning rate scheduler. Gradient clipping was applied when training the transformers. The states and parameters were transformed to be between 0 and 1 before being passed to the autoencoder. The time stepping networks are trained without teacher forcing, and with a limited unrolling. The number of time steps to unroll was treated as a hyperparemeter.

The code for setting up and training the neural networks, including hyperparameter settings, for the test cases can be found in the GitHub repository `https://github.com/nmucke/latent-time-stepping`. The code for simulating training data for Burgers equations and the multi-phase leak location problem can be found in the same repository. The code for simulating the training data for the harmonic wave generation over a submerged bar test case can be made available upon request and in consultation with Associate professor Allan Peter Engsig-Karup. The code for the particle filter implementations as well as the test data can be found in the GitHub repository `https://github.com/nmucke/data-assimilation`.

5.4.1 Viscous Burgers Equation

The first test case is the viscous Burgers equation:

$$\begin{aligned} &\partial_t q(x,t) = \nu \partial_{xx} q(x,t) - q(x,t)\partial_x q(x,t), \\ &q(0,t) = q(L,0) = 0, \\ &q(x,0) = Q \sin\left(\frac{2\pi x}{L}\right), \end{aligned} \tag{5.11}$$

with $x \in [0, L]$, $L = 2$, $\nu = 1/150$, and $Q \sim \mathrm{U}[0.5, 1.5]$. We only perform state estimation so neither the AE nor the time stepping NN receives any parameters as input. The observations used for the data assimilation are simulated, as well as the training data. We add normally distributed noise with a standard deviation of 0.1. Eq. (5.11) is discretized using a second-order finite difference scheme in space and a Runge-Kutta 45 method in time. We consider $t \in [0, 0.3]$ with a step size of 0.001, resulting in 300 time steps. For the data assimilation, we test on a case where the state is observed at 8 spatial locations, $x = (0.0, 0.286, 0.571, 0.857, 1.143, 1.429, 1.714, 2.0)$, $N_y = 8$, at every 10th time step. The latent dimension in the D-LSPF is chosen to be 16.

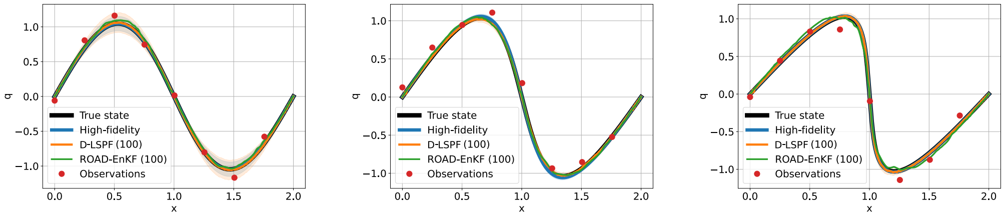
We compare the D-LSPF with 100 and 1000 particles with the Reduced-Order Autodifferentiable Ensemble Kalman Filter (ROAD-EnKF) [22] with a latent dimension of 40. The ROAD-EnKF is trained on the same training data as the D-LSPF with full access to the entire states in space and time. All methods are evaluated on 20 different simulated solutions, with $Q \sim \mathrm{U}[0.5, 1.5]$. We compare the performance by computing the Root-Mean-Square Error (RMSE) and the averaged RMSE of the 2nd, 3rd, and 4th moment of the state ensemble, referred to as the Average Moment RMSE (AMRMSE). The AMRMSE measures how accurately the distributional information of the posterior is approximated and therefore how accurately uncertainty is quantified. Moreover, we also present the negative log-likelihood (NLL) with respect to the high-fidelity posterior.

In Table 5.2, the results for the test case are shown. The D-LSPF shows superior performance with respect to AMRMSE and NLL by one order of magnitude, suggesting that the D-LSPF quantifies the uncertainty in a more accurate way compared to ROAD-EnKF for this case. For the mean state estimation, the D-LSPF also performs 3.75 times better. Regarding timing, the D-LSPF with 100 particles and the ROAD-EnKF are comparable using GPUs, while the ROAD-EnKF is slightly faster using a CPU.

Lastly, in Figure 5.4, we show state estimation results at three different time points. It

Table 5.2: The viscous Burgers equations. In parenthesis is the number of particles. For the high-fidelity (HF) particle filter, 30 CPUs were used in parallel. For the rest of the methods a single CPU or GPU was used. The AMRMSE is computed by comparing the surrogate model ensembles with the HF particle filter solution.

	RMSE ↓	AMRMSE ↓	NLL ↓	GPU ↓	CPU ↓
HF(1000)	$9.0 \cdot 10^{-3}$	-		-	22.7s
D-LSPF(100)	$\mathbf{1.6 \cdot 10^{-2}}$	$1.3 \cdot 10^{-4}$	**0.38**	**0.6s**	1.5s
D-LSPF(1000)	$\mathbf{1.6 \cdot 10^{-2}}$	$\mathbf{1.1 \cdot 10^{-4}}$	0.42	1.2s	13.4s
ROAD-EnKF	$6.0 \cdot 10^{-2}$	$1.2 \cdot 10^{-3}$	4.73	**0.6s**	**0.8s**



(a) $t = 0.03$ (b) $t = 0.15$ (c) $t = 0.29$

Figure 5.4: State estimation for the viscous Burgers equations with observations every 10 time step using the high-fidelity particle filter, the D-LSPF and the ROAD-EnKF. The high-fidelity particle filter was run with 1000 particles and the D-LSPF and ROAD-EnKF were run with 100 particles. we see the state estimation for the using the D-LSPF, ROAD-EnKF, and the high-fidelity model for case 2. It is clear that all methods approximates the state well. However, the ROAD-EnKF is visibly worse at the end.

is clear that the uncertainty bands shrink as time passes and more observations become available as expected. The ROAD-EnKF state estimation is visually slightly worse than the D-LSPF and the high-fidelity particle filter.

5.4.2 Harmonic wave generation over a submerged bar

In this test case, the data comes from a real-world experiment [11]. The setting is a 25m long and 0.4m tall wave tank, with waves being generated from the left side, traveling to the right. At the seabed of the tank, a 0.3m tall submerged bar is placed, see Figure 5.5. Eight sensors measure the surface height of the water at $x = (4, 10.5, 13.5, 14.5, 15.7, 17.3, 19.0, 21.0)$. For the state and parameter estimation, we aim to reconstruct the surface elevation, the velocity potential, and the height of the submerged bar. A similar study was conducted in [12], where the uncertainty of the water wave height was quantified given random perturbations on the seabed; in [12] however only uncertainty of the forward problem was considered, whereas we solve the inverse problem with uncertainty quantification, given the observations. For

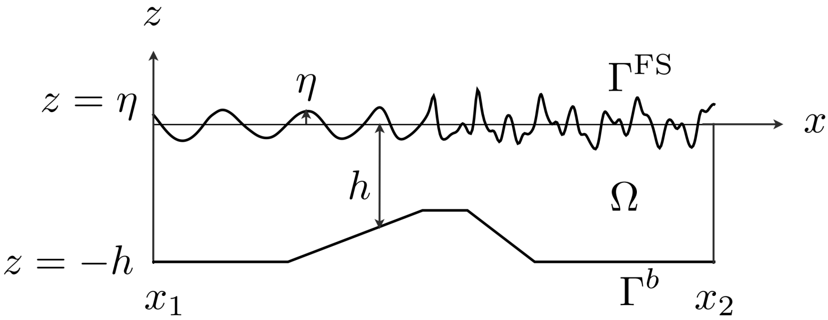


Figure 5.5: Wave tank setup and physical variables. Figure comes from [31].

the neural network surrogate model, we only consider the surface variables η and $\tilde{\phi}$, as the state and the height of the submerged bar is the parameter of interest. This highlights an important advantage of using a non-intrusive surrogate model, as it becomes possible to only model the relevant quantities of interest, bypassing the computations of the velocity potential in the vertical direction.

To generate the training data, we model the setup using the fully nonlinear water wave model for deep fluids as described in [31]. The problem is modeled in 2D by means of a set of 1D PDEs for the free surface boundary conditions, together with a 2D Laplace problem in the full domain. Let x be the horizontal component and z the vertical component, see Figure 5.5. The velocity potential, $\phi : (x, z, t) \mapsto \phi(x, z, t)$, is the scalar function defined on the whole 2D domain, and the free surface elevation, $\eta : (x, t) \mapsto \eta(x, t)$, is defined only on the 1D surface. The free surface boundary conditions can be expressed in the so-called Zakharov form [169], modeled by two 1D PDEs – the wave height, η, and the velocity potential, $\tilde{\phi}$:

$$\begin{aligned}
\frac{\partial \eta}{\partial t} &= -\boldsymbol{\nabla}\eta \cdot \boldsymbol{\nabla}\tilde{\phi} + \tilde{w}(1 + \boldsymbol{\nabla}\eta \cdot \boldsymbol{\nabla}\eta), \\
\frac{\partial \tilde{\phi}}{\partial t} &= -g\eta - \frac{1}{2}\left(\boldsymbol{\nabla}\tilde{\phi} \cdot \boldsymbol{\nabla}\tilde{\phi} - \tilde{w}^2(1 + \boldsymbol{\nabla}\eta \cdot \boldsymbol{\nabla}\eta)\right).
\end{aligned} \tag{5.12}$$

