Simulation-Efficient Structural Health Monitoring via Graph Neural Networks and Amortized Bayesian Inference

Master of Science Thesis

MSc Systems & Control
Tom de Jonge - 4905350

TU Delft

Delft Center for Systems & Control

Simulation-Efficient Structural Health Monitoring via Graph Neural Networks and Amortized Bayesian Inference

Master of Science Thesis

For the degree of Master of Science in Systems and Control at
Delft University of Technology

Tom de Jonge

August 17, 2025

University Supervisor I:	Dr. Sc. Mohammad Khosravi
University Supervisor II:	Dr. Dimitris Boskos
Industry Supervisor I:	Andrés Martínez Colán
Industry Supervisor II:	Christos Lathourakis
Project Duration:	11, 2024 – 08, 2025
Faculty:	Mechanical Engineering, Delft
Cover Image:	K. Soma (Modified)

TU Delft TNO

Summary

Structural health monitoring (SHM) aims to ensure the reliable operation of structures through the use of sensors and data analysis. A commonly used approach is to frame the detection of damage as an inverse problem, where the goal is to estimate uncertain physical parameters, including damage indicators, from measured structural responses. Bayesian inference offers an attractive framework for solving such inverse problems, as it not only enables damage detection, but also provides a measure of uncertainty. The core concept of Bayesian inference is to update a prior belief of the parameters of interest using observational data, yielding a posterior distribution that encodes both the most likely parameter values and the uncertainty surrounding them. However, a challenge in applying Bayesian inference is the need for a likelihood function, which is often infeasible to define in practice, as exact forward models are unavailable or intractable.

To circumvent this issue, likelihood-free inference (LFI) methods have emerged. These approaches bypass the need for an explicit likelihood by relying solely on the ability to generate simulated data from a forward model. One such method is BayesFlow, which is deep learning-based and amortized. Theoretically, BayesFlow is a suitable method to detect damage in structures, including uncertainty quantification. However, large amounts of simulation training data are required to properly train BayesFlow models, which is often expensive to generate. This bottleneck currently hinders the practical application of BayesFlow to real-world SHM.

This work presents a method to address the problematic simulation data requirements. The summary network component of BayesFlow, which converts raw data into informative summary statistics, is identified as an area with potential for enhancement. A physics-informed approach is proposed, in which the summary network is enriched by integrating knowledge about the physical structure. Specifically, a Graph Summary Neural Network (GSNN) is designed, leveraging the sequential nature of sensor measurements and the spatial topology of the structure.

To highlight the effectiveness of the developed method, three case studies of increasing complexity were considered. First, the detection of a single damaged component in a 2D scenario was tested. Subsequently, a more difficult 2D scenario was tested, in which twice the number of elements were eligible to be damaged. Finally, a complex 3D scenario was investigated. In the simplest 2D test case, the GSNN is able to reduce the number of simulations required to reach the bound on model performance by 75%. It is noteworthy that in this case, all models eventually reached a similar performance bound. In the more complex 2D case, the GSNN not only reduced the required number of simulations by 37.5%, but also achieved overall improved posterior estimation. The GSNN reached a model performance score of 0.3918, compared to baseline models, which were only able to achieve model performance scores ranging from 0.5149 to 0.8705. Note that a lower model performance score indicates a better performing model. In the 3D scenario, the advantage of the GSNN over baseline networks in terms of model performance is further highlighted, as a performance score of 0.467 was reached, compared to baseline models, which were only able to achieve performance scores ranging from 3.077 to 1.748. These results indicate that the GSNN surpasses baseline networks in terms of both efficiency *and* performance, thus bringing BayesFlow-based SHM closer to real-world deployment.

Table of Contents

List of Figures

List of Tables

Nomenclature

List of Abbreviations

Abbreviation	Definition
ABC	Approximate Bayesian Computation
CNN	Convolutional Neural Network
CRPS	Continuous Ranked Probability Score
cINN	Conditional Invertible Neural Network
FEM	Finite Element Model
GCN	Graph Convolutional Network
GNN	Graph Neural Network
NN	Neural Network
PCA	Principal Component Analysis
PIML	Physics-Informed Machine Learning
ReLU	Rectified Linear Unit
SHM	Structural Health Monitoring

List of Symbols

Symbol	Definition
A	Adjacency matrix of a graph
D	Degree matrix of a graph
f_ϕ	Inference network
h_ψ	Summary network
J_{f_ϕ}	Jacobian of inference network
$p^*(\cdot)$	Ground truth probability density function
$p(\cdot)$	Approximate probability density function
x	Simulated structural response
x_{obs}	Observed structural response
$\tilde{x}$	Summary statistic of a simulated structural response
$\tilde{x}_{\text{obs}}$	Summary statistic of an observed structural response
z	Latent variable
η	Sufficient summary statistic
θ	Parameter vector
$\hat{\theta}$	Estimated parameter vector
ω	Damage parameter
ϕ	Weights of the inference network
ψ	Weights of the summary network
$\mathcal{F}$	Forward model
$\mathcal{G}$	Graph
$\mathcal{L}$	Loss function
$\mathcal{N}$	Neighborhood of a graph node
$\sigma(\cdot)$	Non-linear activation function

Preface

I would like to thank Andrés Martínez Colán and Christos Lathourakis for their continued support, valuable feedback, friendly coffee chats, and the opportunity to collaborate with TNO. I would like to thank Mohammad Khosravi and Dimitris Boskos for their guidance and supervision throughout the thesis. Finally, I would like to thank Anne van Weezenbeek for her unwavering encouragement, and for always providing a fresh perspective when I needed it most.

Tom de Jonge
Delft, August 2025

1 Introduction

1.1. Structural Health Monitoring

Structures are physical systems designed to withstand loads and provide stability. They are found in diverse sectors, across a wide range of applications. A few examples include civil infrastructure, where structures are used in bridges and buildings; aerospace, where structures can be found in airplanes and spacecrafts; and energy systems, where structures are used in wind turbines and dams. Regardless of their specific form or function, all structures deteriorate over time due to operational stressors and environmental factors. Fatigue from repetitive loads is a common issue that impacts many older structures [1]. Among environmental influences, corrosion can compromise the structural health [2], while large-scale natural disasters like earthquakes and hurricanes can also lead to damage [3]. These stressors can degrade the construction materials, giving rise to damages that can compromise the integrity of the structure [4, 1, 2, 3].

Furthermore, these stressors are often unpredictable and difficult to model [5]. Their nondeterministic nature makes it challenging to estimate the condition of a structure based on its age and usage alone, meaning that the actual structural state may deviate from the predicted structural state. These deviations between predictions and reality can have large implications. Misjudging structural integrity can lead to premature decommissioning or unexpected failures, which can result in severe environmental, financial, and human safety consequences [4, 6, 7]. For example, in 2018, the Morandi bridge in Genoa collapsed, killing 43 people and leaving many hundreds homeless. The structural integrity of the bridge's suspension cables was difficult to estimate, and it was unclear how the cables were behaving under traffic loads. Despite some visible signs of deterioration, the true extent of the internal damage was underestimated, leading to an unexpected failure [8]. If the structure had been monitored more accurately, damage may have been localized in time, and preventative maintenance action could have been taken.

It is clear that the ability to properly evaluate the integrity of structures holds the potential to offer significant economic and life-safety benefits. However, assessing the condition of a structure is a complex task, requiring high levels of specialization [4, 9]. Furthermore, as structures can be found in such a diverse range of sectors, it is difficult to develop methods that are readily applicable to different types of structures. Traditionally, the assessment of structures primarily relies on visual inspections [4, 10]. However, visual inspections are subjective, inconsistent and inaccurate [11]. Developing new methods that make use of modern sensor technologies and smart algorithms could significantly improve the quality of the structural health assessments.

The concept of automating damage detection through sensor-based systems that collect data from the structure under study is referred to as Structural Health Monitoring (SHM) [12, 13, 14]. SHM can be used in five stages to detect, locate, classify, assess, and even predict future

Figure 1.1: Collapsed Morandi bridge, Genua, Italy [19]

damages [15, 16]. These technologies have the potential to provide accurate information about a structure's state whenever necessary, without interrupting its operation (e.g., without closing a bridge to traffic) [4]. A typical SHM strategy is comprised of several key components. Sensors must be carefully chosen, placed in optimized positions, and properly set up to ensure effective monitoring. The collected data must be processed, transforming raw measurements into a form suitable for analysis. Finally, the probabilistic analysis used to interpret the data has to be accurate in describing the structural state of the object [12, 14, 17]. As each of these steps is complex in itself, developing a cohesive SHM strategy requires a well thought-out approach.

Although recent advancements have been made in the field of SHM, the application of the technology to real structures is still in its infancy [4, 10]. Most existing real-life SHM systems are still in the experimental phase. The widespread deployment of SHM is currently obstructed by high equipment and maintenance costs, the complexity of sensor management and data analysis, and uncertainties in interpreting results [4, 18]. Additionally, the lack of standardized frameworks and the variability in structural types make it difficult to generalize SHM strategies across different objects [4]. The full potential of SHM has yet to be realized, motivating the relevance and value of this thesis. In the next section, a well-established framework for damage detection is introduced, which forms the basis for the SHM method developed in this work.

1.2. Inverse Methods for Structural Health Monitoring

A widely adopted SHM framework is to formulate damage detection as an inverse problem [9, 20]. This approach relies on a forward model, which maps a vector of physical parameters to a structural response, i.e.

$$x = \mathcal{F}(\theta), \tag{1.1}$$

where x denotes the structural response computed by the forward model of the structure $\mathcal{F}$, and θ denotes the parameter vector. To incorporate and reflect the effects of damage in the structural behavior, we can include *damage parameters*, denoted as ω, which act as proxies for reductions in stiffness, strength, or other mechanical properties of the affected components. Each damage parameter reflects the structural health of a specific element of the structure, indexed by i. The set of structural elements that are linked to damage parameters is denoted as $\mathcal{B}$.

Damage detection in the elements of $\mathcal{B}$ can then be expressed as an inverse problem: given measurements from the real structure, x_{obs}, the goal is to determine the parameter vector θ such that the forward model's predicted response matches the observed response as closely as possible. Since θ contains the damage parameters of elements in $\mathcal{B}$, the estimated values directly indicate their structural condition. A substantial increase in the inferred damage parameter for an element can be interpreted as evidence of damage in that element.

However, computing solutions to inverse problems is challenging, mainly due to inaccuracies in the forward model and measurement noise caused by imperfect sensors [9, 21]. Modeling inaccuracies are unavoidable, as modeling complex substructures, such as joints, is difficult and can only be done through simplifications. Furthermore, modeling structural damage is also challenging, as it involves defining how variations in the damage parameters translate into changes in the structure's mechanical behaviour. For example, damage modeling can range from a simple stiffness reduction at the affected location to advanced, highly detailed crack models. In addition to the challenges posed by noise and modeling issues, inverse problems are often inherently ill-posed, meaning solutions may not exist, may be non-unique, or may be highly sensitive to small perturbations in the input data [21].

1.3. Solving Inverse Problems with Bayesian Inference

A popular framework for solving inverse problems is Bayesian inference [9]. In this approach, both prior knowledge and measurement data are used to update beliefs about uncertain parameters. This is reflected by Bayes' Theorem

$$p(\theta|x_{\text{obs}}) = \frac{p(x_{\text{obs}}|\theta)\, p(\theta)}{p(x_{\text{obs}})}, \tag{1.2}$$

where $p(\theta|x_{\text{obs}})$ is the posterior distribution after observing x_{obs}, $p(x_{\text{obs}}|\theta)$ is the likelihood function, $p(\theta)$ is the prior distribution and $p(x_{\text{obs}})$ is the evidence, a normalizing constant that ensures the posterior is a valid probability distribution.

Applying this to the context of SHM, the inverse problem for damage detection described in section 1.2 can be solved using Bayesian inference. A prior belief of the parameter vector is created, incorporating expert knowledge, heuristics, and uncertainty. Taking a sample of the parameter vector from the prior belief allows for simulation of the structural response, i.e.

$$x = \mathcal{F}(\theta), \quad \theta \sim p(\theta), \tag{1.3}$$

where $p(\theta)$ represents the prior belief of the parameter vector. Given measurement data from the real structure, denoted as x_{obs}, the inverse problem of estimating the damage parameters can then be solved using (1.2). This yields a posterior distribution, indicating the updated belief about the most plausible values of the damage parameters given the observed structural response. The process of using an inverse problem with Bayesian inference for damage detection is depicted schematically in Figure 1.2.

One of the main reasons Bayesian inference is appealing in this context is that it produces a complete probability distribution over the unknown parameters rather than a single point estimate. This enables uncertainty quantification, which is critical in SHM, where measurements are noisy and damage signatures can be subtle. By combining both measurement data and

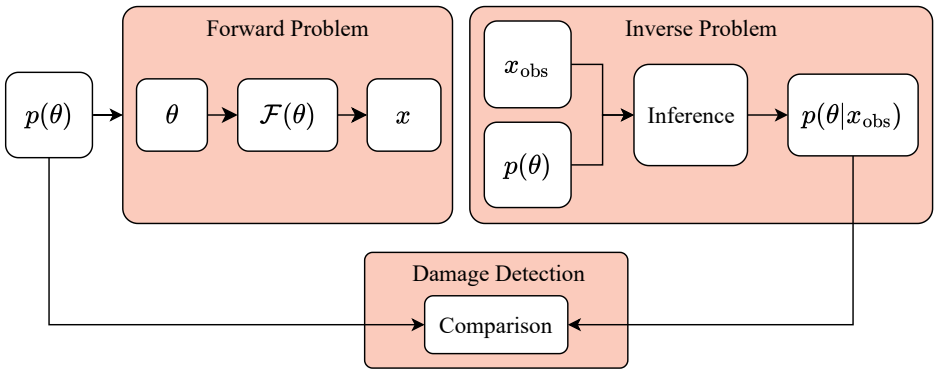


Figure 1.2: Using the inverse problem with Bayesian inference as a framework for SHM. The posterior distribution is compared to the prior distribution. Any significant changes in the damage parameters can be interpreted as damages.

a prior that encodes existing engineering knowledge, the resulting posterior integrates all available information into a coherent and interpretable form. In cases where multiple parameter configurations are consistent with the observations, the posterior can represent the ambiguity directly, avoiding the false certainty of single-solution approaches.

A practical challenge with this approach is that it requires a well-specified likelihood function. The likelihood arises from the data generating process; however, in many applications, this is difficult to quantify, as exact models are not available or computationally expensive [9, 22, 23]. Even when a model is available, it may only be accessible as a complex simulator that does not provide a closed-form expression for the likelihood. Furthermore, using Bayesian inference for SHM would require reusing Bayes' Theorem to calculate a new posterior distribution for each new measurement, resulting in a large number of expensive calculations over time [24].

To mitigate the issues related to the definition of a likelihood function, a special class of Bayesian inference methods has been developed that are likelihood-free. Likelihood-free inference (LFI) refers to a class of methods specifically designed to perform statistical inference using simulators, eliminating the need for an explicit likelihood function. One of the most well-established LFI methods is Approximate Bayesian Computation (ABC) [25, 26], which approximates the likelihood by generating simulations from a model and comparing the simulated data to the observed data [26]. Although this method circumvents the need for an explicit likelihood function, it remains computationally inefficient, particularly due to its low acceptance rates and the large number of required simulations. Furthermore, the fact that a computationally expensive operation based on Bayes' Theorem must be repeated for each new measurement remains an issue [25].

1.4. Amortized & Likelihood-Free Inference with BayesFlow

A novel method that is able to overcome both the issue related to the likelihood function and the issue related to the repeated application of Bayes' Theorem is BayesFlow. This method leverages deep learning to perform amortized Bayesian inference. The amortized nature of BayesFlow means that once the machine learning model is trained, it is able to generate approximate posterior distributions for new measurements almost instantaneously [27, 28, 29]. The working mechanism of the method is discussed in section 2.1.

Although the use of machine learning for SHM is not new, most existing approaches are not able to provide a reliable measure of uncertainty along with their damage detection. This is what sets Bayesian inference-based methods apart. BayesFlow is particularly promising within

the domain of machine learning-based solutions for SHM due to its reliance on a simulator rather than a real-world dataset. Traditionally, supervised machine learning models require labeled data across numerous scenarios to be able to properly generalize. In the context of SHM, this would necessitate intentionally damaging real structures, which is impractical and often infeasible. This issue can be circumvented using unsupervised machine learning (for example to detect anomalies and therefore damages), or by using a simulator, as is the case in BayesFlow.

BayesFlow has already been shown to have potential for SHM in existing studies. Previously, the method was validated by detecting damage in a synthetic, two-dimensional bridge. A posterior distribution of the model parameters was generated based on synthetic sensor data, allowing for the successful identification of damage [9]. However, there is still a large obstacle that must be overcome to facilitate the feasibility of BayesFlow for SHM to *real-world* structures.

1.5. Challenges of Applying BayesFlow to Structural Health Monitoring

The primary obstacle preventing the implementation of BayesFlow in real-world SHM applications lies in its training simulation data requirements. For the machine learning model to effectively detect damage, it must be trained on simulations representing a wide range of damaged states. Without such diversity in the training set, the model may fail to generalize, resulting in poor performance when applied to new, unseen cases [24, 30, 31].

This leads to the current bottleneck: the computational cost of generating a large training set of high-fidelity simulations. Realistic Finite Element Models (FEMs) are generally used for the analysis of high-tech engineering structures. These FEMs are highly detailed and complex, involving a large number of degrees of freedom. This significantly increases the number of unknowns to solve for, which increases the size of the matrices that need to be processed, and therefore also the associated computational burden. Moreover, highly detailed simulators may require iterative or nonlinear solvers, further compounding the computational cost [32, 33].