Eq. (5.12) is defined on the surface part of the domain, Γ^{FS}. $\tilde{w} = \partial_z \phi|_{z=\eta}$ and $\tilde{\phi} = \phi|_{z=\eta}$ are the surface parts of the functions that are defined on the 2D domain and Γ^{FS} is the free surface, as shown in Figure 5.5. The velocity potential on the domain is modeled by the 2D Laplace problem, via the σ-transform:

$$\begin{aligned}
\nabla^\sigma(K(x;t)\nabla^\sigma \phi) = 0, &\quad \text{in} \quad \Omega^c, \\
\phi = \tilde{\phi}, &\quad z = \eta \quad \text{on} \quad \Gamma^{FS} \\
\mathbf{n} \cdot \nabla \phi = 0, &\quad z = -h(x, y) \quad \text{on} \quad \Gamma^b.
\end{aligned} \tag{5.13}$$

where $\sigma = (z + h(x))d(x,t)^{-1}$, $\Omega^c = \{(x,\sigma)|0 \leq \sigma \leq 1\}$, and

$$K(x,t) = \begin{bmatrix} d & -\sigma\partial_x\eta \\ -\sigma\partial_x\eta & \frac{1+(\sigma\partial_x\eta)^2}{d} \end{bmatrix}. \tag{5.14}$$

We use the spectral element method, as described in [31], for the discretization of the equations. We use 103 elements in the horizontal direction and 1 element in the vertical direction, both with 6th order polynomials, to generate the training data. The equations are solved with a step size of 0.03535 with $t \in [0, 42.42]$, resulting in 1200 time steps, and the bar height is uniformly sampled between 0.1 and 0.325. The states are interpolated onto a regular grid of 512 points.

Sensor observations from the experiment are available at a time frequency of 0.03535s, which was also chosen as the step size for the simulations. To demonstrate how the D-LSPF performs with varying time intervals between the observations, we show the results for sensor observations at every 25th and 75th time step, corresponding to every 0.884s and 2.651s, respectively. We refer to these two settings as case 1 and 2. We only observe the wave height and not the velocity potential. Since we deal with real-world data, the true full state is not available. Therefore, we measure the accuracy against the full time series of observations, showcasing that the D-LSPF can accurately estimate the state between observations. We do, however, also compare the results with a high-fidelity simulation with the true bar height. Furthermore, we present the accuracy of the bar height estimates.

Table 5.3: Results for the harmonic wave generation test case using the D-LSPF and the ROAD-EnKF with 100 and 1000 particles. Timings are measured using a single GPU. The PICP is computed using the 2.5th and the 97.5th percentile. An upward pointing arrow means larger values are better and a downward pointing arrow means lower values are better. "S-" and "P-" refer to the state and parameters, respectively.

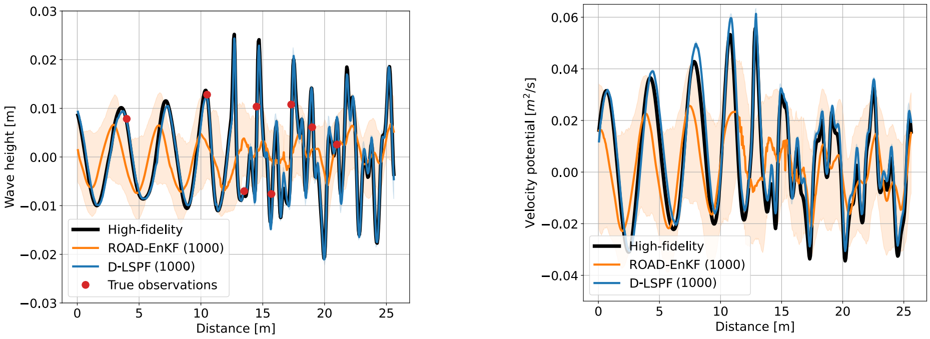
	Case 1 – Every 25 time step			
	S-RRMSE ↓	S-PICP ↑	P-RRMSE ↓	Time ↓
D-LSPF(100)	$\mathbf{3.6 \cdot 10^{-1}}$	$4.4 \cdot 10^{-1}$	$\mathbf{4.4 \cdot 10^{-3}}$	**1.5s**
D-LSPF(1000)	$3.8 \cdot 10^{-1}$	$3.9 \cdot 10^{-1}$	$4.6 \cdot 10^{-3}$	10.5s
ROAD-EnKF(100)	1.1	$5.0 \cdot 10^{-1}$	-	5.4s
ROAD-EnKF(1000)	1.0	$\mathbf{5.5 \cdot 10^{-1}}$	-	31.3s
	Case 2 – Every 75 time step			
D-LSPF(100)	$4.3 \cdot 10^{-1}$	$4.9 \cdot 10^{-1}$	$5.3 \cdot 10^{-3}$	**1.4s**
D-LSPF(1000)	$\mathbf{4.0 \cdot 10^{-1}}$	$5.6 \cdot 10^{-1}$	$\mathbf{4.9 \cdot 10^{-3}}$	10.0s
ROAD-EnKF(100)	1.0	$\mathbf{5.8 \cdot 10^{-1}}$	-	5.1s
ROAD-EnKF(1000)	1.0	$\mathbf{5.8 \cdot 10^{-1}}$	-	29.9s

We compare the D-LSPF with the ROAD-EnKF [22], where we train the ROAD-EnKF model on the same simulated data as the D-LSPF with full access to the states in space and time. To deal with the multiple states, wave height and velocity potential, we introduce a slight modification in the decoder network in the ROAD-EnKF compared to [22], by ensuring that the Fourier decoder networks outputs data with two channels. The latent dimension is chosen to be 8 for the D-LSPF and 40 for the ROAD-EnKF. The neural network architectures for the modified ROAD-EnKF model have been chosen through hyperparameter tuning.

Table 5.3 contains the Relative RMSE (RRMSE), probability interval coverage percentage (PICP), and timings for the D-LSPF and the ROAD-EnKF in both variations of the test case using 100 and 1000 particles. The D-LSPF clearly performs best with respect to the state estimation with an improvement of one order of magnitude, for both 100 and 1000 particles. For the PICP, the ROAD-EnKF does slightly better, however, inspecting Figures 5.6a and 5.6b, we see that the ROAD-EnKF has large uncertainty intervals while being quite inaccurate on average compared to the D-LSPF. In general, the PICP is less relevant when the RRMSE is bad. This is further highlighted in Figure 5.7a and 5.7b, plotting the windowed RRMSE versus time. The windowed RRMSE measures the RRMSE in a time window in order to show how the state estimation improves when more observations become available. The D-LSPF converges and even surpasses the high-fidelity simulation, showcasing how assimilating observations impactfully improves accuracy. We also notice for both the D-LSPF and ROAD-EnKF that there are only minor differences between using 100 and 1000 particles. Furthermore, the RRMSE only varies slightly between the two cases, suggesting that both methods are stable with respect to observation frequency.

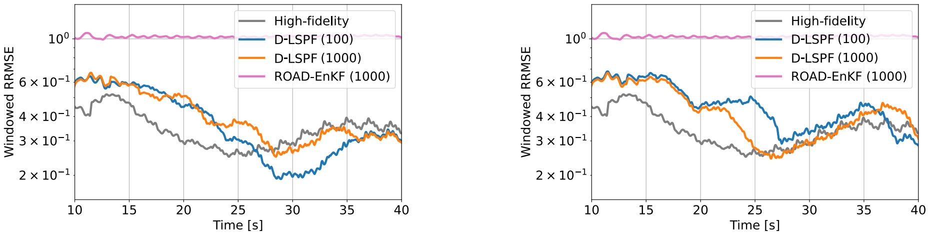
Besides the accuracy, Table 5.3 also notes the computation time. In both cases, both the D-LSPF and the ROAD-EnKF are faster than real-time, as the data assimilation takes place over 40s in physical time and the D-LSPF and ROAD-EnKF computation times vary between 1.4s and 29.9s. In general, the D-LSPF is between 3 and 4 times faster than the ROAD-EnKF. For comparison, a single high-fidelity model simulation takes on average some 1062s. Hence, running the particle filter using the high-fidelity model on 30 CPU cores, assuming no overhead associated with the parallelization, would take approximately 3542s with 100 particles and 35420s for 1000 particles, yielding a speed-up of 2345 for 100 particles and 3376 for 1000 particles on a GPU when using the D-LSPF. When deploying the D-LSPF and running it as the data comes in, it is not possible to perform the data assimilation task faster than the arrival of observations. However, the timings show that the method assimilates the data without any delay.

Figure 5.8 shows the quality of the D-LSPF estimates of the state and the sensor locations between observations. Furthermore, the difference between using 100 and 1000 particles is negligible for the state estimates. However, when zooming in, it becomes clear that using more particles result in better uncertainty intervals. For an overview of the estimation at all sensor locations, including ROAD-EnKF results, see Figure 5.9.



(a) Wave height estimation, $\eta \pm 2\sigma$. (b) Velocity potential estimation, $\tilde{\phi} \pm 2\sigma$.

Figure 5.6: State estimation at t=40s for the harmonic wave test case with observations every 75 time steps. Note that only wave height is observed and not velocity potential.