Implementing BayesFlow in real-life settings would require some of the simplifying assumptions made in previous research to be revisited. As previous work [9] was mainly meant as a proof of concept to demonstrate the applicability of BayesFlow to SHM as described in section 1.2, the research focused on a synthetic model, in which damage was restricted to a small subset of elements, and only one element was assumed to be damaged at a time. However, in practice, these assumptions do not hold. Damage can occur in multiple elements simultaneously and is not confined to predefined locations. Relaxing these assumptions increases both the number of parameters and the dimensionality of the parameter space. As a result, the number of simulations required to adequately cover this space grows exponentially, while each individual simulation remains expensive. This implies that each simulation is not only more costly, but we also require a greater number of simulations in total.

The combination of high simulation cost and the need for a large, diverse training dataset presents a fundamental challenge to the practical deployment of BayesFlow for SHM. Overcoming this issue is crucial for transitioning from synthetic benchmarks to real-world applications of BayesFlow in SHM.

1.6. Potential for Efficient Learning with a Physics-Informed Summary Network

A potential strategy to mitigate the need for large and computationally expensive datasets is to maximize the information extracted from each individual simulation. In the BayesFlow architecture, this task is performed by the summary network, which compresses high-dimensional observational data into low-dimensional summary statistics. These summary statistics are then

used by the inference network to approximate the posterior distribution. While compressing the raw input data, the summary network ensures that specifically the information that is most relevant for inferring the model parameters further on in the pipeline is retained [27, 29]. The quality of the summary network therefore directly affects the efficiency and accuracy of the entire inference process [24, 34]. More details regarding the use of summary statistics in machine learning and Bayesian inference processes can be found in Section 2.3.

The proposed strategy in this work is to improve upon the standard summary network by designing a summary network that reflects the underlying physics and structure of the problem domain. This aligns with the principles of Physics-Informed Machine Learning (PIML), a field of machine learning which focuses on integrating domain knowledge into learning algorithms. Physics is incorporated through biases in the model, as well as by strategically choosing the training data and by influencing the training process itself. This is discussed in more detail in section 2.4. Embedding physical priors helps machine learning models focus on meaningful features, reduce overfitting, and improve generalization to unseen damage scenarios [34]. By leveraging a physics-informed design, the aim is to ensure the summary network extracts more useful information from fewer simulations, thereby mitigating the bottleneck of data generation.

In the context of structures, prior knowledge of the topology and data format of the sensor network can be encoded into the summary network. Accordingly, the Graph Neural Network (GNN) is identified as an architecture for the summary network that can naturally exploit the spatial relationships between sensors and the structural connectivity. This concept is described in more detail in Section 3.2.

1.7. Motivation & Research Question

The previous sections have established a clear motivation for advancing SHM. The limitations of traditional inspection methods have been illustrated, and the need for reliable monitoring systems has been emphasized. BayesFlow is presented as a state-of-the-art solution, offering potential for rapid and uncertainty-quantified damage assessments. However, there is a practical bottleneck that currently limits its use: the high computational cost associated with the generation of a large and diverse set of training simulations.

To address this issue, a strategy for tailoring the summary network to the structure and the context of the available data has been proposed. Such a network could enable likelihood-free, amortized Bayesian inference, including uncertainty quantification, in real-world SHM settings. This directly leads to the research gap addressed in this work: the design of physics-informed summary networks for SHM with BayesFlow remains largely unexplored. By extension, the additional gain of incorporating physics knowledge into the summary network has not yet been quantified. This thesis aims to fill these research gaps, guided by the research question:

To what extent can the simulation data requirements for BayesFlow-based structural health monitoring be reduced by using a Physics-Informed Summary Network tailored specifically to structural health monitoring data?

1.8. Thesis Outline

The thesis begins by presenting background information, which is necessary to justify the choices made during the design process of the physics-informed summary network. Next, the proposed physics-informed summary network for BayesFlow-based SHM is discussed in detail. Additionally, the developed summary network is used to perform damage detection with BayesFlow in a several numerical experiments, and its performance is compared to that of several baseline summary networks. Based on the results of the test case, conclusions are drawn and recommendations for future work are provided.

2

Background Information

In this chapter, the concepts necessary to understand the development of a physics-informed summary network, specifically tailored for BayesFlow-based SHM, are explained. This includes an in-depth explanation of BayesFlow and its working principles, an illustration of the importance of summary statistics in Bayesian inference and machine learning, and a discussion of the PIML techniques relevant to this work.

2.1. Working Principles of BayesFlow

We begin by describing the general working principle of BayesFlow. The algorithm consists of a training and an inference phase [27, 28, 29], depicted in Figure 2.1.

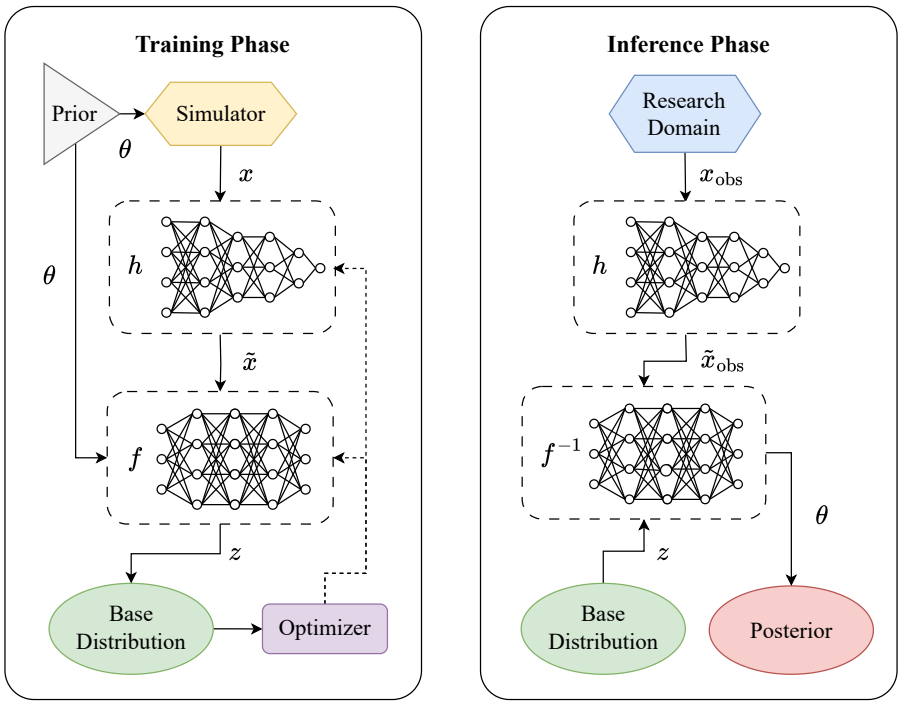


Figure 2.1: Training and Inference Phases of BayesFlow, adapted from [29].

2.1.1. Training Phase

During the training phase, a summary neural network and an invertible inference neural network are jointly optimized. Training data is generated by the simulator, which draws samples of θ from the prior, and passes them to the forward model, resulting in a simulated response

$$x = \mathcal{F}(\theta), \quad \theta \sim p(\theta), \tag{2.1}$$

where x denotes the simulated response, $\mathcal{F}$ denotes the forward model, and θ denotes the parameter vector. Each simulation is then passed through the summary neural network, producing a summary vector

$$\tilde{x} = h_\psi(x), \tag{2.2}$$

where $\tilde{x}$ denotes the summary vector, and h_ψ denotes the summary network, which is characterized by weights ψ. Next, the sample θ is passed to the invertible inference network together with its respective summary vector. The inference network maps the parameter vector to a latent variable, conditioned on the summary vector, i.e.

$$z = f_\phi(\theta; \tilde{x}), \tag{2.3}$$

where z is the latent variable, and f_ϕ denotes the invertible inference network, which is characterized by weights ϕ. The final step of an iteration of the training loop is to update the weights of the summary network and the weights of the inference network to minimize the loss function. The objective of the loss function is to penalize deviations of the distribution of the latent variable z from a base distribution, which is chosen to be a multivariate Gaussian with zero mean and a diagonal covariance matrix. The loss function is discussed in more detail in subsection 2.1.4.

Each update is performed by computing the loss in a forward pass and using backpropagation. This process is repeated for each batch of simulations, using the updated weights of the neural networks in each new pass.

2.1.2. Inference Phase

Once the summary and inference networks are sufficiently trained, they can be used to generate an approximation of the posterior distribution for any new measurement obtained from the structure [29, 27]. The measured observation is passed through the (now trained and optimized) summary network, yielding the summary vector of the observed data

$$\tilde{x}_{\text{obs}} = h_\psi(x_{\text{obs}}), \tag{2.4}$$

where $\tilde{x}_{\text{obs}}$ denotes the summary vector of the measurement. Next, the invertible nature of the inference network is leveraged, generating a map from the latent variable z space to the parameter θ space, conditioned on the summary vector of the observed dataset. L samples are taken from the z space, which is the known base distribution, i.e.,

$$\{z^{(l)}\}_{l=1}^{L}, \quad z \sim \mathcal{N}(0, I) \tag{2.5}$$

Each of these samples is input into the inverted inference network, conditioned on the summary vector of the observation, generating an estimate of the parameter vector

$$\theta^{(l)} = f_\phi^{-1}(z^{(l)}; \tilde{x}_{\text{obs}}) \tag{2.6}$$

This process is repeated for all L samples, yielding a collection of estimates of the parameter vector that can be interpreted as a posterior distribution

$$\{\theta^{(l)}\}_{l=1}^{L} \sim p(\theta | x_{\text{obs}}) \tag{2.7}$$

The principle of passing samples from the base distribution to the inference network, conditioned on the summary vector of an observation, is displayed schematically in Figure 2.2.

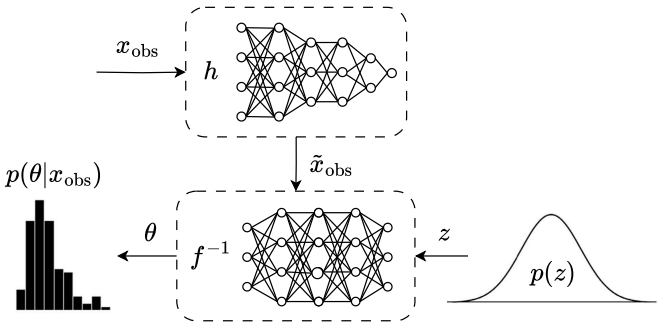


Figure 2.2: Schematic illustration of the inference phase working principle, adapted from [29].

2.1.3. The Summary and Inference Networks

In BayesFlow, the summary network serves as an encoder that transforms high-dimensional simulator outputs into compact, informative feature representations, which are subsequently used as conditioning inputs for the invertible inference network. The architecture of the summary network is domain-dependent and can incorporate biases tailored to the application [29].

The inference network is a conditional invertible neural network (cINN), which is a type of normalizing flow designed for conditional density estimation. Normalizing flows refers to a family of models that transform a simple base distribution into a complex target distribution through a sequence of invertible and differentiable mappings. Each transformation is parameterized by a neural network and designed such that both the forward mapping, which generates samples, and the inverse mapping, which evaluates densities, can be computed efficiently. This property allows the model to exploit the change-of-variables theorem, ensuring exact likelihood computation and stable training via maximum likelihood estimation.

In the context of the cINN, the transformations are conditioned on observed data, enabling the network to learn the posterior distribution of parameters given observations. Conditioning is introduced by passing the summary of the observed structural responses into each transformation block, allowing the latent variable transformations to adapt to different input conditions. The architecture of a cINN is commonly based on coupling layers or affine transformations, where only part of the input is transformed at each step while the rest remains unchanged, ensuring efficient invertibility. The unchanged part is swapped in subsequent layers so that all components of the input are eventually transformed [35, 36, 29].

2.1.4. The Loss Function

The overall goal of the BayesFlow model is to solve an inverse problem by generating an approximate posterior distribution, denoted as $p(\theta|x)$, that accurately approximates the true posterior, denoted as $p^*(\theta|x)$, for any possible observation x, i.e.

$$p(\theta|x) \approx p^*(\theta|x) \qquad \forall x \tag{2.8}$$

The difference between the true and the approximated posterior can be quantified using the Kullback-Leibler (KL) divergence. Plugging the true posterior distribution and the estimation of the posteriorinto the definition of the KL divergence yields

$$\mathbb{KL}(p^*(\theta|x)||p(\theta|x)) = \mathbb{E}_{p^*(\theta|x)}\left[\log p^*(\theta|x) - \log p(\theta|x)\right], \tag{2.9}$$

where $\mathbb{KL}$ denotes the KL divergence. However, since the estimation of the posterior distribution $p(\theta|x)$ should be accurate for all possible observations x, this metric should not just be minimized for a single x, but minimized on average across the entire distribution of possible observations.

This leads to the definition of the loss function as the expectation of the KL divergence over the marginal data distribution $p(x)$, i.e.

$$\mathcal{L} = \mathbb{E}_{p(x)} \left[\mathbb{E}_{p^*(\theta|x)} \left[\log p^*(\theta|x) - \log p(\theta|x) \right] \right] \tag{2.10}$$

First, to illustrate the core mechanism of the loss function, consider a simplified version of BayesFlow that does not use a summary network, i.e. where the raw data x is directly used for inference. The objective then becomes to find the weights of the inference network ϕ that minimize the loss function, i.e.

$$\begin{aligned} \phi^* &= \underset{\phi}{\operatorname{argmin}}\, \mathbb{E}_{p(x)} \left[\mathbb{E}_{p^*(\theta|x)} \left[\log p^*(\theta|x) - \log p_\phi(\theta|x) \right] \right] \\ &= \underset{\phi}{\operatorname{argmin}}\, \mathbb{E}_{p(x)} \left[\mathbb{E}_{p^*(\theta|x)} \left[-\log p_\phi(\theta|x) \right] \right] \\ &= \underset{\phi}{\operatorname{argmax}}\, \mathbb{E}_{p(x)} \left[\mathbb{E}_{p^*(\theta|x)} \left[\log p_\phi(\theta|x) \right] \right] \end{aligned} \tag{2.11}$$

Note that the term $\log p^*(\theta|x)$ does not depend on ϕ and can thus be omitted during optimization. Next, the nested expectation is written as an integral using the definition of the expectation of a continuous random variable, i.e.

$$\phi^* = \underset{\phi}{\operatorname{argmax}} \int p(x) \left(\int p^*(\theta|x) \log p_\phi(\theta|x) \mathrm{d}\theta \right) \mathrm{d}x \tag{2.12}$$

Next, Bayes' Theorem is used to substitute $p^*(\theta|x)$. Additionally, the law of joint probability is applied, i.e.

$$p^*(\theta|x) = \frac{p(x|\theta)p(\theta)}{p(x)} = \frac{p(x,\theta)}{p(x)} \quad \Rightarrow \quad p(x)p^*(\theta|x) = p(x,\theta), \tag{2.13}$$

allowing us to write

$$\begin{aligned} \phi^* &= \underset{\phi}{\operatorname{argmax}} \iint p(x,\theta) \log p_\phi(\theta|x) \mathrm{d}\theta \, \mathrm{d}x \\ &= \underset{\phi}{\operatorname{argmax}} \iint p(x,\theta) \log p_\phi(\theta|x) \mathrm{d}x \, \mathrm{d}\theta \end{aligned} \tag{2.14}$$

Next, we want to express the approximation of the posterior distribution $p_\phi(\theta|x)$ in terms of the known base distribution of the latent variable z. Since we construct the approximate posterior by applying an invertible transformation $f_\phi(\theta; x)$ to map samples from the parameter space to the latent space, we can use the change of variables formula to compute the density in the original space. Specifically, for an invertible and differentiable function f_ϕ, the density of θ is obtained by evaluating the density of its output $z = f_\phi(\theta; x)$ in the latent space, and correcting for the distortion of space introduced by the transformation. This distortion is quantified by the determinant of the Jacobian matrix of f_ϕ with respect to θ. As a result, we obtain

$$p_\phi(\theta|x) = p(z) \left| \det \left(\frac{\partial z}{\partial \theta} \right) \right| = p(f_\phi(\theta; x)) \left| \det \left(\frac{\partial f_\phi(\theta; x)}{\partial \theta} \right) \right|, \tag{2.15}$$

where the first term evaluates the base density at the transformed point, and the second term accounts for the local change in volume caused by the transformation. This allows for the evaluation of the approximate posterior $p_\phi(\theta|x)$ without modeling it directly. Plugging (2.15)

into the objective function yields

$$\begin{aligned}
\phi^* &= \underset{\phi}{\operatorname{argmax}} \iint p(x,\theta) \log p_\phi(\theta|x)\, \mathrm{d}x\, \mathrm{d}\theta \\
&= \underset{\phi}{\operatorname{argmax}} \iint p(x,\theta) \log \left(p(f_\phi(\theta;x)) \left| \det \left(\frac{\partial f_\phi(\theta;x)}{\partial \theta} \right) \right| \right) \mathrm{d}x\, \mathrm{d}\theta \\
&= \underset{\phi}{\operatorname{argmax}} \iint p(x,\theta) \left[\log p(f_\phi(\theta;x)) + \log \left| \det \left(\frac{\partial f_\phi(\theta;x)}{\partial \theta} \right) \right| \right] \mathrm{d}x\, \mathrm{d}\theta \\
&= \underset{\phi}{\operatorname{argmax}} \iint p(x,\theta) \left[\log p(f_\phi(\theta;x)) + \log \left| \det \left(J_{f_\phi} \right) \right| \right] \mathrm{d}x\, \mathrm{d}\theta,
\end{aligned} \tag{2.16}$$

where a simplification of notation has been introduced, letting J_{f_ϕ} denote the Jacobian of f_ϕ evaluated at θ and x, i.e.

$$J_{f_\phi} = \frac{\partial f_\phi(\theta;x)}{\partial \theta} \tag{2.17}$$

The integral obtained in equation 2.16 is an expectation over the joint distribution $p(x,\theta)$, and can therefore also be written as

$$\phi^* = \underset{\phi}{\operatorname{argmax}}\, \mathbb{E}_{(x,\theta)\sim p(x,\theta)} \left[\log p(f_\phi(\theta;x)) + \log \left| \det \left(J_{f_\phi} \right) \right| \right] \tag{2.18}$$