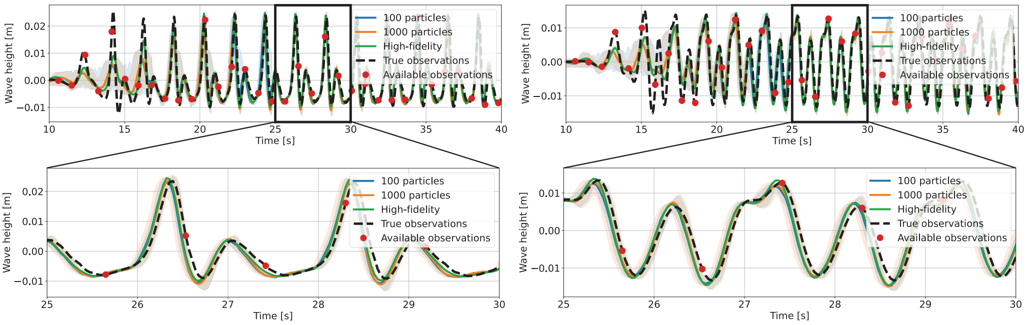
(a) Observations every 25 time step. (b) Observations every 75 time step.

Figure 5.7: Windowed RRMSE for the harmonic wave test case with a window size of 75 time steps.

Lastly, in Figure 5.10, the posterior distributions of the bar height for varying numbers of particles and observation frequency are shown. While the average bar height estimates are very similar for the 100 and 1000 particles, the distributions change from being multimodal to unimodal.

5.4.3 Multi-phase leak localization

Here, we consider leak localization in a two-phase pipe flow for liquid (water) and air, and we assume that they are mixed. We consider a case where only a few sensors are placed along a 5km long pipe, measuring pressure. The system is in a steady state when a leak occurs, where we then use the particle filter to compute a distribution over the leak location, leak size, liquid holdup, mixture velocity, and pressure. This problem is similar to the setup in [153], where single-phase CO_2 was considered, but state estimation was only performed in a non-leakage case.



(a) Sensor located at $13.5m$. (b) Sensor located at $14.5m$.

Figure 5.8: State estimation at two sensor locations computed with the D-LSPF with 100 and 100 particles compared with a high-fidelity simulation with the true bar height and the true sensor data.

The state for data assimilation and our surrogate model is the primitive state, consisting of the liquid holdup, pressure, and mixture velocity, $q = (\alpha_l, p, u_m)$. Both training, testing, and observations data are simulated and noise is artificially added to the observations with a standard deviation of 0.01. We use the homogeneous equilibrium model (HEM) [109] for the simulations. The HEM is a set of nonlinear, hyperbolic one-dimensional PDEs:

$$\begin{aligned}
\partial_t(A_g\rho_g) + \partial_x(A_g\rho_g u_m) &= -\alpha_g L(\rho_m)\delta(x - x_l), \\
\partial_t(A_l\rho_l) + \partial_x(A_l\rho_l u_m) &= -\alpha_l L(\rho_m)\delta(x - x_l), \\
\partial_t(A\rho_m u_m) + \partial_x\left(\rho_m u_m^2 A + p(\rho_g)A\right) &= -\frac{\rho u|u|}{2} f_f(\rho, u),
\end{aligned} \tag{5.15}$$

with boundary conditions $\rho_g u_m(0,t) = (\rho_g u_m)_0$ and $\rho_l u_m(0,t) = (\rho_l u_m)_0$ at the pipe inlet on the left side, and $p(L,t) = p_L$ at the outlet on the right side. f_f is the wall friction, Re is the Reynolds number, and L is the leak size. δ is the Dirac delta function, ensuring that the leak is only active at $x = x_l$. A is the cross section area of the pipe, A_g is the area occupied by gas and A_l the area occupied by liquid. Hence, for $\alpha_g \in [0,1]$ and $\alpha_l \in [0,1]$ being the fraction of gas and liquid, respectively, we have:

$$A_g = \alpha_g A, \quad A_l = \alpha_l A, \quad A = A_g + A_l, \quad \alpha_g + \alpha_g = 1. \tag{5.16}$$

ρ_l is the liquid density assumed to be constant, ρ_g is the gas density, and ρ_m is the mixture density (not to be confused with the probability density functions, ρ),

$$\rho_m = \alpha_g\rho_g + \alpha_l\rho_l. \tag{5.17}$$

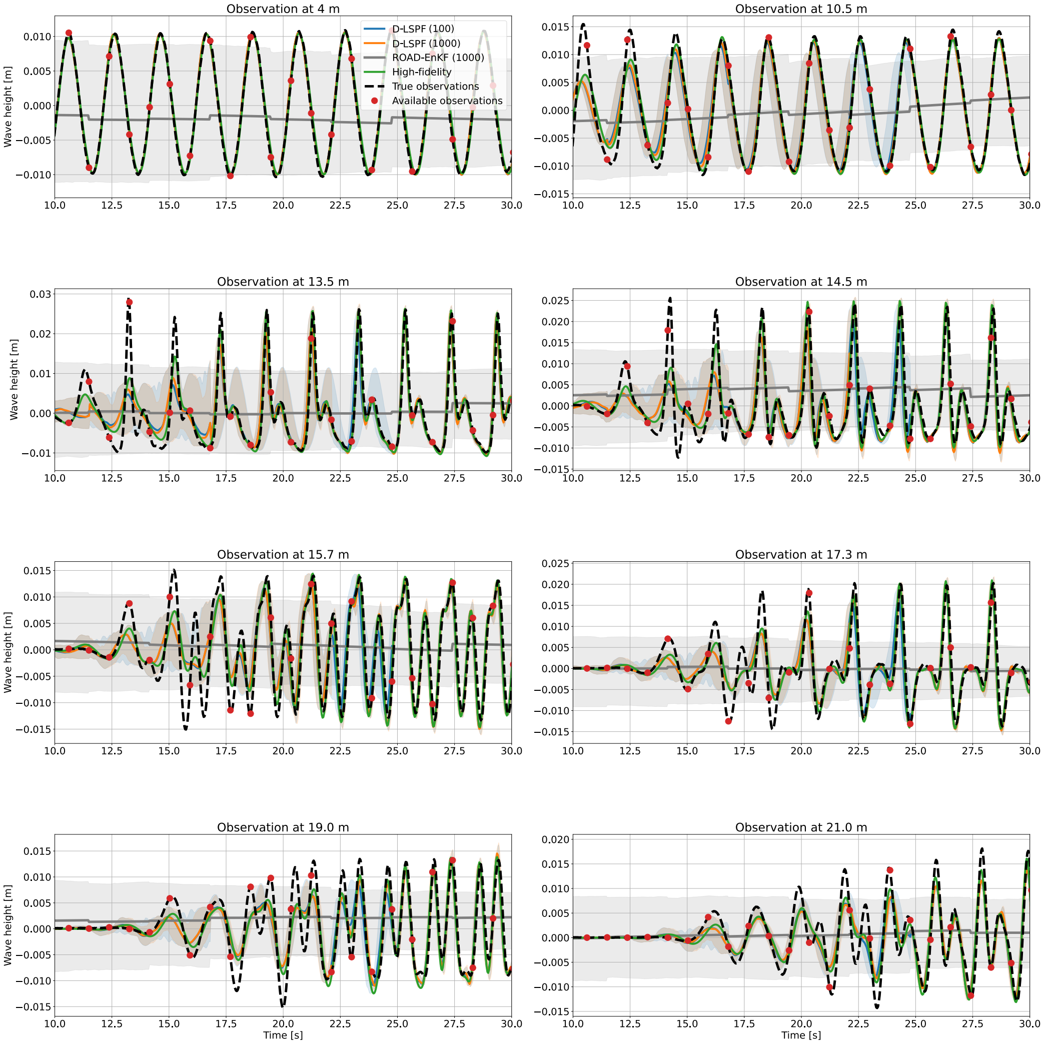
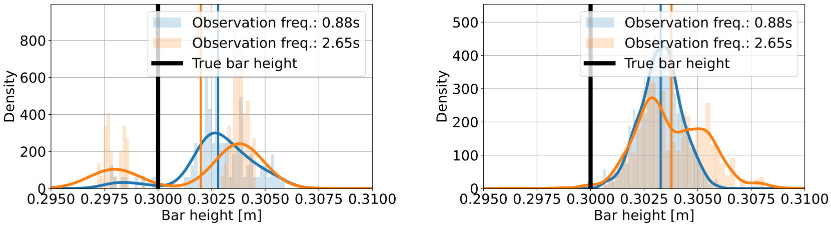


Figure 5.9: State values at the sensor locations compared with the true sensor observations for the D-LSPF with 100 and 1000 particles, as well as for a high-fidelity simulation with the true parameters chosen.



(a) D-LSPF with 100 particles. (b) D-LSPF with 1000 particles.

Figure 5.10: Posterior distribution over the bar height at $t = 40s$ for varying observation frequency and number of particles. The density outlines are computed using kernel density estimation based on the posterior particles. Vertical lines represent the mean of the distributions.

The wall friction, f_f, is given by,

$$f_f = 2\left(\left(\frac{8}{Re}\right)^{12} + (a+b)^{-1.5}\right)^{1/12}, \quad a = \left(-2.457\ln\left(\frac{7}{Re}\right)^{0.9} + 0.27\frac{\varepsilon}{2r}\right)^{16}, \quad b = \left(\frac{37530}{Re}\right)^{16}, \tag{5.18}$$

the Reynolds number, Re, is given by

$$Re = \frac{2r\rho_m u_m}{\mu_m}, \qquad \mu_m = \alpha_g\mu_g + \alpha_l\mu_l, \tag{5.19}$$

and the leak size, L, is given by,

$$L(\rho_m) = C_v\sqrt{\rho_m(p(\rho_g) - p_{\text{amb}})}. \tag{5.20}$$

For specific values of all constants in Eq. (5.15), see Table 5.4.