In practice, (2.18) is not actually computed, due to the fact that $p(x,\theta) = p(x|\theta)p(\theta)$ is not tractable, as there is no explicit form for $p(x|\theta)$. Therefore, it is approximated using the Monte Carlo approximation and samples from $p(x,\theta)$ by drawing $\theta^{(m)} \sim p(\theta)$ and simulating $x^{(m)}$. Suppose we generate M training samples, then we can write

$$\mathbb{E}_{(x,\theta)\sim p(x,\theta)} \left[\log p(f_\phi(\theta;x)) + \log \left| \det \left(J_{f_\phi} \right) \right| \right] \approx \frac{1}{M} \sum_{m=1}^{M} \left[\log p(f_\phi(\theta^{(m)};x^{(m)})) + \log \left| \det \left(J_{f_\phi}^{(m)} \right) \right| \right] \tag{2.19}$$

Consequently, the optimization problem is written as

$$\phi^* = \underset{\phi}{\operatorname{argmax}} \frac{1}{M} \sum_{m=1}^{M} \left[\log p(f_\phi(\theta^{(m)};x^{(m)})) + \log \left| \det \left(J_{f_\phi}^{(m)} \right) \right| \right] \tag{2.20}$$

$$= \underset{\phi}{\operatorname{argmin}} \frac{1}{M} \sum_{m=1}^{M} \left[-\log p(f_\phi(\theta^{(m)};x^{(m)})) - \log \left| \det \left(J_{f_\phi}^{(m)} \right) \right| \right] \tag{2.21}$$

Next, the fact that the base distribution of the latent variable should be a standard Gaussian is used to write

$$f_\phi(\theta^{(m)};x^{(m)}) = z^{(m)} \sim \mathcal{N}(0,I) \tag{2.22}$$

Therefore,

$$p(f_\phi(\theta^{(m)};x^{(m)})) = p(z^{(m)}) = \frac{1}{(2\pi)^{(D/2)}} \exp\left(-\frac{1}{2} ||z^{(m)}||^2 \right) \tag{2.23}$$

Taking the logarithm yields

$$\log p(f_\phi(\theta^{(m)};x^{(m)})) = \log p(z^{(m)}) = -\frac{D}{2}\log(2\pi) - \frac{1}{2}||z^{(m)}||^2 \tag{2.24}$$

Plugging this back into the optimization problem and dropping constants yields the final form that is actually computed , i.e.

$$\phi^* = \underset{\phi}{\operatorname{argmin}} \frac{1}{M} \sum_{m=1}^{M} \left[\frac{D}{2} \log(2\pi) + \frac{1}{2} \left\| f_\phi(\theta^{(m)}; x^{(m)}) \right\|^2 - \log \left| \det \left(J_{f_\phi}^{(m)} \right) \right| \right] \tag{2.25}$$

$$= \underset{\phi}{\operatorname{argmin}} \frac{1}{M} \sum_{m=1}^{M} \left[\frac{1}{2} \left\| f_\phi(\theta^{(m)}; x^{(m)}) \right\|^2 - \log \left| \det \left(J_{f_\phi}^{(m)} \right) \right| \right] \tag{2.26}$$

If we introduce a summary network, denoted by $h_\psi(x)$, together with the invertible inference network, the main objective function from equation 2.11 becomes

$$\phi^*, \psi^* = \underset{\phi,\psi}{\operatorname{argmax}} \, \mathbb{E}_{p(x)} \left[\mathbb{E}_{p^*(\theta|x)} \left[\log p_\phi(\theta | h_\psi(x)) \right] \right] \tag{2.27}$$

Similarly, the Monte Carlo estimate changes to

$$\phi^*, \psi^* = \underset{\phi,\psi}{\operatorname{argmin}} \frac{1}{M} \sum_{m=1}^{M} \left[\frac{1}{2} \left\| f_\phi(\theta^{(m)}; h_\psi(x^{(m)})) \right\|_2^2 - \log \left| \det \left(J_{f_\phi}^{(m)} \right) \right| \right] \tag{2.28}$$

Intuitively, the first term of (2.28),

$$\frac{1}{2} || f_\phi(\theta^{(m)}; h_\psi(x^{(m)})) ||_2^2, \tag{2.29}$$

penalizes the distance of the distribution of the latent variable from the base normal distribution. This term encourages the transformed outputs of the inference network to behave like samples from a standard Gaussian, concentrated around zero with unit variance. The closer the transformed samples are to this ideal Gaussian distribution, the smaller this penalty becomes.

The second term of (2.28),

$$- \log \left| \det \left(J_{f_\phi}^{(m)} \right) \right|, \tag{2.30}$$

accounts for how much the transformation distorts volume when transforming parameters into the Gaussian space. As the inference network transforms parameters θ into latent variables z, it stretches or compresses regions of the parameter θ space to match the Gaussian form in z-space. This reshaping affects how the probability mass should be redistributed: expanding space should reduce local density, while compressing space should increase it, to conserve total probability. The determinant of the Jacobian quantifies this local expansion or compression. Including the negative log of this determinant in the loss forces the network to adjust densities properly when transforming from the original parameter posterior to the latent Gaussian.

Without this correction, the networks could incorrectly manipulate densities, such as collapsing all samples to the center of the Gaussian, without respecting probability conservation. This term therefore ensures that the uncertainty structure of the posterior is accurately preserved during the transformation.

2.2. Summary Statistics for Machine Learning & Bayesian Inference

Recent advances in simulation and sensor technology have resulted in the use of high-dimensional data. This also applies to the context of SHM, where networks built up of different types of sensors can produce vast amounts of high-dimensional data. However, directly using this raw, high-dimensional data in machine learning is not desirable. The curse of dimensionality leads to

data sparsity and unreliable distance metrics, increasing the risk of overfitting and escalating computational costs [37, 38].

Issues related to high-dimensional data also arise when using likelihood-free Bayesian inference methods, such as Approximate Bayesian Computation (ABC). These approaches bypass direct likelihood evaluation by comparing observed and simulated data. In classical ABC, parameter samples are used to compute simulated data, which is compared to observed data. Only those parameters for which the simulated data is sufficiently close to the observed data are accepted, forming an approximate posterior [25]. However, as distance metrics become less informative in high-dimensional spaces, comparisons based on raw data become unreliable [26, 39]. As BayesFlow is both machine learning based and likelihood-free, addressing the problems caused by the curse of dimensionality becomes particularly critical.

To address this issue, comparisons between observed and simulated data are performed on their respective *summary statistics*, rather than the raw high-dimensional inputs directly. A summary statistic is a compact, low-dimensional representation of the raw data. Transforming a high-dimensional input $x \in \mathbb{R}^n$ to a summary statistic $\tilde{x} \in \mathbb{R}^k$ involves applying a function $\mathcal{S}$, i.e.

$$\mathcal{S} : \mathbb{R}^n \to \mathbb{R}^k, \tag{2.31}$$

with $k \ll n$. Classical examples include the mean, variance, and autocorrelation. However, dimensionality reduction may discard essential information. For instance, data drawn from two very different distributions may share the same mean, requiring additional descriptive statistics to distinguish between them. Effective summary statistics must therefore balance compression with the retention of relevant information for downstream processes.

If the downstream process is inference, a summary statistic $\tilde{x} = \mathcal{S}(x)$ is said to be sufficient for a parameter θ if the posterior remains unchanged, i.e.

$$p(\theta|x) = p(\theta|\eta), \tag{2.32}$$

where η denotes a sufficient summary statistic. In other words, given η, the raw data x offers no additional information about θ. In practice, sufficiency is rarely achievable for complex simulators when using low-dimensional summaries [40]. Therefore, the goal becomes to find approximately sufficient statistics that are low-dimensional enough for efficient comparison, but rich enough to retain enough information for accurate inference. Poorly chosen summaries can lead to biased or uncertain posteriors, whereas well-crafted summaries can enable effective inference without an explicit likelihood. This trade-off underpins how critical the role of the summary statistic is in LFI, and motivates researching effective approaches to summary statistic design.

2.3. Neural Networks for Learning Summary Statistics

Traditionally, machine learning and Bayesian inference techniques for SHM have relied on hand-crafted summary statistics based on expert knowledge. Common examples include modal features like natural frequencies, and damping ratios [41], as well as spectral features such as power spectral density [41, 42]. Time-domain features such as peak-to-peak values are also widely used due to their simplicity and ability to indicate anomalies [41, 43]. These features are sensitive to structural changes, and therefore useful for damage detection. While interpretable and grounded in engineering principles, such features often require expert tuning and may not generalize well to new structures or damage scenarios.

Unsupervised learning methods offer a data-driven alternative by automatically extracting low-dimensional features without prior knowledge. Principal Component Analysis (PCA) is a popular technique that captures dominant variance directions by projecting data onto the leading eigenvectors of its covariance matrix [44, 45]. While effective, PCA features lack the

interpretability of hand-crafted statistics, such as mode shapes or damping ratios, which are tied to physical behavior. This highlights a trade-off between interpretability and generalization. Additionally, unsupervised methods often rely on assumptions like linearity or stationarity, which may not hold in practice. Nevertheless, they can reveal hidden structures not captured by manually constructed summary statistics.

An alternative to hand-crafted and unsupervised summary statistics is a third class called learned summary statistics, which are features that are extracted by machine learning models that have been trained end-to-end for specific tasks. As a matter if fact, using neural networks to learn summary statistics is one of the aspects that makes BayesFlow especially effective. As explained in section 2.1, the summary network learns to generate informative summaries directly from data during training.

A key feature of these learned summaries is that they are task-aware, unlike hand-crafted or unsupervised features. This is a consequence of the joint optimization of the summary network and the inference network, specifically with the goal of improving posterior estimation. The joint optimization ensures that the extracted features are highly relevant for the inference objective, often outperforming fixed or unsupervised features [29].

Building on the concept of learned summary statistics, we arrive at the use of *physics-informed* summary neural networks. The method proposed in this work is to incorporate prior physical knowledge into the architecture of the summary network, allowing the model to learn task specific summary statistics more efficiently. The concepts from PIML that are relevant for the methodology developed in this work are explained in more detail in section 2.4.

2.4. Physics-Informed Machine Learning

PIML is a field of research that focuses on combining data-driven machine learning models with physical principles. In PIML, known relationships, symmetries or conservations are incorporated into the machine learning process, enabling models to capture true system behavior more efficiently than black-box approaches can [34, 46, 47, 48]. For this reason, PIML methods are particularly useful in settings where training data is scarce or expensive to generate, as is the case in simulation based SHM. A common way to categorize PIML approaches is by where the physics information enters the model. Three *biases* have been outlined for this purpose: observational bias (physics enters via the data), inductive bias (physics enters via the network architecture), and learning bias (physics enters via the training or loss function) [34]. Subsections 2.4.1, 2.4.3, and 2.4.2 discuss each type of bias in more detail.

2.4.1. Observational Bias

Observational biases are the most fundamental manner of introducing physics into the machine learning process. As a matter of fact, observational biases form the base of what makes machine learning possible in general. As machine learning models learn from training data, the data must obey the underlying laws of physics that the model is aiming to understand. If a dataset would not reflect the relevant physics, a machine learning model would never be able to learn the correct relationships from the data. In other words, training on physics-informed data facilitates the learning of physically correct behavior [34, 46, 48]. Building further on this principle, in some cases it is possible to augment existing data to expand the training set. This can be done as long as the transformations applied to the data are consistent with physical laws. For example, in fluid dynamics, flow fields can be rotated or reflected when symmetry exists in the governing equations. In image classification, a picture of an animal that is to be classified can be mirrored without altering the fact that the image contains the same type of animal. Both of these examples illustrate how data augmentation techniques can help to expand the size of a training dataset without violating any physics, or requiring the acquisition of completely new data.

2.4.2. Learning Bias

Learning biases focus on incorporating physics knowledge during the training process. This is typically done through the loss function, which can include terms that penalize behaviour that deviates from the laws of physics, or through constraints that explicitly favor convergence towards solutions that follow physics [34, 46]. A classic example is the physics-informed neural network, in which the loss function is augmented with a term measuring the residual of a partial differential equation [49]. The network is trained to not only fit the available data, but also to minimize the violation of the governing differential equations. Any physics-based penalty or constraint added to the loss function thus acts as a bias.

2.4.3. Inductive Bias

Inductive bias refers to encoding physical knowledge directly into the model's architecture [34, 46]. Instead of relying solely on data-driven learning or the enforcing of physical consistency through loss penalties, the main principle of inductive bias is to ensure the model architecture itself is designed to inherently respect and leverage any physical relationships. For example, if a system exhibits spatial invariance, architectures like Convolutional Neural Networks (CNNs) are preferred, as their local receptive fields and shared weights inherently assume translational symmetry. If a system evolves over time, architectures like Recurrent Neural Networks (RNNs) or 1D Convolutional Networks are used to reflect causal, sequential dependencies. If relationships are structured as networks of interactions, GNNs are typically used to encode the topological structure directly into the model architecture [34]. The key idea is that selecting a model architecture with a suitable inductive bias effectively integrates physical knowledge directly into the learning process, even before any data is seen.

2.4.4. Physics-Informed Machine Learning within BayesFlow

After discussing a number of different strategies to inject physical knowledge into machine learning processes, the logical next step is to determine how to apply PIML concepts within the BayesFlow framework. Observational biases are inherently present in BayesFlow, as the use of a simulator already ensures that the training data is consistent with the process generating it. Learning biases are difficult to introduce into the inference network. This is mainly due to the fact that during the training phase, the inference network is not optimized to approximate a physical quantity, but rather to ensure that a latent variable follows a Gaussian distribution. Therefore, there are no physical relationships to penalize. Similarly, the output of the summary network is not a physical quantity or known variable, meaning there are no physical constraints that can be placed upon the summary statistics.

Therefore, the most logical method is to use inductive biases, informing the architectural designs of the neural networks based on their input data. There are two components in which knowledge of the physical system could theoretically be integrated using inductive biases: the summary network and the inference network. However, using inductive biases to inform the inference network is difficult. The inference network takes the summary vector as an input, which is already an abstract representation. There is little physical structure left in the input for the inference network to exploit directly. Furthermore, the inference network has a specific architecture using normalizing flows that cannot be altered easily, and it must support invertibility. Therefore, embedding inductive biases here is both difficult and less meaningful.

In contrast, the summary network directly processes physical data, and is therefore the natural component to include architectural choices that reflect the structure of the physical system. Designing the summary network architecture in such a way that known symmetries, conservation laws, or other relationships in the data are leveraged immediately biases the feature extraction toward physically meaningful representation. In combination with the joint optimization of the summary and inference networks used in BayesFlow, injecting inductive biases into the summary network should ensure suitable features for downstream inference.

3

Methodology

The strategy proposed in this work is to inform the design of the summary network based on the characteristics of SHM data. Therefore, it is essential to first clearly define exactly what prior physical knowledge is available, and which specific characteristics we intend to leverage. The key properties that have been identified are sequentiality and spatial connectivity. Based on these characteristics, a Graph Summary Neural Network (GSNN) is designed, which incorporates both characteristics into its model architecture. Additionally, a method is developed that allows for the qualitative comparison of models when performing BayesFlow-based damage detection.

3.1. Leveraging the Characteristics of Health Monitoring Data

This section introduces the format of SHM data and explains the concept of influence lines. Subsequently, the key characteristics of influence line based SHM data are discussed. The primary characteristics that have been identified are sequential structure *within* each influence line and spatial connectivity *between* influence lines, i.e. sensor locations.

3.1.1. Structural Health Monitoring using Influence Lines

In the domain of SHM, two commonly used sensor types are displacement sensors and axial force sensors. The case study investigated in Chapter 4 also assumes a structure equipped with these sensor types. However, the characteristics described in this thesis generalize to any sensor type that can be used to measure a structural response at a specific location on the structure.

A tool that allows for the systematical interpretation of the measurements generated by these types of sensors is the influence line. An influence line characterizes how a structural quantity varies at a particular point on the structure as a function of the position of a load moving along the structure. As they are well suited to represent the effects of moving vehicles, they are often used in settings such as bridges. In contrast to the time-domain or frequency-domain plots control engineers may be used to, influence lines are plotted with respect to the load position, linking the spatial position of a load to a structural response. The influence line of sensor s is represented as

$$x_s(n) = [x_s(1), x_s(2) \dots, x_s(N)] \in \mathbb{R}^N, \tag{3.1}$$

where x_s denotes the steady state response measured by sensor s, and N denotes the number of positions along the structure at which a load is applied. Examples can be found in Figures C.1a and C.1b in Appendix D. As multiple sensors are typically installed to monitor a structure, the data can be formatted into a matrix $x \in \mathbb{R}^{S \times N}$, where each row represents the influence line of sensor $s = 1, \dots, S$, made up of measurements collected at $n = 1, \dots, N$ load positions.

A collection of influence lines contains valuable information, and captures more than just the local structural response at each specific sensor location. A collection of influence lines also reflects the global behavior of a structure as an interconnected system. When a load is applied, its effects propagate throughout the structure, meaning that even distant sensors can register a response. A collection of influence lines contains information on how the entire structure redistributes loads applied at different positions through its supports, spans, and joints. As influence lines encode both local stiffness characteristics and broader system-level interactions, they are a powerful tool for assessing the health of a structure.

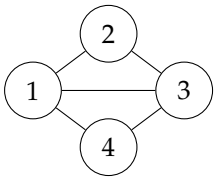
3.1.2. Characteristics of Structural Health Monitoring Data

A collection of influence lines exhibits two key properties: sequentiality and spatial correlation, rooted in the inherent definition of an influence line and the geometry of the structure. Each influence line reflects spatial progression, with the index n corresponding to the position at which a load is applied. Permuting the sequence destroys its physical meaning, making the data inherently sequential. Influence lines exhibit local continuity and smoothness due to the structure's physical continuity. Deviations, such as sharp peaks or discontinuities, can indicate damage, highlighting the importance of preserving sequential structure during analysis.

A collection of influence lines also contains spatial correlations *between* the sensors (i.e. between the rows of a data matrix). Sensor readings are influenced by the geometry of the structure, with sensors on the same load path or in close proximity showing correlated responses. This is especially true in truss systems, where mechanical interactions clearly follow structural topology. The data is thus jointly structured: each sensor provides a sequence, and these sequences are spatially arranged according to the structure's topology. This dual structure calls for a summary network that can incorporate both sequential and spatial dependencies.

3.1.3. Representing Sensor Networks as Graphs

The relationship between the sensors naturally lends itself to graph based analysis. A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ can be represented using a set of nodes, denoted as $\mathcal{V}$, and a set of edges, denoted as $\mathcal{E}$. A graph can be efficiently captured by its adjacency matrix, denoted as A. For an undirected graph, each entry $A_{i,j}$ is 1 if there is an edge between nodes i and j, and 0 otherwise. In addition to the adjacency matrix, a graph can also be described using the degree matrix D, which is a diagonal matrix where each entry D_{ii} equals the degree (i.e., number of connections) of node i. For example, the graph shown in Figure 3.1 has adjacency matrix A and degree matrix D:



Graph G

$$A = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

Adjacency matrix A

$$D = \begin{bmatrix} 3 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix}$$

Degree matrix D

Figure 3.1: Graph G and its corresponding adjacency matrix A and degree matrix D.

An essential aspect of this work is to make use of the fact that structures can be intuitively cast into a graph, by interpreting sensors as graph nodes and the structural elements connecting the sensors as edges [50]. The proposed method is to introduce an inductive bias into the summary network that leverages this graph structure. Incorporating the information about the spatial connectivity of the sensors should allow the machine learning model to reason more effectively about how changes in one part of the structure may impact connected regions.

3.2. Design of a Graph Summary Neural Network

The summary network designed to leverage the physics described in subsection 3.1.2 is a GSNN. Following the graph notation provided in subsection 3.1.3, a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is constructed for a given sensor network, where $\mathcal{V} = \{v_1, \ldots, v_S\}$, meaning each node corresponds to a sensor s, and $\mathcal{E}$ is chosen so that sensors connected by structural elements in real-life are connected by edges in the graph. This graph is incorporated in the model architecture, introducing an inductive bias based on the relationships between sensors.

The GSNN uses a message-passing layer. Each node in the graph is assigned a feature vector, which is initially the influence line of the sensor corresponding to that node. Note that instead of using the raw influence line, a preprocessed feature representation of the influence line could also be used as the feature vector. Each node updates its feature vector by aggregating information from its neighbors, and integrating this with its own feature vector. The update for node v_s is

$$h_{v_s} = \sigma\left(W_{\text{self}} x_{v_s} + \frac{1}{|\mathcal{N}(v_s)|} \sum_{j \in \mathcal{N}(v_s)} W_{\text{msg}} x_{v_j} + b\right), \tag{3.2}$$

where $h_{v_s} \in \mathbb{R}^H$ is the updated feature vector of node v_s. Furthermore, $W_{\text{self}}, W_{\text{msg}} \in \mathbb{R}^{N \times H}$ denote the learnable weight matrices for self and neighbor contributions, $x_{v_s} \in \mathbb{R}^N$ denotes the feature vector of node v_s, $x_{v_j} \in \mathbb{R}^N$ is the feature vector of node j, $b \in \mathbb{R}^H$ is a learnable bias vector, H is the message-passing output dimension, and $\sigma(\cdot)$ is a non-linear activation function.

Equation 3.2 is applied to each node of the graph in a single loop, with the resulting feature vectors of each node becoming rows of the output of the message passing layer, i.e.,

$$h_{\text{MPL}} = \begin{bmatrix} h_{v_1}^\top & h_{v_2}^\top & \ldots & h_{v_S}^\top \end{bmatrix} \in \mathbb{R}^{S \times H}. \tag{3.3}$$

This loop-based approach computes the mean of transformed neighbor features per node sequentially. While flexible and easy to understand, it can become computationally inefficient for larger sensor networks, as each node's update is handled one by one. To improve computational efficiency the same operation can be reformulated using matrix multiplication and vectorized tensor operations. To achieve this, the sensor connectivity is encoded into a row-normalized adjacency matrix (excluding self-loops) $\tilde{A} \in \mathbb{R}^{S \times S}$, constructed according to

$$\tilde{A}_{ij} = \begin{cases} \frac{1}{|\mathcal{N}(i)|} & \text{if } j \in \mathcal{N}(i) \\ 0 & \text{otherwise,} \end{cases} \tag{3.4}$$

Given an input $x \in \mathbb{R}^{S \times N}$, in which each row corresponds to an influence line, the aggregated neighbor messages can be computed in a single operation according to

$$h_{\text{agg}} = \tilde{A}(x W_{\text{agg}}) \in \mathbb{R}^{S \times H} \tag{3.5}$$

The self-message is computed in parallel as

$$h_{\text{self}} = x W_{\text{self}} \in \mathbb{R}^{S \times H}, \tag{3.6}$$

The final update is then computed as

$$h_{\text{MPL}} = \sigma\left(h_{\text{self}} + h_{\text{agg}} + b\right) \in \mathbb{R}^{S \times H} \tag{3.7}$$

Multiple message passing layers can be used sequentially to propagate the information from each node throughout the network. To transform the output of the message passing layer into a fixed-size summary vector, the updated sensor features are flattened across the sensor and feature dimensions. A flattening function is defined, transforming a matrix into a vector, i.e.

$$f_{\text{vec}} : \mathbb{R}^{m \times n} \to \mathbb{R}^{mn} \tag{3.8}$$

The flattening function is applied to the output of the message passing layers

$$f_{\text{vec}}(h_{\text{MPL}}) = h_{\text{flat}} \in \mathbb{R}^{SH}, \tag{3.9}$$

resulting in a vector containing the concatenated feature representations of each node. This vector is then passed through a fully connected layer, called the output layer, to reduce the dimension of the vector to the desired size of the summary vector according to

$$\tilde{x} = \sigma(W_{\text{out}} h_{\text{flat}} + b_{\text{out}}) \in \mathbb{R}^{d_s}, \tag{3.10}$$

where $W_{\text{out}} \in \mathbb{R}^{d_s \times SH}$ is a matrix containing learnable weights, $b_{\text{out}} \in \mathbb{R}^{d_s}$ is a vector containing learnable biases, and $\sigma(\cdot)$ is a non-linear activation function.

3.3. Evaluating the Quality of a Posterior

In addition to the GSNN, a method has been developed that facilitates quantitative comparison between BayesFlow models with different summary networks. This is important, as the performance of the GSNN should be evaluated against baseline neural networks across varying training dataset sizes to determine whether it is indeed able to learn more efficiently.

The first step in comparing model performances is to define a measure that reflects the quality of the model output. In other words, defining what makes one posterior more suitable than another. A posterior represents the updated belief about a parameter after observing data, combining prior information and observed evidence. Therefore, a suitable posterior is not necessarily one that is as certain as possible, but rather one that appropriately reflects the true uncertainty in the inference problem. For example, when data is scarce or noisy, the posterior should express high uncertainty, rather than overconfident estimates. However, in this work, predicted damage parameters are compared to ground truth values, which are known exactly. As there is no uncertainty, the ground truth is a deterministic value rather than a distribution.

When comparing to a deterministic ground truth, a suitable posterior has a mean that is close to the ground truth and a standard deviation that is as low as possible. To illustrate this, 3 toy models are compared. In the example, the ground truth of the parameter that is to be estimated is 0.8. Figure 3.2 shows the posterior distribution for the parameter computed using model 1. As the mean of the posterior distribution is far from the ground truth, the estimate is inaccurate. In figure 3.4 the output of model 2 is shown. The mean of the posterior distribution is close to the ground truth value, but the standard deviation is high. This indicates that the estimate is accurate, but uncertain. In Figure 3.6, the output of model 3 is shown. The mean of the posterior distribution is close to the ground truth value, and the standard deviation is low, indicating the resulting estimate is both accurate and certain.

3.3.1. The Continuous Ranked Probability Score

The Continuous Ranked Probability Score (CRPS) is used to quantify the quality of the posterior when comparing to a deterministic value. CRPS is a scoring rule that is often used to evaluate the accuracy of probabilistic forecasts. It measures the difference between the predicted cumulative distribution function (CDF) and the CDF of the ground truth posterior. A lower CRPS value therefore indicates a better match between the predicted distribution and the ground truth. The CRPS is defined as

$$\text{CRPS}(p(\theta|x_{\text{obs}}), \theta^*) = \int_{-\infty}^{\infty} (\text{CDF}(\theta) - \mathcal{H}\{\theta \geq \theta^*\})^2 \, \mathrm{d}\theta, \tag{3.11}$$

where θ^* is the ground truth, $\text{CDF}(\theta) = \int_{-\infty}^{\theta} p(t|x_{\text{obs}}) \, \mathrm{d}t$ is the CDF of the posterior $p(\theta|x_{\text{obs}})$, and $\mathcal{H}\{\theta \geq \theta^*\}$ is the step function centered at θ^*, which is 0 when $\theta < \theta^*$ and 1 otherwise.

Figure 3.2: PDF of the posterior computed using model 1 vs ground truth. The mean of the posterior is far from the ground truth, meaning the estimate is inaccurate.

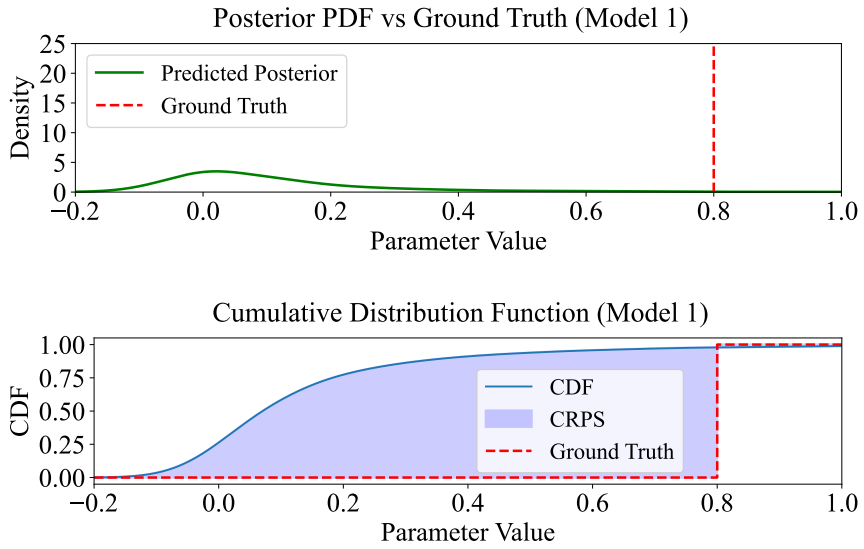


Figure 3.3: Due to the offset of the mean of the posterior, the area between the CDF and ground truth step function is large, resulting in a large CRPS.

Figure 3.4: PDF of the posterior computed using model 2 vs ground truth. The mean of the posterior distribution is close to the ground truth value, but the standard deviation is large. The estimate is accurate, but uncertain.

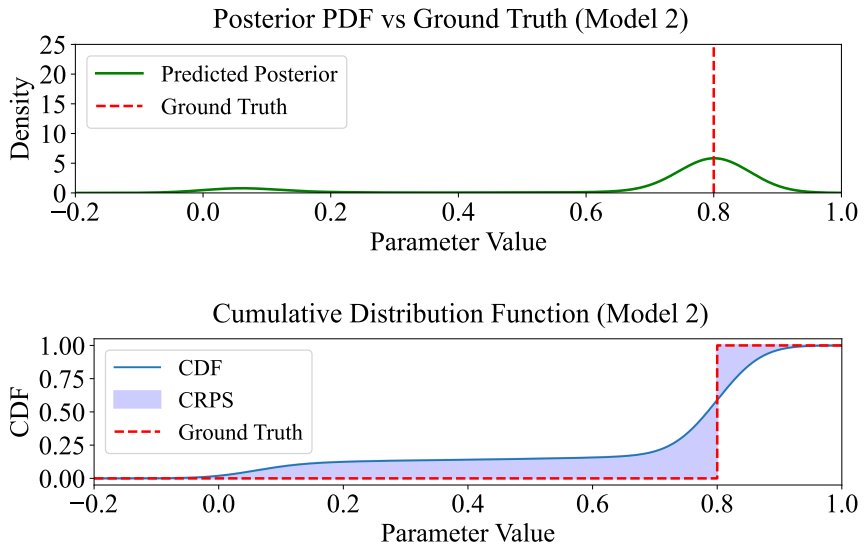


Figure 3.5: The offset of the mean of the posterior is small. However, due to the large standard deviation, the slope of the CDF is low, increasing the area between the CDF and ground truth step function, resulting in a larger CRPS.

Figure 3.6: PDF of the posterior computed using model 3 vs ground truth. The mean of the posterior distribution is close to the ground truth value, and the standard deviation is low. The estimate is accurate and certain.

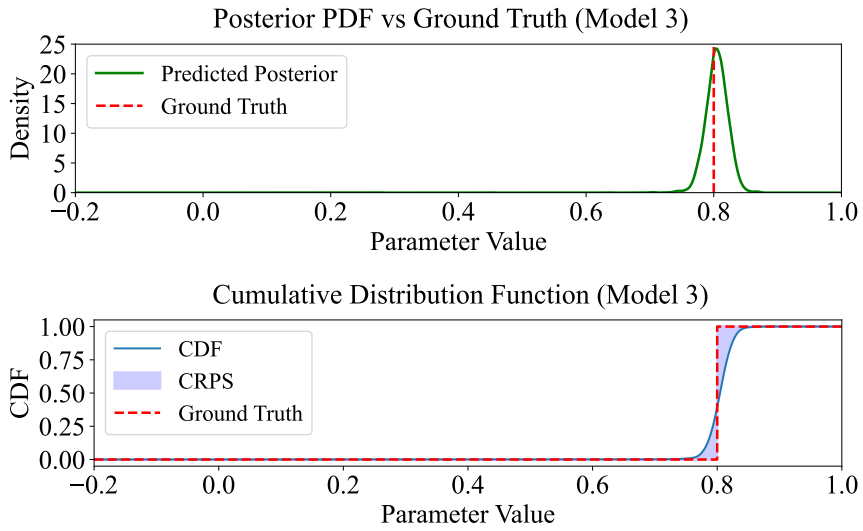


Figure 3.7: The offset of the mean of the posterior is small. Due to the low standard deviation, the slope of the CDF is steep, decreasing the area between the CDF and ground truth step function, resulting in a lower CRPS.

When the predictive distribution is represented via N samples, denoted as $\{\theta_i\}_{i=1}^{N} \sim \text{CDF}(\theta)$, instead of closed-form distributions, the CRPS can be approximated as

$$\text{CRPS}(\{\theta_i\}, \theta^*) = \frac{1}{N}\sum_{i=1}^{N}|\theta_i - \theta^*| - \frac{1}{2N^2}\sum_{i=1}^{N}\sum_{j=1}^{N}|\theta_i - \theta_j| \tag{3.12}$$

The same example used in subsection 3.3.1 is used to demonstrate how the CRPS penalizes inaccurate posteriors when comparing to a deterministic ground truth. Figure 3.3 shows the CRPS computed for the posterior generated using model 1. Due to the offset of the mean of the posterior, the area between the CDF and ground truth step function is large, resulting in a large CRPS. Figure 3.5 shows the CRPS computed for the posterior generated using model 2. The offset of the mean of the posterior is small. However, due to the large standard deviation, the slope of the CDF is low, increasing the area between the CDF and ground truth step function, resulting in a larger CRPS. Figure 3.7 shows the CRPS computed for the posterior generated using model 3. The offset of the mean of the posterior is small. Due to the low standard deviation, the slope of the CDF is steep, decreasing the area between the CDF and ground truth step function, resulting in a lower CRPS.

3.3.2. Evaluating a Model

As the CRPS is a metric for *one* posterior, it only conveys how well the damage parameter of *one* element within the structure is estimated. However, the goal is to create a metric that reflects the accuracy of the model when estimating an entire parameter vector. Therefore, a weighted average of the CRPS of each estimated damage parameters is used. As the damage parameter of the damaged bar is the parameter we are most interested in, the CRPS of this parameter is awarded a heavier weight. However, it is still important to include the CRPS of the other bars to reflect the model's ability to correctly identify undamaged elements and avoid false positives, ensuring that the posterior is not only accurate in detecting damage but also well-calibrated across the entire parameter space.

An instance of a ground truth parameter vector, together with its corresponding simulated structural response is referred to as a *scenario*. For each scenario, a weighted score is computed, which reflects how well the model recovers the true damage parameters based on the given observation. The weighted score α of scenario j is calculated as

$$\alpha^{(j)} = \frac{w_{i=i_{\text{dam}}} \text{CRPS}^{(j)}_{i=i_{\text{dam}}} + \sum_{i \neq i_{\text{dam}}} w_{i \neq i_{\text{dam}}} \text{CRPS}^{(j)}_{i}}{\sum w}, \tag{3.13}$$

where w_i denotes the weight for the CRPS of element i. The performance of an entire model, denoted as β, given a set of J scenarios, is computed by averaging $\alpha^{(j)}$ over the scenarios, i.e.

$$\beta = \frac{1}{J} \sum_{j=1}^{J} \alpha^{(j)}, \tag{3.14}$$

3.4. Baseline Summary Networks

The GSNN is compared against several baseline summary network architectures. These architectures incrementally incorporate more knowledge about the physical system.

The first baseline summary network is a fully connected, *dense* feedforward neural network. This architecture does not make use of any inductive bias, meaning it does not exploit any spatial or sequential structure of the sensor data. Instead, it treats the input data as a flat array of features. While such a network can, in principle, approximate any mapping given sufficient capacity and data, it relies entirely on the learning process to extract meaningful patterns from raw values. As a result, these neural networks often require significantly more training data to reach competitive performance, making them a useful lower bound for comparison.