We discretize the PDEs using the nodal discontinuous Galerkin method [66] with Legendre polynomials for the modal representation and Lagrange polynomials for the nodal degrees of freedom. We use the Lax-Friedrichs discretization for the numerical flux and BDF2 for time stepping. The nonlinear equations coming from the implicit time stepping are solved using Newton's method, where the resulting linear systems are solved via an LU factorization. Furthermore, the Jacobian matrix is only updated every 500 time-steps to speed up the computations.

The true state and observations used for the data assimilation are simulated using 3000 elements and third-order polynomials. The states are then evaluated on regular grid of 512 points.

The training data is simulated using 2000 elements and second-order polynomials. There-

Table 5.4: Parameters for the multi phase pipe flow equations, (5.15). Note that the discharge coefficient and the leakage location have values denoted by intervals, as they are the parameters to determine.

Physical quantity	Constant	Value	Unit
Pipe length	L	5000	m
Diameter	d	0.2	m
Cross-sectional area	A	0.0314	m^2
Speed of sound in gas	c	308	m/s
Ambient pressure	p_{amb}	$1.01325 \cdot 10^5$	Pa
Reference pressure	p_{ref}	$1 \cdot 10^5$	Pa
Reference density (gas)	ρ_g	1.26	$\mathrm{kg/m}^3$
Reference density (liquid)	ρ_l	1003	$\mathrm{kg/m}^3$
Inflow velocity	v_0	4.0	m/s
Outflow pressure	p_L	1.0e6	Pa
Pipe roughness	ε	10^{-8}	m
Fluid viscosity (gas)	μ_g	$1.8 \cdot 10^{-5}$	$\mathrm{N \cdot s/m}^2$
Fluid viscosity (liquid)	μ_l	$1.516 \cdot 10^{-5}$	$\mathrm{N \cdot s/m}^2$
Temperature	T	278	Kelvin
Discharge coefficient	C_d	$[1, 2]$	m
Leakage location	x_l	$[10, 5990]$	m

after, it is evaluated on a regular grid consisting of 512 grid points. The equations are solved with a time-step size of 0.01, for $t = [0, 120]$ seconds. Hence, there are 12000 time steps. The surrogate model is trained to take steps 10 times larger than the high-fidelity model.

For testing the D-LSPF, we compute a high-fidelity particle filter solution with 5000 particles as a baseline to measure if we estimate the distributional information accurately. The high-fidelity solver uses 700 elements and 3rd order polynomials.

The available observations are located at 8 spatial locations: x=(489.24, 978.47, 1467.71, 1956.95, 2446.18, 2935.42, 3424.66, 3913.89, 4403.13). Only pressure is observed. The observations arrive with a time frequency of 4s, corresponding to every 400 time steps of the PDE model. To ensure consistent performance across various configurations, we compute the metrics over 8 different test trajectories with varying leak locations and sizes and take the average. We compute the state and parameter RRMSE against the true solution, the state AMRMSE against a HF particle filter solution, the state and parameter NLL against the HF particle filter solution, as well as the Wasserstein-1 distance of the posterior parameter distribution against the HF posterior.

We use this test case as an ablation study: we use the same particle filter setting for all approaches and only replace the chosen architectures. We compare the presented architectures with three alternatives: One where the transformer-based AE layers are replaced with convolutional ResNet layers. The dimensionality reduction in this network is performed through strided convolutions and the dimensionality expansion is performed using transposed convolutions. The second alternative is using the proposed ViT architecture but without the Wasserstein distance and consistency regularization in the latent space for the autoencoder. Lastly, we compare with a setup that still uses the proposed ViT AE with regularization, but replaces the transformer based time stepping with a neural ODE (NODE) [21]. In all cases, a latent dimension of 8 is chosen.

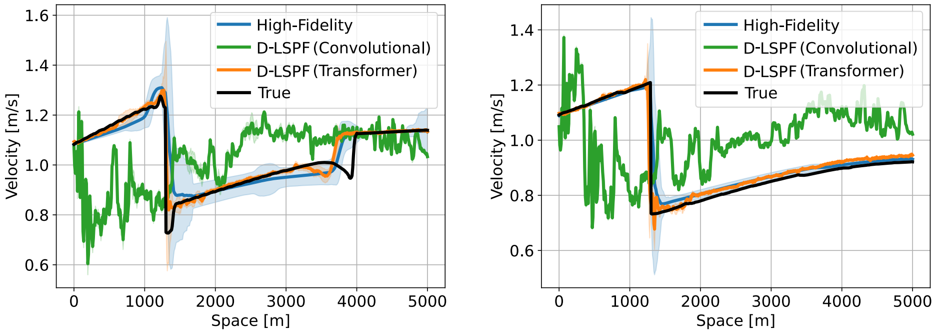
Remark We also trained a Fourier Neural Operator (FNO) [91] as a surrogate model, but without success. In all attempts, the solutions exploded after a certain number of time steps. This may be due to the fact that there is a clear discontinuity which is located at the parameter, x_l. Fourier series may not be good approximators for such tasks. For this reason, FNO results are omitted from the chapter.

Table 5.5: Computation times using a GPU for the D-LSPF applied to the multi phase leak localization test case. All timings are computed with 5000 particles. The high-fidelity solver makes use of 100 CPU cores and the neural network uses one GPU. "P-" refers to parameter estimation and "S-" refers to state estimation.

	HF	Reg-ViT-Trans	NoReg-ViT-Trans	Reg-Conv-Trans	Reg-ViT-NODE
P-RRMSE $\downarrow$	$5.2 \cdot 10^{-1}$	$\mathbf{9.6 \cdot 10^{-3}}$	$1.1 \cdot 10^{-2}$	$7.4 \cdot 10^{-1}$	$1.5 \cdot 10^{-2}$
P-Wasserstein-1 $\downarrow$	-	169.9	**168.4**	875.7	172.7
S-RRMSE $\downarrow$	$7.9 \cdot 10^{-2}$	$\mathbf{2.5 \cdot 10^{-2}}$	$2.9 \cdot 10^{-2}$	$8.3 \cdot 10^{-2}$	$3.1 \cdot 10^{-2}$
S-AMRMSE $\downarrow$	-	$\mathbf{4.3 \cdot 10^{-3}}$	$4.4 \cdot 10^{-3}$	$4.5 \cdot 10^{-3}$	$4.5 \cdot 10^{-3}$
P-NLL $\downarrow$	-	**5.09**	5.70	6.01	12.54
S-NLL $\downarrow$	-	16.87	**15.37**	18.37	17.32
Time (GPU) $\downarrow$	-	24.89s	24.90s	23.58s	**6.52**
Speed-up (GPU) $\downarrow$	-	1807.95	1807.23	1908.40	**6901.84**
Time (CPU) $\downarrow$	45000s	332.3s	317.0s	328.8s	**45.9s**
Speed-up (CPU) $\downarrow$	-	135.4	142.0	136.9	**980.8**

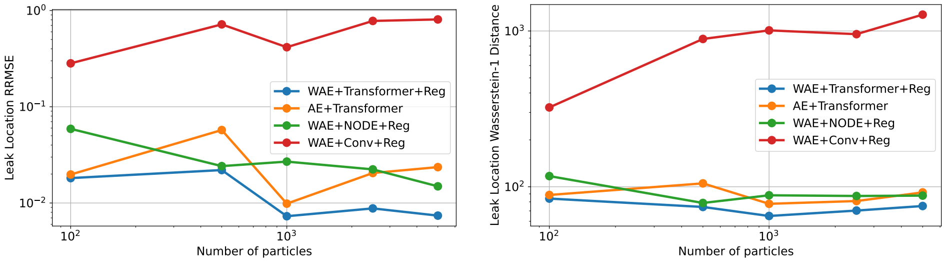
The results are summarized in Table 5.5. Our chosen architecture demonstrates superior performance in most metrics. Note in particular that the results using the convolutional AE are significantly worse compared to the ViT AE, emphasizing the advantages of the transformer-based architecture across different combinations. Furthermore, the D-LSPF outperforms the high-fidelity particle filter in estimating the leak location and size as well as the state. This can be explained from the fact that the D-LSPF is trained on higher resolution trajectories. As this came at a higher cost only at the training stage, it does not add to the computation time when running the particle filter.

In Figure 5.11, the true velocity at $t = 40$ and $t = 120$ are compared with estimates using a high-fidelity model, the D-LSPF with the ViT AE, and the D-LSPF with convolutional AE. The convolutional AE clearly fails to reconstruct the state in any meaningful way, while the D-LSPF with the ViT AE accurately estimates the velocity with the uncertainty concentrated around the leak location as expected.



(a) Velocity estimation at $t = 40$s. (b) Velocity estimation at $t = 120$s.

Figure 5.11: Velocity estimation results for the multi phase pipeflow test case.



(a) Leak location and size Relative RMSE (b) Wasserstein distance for the leak location.

Figure 5.12: Results for the leak size and location estimation at time 120s, for varying neural network specifications and number of particles.

In Figure 5.12, we show the convergence accuracy of the leak location estimation for all the neural network setups. We see in Figure 5.12a that the proposed setup is superior than the alternatives with respect to the RRMSE for all number of particles. Similarly, we reach the same conclusion with respect to the Wasserstein-1 distance of the posterior distribution of the leak location.

5.5 Conclusion

We presented a novel particle filter, the D-LSPF, for fast and accurate data assimilation with uncertainty quantification. The D-LSPF was based on a surrogate model utilizing dimensionality reduction and latent space time stepping. The use of particle filters provided estimates for the state and parameters as well as for the associated uncertainties. For the AE, we made use of a novel extension of the vision transformer for dimensionality reduction and reconstruction. Furthermore, we discussed a number of regularization techniques to improve the performance of the D-LSPF.