The second baseline network is referred to as a *sequential network*. It takes into account the sequential nature of the influence lines, but does not leverage any spatial structure between the sensors. The network applies 1D convolutions to each sensor's influence line independently to extract local sequential features, and then uses an LSTM followed by a dense layer to aggregate these features into a fixed-length summary vector. The processing is sensor-wise and independent, meaning that potential correlations or interactions between sensors are not modeled explicitly. This architecture is based on an existing summary network in the BayesFlow package, and was previously used by researchers at TNO [9] for similar BayesFlow-based SHM applications. Its performance therefore serves as a reference for methods that exploit sequential dynamics but no spatial relationships.

The third baseline is the *2D-CNN* summary network, which processes the influence line data as if it were a grayscale image, where the first axis corresponds to the sensor index and the second to the load application location. The data is passed through a series of convolutional blocks that capture local patterns across both axes, followed by a global average pooling layer to produce the summary vector. This allows the network to model limited spatial interactions between neighboring sensors, but only through a predefined ordering in the sensor dimension. While this ordering can be chosen to reflect physical relationships, this spatial modeling is much less representative than when using an explicit graph.

All baseline summary networks are implemented to be fully compatible with the BayesFlow framework, ensuring they can be used interchangeably within the same amortized Bayesian inference pipeline. They process the same raw influence line data and are configured to output summary vectors of the same dimensionality, denoted as d_s. Fixing this dimension ensures that differences in performance can be attributed to architectural choices. More mathematical details about each of the baseline summary networks can be found in Appendix G.

4

Experiments and Simulations

The performance of the GSNN is compared to benchmark summary networks, which progressively incorporate more physics knowledge into their design. The comparison is conducted through a case study in which BayesFlow-based SHM is applied to three different synthetic idealizations of a model bridge. For each idealization, performance was evaluated across varying training data sizes to highlight the difference in training efficiency between the machine learning models.

4.1. System Description

The case study is based on a small-scale bridge, constructed in 2024 by Bundesanstalt für Materialforschung und -prüfung (BAM) and TNO, shown in Figure 4.1. The model consists of ABS plastic elements, with an aluminum core anchoring steel cables for vertical stability. A flexible roadbed on top of the plastic elements is used to guide a self-driving car across the bridge.

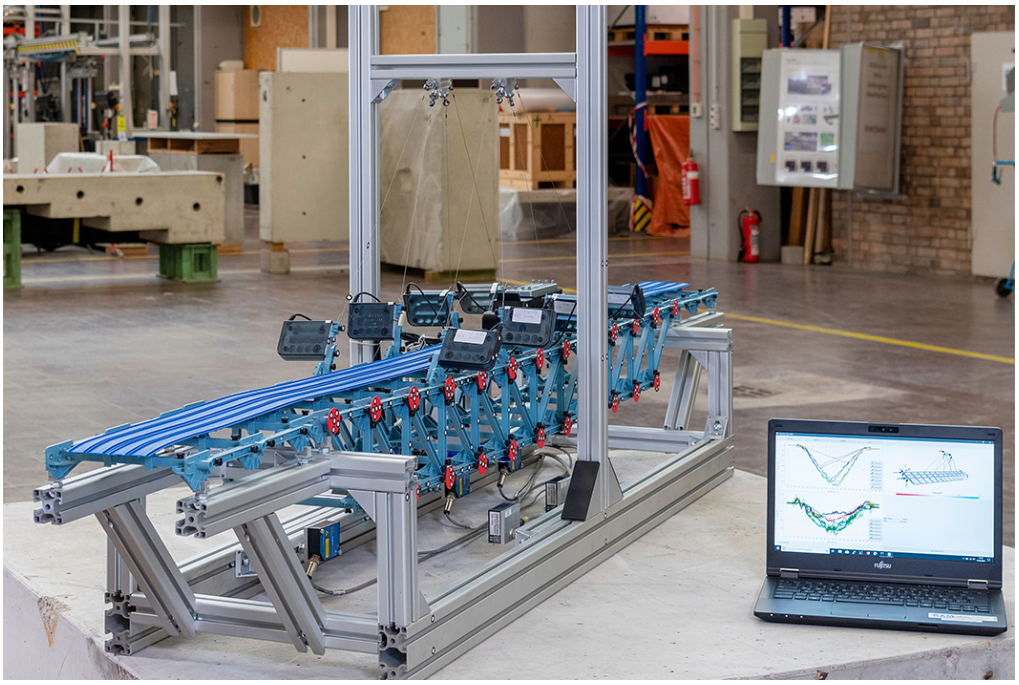


Figure 4.1: Small scale model of a cable-stayed truss bridge, constructed in 2024 by BAM and TNO [51].

The model represents a cable-stayed truss bridge. This entails that the bridge deck is supported by a truss, which is a framework composed of straight elements connected at nodes to form triangular units. Trusses ensure that loads are effectively distributed through axial forces in individual members. As a result, the structure can carry significant loads with relatively little material. Additionally, the deck is supported by inclined cables anchored to towers. The small-scale bridge is equipped with both vertical displacement sensors and axial cable force sensors. Damage in the structure can be emulated by replacing the ABS elements with lower-stiffness variants to simulate a reduction in elastic modulus. In the case study covered in this work, the goal is to detect damage in *idealized versions* of this small-scale bridge.

4.2. Experiment Setup

Damage detection in the idealized models is performed by estimating the damage parameters of certain truss elements, as described in section 1.2. To verify if the GSNN does indeed learn more efficiently, model performance was evaluated across varying training data sizes for each of the baseline models and the GSNN. This section outlines the setup and procedure of these experiments.

4.2.1. Idealization Scenarios

Idealization 1: The first idealization is a two-dimensional simplification of the bridge, created using the structure's transversal symmetry. The mesh of the idealization is shown in Figure 4.2, consisting of 45 bars and 4 cables arranged in a Pratt truss configuration. It is assumed that all 22 unconstrained nodes, depicted in red, are equipped with vertical displacement sensors, and all 4 cables are equipped with axial force sensors, yielding a total of $S = 26$ sensors. An important addition is the assumptions made in this idealization. It is assumed that only bars in the bottom chord (bars 12 through 21) are able to be damaged, and at most one bar can be damaged at a time.

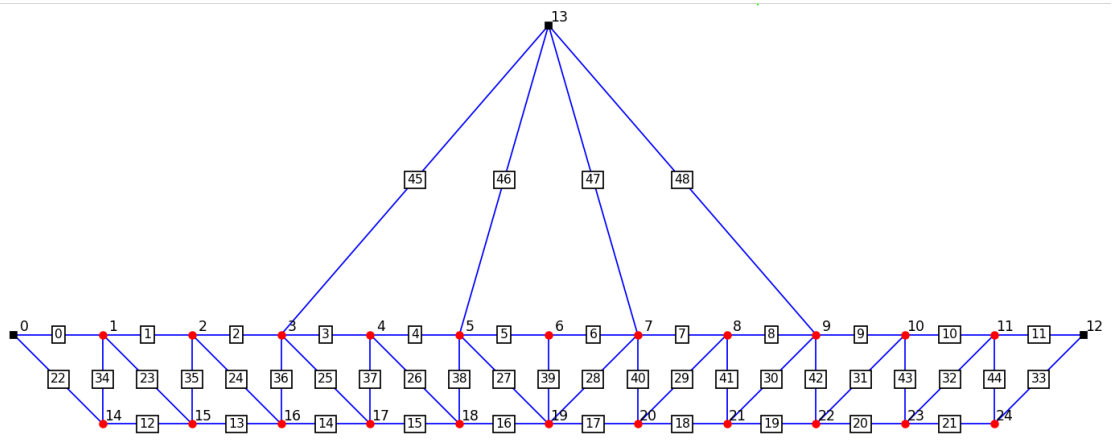


Figure 4.2: Mesh of the first and second idealizations of the small-scale model bridge.

Idealization 2: The second idealization uses the same mesh as the first idealization, displayed in Figure 4.2. The key difference lies in the assumptions. In this idealization, it is assumed bars in both the top and bottom chords (bars 0 through 21) are able to be damaged. This increases the parameter space significantly, as 12 additional damage parameters are introduced. Again, it is assumed that only one bar can be damaged at a time.

Idealization 3: The third idealization focuses on a more complex, three-dimensional mesh, displayed in Figure 4.3. The mesh consists of 138 bars and 8 cables. It is assumed that all 44 unconstrained nodes are equipped with vertical displacement sensors, and all 8 cables are equipped with axial force sensors, yielding a total of $S = 52$ sensors. For this idealization, the assumption is made that all bars in the bottom chord of the bridge (bars 12 through 21 and 99 through 108) are able to be damaged, and at most one bar can be damaged simultaneously.

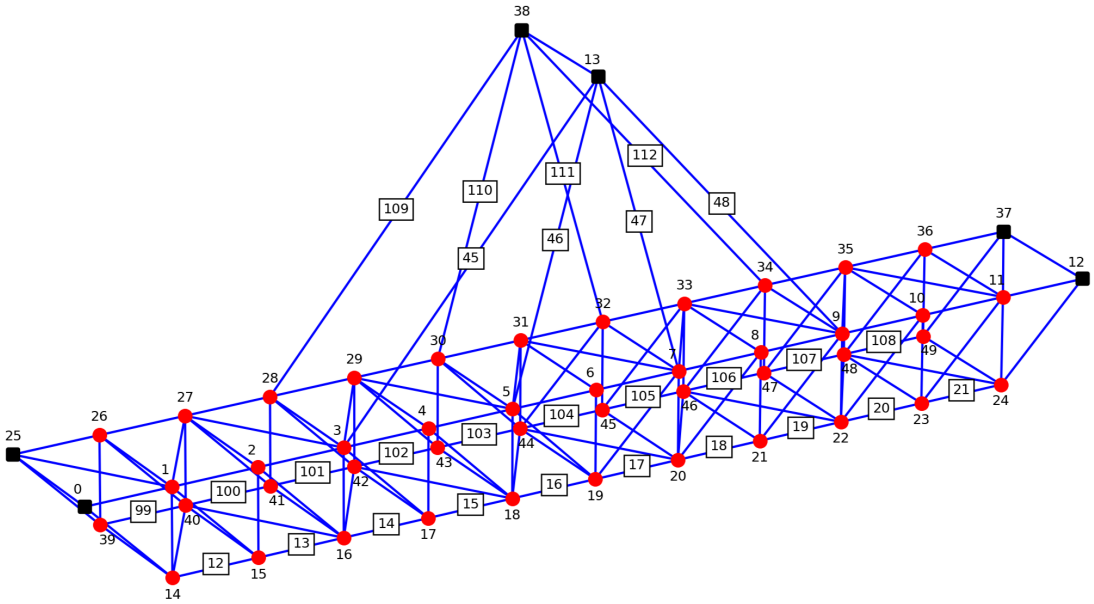


Figure 4.3: Mesh of the third idealization of the small-scale model bridge.

4.2.2. Forward Model

A FEM, able to compute the reactions of a structure for a given parameter vector and load, is used as the simulator. The output of the FEM is used to construct an influence line for each sensor present in the simplification of the bridge. An example output for the 2D mesh is plotted in Figures C.1a and C.1b. More information about the working principles and validation process of the FEM can be found in Appendix B.

4.2.3. Prior Definitions

For each idealization, the parameter vector θ includes the base Young's modulus of the bars, denoted as E_{bar}, and the base Young's modulus of the cables, denoted as E_{cable}. Additionally, damage parameters ω are added to the prior for a selected subset of the bars, denoted as $\mathcal{B}$. To define which bars are added to $\mathcal{B}$, the bar elements in truss are indexed using i, corresponding to the numeration displayed in Figure 4.2 for idealizations 1 and 2, and corresponding to Figure 4.3 for idealization 3. The set of damage parameters can thus be defined as

$$\omega = \{\omega_i\} \; \forall i \in \mathcal{B}, \tag{4.1}$$

where $\omega_i \in [0, 1]$ denotes the relative stiffness reduction of bar i. The damage parameters reduce the base stiffness of the bar elements according to

$$E_i = E_{\text{bar}} \cdot (1 - \omega_i), \quad \forall i \in \mathcal{B} \tag{4.2}$$

To constrain the inference problem, it is assumed that only one bar in the damageable set can be damaged bar at a time. That is, the damaged bar index is drawn from a uniform discrete distribution over $\mathcal{B}$, i.e.

$$i_{\text{dam}} \sim \mathcal{U}(\mathcal{B}), \tag{4.3}$$

where i_{dam} is the index of the damaged bar, and $\mathcal{U}(\mathcal{B})$ is the uniform distribution over the elements of $\mathcal{B}$. The damage severity of bar i_{damaged} is drawn from a uniform prior, i.e.

$$\omega_{i_{\text{dam}}} \sim \mathcal{U}([\omega_{\text{min, dam}}, \omega_{\text{max, dam}}]), \tag{4.4}$$

where, $\omega_{\text{min, dam}}$ is the lower bound of the damage parameter of the damaged bar, and $\omega_{\text{max, dam}}$ is the upper bound of the damage parameter of the damaged bar. The damage parameters of all other bars are drawn from a narrow triangular distribution centered at 0, i.e.

$$\omega_i \sim \mathcal{T}(\omega_{\text{min, undam}}, \mu, \omega_{\text{max, undam}}), \ \forall i \neq i_{\text{dam}} \tag{4.5}$$

where $\mathcal{T}$ denotes a triangular distribution, $\omega_{\text{min, undam}}$ denotes the lower bound of the damage parameter of an undamaged bar, μ denotes the center of the triangular distribution, $\omega_{\text{max, undam}}$ denotes the upper bound of the damage parameter of an undamaged bar.

Idealization 1: In this case, the 10 bars along the bridge bottom chord are assumed to be damageable, i.e.

$$\mathcal{B} = \{12, 13, \ldots, 21\}, \tag{4.6}$$

The choice for this relatively small set is made in order to constrain the parameter space and keep the inference process relatively simple for demonstration purposes. This results in a prior built up of 12 parameters in total: 2 elastic moduli and 10 damage parameters, of which only one is active.

Idealization 2: In this case, the bars along both the top and bottom chords are assumed to be damageable, i.e.

$$\mathcal{B} = \{0, 1, \ldots, 21\}, \tag{4.7}$$

This results in a prior built up of 24 parameters in total: 2 elastic moduli and 22 damage parameters, of which only one is active.

Idealization 3: In this case, the bars along both the bottom chord of the 3D mesh are assumed to be damageable, i.e.

$$\mathcal{B} = \{12, \ldots, 21\} \cup \{99, \ldots, 108\}, \tag{4.8}$$

This results in a prior built up of 22 parameters in total: 2 elastic moduli and 20 damage parameters, of which only one is active. Table 4.1 summarizes the prior distributions for each of the idealizations.

Table 4.1: Prior distributions of model parameters for the three idealizations

Parameter	Unit	Distribution	Hyperparameters	Idealization
E_{bar}	GPa	Normal	$(\mu,\ \sigma^2)$ = (2.3, 0.46)	1–3
E_{cable}	GPa	Normal	$(\mu,\ \sigma^2)$ = (210, 20)	1–3
$\omega_i,\ i \neq i_{\text{dam}}$	[-]	Triangular	(lb, μ, ub) = (-0.1, 0, 0.1)	1–3
$\omega_i,\ i = i_{\text{dam}}$	[-]	Uniform	(lb, ub) = (-0.1, 1)	1–3
i_{dam}	[-]	Discrete Uniform	(lb, ub) = (12, 21)	1
i_{dam}	[-]	Discrete Uniform	(lb, ub) = (0, 21)	2
i_{dam}	[-]	Discrete Uniform	(lb, ub) = (12, 21) $\cup$ (99, 108)	3

4.2.4. Generation of Training Data

Synthetic training data is generated using the simulator, i.e. the FEM. For each idealization, samples of the parameter vector are drawn from their respective prior. For each sampled parameter vector, the influence lines of all S sensors are computed by applying identical vertical loads at N equidistant positions along the deck of the bridge, resulting in a data matrix

$$x_{\text{FEM}} = \mathcal{F}(\theta) \in \mathbb{R}^{S \times N} \tag{4.9}$$

The clean FEM output for sensor is then corrupted with noise to simulate measurement uncertainty. Different noise levels are used for cable force and displacement sensors to reflect the higher sensitivity of the displacement sensors. For cable force sensors, noise is drawn from a Gaussian distribution with zero mean and variance σ^2_{force}, i.e.

$$\epsilon_{\text{force}} \sim \mathcal{N}(0, \sigma^2_{\text{force}}) \tag{4.10}$$

For vertical displacement sensors, noise is drawn from a different Gaussian distribution, again with zero mean but this time with variance σ^2_{disp}, i.e.

$$\epsilon_{\text{disp}} \sim \mathcal{N}(0, \sigma^2_{\text{disp}}) \tag{4.11}$$

Each data matrix, augmented with noise, is also referred to as a *simulation*. Before being fed to the machine learning models, the training simulations are normalized by subtracting the mean and dividing by the standard deviation per variable independently.

4.2.5. Network Settings

The inference network is kept identical across all comparisons. A coupling flow is chosen with the same number of layers as parameters to be estimated, as prescribed in [29]. As the goal is to demonstrate the training efficiency of the GSNN, BayesFlow models are created using several different summary networks as baselines, allowing for comparison. These summary networks include a Dense Feedforward Neural Network, a 2D-CNN, and a Sequence Network. Each of these summary networks is discussed in more detail in section Appendix G. When using a GSNN, the connectivity graph is constructed from the structure geometry as explained in section 3.1.3. An illustration of the sensor connectivity graph for the first and second idealizations is depicted in Figure 4.4. It is clear that sensors neighboring in the mesh become neighbors in the graph.

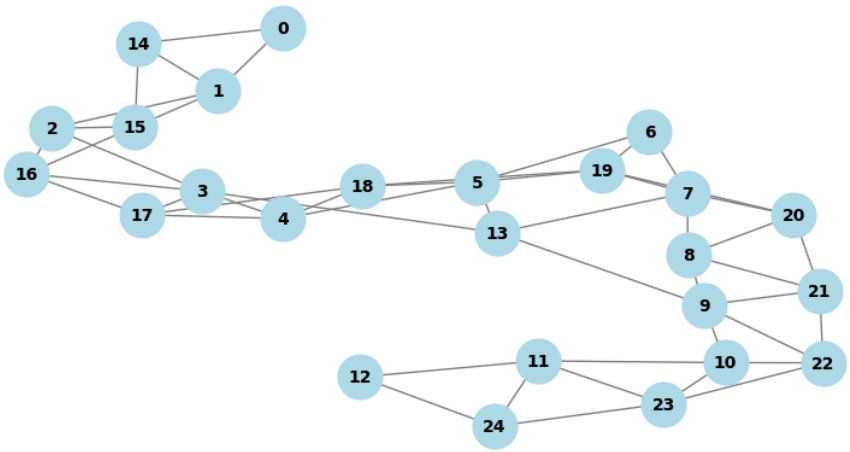


Figure 4.4: Connectivity graph of the mesh in Figure 4.2.

An illustration of the sensor connectivity graph for the third idealization is depicted in Figure 4.5. Again, sensors neighboring in the mesh become neighbors in the graph. The hyperparameters of all networks can be found in Appendix H.

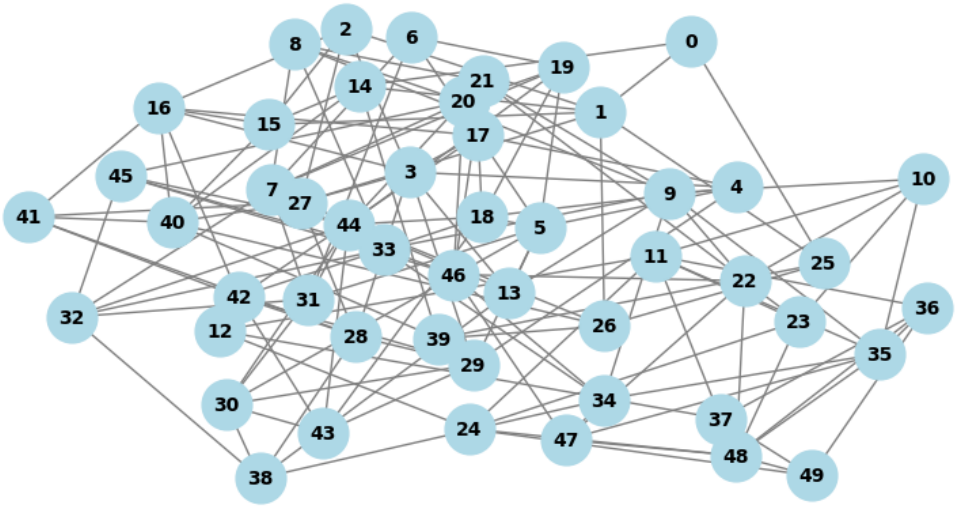


Figure 4.5: Connectivity graph of the mesh in Figure 4.3.

4.2.6. Training Process

The amount of training data available to the model is taken as a dependent variable in this work, and is therefore incrementally increased between models. All other aspects of the training process are kept equal. The largest number of training simulations used per comparison is determined based on the performance of the models, where the goal is to ensure each model has the opportunity to finish learning. All models are trained for 50 epochs, using a batch size of 32 training simulations, the Adam optimizer with a learning rate of 1e-4, and the same 200 validation simulations.

4.2.7. Testing Process

The performance of each model is verified using a set of ground truth scenarios. The ground truth scenarios that are used include a completely undamaged scenario ($\omega_i = 0,\ \forall i$), and one scenario per damageable bar, in which the damaged bar i has damage parameter $\omega_{i_{\text{dam}}} = 0.8$ and all other bars are undamaged. Example influence lines generated for these ground truth parameter vectors for the first idealization can be found in Appendix D, Figures D.1a and D.1b. The weights of the CRPS scores of undamaged bars and the damaged bar used to determine the model performance score β are 0.1 and 0.9 respectively.

5

Results

This chapter evaluates the performance of the GSNN against the baseline models across the three idealizations of increasing complexity described in Chapter 4. The primary metrics for comparison are data efficiency, meaning the number of training simulations required to achieve the bound on performance, and final model accuracy, quantified by the model score at the performance bound, where a lower score indicates a better performance. The results, summarized in Table 5.1, are discussed separately for each idealization.

5.1. Idealization 1: 2D Model, Bottom Chord

We begin by presenting the results for the simplest idealization, i.e. the 2D mesh in which only the bottom bridge chord is assumed to be damageable. Figure 5.1 shows the model score as a function of the training dataset size for each different summary network. Exponential curves have been fitted to the data points to highlight the asymptotic convergence of the model score.

Figure 5.1: Size of training dataset vs performance with fitted exponential decay curves. The GSNN learns most efficiently, followed by the 2D-CNN, Dense Network and finally the Sequential Network.

The Sequential Network, which was used in previous work on BayesFlow based SHM [9], requires the most simulations (≈80,000) to reach peak performance. The Dense Network stabilizes after ≈40,000 simulations. The 2D-CNN converges after ≈30,000 simulations. Notably, the GSNN reaches its final performance after just ≈20,000 simulations. This indicates the GSNN indeed requires fewer (75% less) training simulations in comparison to the original Sequential Network. Another notable result is the fact that the performance of the 2D-CNN and Dense Networks plateaus at a higher model score than the Sequence Network and GSNN. After analyzing the posteriors, it is evident that the difference in performance score is caused by the fact that the 2D-CNN and Dense Networks produce posteriors in which the uncertainty is slightly higher than in the posteriors produced by the Sequence Network and GSNN.

To show how model performance differs at the point where adding more data provides little further benefit to the GSNN, Figures 5.2, 5.3, 5.4, and 5.5 show the posteriors produced by each model trained with 10,000 simulations for a scenario in which bar $i = 15$ is damaged.

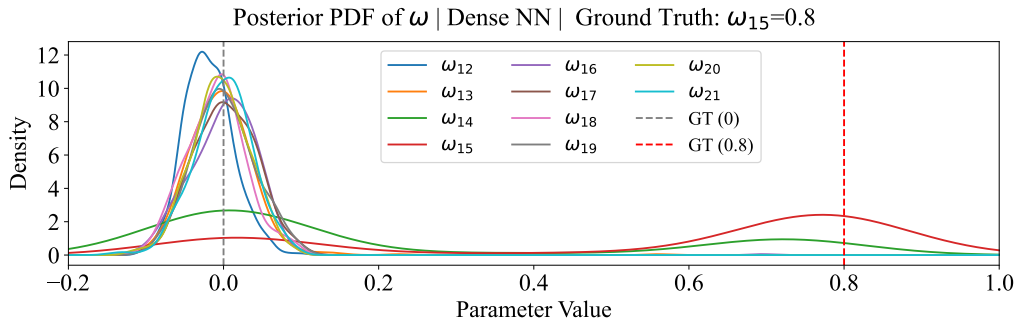


Figure 5.2: Posterior generated using a Dense summary network with 10,000 training simulations.

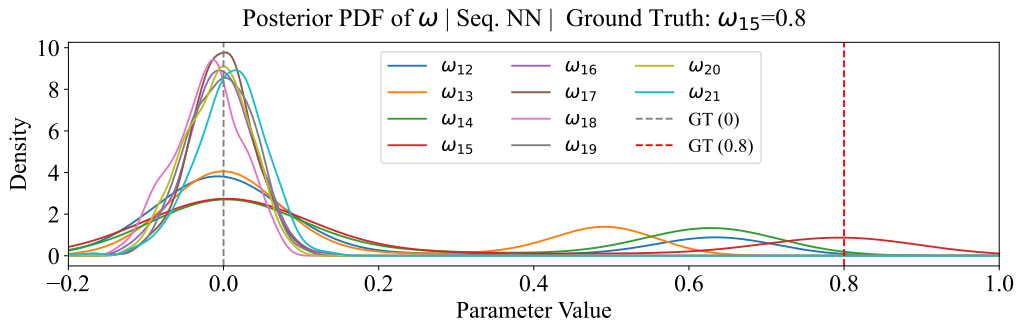


Figure 5.3: Posterior generated using a Sequential summary network with 10,000 training simulations.

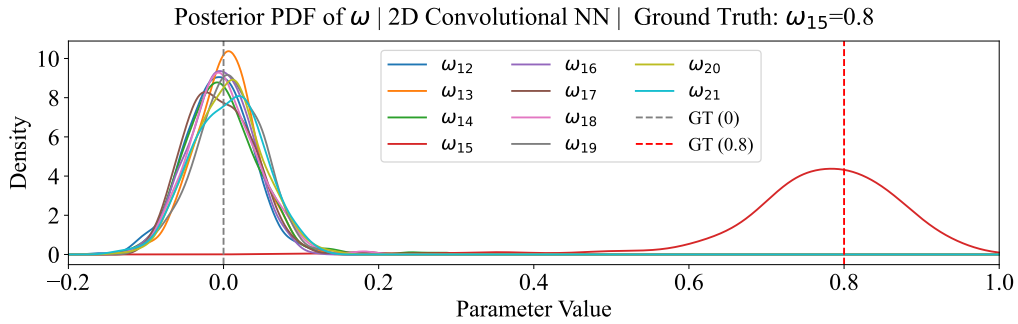


Figure 5.4: Posterior generated using a 2D-CNN with 10,000 training simulations.

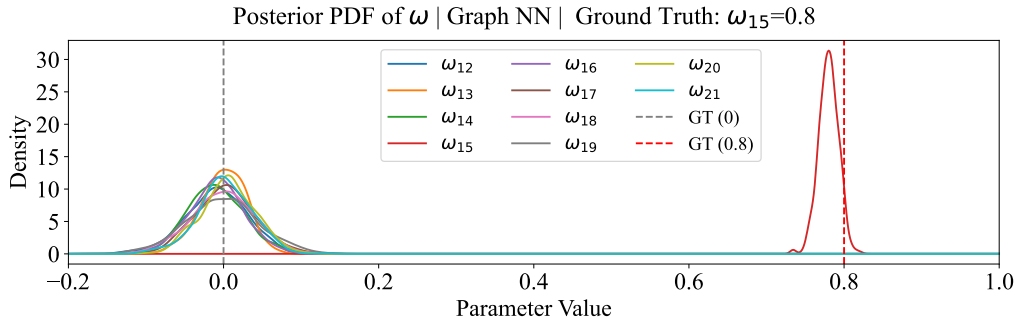


Figure 5.5: Posterior generated using a GSNN with 10,000 training simulations.

Figure 5.5 indicates that the GSNN is already able to localize and quantify the damage relatively accurately, including a high level of certainty. In contrast, the posterior generated by the Sequential Network, depicted in Figure 5.3 is very inaccurate and clearly indicates the model does not grasp the relationship between damage parameters and structural responses. Although the posterior indicates the model is aware there is damage in the vicinity of bars 12, 13, 14 and 15, it is not able to confidently identify specifically which bar has been damaged or to what extent. The performance of the Dense Network is slightly better, as depicted in Figure 5.2. The model is only unsure whether the damage is in bar 14 or 15. Figure 5.4 shows that the 2D-CNN is able to suitably identify the bar with damage, but is less certain about its prediction than the GSNN.

5.2. Idealization 2: 2D Model, Expanded Parameter Space

Figure 5.6 shows the results for the second idealization, in which the same 2D mesh is used, but 22 damage parameters are estimated in addition to the base Young's Moduli of the bars and cables (as opposed to 10 damage parameters in the first idealization).

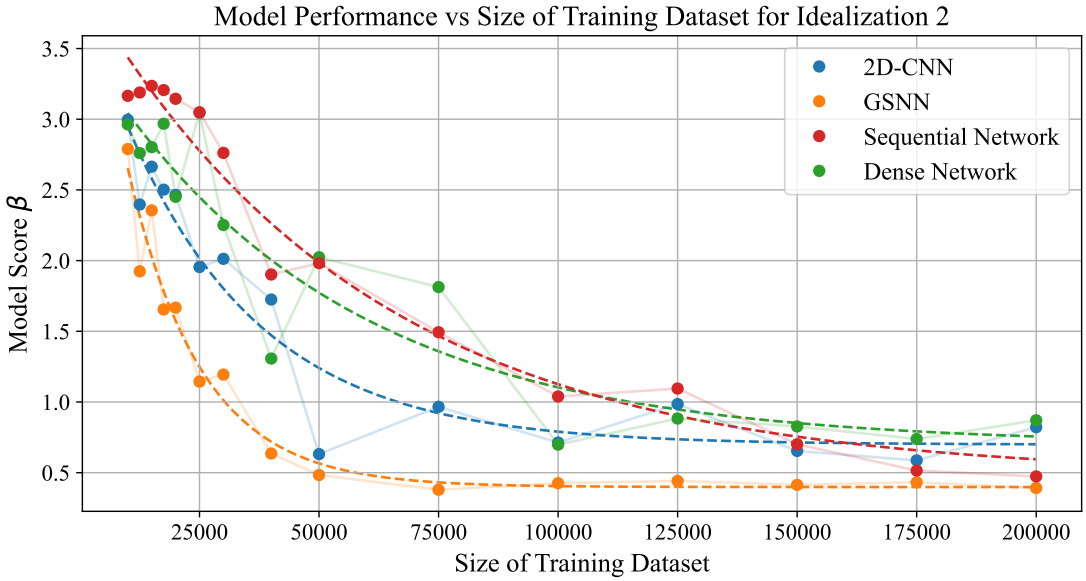


Figure 5.6: Size of training dataset vs performance including exponential decay curves. The GSNN learns most efficiently, followed by the 2D-CNN. The Dense Network and Sequential Network perform similarly.

All models require significantly more data to reach their peak performance, as is to be expected when estimating more parameters. Here, the advantage of the GSNN becomes even more apparent. It not only requires the fewest simulations to converge (≈75,000) but, as shown in Table

5.1, it also achieves a significantly better final model score (0.3918) than all baseline models. To place this into perspective, the Sequential and Dense Networks require over 200,000 simulations and only reach scores of 0.5149 and 0.8705, respectively. It can also be observed that the final performance of the GSNN is substantially better than that of the 2D-CNN, with the difference being more prominent than in the first idealization. This suggests that as the parameter space grows, the unguided learning approach of the baseline models becomes less effective, while the physics-informed GSNN continues to extract more relevant features efficiently.

5.3. Idealization 3: 3D Model

The third and final idealization is the most complex challenge: a full 3D bridge model with 52 sensors and 20 possible damage locations in the bottom chords of the bridge. The results, shown in Figure 5.7, reveal a dramatic divergence in the capabilities of the summary networks.

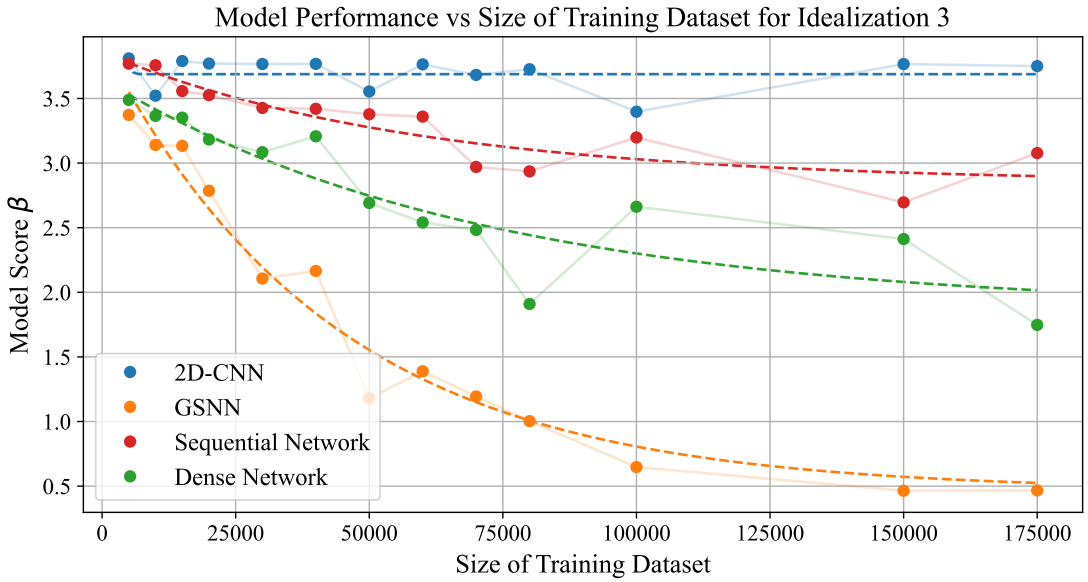


Figure 5.7: Size of training dataset vs performance including exponential decay curves.

The GSNN is the architecture that clearly demonstrates the most effective learning in this scenario. Its performance score improves quickly, with a steep learning curve as larger training datasets are used. Furthermore, the GSNN reaches a final score of 0.467, which is almost an order of magnitude better than its competitors.

The other three baseline networks learn much less effectively. The Dense Network and Sequential Network are able to learn some of the investigated relationships, although their performances plateau at poor scores (1.748 and 3.077, respectively). Most strikingly, the 2D-CNN, which was a respectable performer in the 2D cases, fails to learn completely, yielding the worst final score of 3.750. This highlights the limitation of its inductive bias. Treating the sensor data as a 2D image is a viable approximation for a simple, 2D sensor layout, but breaks down when faced with the complex and non-uniform spatial arrangement of sensors on a 3D structure.

The GSNN, however, is not constrained by such assumptions. By design, it directly encodes the physical connectivity of the sensor network, regardless of its geometric complexity. This allows it to effectively model the propagation of structural responses through the bridge. These results underscore the central hypothesis of this work: for complex, real-world structures, incorporating an accurate, physics-informed inductive bias into the summary network is not just beneficial, but essential for enabling accurate amortized SHM.