We demonstrated the D-LSPF on three different test cases with varying characteristics and complexities. In the first test, we compared with alternative deep learning-based data assimilation methods for the viscous Burgers equation. The D-LSPF showed superior performance by an order of magnitude regarding uncertainty estimation as well as almost 4 times better mean reconstruction. In the second test case, we performed state and parameter estimation on a wave tank experiment with few observations available in time and space. The D-LSPF performed up to 3 times better than alternative approaches while also being faster and successfully estimated both the state and parameter. Lastly, we applied the D-LSPF on a leak localization problem for multi-phase flow in a long pipe. We showed how the ViT AE and the regularization techniques drastically improved the state and parameter estimation: the D-LSPF provided speed-ups of 3 orders of magnitude compared with a high-fidelity particle filter while also being more accurate.

Several aspects of our work could benefit from future investigations. In particular, we made use of the bootstrap particle filter. While this particular version of the particle filter has many advantages such as relative ease of implementation and convergence, there are alternatives. It might be fruitful to analyze the quality of filters that utilize the differentiability of neural networks, such as the nudging particle filter with gradient nudging [3] or particle filters based on Stein variational gradient descent [34, 100].

In all, the D-LSPF significantly sped up complex data assimilation tasks without sacrificing accuracy, thus enabling fusing of increasingly complex models with data in real-time. This work promises a solid foundation for future digital twins.

6

CONCLUSION

In this thesis, we discussed several methods to speed up inverse problems and data assimilation with uncertainty quantification. The aim was to enable real-time state and parameter estimation to be used in digital twins, which before was limited due to excessive amount of computations related to such tasks. The integration of advanced deep learning techniques with traditional scientific computing methods has led to novel developments in reduced order modeling, Bayesian inverse problem solving, digital twin technology for leak localization, and real-time nonlinear data assimilation. Specifically, techniques like autoencoders, LSTM, CNNs, and transformers from the realm of deep learning were shown to work well with conventional methods from scientific computing, such particle filters and Markov Chain Monte Carlo methods.

6.1 Chapter Conclusions

Each chapter contributed to the overarching theme by building a suite of tools and methodologies. These methods enhance computational efficiency in various real-world scenarios. Below, we conclude the findings in each of the chapters.

Chapter 2 In this chapter, we have successfully demonstrated the application of spatially- and memory-aware deep learning techniques in reduced order modeling (ROM) for parameterized time-dependent partial differential equations (PDEs). The development of a nonintrusive framework, integrating convolutional autoencoders for nonlinear dimensionality reduction and memory-aware neural networks for time stepping, marks a significant advancement in computational efficiency and accuracy. This approach allows for a more precise capture of the complex dynamics inherent in parameterized PDEs, setting a new benchmark for ROM in scientific computing.

Chapter 3 Chapter 3 presented the Markov Chain Generative Adversarial neural Network (MCGAN) method for solving Bayesian inverse problems in physics applications. The innovative approach of embedding GANs into a Hamiltonian Monte Carlo sampling scheme enhances the efficiency of sampling from complex distributions. This chapter not only presented a theoretical convergence proof for the method but also highlighted its practical efficacy in scenarios like Darcy flow and leakage detection in pipe flow, underscoring its potential in various physics-based applications.

Chapter 4 In this chapter, we presented a probabilistic digital twin framework utilizing generative deep learning for leak localization in water distribution networks. The use of a supervised Wasserstein Autoencoder for generating complete states of flow and pressure, coupled with Bayesian inference, represents a major leap in the efficiency and accuracy of leak localization methods. The innovative transformer-based architecture further elevated the model's performance, demonstrating the practicality of this approach in real-world scenarios.

Chapter 5 The final chapter introduced the Deep Latent Space Particle Filter (D-LSPF), a novel method for real-time nonlinear data assimilation with uncertainty quantification. By integrating transformer-based dimensionality reduction in a Wasserstein Autoencoder with a time-stepping transformer, this methodology significantly enhanced the efficiency of particle filtering. This approach not only estimated the state accurately but also captured the full posterior distribution, offering robust solutions for complex data assimilation problems.

6.2 Outlook and Future Work

This thesis has explored the integration of deep learning in the realm of digital twins, focusing on real-time inverse problems and data assimilation. The advancements made here pave the way for further research and development in several key areas. The future directions can be categorized into three major subsections:

Deep Learning-Based Surrogate Models The use of deep learning-based surrogate models in scientific computing has shown promising results in this thesis. Future work should aim to enhance these models' accuracy and efficiency even further.

In future developments, one could focus on the integration of transfer learning to leverage pre-trained models for new but related problems to significantly reduce computational costs [141]. This could especially lower the offline training costs by not training from scratch every time a new problem arises.

Furthermore, one could look into incorporating physics knowledge directly into the neural network architectures or training [125, 154]. In this thesis, we mainly focused on purely data-driven approaches. By incorporating physical constraints, one could ensure that the surrogate model adheres to fundamental principles of physics to higher degree with less data.

Lastly, instead of only training the surrogate in the offline stage, one could continue the training when new data arrives by using online learning [35]. With such approaches, the surrogate could potentially improve over time, ensuring long-term stability.

Inverse Problems and Data Assimilation In the area of inverse problems and data assimilation, there is still much to be explored. We primarily used sampling based methods, such MCMC and particle fitlers, in this thesis. However, by utilizing the differentiability of neural networks, one could potentially gain further speed-ups and accuracy. This could be achieved with alternative Bayesian approaches such as nudging particle filters [3] and Stein variational gradient descent methods [34, 100].

Furthermore, one could bypass the expensive likelihood computation by exploring likelihood-free methods [2]. With such methods, one has to define alternative formulations of the posterior distribution as well as alternative ways to estimate it, using e.g. adversarial losses directly in the inference schemes [44, 97].

Generative Modeling for Scientific Computing Generative models based on neural networks have shown substantial promise in scientific computing applications. We focused on GAN- and autoencoder-based generative models in this thesis. However, denoising diffusion models (DDMs) have already shown great promise outside of scientific computing. DDMs for physics applications are still widely unexplored and might prove to be useful [70, 92]. However, a drawback of DDMs is the computation time associated with sampling, which might make them difficult to use when real-time computations are important.

BIBLIOGRAPHY

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, 2016.

[2] J. Adler and O. Öktem. Deep bayesian inversion. *arXiv preprint arXiv:1811.05910*, 2018.

[3] Ö. D. Akyildiz and J. Míguez. Nudging the particle filter. *Statistics and Computing*, 30:305–330, 2020.

[4] A. Albarakati, M. Budišić, R. Crocker, J. Glass-Klaiber, S. Iams, J. Maclean, N. Marshall, C. Roberts, and E. S. Van Vleck. Model and data reduction for data assimilation: Particle filters employing projected forecasts and data with application to a shallow water model. *Computers & Mathematics w. Applications*, 116:194–211, 2022.

[5] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein generative adversarial networks. In *International Conference on Machine Learning*, pages 214–223. PMLR, 2017.

[6] M. Asch. *A toolbox for digital twins: from model-based to data-driven.* SIAM, 2022.

[7] M. Asch, M. Bocquet, and M. Nodet. *Data assimilation: methods, algorithms, and applications.* SIAM, 2016.

[8] C. Audouze, F. De Vuyst, and P. B. Nair. Nonintrusive reduced-order modeling of parametrized time-dependent partial differential equations. *Numerical Methods for Partial Differential Equations*, 29(5):1587–1628, 2013.

[9] N. Baker, F. Alexander, T. Bremer, A. Hagberg, Y. Kevrekidis, H. Najm, M. Parashar, A. Patra, J. Sethian, S. Wild, et al. Workshop report on basic research needs for scientific machine learning: Core technologies for artificial intelligence. Technical report, USDOE Office of Science (SC), Washington, DC (United States), 2019.

[10] D. H. Ballard. Modular learning in neural networks. In *Proceedings of the Sixth National Conference on Artificial Intelligence - Volume 1*, pages 279–284, 1987.

[11] S. Beji and J. A. Battjes. Numerical simulation of nonlinear wave propagation over a bar. *Coastal Engineering*, 23(1-2):1–16, 1994.

[12] D. Bigoni, A. P. Engsig-Karup, and C. Eskilsson. Efficient uncertainty quantification of a fully nonlinear and dispersive water wave model with random inputs. *Journal of Engineering Mathematics*, 101:87–113, 2016.

[13] V. I. Bogachev. *Measure theory*, volume 1. Springer Science & Business Media, 2007.

[14] A. Borovykh, S. Bohte, and C. W. Oosterlee. Conditional time series forecasting with convolutional neural networks. *arXiv preprint arXiv:1703.04691*, 2017.

[15] P. Brémaud. *Probability Theory and Stochastic Processes.* Springer Nature, 2020.

[16] S. Brooks, A. Gelman, G. Jones, and X.-L. Meng. *Handbook of Markov chain Monte Carlo.* CRC press, 2011.

[17] S. Brunton, B. Noack, and P. Koumoutsakos. Machine learning for fluid mechanics. *arXiv preprint arXiv:1905.11075*, 2019.

[18] S. L. Brunton, J. L. Proctor, and J. N. Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15):3932–3937, 2016.

[19] S. L. Brunton, B. R. Noack, and P. Koumoutsakos. Machine learning for fluid mechanics. *Annual Review of Fluid Mechanics*, 52:477–508, 2020.

[20] K. Champion, B. Lusch, J. N. Kutz, and S. L. Brunton. Data-driven discovery of coordinates and governing equations. *Proceedings of the National Academy of Sciences*, 116(45):22445–22451, 2019.

[21] T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, pages 6571–6583, 2018.

[22] Y. Chen, D. Sanz-Alonso, and R. Willett. Reduced-order autodifferentiable ensemble kalman filters. *Preprint arXiv:2301.11961*, 2023.

[23] S. Cheng, C. Quilodrán-Casas, S. Ouala, A. Farchi, C. Liu, P. Tandeo, R. Fablet, D. Lucor, B. Iooss, J. Brajard, et al. Machine learning with data assimilation and uncertainty quantification for dynamical systems: a review. *IEEE/CAA Journal of Automatica Sinica*, 10(6):1361–1387, 2023.

[24] B. Cockburn, J. Guzmán, and H. Wang. Superconvergent discontinuous galerkin methods for second-order elliptic problems. *Mathematics of Computation*, 78(265): 1–24, 2009.

[25] D. L. Colton, R. Kress, and R. Kress. *Inverse acoustic and electromagnetic scattering theory*, volume 93. Springer, 1998.

[26] P. Dhariwal and A. Nichol. Diffusion models beat gans on image synthesis. *arXiv preprint arXiv:2105.05233*, 2021.

[27] Z. Ding and Q. Li. Ensemble kalman inversion: mean-field limit and convergence analysis. *Statistics and Computing*, 31(1):1–21, 2021.

[28] S. Domesová and M. Beres. Solution of inverse problems using bayesian approach with application to estimation of material parameters in darcy flow. *Advances in Electrical & Electronic Engineering*, 15(2), 2017.

[29] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[30] C. Drygala, B. Winhart, F. di Mare, and H. Gottschalk. Generative modeling of turbulence. *Physics of Fluids*, 34(3):035114, 2022.

[31] A. P. Engsig-Karup, C. Eskilsson, and D. Bigoni. A stabilised nodal spectral element method for fully nonlinear water waves. *Journal of Computational Physics*, 318:1–21, 2016.

[32] N. B. Erichson, M. Muehlebach, and M. W. Mahoney. Physics-informed autoencoders for lyapunov-stable fluid flow prediction. *arXiv preprint arXiv:1905.10866*, 2019.

[33] G. Evensen, F. C. Vossepoel, and P. J. van Leeuwen. *Data assimilation fundamentals: A unified formulation of the state and parameter estimation problem.* Springer Nature, 2022.

[34] J. Fan, A. Taghvaei, and Y. Chen. Stein particle filtering. *Preprint arXiv:2106.10568*, 2021.

[35] A. Farchi, M. Chrust, M. Bocquet, P. Laloyaux, and M. Bonavita. Online model error correction with neural networks in the incremental 4d-var framework. *Journal of Advances in Modeling Earth Systems*, 15(9):e2022MS003474, 2023.

[36] P. Fearnhead and H. R. Künsch. Particle filters and data assimilation. *Annual Review of Statistics and Its Application*, 5:421–449, 2018.

[37] J. Feinberg and H. P. Langtangen. Chaospy: An open source tool for designing methods of uncertainty quantification. *Journal of Computational Science*, 11:46–57, 2015.

[38] M. Frangos, Y. Marzouk, K. Willcox, and B. van Bloemen Waanders. Surrogate and reduced-order modeling: a comparison of approaches for large-scale statistical inverse problems. *Large-Scale Inverse Problems and Quantification of Uncertainty*, pages 123–149, 2010.

[39] S. Fresca, L. Dede, and A. Manzoni. A comprehensive deep learning-based approach to reduced order modeling of nonlinear time-dependent parametrized pdes. *arXiv preprint arXiv:2001.04001*, 2020.

[40] S. Fresca, L. Dede', and A. Manzoni. A comprehensive deep learing-based approach to reduced order modeling of nonlinear time-dependent parametrized pdes. *Journal of Scientific Computing*, 87:1–36, 2021.

[41] J. Friedman, T. Hastie, and R. Tibshirani. *The elements of statistical learning*, volume 1. Springer Series in Statistics New York, 2001.

[42] D. Gamerman and H. F. Lopes. *Markov chain Monte Carlo: stochastic simulation for Bayesian inference.* CRC Press, 2006.

[43] N. Geneva and N. Zabaras. Transformers for modeling physical systems. *Neural Networks*, 146:272–289, 2022.

[44] M. Gillhofer, H. Ramsauer, J. Brandstetter, B. Schäfl, and S. Hochreiter. A gan based solver of black-box inverse problems. In *NeurIPS 2019 Workshop on Solving Inverse Problems with Deep Networks*, 2019.

[45] C. Gin, B. Lusch, S. L. Brunton, and J. N. Kutz. Deep learning models for global coordinate transformations that linearize pdes. *arXiv preprint arXiv:1911.02710*, 2019.

[46] K. Goda. A multistep technique with implicit difference schemes for calculating two-or three-dimensional cavity flows. *Journal of Computational Physics*, 30(1):76–95, 1979.

[47] H. Goh, S. Sheriffdeen, J. Wittmer, and T. Bui-Thanh. Solving bayesian inverse problems via variational autoencoders. *arXiv preprint arXiv:1912.04212*, 2019.

[48] A. Gonczarek and J. M. Tomczak. Articulated tracking with manifold regularized particle filter. *Machine Vision and Applications*, 27:275–286, 2016.

[49] F. J. Gonzalez and M. Balajewicz. Deep convolutional recurrent autoencoders for learning low-dimensional feature dynamics of fluid systems. *arXiv preprint arXiv:1808.01346*, 2018.

[50] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning.* MIT press, 2016.

[51] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[52] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks. *arXiv preprint arXiv:1406.2661*, 2014.

[53] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. Smola. A kernel two-sample test. *The Journal of Machine Learning Research*, 13(1):723–773, 2012.

[54] R. Gribonval, G. Kutyniok, M. Nielsen, and F. Voigtlaender. Approximation spaces of deep neural networks. *Constructive Approximation*, pages 1–109, 2021.

[55] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville. Improved training of wasserstein gans. *arXiv preprint arXiv:1704.00028*, 2017.

[56] M. Guo and J. S. Hesthaven. Reduced order modeling for nonlinear structural analysis using gaussian process regression. *Computer Methods in Applied Mechanics and Engineering*, 341:807–826, 2018.

[57] G. Hager and G. Wellein. *Introduction to high performance computing for scientists and engineers.* CRC Press, 2010.

[58] X. Han, H. Gao, T. Pffaf, J.-X. Wang, and L.-P. Liu. Predicting physics in mesh-reduced space with temporal attention. *arXiv preprint arXiv:2201.09113*, 2022.

[59] J. Harlim. *Data-driven computational methods: parameter and operator estimations.* Cambridge University Press, 2018.

[60] T. Hastie, R. Tibshirani, J. H. Friedman, and J. H. Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.

[61] W. K. Hastings. Monte carlo sampling methods using markov chains and their applications. 1970.

[62] E. Hauge, O. M. Aamo, and J.-M. Godhavn. Model based pipeline monitoring with leak detection. *IFAC Proceedings Volumes*, 40(12):318–323, 2007.

[63] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[64] B. Heo, S. Yun, D. Han, S. Chun, J. Choe, and S. J. Oh. Rethinking spatial dimensions of vision transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 11936–11945, 2021.

[65] J. S. Hesthaven and S. Ubbiali. Non-intrusive reduced order modeling of nonlinear problems using neural networks. *Journal of Computational Physics*, 363:55–78, 2018.

[66] J. S. Hesthaven and T. Warburton. *Nodal discontinuous Galerkin methods: algorithms, analysis, and applications.* Springer Science & Business Media, 2007.

[67] J. S. Hesthaven, G. Rozza, B. Stamm, et al. *Certified reduced basis methods for parametrized partial differential equations*, volume 590. Springer, 2016.

[68] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9 (8):1735–1780, 1997.

[69] M. D. Hoffman and A. Gelman. The no-u-turn sampler: adaptively setting path lengths in hamiltonian monte carlo. *J. Mach. Learn. Res.*, 15(1):1593–1623, 2014.

[70] B. Holzschuh, S. Vegetti, and N. Thuerey. Solving inverse physics problems with score matching. *Advances in Neural Information Processing Systems*, 36, 2023.

[71] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[72] A. Hørsholt, L. H. Christiansen, K. Meyer, J. K. Huusom, and J. B. Jørgensen. Spatial discretization and kalman filtering for ideal packed-bed chromatography. In *2019 18th European Control Conference (ECC)*, pages 2356–2361. IEEE, 2019.

[73] P. L. Houtekamer and H. L. Mitchell. Data assimilation using an ensemble kalman filter technique. *Monthly Weather Review*, 126(3):796–811, 1998.

[74] A. Jabbar, X. Li, and B. Omar. A survey on generative adversarial networks: Variants, applications, and training. *arXiv preprint arXiv:2006.05132*, 2020.

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

[76] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

[77] T. Kadeethum, D. O'Malley, J. N. Fuhg, Y. Choi, J. Lee, H. S. Viswanathan, and N. Bouklas. A framework for data-driven solution and parameter estimation of pdes using conditional generative adversarial networks. *arXiv preprint arXiv:2105.13136*, 2021.