Table 5.1: Results for all Idealization Experiments

Idealization	Summary Network	Simulations Required to Reach Peak Performance	Final Score	Improvement over Sequential
1	Sequential	80,000	0.2168	[-]
	Dense	40,000	0.4535	50%
	2D-CNN	30,000	0.4057	37.5%
	GSNN	**20,000**	**0.2157**	**75%**
2	Sequential	>200,000	0.5149	[-]
	Dense	200,000	0.8705	0%
	2D-CNN	150,000	0.8217	25%
	GSNN	**75,000**	**0.3918**	**37.5%**
3	Sequential	[-]	3.077	[-]
	Dense	[-]	1.748	[-]
	2D-CNN	[-]	3.750	[-]
	GSNN	[-]	**0.467**	[-]

6

Conclusion

The main contribution of this work is the development of a GSNN that can be used as the summary network component of BayesFlow, improving both the simulation efficiency and approximated posterior quality, thereby paving the way to amortized, likelihood-free SHM with uncertainty quantification in *real-life* settings.

To demonstrate the potential of the developed method, three progressively more complex case studies were investigated. The first involved identifying a single damaged element in a 2D setting. Subsequently, a more challenging 2D case was covered, in which additional bars were eligible to be damaged. Last, a 3D case was investigated, with a significantly larger and more complex structure. In the simplest case, the GSNN reduced the number of simulations needed to reach the performance bound by 75%. Note that in this case, all models were eventually able to reach a similar performance bound. In the more complex 2D setting, the GSNN not only decreased the required number of simulations by 37.5% but also yielded superior posterior estimates, reaching a model performance score of 0.3918, while baseline networks only reached scores between 0.5149 and 0.8705. In the 3D scenario, the advantage of the GSNN over baseline networks in terms of model performance is further highlighted, as a performance score of 0.467 was reached, compared to baseline models which were only able to achieve performance scores ranging from 3.077 to 1.748. These results indicate that the GSNN surpasses baseline networks in terms of both efficiency *and* performance. This is a valuable development, as it highlights the potential of the GSNN to open the door to amortized, likelihood-free SHM with uncertainty quantification in real-life settings.

Although this work used a cable-stayed truss bridge as a case study to demonstrate the feasibility of amortized Bayesian inference with a GSNN as the summary network, the approach is not limited to this specific bridge, or even bridges in general. Using the topology of a structure to inform the construction of the GSNN can be generalized. Structures such as space frames, transmission towers, lattice girders, and crane booms all consist of discrete elements connected at nodes, forming a natural graph representation. The same modeling approach can be directly applied to these systems, requiring only adjustments to the graph to reflect the specific topology of each structure.

In addition, the method can be extended to non-truss structures by abstracting the physical system into a representation appropriate for applying graph theory. In FEMs of plates or solid structures, individual elements can serve as nodes, with edges defined based on mesh connectivity or physical adjacency. Alternatively, in large-scale structures such as bridges, buildings, or offshore platforms, a more conceptual graph representation can be adopted. Here, major structural components, such as foundations, columns, beams, girders, and slabs, can be treated as graph nodes, and their supports, joints, or interfaces can define the graph edges. For instance, in a

highway bridge, the foundation, piers, crossheads, girders, and deck could be modeled as nodes, connecting them according to their load transfer relationships. This flexibility in representing different structural systems as graphs suggests that the developed GSNN could be adapted for damage detection tasks in a broad range of SHM applications.

While the current implementation of the GSNN has demonstrated strong capabilities, future work could focus on incorporating additional physics knowledge into the model. The most direct way to infuse more physics into the GSNN is by treating the edges of the graph not just as connections, but as the physical elements they represent. For example, the length of a member is directly related to its axial stiffness. For a given material and cross-section, a longer member is more flexible. Including member length as an edge feature would allow the GSNN to learn this relationship explicitly. Additionally, the angle of a truss member determines how its internal axial force resolves into horizontal and vertical components at the nodes. By providing the orientation as an edge feature, the GSNN could be able to learn how forces are distributed along different directions at the nodes.

Furthermore, it would be valuable to apply the methods described in this work to more complex cases. Expanding the test case by using the physical bridge demonstrator would add valuable insights that are not covered in synthetic cases. Working with real-life structures and measurements is vastly more difficult, as demonstrated in Appendix J, which contains the first steps towards the application of the method described in this work to the real-life bridge demonstrator. One obstacle is the logistical issue of collecting data, which is challenging due to practical limitations in sensor placement and the accessibility of structural components. Additionally, the collected data typically has higher levels of noise and stochasticity, as external influences and sensor imperfections introduce uncertainties and disturbances that are absent in controlled simulations.

Another challenge that arises when extending this work to real-life scenarios is the issue of model discrepancies. In the test cases discussed in this thesis, the simulator used to generate the training data is identical to the model generating the synthetic observations. However, in reality, the physical structure will never be represented perfectly by the numerical model, as simplifications, approximations, and uncertainties in material properties, boundary conditions, and loading scenarios inevitably introduce mismatches. These discrepancies mean that the inference network, which is trained purely on synthetic data, may not generalize well when applied to real measurements, as the process generating observed responses may deviate from the model behaviour learned during training.

Addressing this model misspecification is a difficult task. Recommended directions to tackle this issue include using state-of-the-art modeling techniques, such as well-developed FEMs, able to accurately capture the structural behavior, along with hybrid approaches that can enrich physics-based simulations with data-driven corrections. Additionally, advanced system identification methods should be used to accurately identify the base structural properties and dynamic behavior from measurements, ensuring that models are properly calibrated to reflect the real system.

Appendices

Appendix A. Justification of a Summary Network

In this appendix, a proof is given to show that a perfectly trained inference network is able to perfectly reproduce the posterior. This further justifies the use of a summary network instead of handcrafted or unsupervised statistics.

If the invertible neural network is perfectly trained, the latent variables z always follow the same standard multivariate Gaussian distribution regardless of the data x. In other words: z and x are independent. If the network is trained perfectly, the global minimum of the loss function in equation 2.10 is reached. As the KL-divergence is always larger than or equal to zero, the global minimum is achieved when the argument becomes 0. This implies

$$\mathbb{KL}(p^*(\theta|x)||p(\theta|x) = 0 \qquad \forall x \tag{1}$$

Using the fact that the KL-divergence is preserved under smooth, invertible transformations (which we enforce our neural network to be),

$$\mathbb{KL}(p^*(\theta|x)||p(\theta|x) = \mathbb{KL}(p^*(z)||p(z|x) = 0 \tag{2}$$

Therefore, the two distributions must be identical, and no matter what x is, z will always follow the standard Gaussian distribution, i.e.

$$p^*(z) = p(z|x) \Leftrightarrow z \perp x \tag{3}$$

This is important, as in this case, sampling $z \sim \mathcal{N}(0, I)$ and passing the samples through the inverse network outputs valid samples from the true posterior $p^*(\theta|x)$.

Appendix B. Finite Element Model Details

This Appendix describes the FEM base, which was used as the simulator. The FEM provided training data, validation data and synthetic observations used as test data. The FEM was validated using COMSOL Multiphysics. The assumptions and working principles are discussed in detail.

Appendix B.1 Assumptions of the Finite Element Model

The FEM assumes the geometry consists only of straight line trusses, which are assumed to be massless and have a constant cross-sectional area. The trusses are modeled to only carry axial forces (tension/compression), ignoring any bending and shear. Nodes are modeled as points with no physical size, shape, mass or rotational properties. Nodes are assumed to have only degrees of freedom in the translation directions. Details of the structure, such as stiffeners, gusset plates, or diaphragms are neglected. The coarse mesh fails to capture stress gradients accurately.

Regarding material properties, the deformation of the trusses is assumed to follow a linear elastic model, dependent on the Young's modulus and the cross-sectional area. The cables and bars each have their own cross-sectional area and Young's modulus. It is also important to note that there is currently no implementation of the fact that cables are not able to handle any compressive force.

Next, the boundary condition assumptions are discussed. The nodes connecting the structure to the ground and the nodes connecting the cables to the frame are assumed to be completely fixed,

meaning they do not have the freedom to translate or rotate in any direction. Although these boundary conditions may be perfectly rigid supports, in reality some flexibility exists.

The final set of assumptions are the loading condition assumptions. In the FEM, forces are applied as static, concentrated loads directly at the nodes. It also assumes that the forces are instantaneously applied, uniformly distributed, and remain constant over time, neglecting any dynamic effects of moving loads such as acceleration of a vehicle.

Appendix B.2 Mathematical Basis of the Finite Element Model

Each truss element is modeled to only carry axial forces (tension or compression). No bending moments, shear forces, or torsional effects are included. The joints connecting the elements are idealized as frictionless and pin-connected. The model assumes linear elasticity, meaning that the relationship between stress and strain is linear and follows Hooke's Law. Furthermore, the model does not account for initial geometric imperfections, residual stresses, or pre-stress in the elements. All elements are assumed to be stress-free in their undeformed configuration.

The FEM base is built as a Python class, which can be used to compute the displacements and internal forces of the structure when under load. The FEM base is supplied with a mesh file containing the geometry of the structure, defined by a set of nodes and trusses. Each node i is defined by its index, coordinates (x_i, y_i), and a Boolean flag indicating whether it is constrained or not. Nodes are instances of the `Node` class and have two degrees of freedom: horizontal and vertical displacements. The global degrees of freedom for node i are given by

$$\text{dofs}_i = [2i, 2i + 1] \tag{4}$$

Each truss element connects two nodes and is defined by its own index t, the indexes of its start and end nodes, and its type (either 'bar' or 'cable'). These are instances of the `Truss` class. The use of a mesh describing the nodes and truss elements means that the FEM base is reusable for different bridges, as the geometry can be altered easily. To illustrate the FEM, the mesh of the idealization shown in Figure 4.2 is used. Blue elements 0–44 represent steel bars building up the bridge truss. Elements 45–48 represent the steel cables. Nodes 0, 12, and 13 are shown in black, indicating that they are constrained. All other (red) nodes have two degrees of freedom.

After defining the overall geometry, the material properties are assigned. Each truss element is assumed to behave according to linear elasticity and is characterized by its axial stiffness EA, where E is the Young's modulus and A is the cross-sectional area. Each bar is assumed to have the same base cross-sectional area A_{bar} and base stiffness E_{bar}. Similarly, each cable is assumed to have the same base cross-sectional area A_{cable} and base stiffness E_{cable}. As the cables have a uniform cylindrical shape, A_{cable} is estimated using the known diameter of the cables d_{cable} as

$$A_{\text{cable}} = \frac{\pi d_{\text{cable}}^2}{4} \tag{5}$$

The initial length of the truss element is denoted as l_0. Using the material properties and length of the element, the local stiffness matrix in the local coordinate system is constructed as

$$k_{\text{local}} = \frac{EA}{l_0} \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{6}$$

This matrix is then transformed to the global coordinate system using the rotation matrix $\mathbf{R}$, which depends on the angle θ of the element relative to the global x-axis. The rotation matrix is given by

$$R = \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & \cos\theta & \sin\theta \\ 0 & 0 & -\sin\theta & \cos\theta \end{bmatrix}, \tag{7}$$

where the directional cosines are computed as

$$\cos\theta = \frac{x_2 - x_1}{l_0}, \quad \sin\theta = \frac{y_2 - y_1}{l_0} \tag{8}$$

The global element stiffness matrix is then obtained as

$$k_{\text{global}} = R^\top k_{\text{local}} R \tag{9}$$

The global stiffness matrix of the entire structure, $K \in \mathbb{R}^{2n \times 2n}$, is assembled by summing all individual element contributions according to the connectivity of the elements. For each element, the global degrees of freedom are identified, and its k_{global} is inserted into the corresponding block of K. Once the global stiffness matrix has been assembled, external loads are applied via the global force vector $F \in \mathbb{R}^{2n}$, where each entry corresponds to a nodal force. In our case, vertical loads are used to simulate a vehicle moving across the bridge. These loads can be applied directly to a node or interpolated across neighboring nodes. The equilibrium equation of the FEM system is given as

$$Ku = F \tag{10}$$

To enforce boundary conditions, a penalty method is applied. For each constrained degree of freedom d, the corresponding row and column of K are zeroed out, and the diagonal entry is set to one. The corresponding entry in F is also set to zero, i.e.

$$K_{dd} = 1, \quad K_{di} = K_{id} = 0 \quad \forall i \neq d, \quad F_d = 0 \tag{11}$$

This yields a constrained system, given as

$$K_c u = F_c, \tag{12}$$

which is solved using LU decomposition to obtain the displacement vector $\mathbf{u}$, containing all degrees of freedom in the structure.

The axial force in a truss element is computed by transforming the global displacements into the element's local coordinate system and evaluating the strain

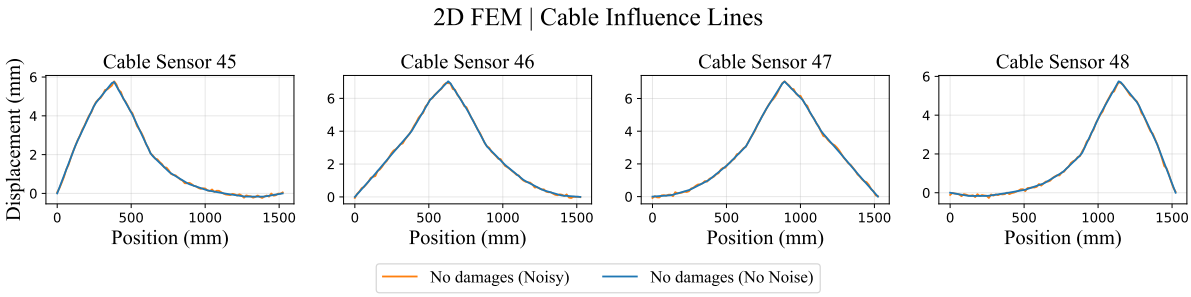
$$\varepsilon = \frac{u_{\text{local},2} - u_{\text{local},1}}{l_0}, \tag{13}$$

which gives the axial force

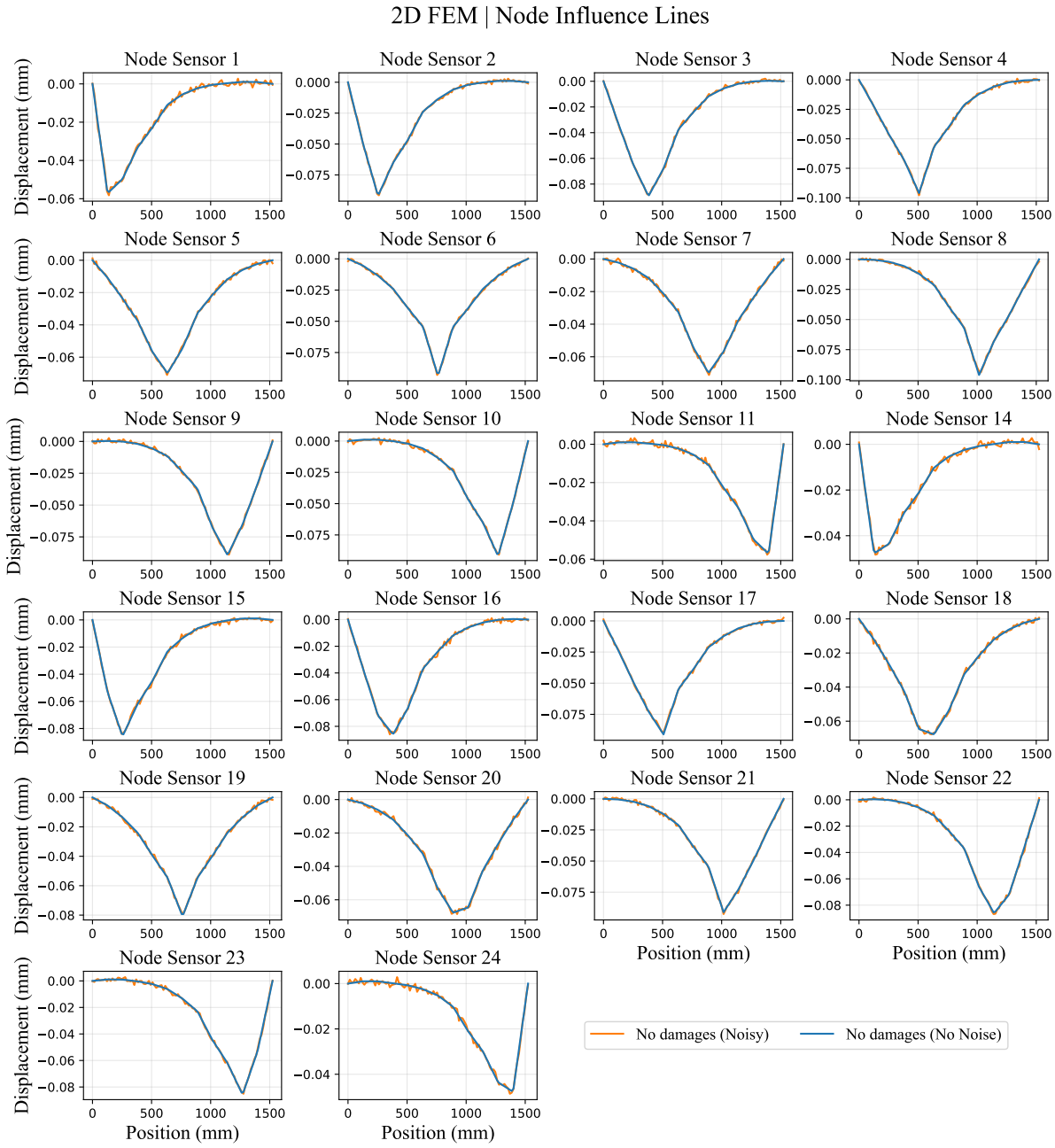
$$F_{\text{axial}} = EA\,\varepsilon \tag{14}$$

To compute influence lines for displacements or axial cable forces, a moving point load is sequentially applied at various positions along the bridge. For each load position, the FEM system is solved and the sensor outputs (displacements or forces) are recorded.