[78] J. Kaipio and E. Somersalo. *Statistical and computational inverse problems*, volume 160. Springer Science & Business Media, 2006.

[79] M. G. Kapteyn, J. V. Pretorius, and K. E. Willcox. A probabilistic graphical model foundation for enabling predictive digital twins at scale. *Nature Computational Science*, 1(5):337–347, 2021.

[80] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[81] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[82] G. Kitagawa. Monte carlo filter and smoother for non-gaussian nonlinear state space models. *Journal of Computational and Graphical Statistics*, pages 1–25, 1996.

[83] D. A. Kopriva. *Implementing spectral methods for partial differential equations: Algorithms for scientists and engineers.* Springer Science & Business Media, 2009.

[84] M. A. Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE journal*, 37(2):233–243, 1991.

[85] P. Kumar, P. Luo, F. J. Gaspar, and C. W. Oosterlee. A multigrid multilevel monte carlo method for transport in the darcy–stokes system. *Journal of Computational Physics*, 371:382–408, 2018.

[86] P. K. Kundu and I. M. Cohen. Fluid mechanics. 2002.

[87] K. Lee and K. T. Carlberg. Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders. *Journal of Computational Physics*, 404:108973, 2020.

[88] S. Lee and N. Baker. Basic research needs for scientific machine learning: Core technologies for artificial intelligence. Technical report, USDOE Office of Science (SC)(United States), 2018.

[89] R. J. LeVeque. *Finite difference methods for ordinary and partial differential equations: steady-state and time-dependent problems*, volume 98. SIAM, 2007.

[90] G. Li, D. Jin, Q. Yu, and M. Qi. Ib-transunet: Combining information bottleneck and transformer for medical image segmentation. *Journal of King Saud University-Computer and Information Sciences*, 35(3):249–258, 2023.

[91] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020.

[92] P. Lippe, B. Veeling, P. Perdikaris, R. Turner, and J. Brandstetter. Pde-refiner: Achieving accurate long rollouts with neural pde solvers. *Advances in Neural Information Processing Systems*, 36, 2024.

[93] S. Liu, O. Bousquet, and K. Chaudhuri. Approximation and convergence properties of generative adversarial learning. *arXiv preprint arXiv:1705.08991*, 2017.

[94] A. Logg, K.-A. Mardal, and G. Wells. *Automated solution of differential equations by the finite element method: The FEniCS book*, volume 84. Springer Science & Business Media, 2012.

[95] I. Loshchilov and F. Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

[96] F. Lu, M. Morzfeld, X. Tu, and A. J. Chorin. Limitations of polynomial chaos expansions in the bayesian solution of inverse problems. *Journal of Computational Physics*, 282:138–147, 2015.

[97] S. Lunz, O. Öktem, and C.-B. Schönlieb. Adversarial regularizers in inverse problems. *Advances in Neural Information Processing Systems*, 31, 2018.

[98] C. J. Maddison, J. Lawson, G. Tucker, N. Heess, M. Norouzi, A. Mnih, A. Doucet, and Y. Teh. Filtering variational objectives. *Advances in Neural Information Processing Systems*, 30, 2017.

[99] G. Maier, G. Bolzon, V. Buljak, T. Garbowski, B. Miller, et al. Synergic combinations of computational methods and experiments for structural diagnoses. *Computer Methods in Mechanics*, 1:453–476, 2010.

[100] F. A. Maken, F. Ramos, and L. Ott. Stein particle filter for nonlinear, non-gaussian state estimation. *IEEE Robotics and Automation Letters*, 7(2):5421–5428, 2022.

[101] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey. Adversarial autoencoders. *arXiv preprint arXiv:1511.05644*, 2015.

[102] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953.

[103] E. G. Mohammed, E. B. Zeleke, and S. L. Abebe. Water leakage detection and localization using hydraulic modeling and classification. *Journal of Hydroinformatics*, 23(4):782–794, 2021.

[104] A. K. Moretti, Z. Wang, L. Wu, I. Drori, and I. Pe'er. Particle smoothing variational objectives. *arXiv preprint arXiv:1909.09734*, 2019.

[105] N. T. Mücke, S. M. Bohté, and C. W. Oosterlee. Reduced order modeling for parameterized time-dependent pdes using spatially and memory aware deep learning. *Journal of Computational Science*, page 101408, 2021.

[106] N. T. Mücke, P. Pandey, S. Jain, S. M. Bohté, and C. W. Oosterlee. A probabilistic digital twin for leak localization in water distribution networks using generative deep learning. *Sensors*, 23(13):6179, 2023.

[107] N. T. Mücke, B. Sanderse, S. M. Bohté, and C. W. Oosterlee. Markov chain generative adversarial neural networks for solving bayesian inverse problems in physics applications. *Computers & Mathematics w. Applications*, 147:278–299, 2023.

[108] N. T. Mücke, L. H. Christiansen, A. P. Engsig-Karup, and J. B. Jørgensen. Reduced order modeling for nonlinear pde-constrained optimization using neural networks. In *2019 IEEE 58th Conference on Decision and Control (CDC)*, pages 4267–4272. IEEE, 2019.

[109] C. Omgba-Essama. *Numerical modelling of transient gas-liquid flows (application to stratified & slug flow regimes).* PhD thesis, Cranfield University, CERES, 2004.

[110] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[111] O. Ovadia, A. Kahana, P. Stinis, E. Turkel, and G. E. Karniadakis. Vito: Vision transformer-operator. *Preprint arXiv:2303.08891*, 2023.

[112] S. Pan and K. Duraisamy. Long-time predictive modeling of nonlinear dynamical systems using neural networks. *Complexity*, 2018, 2018.

[113] V. M. Panaretos and Y. Zemel. *An invitation to statistics in Wasserstein space.* Springer Nature, 2020.

[114] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. 2017.

[115] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

[116] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32, 2019.

[117] D. Patel and A. A. Oberai. Bayesian inference with generative adversarial network priors. *arXiv preprint arXiv:1907.09987*, 2019.

[118] D. V. Patel, D. Ray, H. Ramaswamy, and A. Oberai. Bayesian inference in physics-driven problems with adversarial priors. In *NeurIPS 2020 Workshop on Deep Learning and Inverse Problems*, 2020.

[119] S. Pawar, S. Rahman, H. Vaddireddy, O. San, A. Rasheed, and P. Vedula. A deep learning enabler for nonintrusive reduced order modeling of fluid flows. *Physics of Fluids*, 31(8):085101, 2019.

[120] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[121] P. Petersen and F. Voigtlaender. Equivalence of approximation by convolutional neural networks and fully-connected networks. *Proceedings of the American Mathematical Society*, 148(4):1567–1581, 2020.

[122] A. Quarteroni and S. Quarteroni. *Numerical models for differential problems*, volume 2. Springer, 2009.

[123] A. Quarteroni, A. Manzoni, and F. Negri. *Reduced basis methods for partial differential equations: an introduction*, volume 92. Springer, 2015.

[124] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations. *arXiv preprint arXiv:1711.10561*, 2017.

[125] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

[126] R. Ran, T. Gao, and B. Fang. Transformer-based dimensionality reduction. *Preprint arXiv:2210.08288*, 2022.

[127] A. Rasheed, O. San, and T. Kvamsdal. Digital twin: Values, challenges and enablers from a modeling perspective. *Ieee Access*, 8:21980–22012, 2020.

[128] S. Reich and C. Cotter. *Probabilistic forecasting and Bayesian data assimilation.* Cambridge University Press, 2015.

[129] D. Rezende and S. Mohamed. Variational inference with normalizing flows. In *International Conference on Machine Learning*, pages 1530–1538. PMLR, 2015.

[130] L. Romero, J. Blesa, V. Puig, G. Cembrano, and C. Trapiello. First results in leak localization in water distribution networks using graph-based clustering and deep learning. *IFAC-PapersOnLine*, 53(2):16691–16696, 2020.

[131] L. A. Rossman et al. Epanet 2: users manual. 2000.

[132] S. Ruchi, S. Dubinkina, and M. Iglesias. Transform-based particle filtering for elliptic bayesian inverse problems. *Inverse Problems*, 35(11):115005, 2019.

[133] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[134] L. Ruthotto and E. Haber. Deep neural networks motivated by partial differential equations. *Journal of Mathematical Imaging and Vision*, pages 1–13, 2019.

[135] L. Ruthotto and E. Haber. An introduction to deep generative modeling. *GAMM-Mitteilungen*, page e202100008, 2021.

[136] Y. Saad. *Iterative methods for sparse linear systems*, volume 82. SIAM, 2003.

[137] B. Sanderse, V. V. Dighe, K. Boorsma, and G. Schepers. Efficient bayesian calibration of aerodynamic wind turbine models using surrogate modeling. *Wind Energy Science Discussions*, pages 1–34, 2021.

[138] J. A. Schetz and A. E. Fuhs. *Handbook of fluid dynamics and fluid machinery*, volume 1. Wiley New York, 1996.

[139] G. Seabra, N. Mücke, V. Silva, D. Voskov, and F. Vossepoel. Ai enhanced data assimilation and uncertainty quantification applied to geological carbon storage. *arXiv preprint arXiv:2402.06110*, 2024.

[140] M. Shekofteh, M. Jalili Ghazizadeh, and J. Yazdi. A methodology for leak detection in water distribution networks using graph theory and artificial neural network. *Urban Water Journal*, 17(6):525–533, 2020.

[141] J. Shen, T. Marwah, and A. Talwalkar. Ups: Towards foundation models for pde solving via cross-modal adaptation. *arXiv preprint arXiv:2403.07187*, 2024.

[142] V. L. S. Silva, C. E. Heaney, and C. C. Pain. Generative network-based reduced-order model for prediction, data assimilation and uncertainty quantification. In *LatinX in AI Workshop at ICML 2023 (Regular Deadline)*, 2023.

[143] A. Simpson and S. Elhay. Jacobian matrix for solving water distribution system equations with the darcy-weisbach head-loss model. 2011.

[144] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, pages 2256–2265. PMLR, 2015.

[145] B. Sprungk. On the local lipschitz stability of bayesian inverse problems. *Inverse Problems*, 36(5):055015, 2020.

[146] A. Stuart and A. Teckentrup. Posterior consistency for gaussian process approximations of bayesian posterior distributions. *Mathematics of Computation*, 87(310):721–753, 2018.

[147] A. M. Stuart. Inverse problems: a bayesian perspective. *Acta Numerica*, 19:451–559, 2010.

[148] M. Sugiyama. *Statistical reinforcement learning: modern machine learning approaches.* CRC Press, 2015.

[149] C. Sun, B. Parellada, V. Puig, and G. Cembrano. Leak localization in water distribution networks using pressure and data-driven classifier approach. *Water*, 12(1):54, 2019.

[150] R. Swischuk, L. Mainini, B. Peherstorfer, and K. Willcox. Projection-based model reduction: Formulations for physics-based machine learning. *Computers & Fluids*, 179: 704–717, 2019.

[151] I. Tolstikhin, O. Bousquet, S. Gelly, and B. Schoelkopf. Wasserstein auto-encoders. *arXiv preprint arXiv:1711.01558*, 2017.

[152] U. Trottenberg, C. W. Oosterlee, and A. Schuller. *Multigrid.* Elsevier, 2000.

[153] F. E. Uilhoorn. A particle filter-based framework for real-time state estimation of a non-linear hyperbolic pde system describing transient flows in co2 pipelines. *Computers & Mathematics w. Applications*, 68(12):1991–2004, 2014.

[154] T. van Gastelen, W. Edeling, and B. Sanderse. Energy-conserving neural network for turbulence closure modeling. *arXiv preprint arXiv:2301.13770*, 2023.

[155] G. van Lagen, E. Abraham, and P. M. Esfahani. A bayesian approach for active fault isolation with an application to leakage localization in water distribution networks. *IEEE Transactions on Control Systems Technology*, 2022.

[156] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 30, 2017.

[157] R. Vidal, Y. Ma, and S. Sastry. Generalized principal component analysis (gpca). *IEEE transactions on pattern analysis and machine intelligence*, 27(12):1945–1959, 2005.

[158] Z. Y. Wan, L. Zepeda-Núñez, A. Boral, and F. Sha. Evolve smoothly, fit consistently: Learning smooth latent dynamics for advection-dominated systems. *Preprint arXiv:2301.10391*, 2023.

[159] H. Wang and J. Li. Adaptive gaussian process approximation for bayesian inference with expensive likelihood functions. *Neural Computation*, 30(11):3072–3094, 2018.

[160] Q. Wang, J. S. Hesthaven, and D. Ray. Non-intrusive reduced order modeling of unsteady flows using artificial neural networks with application to a combustion problem. *Journal of Computational Physics*, 384:289–307, 2019.

[161] J. Whang, E. Lindgren, and A. Dimakis. Composing normalizing flows for inverse problems. In *International Conference on Machine Learning*, pages 11158–11169. PMLR, 2021.

[162] Y. Xia and N. Zabaras. Bayesian multiscale deep generative model for the solution of high-dimensional inverse problems. *Journal of Computational Physics*, 455:111008, 2022.

[163] Z. Xiao, K. Kreis, and A. Vahdat. Tackling the generative learning trilemma with denoising diffusion gans. *arXiv preprint arXiv:2112.07804*, 2021.

[164] D. Xiu. *Numerical methods for stochastic computations.* Princeton University Press, 2010.

[165] J. Xu and K. Duraisamy. Multi-level convolutional autoencoder networks for parametric prediction of spatio-temporal dynamics. *arXiv preprint arXiv:1912.11114*, 2019.

[166] J. Xu and K. Duraisamy. Multi-level convolutional autoencoder networks for parametric prediction of spatio-temporal dynamics. *Computer Methods in Applied Mechanics and Engineering*, 372:113379, 2020.

[167] Y. Yang, J. A. Stork, and T. Stoyanov. Particle filters in latent space for robust deformable linear object tracking. *IEEE Robotics and Automation Letters*, 7(4):12577–12584, 2022.

[168] S. Yu and J. C. Principe. Understanding autoencoders with information theoretic concepts. *Neural Networks*, 117:104–123, 2019.

[169] V. E. Zakharov. Stability of periodic waves of finite amplitude on the surface of a deep fluid. *Journal of Applied Mechanics and Technical Physics*, 9(2):190–194, 1968.

CURRICULUM VITAE

Nikolaj Takata Mücke

18-03-1993	Born in Greve, Denmark

Education

2019-2024	PhD in Applied Mathematics Centrum Wiskunde & Informatica and Utrecht University, The Netherlands **Thesis:** *Deep Learning for Real-Time Inverse Problems and Data Assimilation with Uncertainty Quantification for Digital Twins* **Promotors:** Prof. dr. S. M. Bohté and Prof. dr. ir. C. W. Oosterlee
2016-2018	Master of Science in Mathematical Modeling and Computation Technical University of Denmark, Denmark
2013-2016	Bachelor of Science in Mathematics and Technology Technical University of Denmark, Denmark

LIST OF PUBLICATIONS

1. **Mücke, Nikolaj T.**, Sander M. Bohté, and Cornelis W. Oosterlee. "Reduced order modeling for parameterized time-dependent PDEs using spatially and memory aware deep learning." Journal of Computational Science 53 (2021): 101408..
2. **Mücke, Nikolaj T.**, et al. "Markov chain generative adversarial neural networks for solving Bayesian inverse problems in physics applications." Computers & Mathematics with Applications 147 (2023): 278-299.
3. **Mücke, Nikolaj T.**, et al. "A Probabilistic Digital Twin for Leak Localization in Water Distribution Networks Using Generative Deep Learning." Sensors 23.13 (2023): 6179.
4. **Mücke, Nikolaj T.**, Sander M. Bohté, and Cornelis W. Oosterlee. "The Deep Latent Space Particle Filter for Real-Time Nonlinear Data Assimilation with Uncertainty Quantification." **(Submitted for publication)**

LIST OF PRESENTATIONS

Oral presentations

Conferences

1. **ECMI Conference on Industrial and Applied Mathematics**, Wuppertal, Germany, 2021.
2. **Mechanistic Machine Learning and Digital Twins for Computational Science, Engineering & Technology (MMLDT)**, San Diego, USA, 2021
3. **DTU Compute SciML Lunch Seminar**, Lyngby, Denmark, 2022
4. **SIAM Uncertainty Quantification**, Atlanta, USA, 2022
5. **Computational Fluids Conference (CFC)**, Cannes, France, 2023
6. **SIAM Computational Science and Engineering**, Amsterdam, The Netherlands, 2023

Workshops

1. **AI and IoT for Flow Modeling**, Amsterdam, The Netherlands, 2020
2. **Workshop on Machine Learning for Physics-Based Modeling**, Amsterdam, The Netherlands, 2021
3. **Computational Imaging Masterclass**, Amsterdam, The Netherlands, 2022
4. **Digital Twins for Pipe Transport Networks**, Amsterdam, The Netherlands, 2023

Poster presentations

1. **Dutch-Flemish Scientific Computing Society**, Woudschoten, The Netherlands, 2021
2. **SIAM Mathematics of Data Science**, San Diego, USA, 2022

ACKNOWLEDGEMENTS

First of all I would like to thank my two supervisors, Kees and Sander. You were a great help and support throughout the entire process. Without all the weekly discussions, the work would not have turned out as great as it did. It would also have been impossible for me to go through the process without your guidance on the intricacies of academic research and publishing.

Kees, you were an amazing help in structuring my work and made sure I stayed on track. And without your careful readings and corrections, my (often subpar) writing would not have been elevated to the next level.

Sander, your help in staying up to date with the latest developments in machine learning was invaluable and challenged me to constantly try out new stuff. I also really appreciate all the conversations and advise you provided when I spontaneously dropped by your office.

I would also like to thank all the great colleagues at CWI for nice discussions and input. Especially, my two office mates, Kristoffer and Robin, deserve an additional gratitude for keeping me company in the office.

Hema, Robin, and Jurriaan, you were great company during the slightly less optimal conditions of the COVID lockdown. At the otherwise empty CWI, it was great to have a group of people to have lunch with.

Benjamin also deserves a special thanks. While not being an official supervisor, you often took on that role and helped me with several problems regarding fluid dynamics, mathematics, boundary conditions, presentations, and writing.

Besides the academic colleagues, one of the most important people was, of course, Nada. Without your help, it would have been impossible to get my money reimbursed, organize workshops, and in general getting things done.

Unfortunately, I didn't end up doing as much traveling to Bangalore as originally planned, but I definitely enjoyed the two trips I did have. Thank you, Shashi and Prerna, for being exceptional hosts and thank you for the fruitful collaboration.

Lastly, I want to thank my amazing girlfriend, Kyra. Even though you didn't provide much help with the mathematics, you helped with everything else. I am so happy we met while I was doing my PhD. I Love you!