Appendix C. Example 2D FEM Output (Undamaged Case)



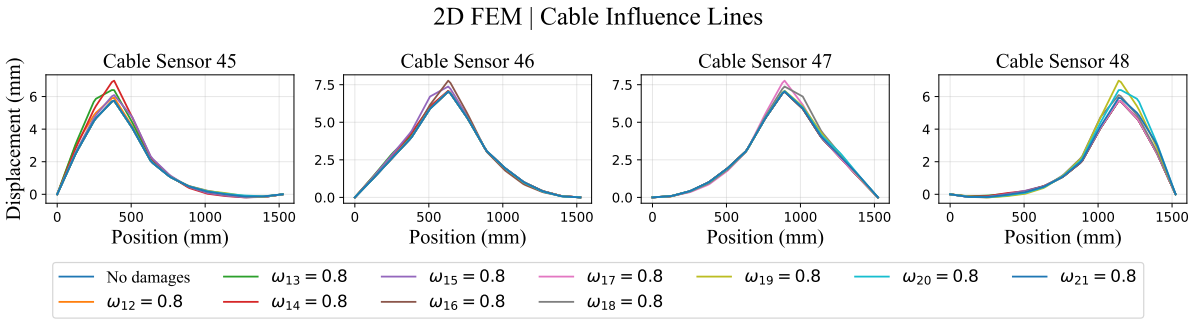
(a) Synthetic (noisy) axial force influence lines of the cables in the mesh depicted in Figure 4.2 in an undamaged scenario.



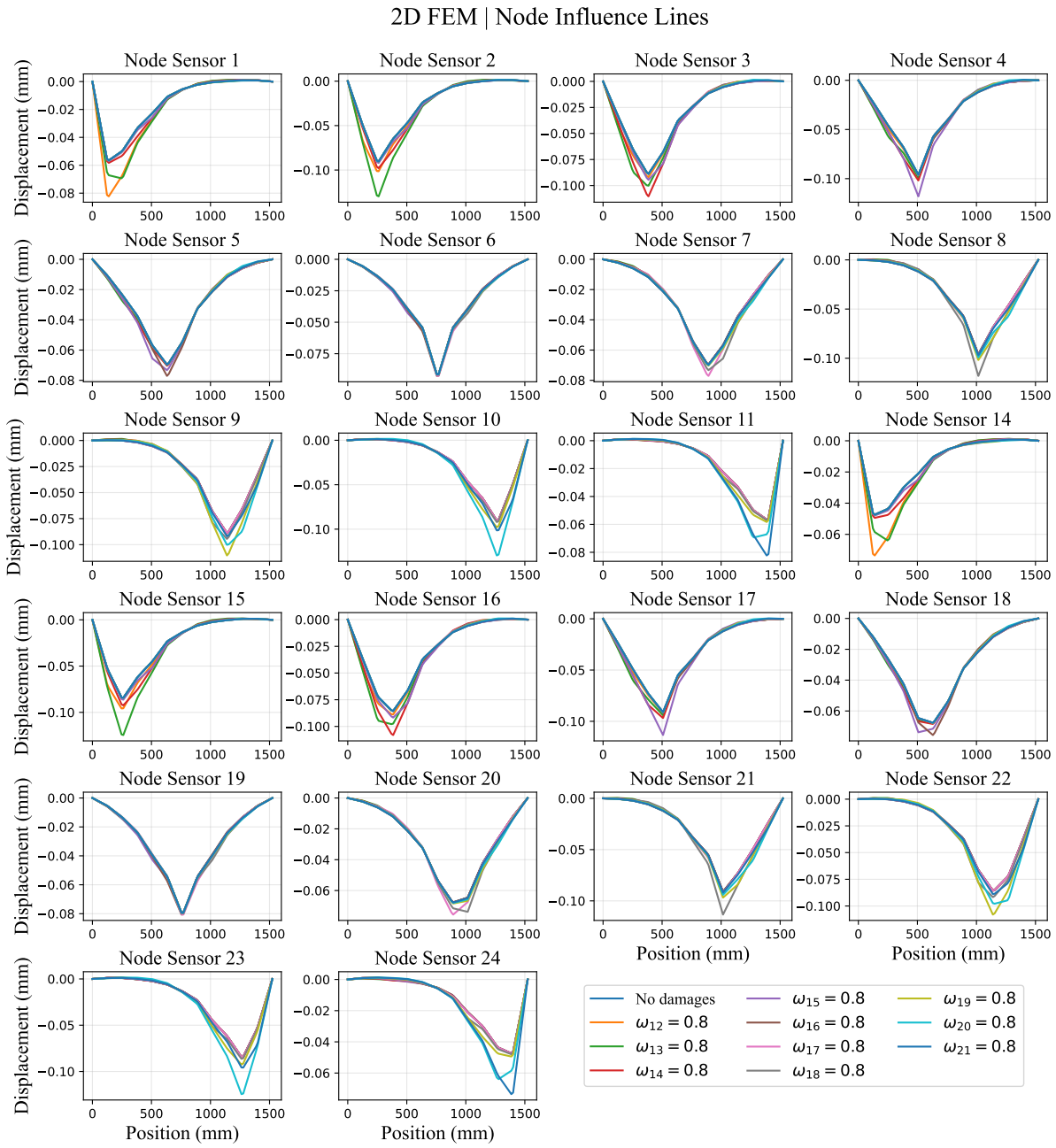
(b) Synthetic (noisy) vertical node displacement influence lines of the nodes in the mesh depicted in Figure 4.2 in an undamaged scenario.

Figure C.1: Synthetic influence lines produced using the 2D FEM and mesh from Figure 4.2 in an undamaged scenario.

Appendix D. Ground Truth Influence Lines (No Noise)



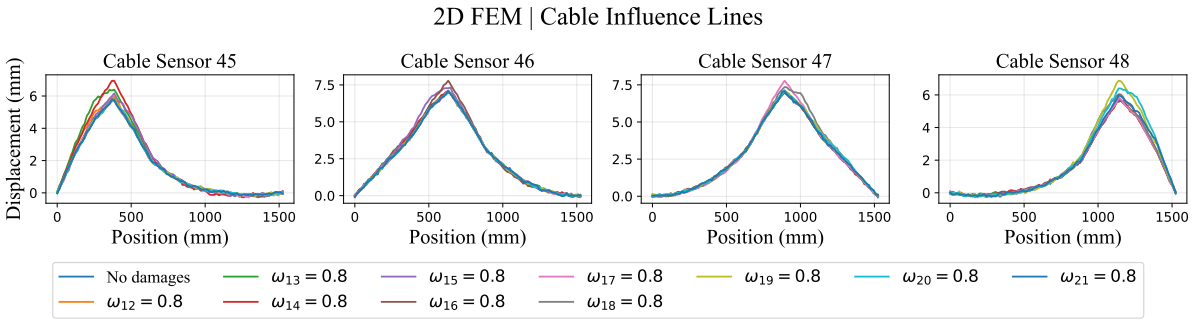
(a) Synthetic cable force influence lines generated by the 2D FEM using the ground truth parameter vectors.



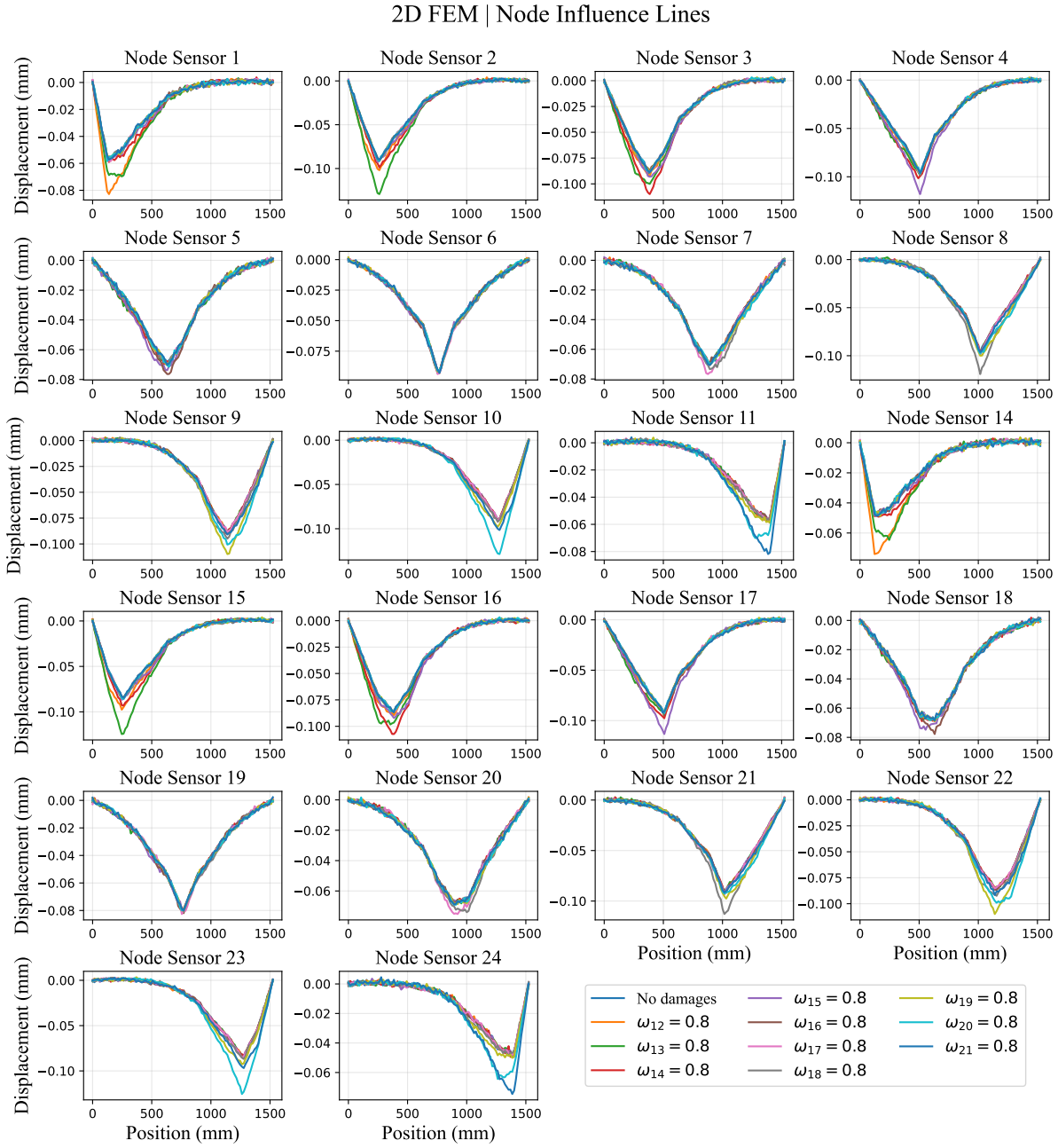
(b) Synthetic node displacement influence lines generated by the 2D FEM using the ground truth parameter vectors.

Figure D.1: Synthetic influence lines produced using the 2D FEM and ground truth parameter vectors.

Appendix E. Ground Truth Influence Lines (With Noise)



(a) Synthetic noisy cable force influence lines generated by the 2D FEM using the ground truth parameter vectors.



(b) Synthetic noisy node displacement influence lines generated by the 2D FEM using the ground truth parameter vectors.

Figure E.1: Synthetic noisy influence lines produced using the 2D FEM and ground truth parameter vectors.

Appendix F. COMSOL Validation of the FEM

A model of the 3D bridge mesh was replicated in COMSOL Multiphysics, shown in Figure F.1. This allowed for the validation of the 3D FEM. As the displacements and cable forces returned by the FEM were identical to those computed in COMSOL, the FEM was concluded to be accurate.

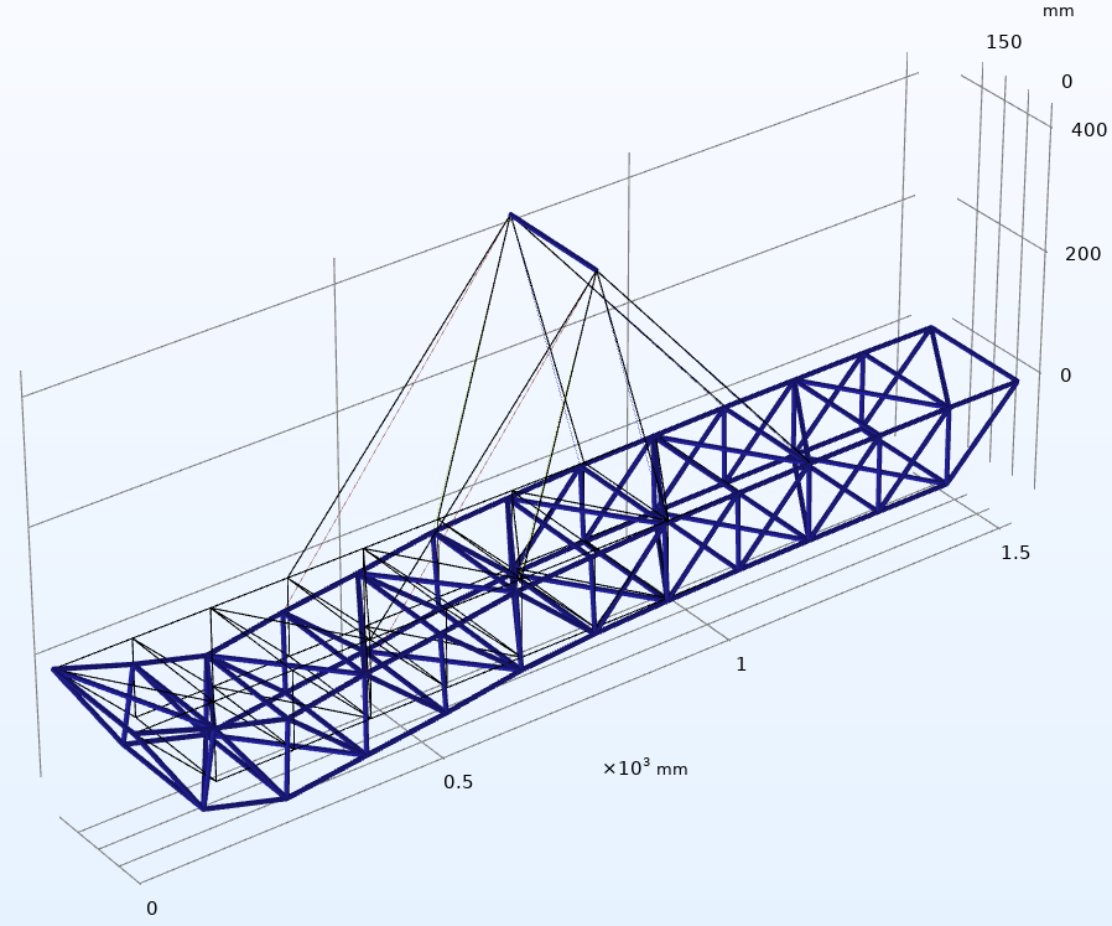


Figure F.1: COMSOL model of the 3D Mesh, used to validate the FEM.

Appendix G. Baseline Summary Network Designs

In this appendix, the baseline summary network architectures used for comparison are described in detail. The input tensor, which is identical for all architectures, consists of B data instances (simulations or observations) making up a batch. Each batch is made up of data $x \in \mathbb{R}^{B \times S \times N}$, a collection of B matrices in which each row is the influence line of sensor $s = 1, \ldots, S$, made up of measurements collected at $n = 1, \ldots, N$ load position. Within each summary network, all operations are vectorized across the batch dimension B. This entails that the same weights are applied to every sample in batch $b = 1, \ldots, B$. To simplify notation and improve clarity, the batch dimension is omitted in the descriptions of the networks. The input tensor to each neural network is therefore simplified to $x \in \mathbb{R}^{S \times N}$.

Appendix G.1 Dense Feedforward Neural Network

The first baseline summary network is a dense, fully connected, feedforward neural network. The architecture of the dense neural begins by flattening the input with a flattening function as defined in (3.8), which transforms a matrix into a vector. The flattening function is applied to an instance of training data $x \in \mathbb{R}^{S \times N}$ according to

$$f_{\text{vec}}(x) = \vec{x} \in \mathbb{R}^{SN} \tag{15}$$

The flattened sample is then passed through a total of L fully connected dense layers, according to

$$\begin{aligned} h_1 &= \sigma(W_1 \vec{x} + b_1) \\ h_2 &= \sigma(W_2 h_1 + b_2) \\ &\vdots \\ \tilde{x} &= \sigma(W_L h_{L-1} + b_L), \end{aligned} \tag{16}$$

where $h_l \in \mathbb{R}^{d_l}$ is the hidden representation output after layer l, W_l is a matrix containing learnable weights for layer l, b_l is a corresponding vector containing learnable biases for layer l, $\sigma(\,)$ is a non-linear activation function, and $\tilde{x} \in \mathbb{R}^{d_s}$ is the summary vector. Note that the sizes of W_l and b_l can differ per layer. Their sizes can be generalized according to

$$W_l \in \mathbb{R}^{d_l \times d_{l-1}}, \quad b_l \in \mathbb{R}^{d_l}, \quad d_0 = SN, \quad d_L = d_s \tag{17}$$

Appendix G.2 Sequential Neural Network

The second baseline network is referred to as a sequential network. The input is treated as a collection of S independent 1D sequences of length N, with no interaction between them. Each sensor's influence line $x_s \in \mathbb{R}^N$, corresponding to a row of x, is individually passed through a 1D convolutional block. Within this 1D convolutional block, a 1D convolutional layer is created for each kernel size in a set of kernel sizes $\mathcal{K}$.

In each 1D convolutional layer, corresponding to a kernel size $k \in \mathcal{K}$, a total of F learnable filters are applied separately to each sensors influence line, treating each influence line as an independent 1D signal along the load position axis. The set of filters is shared across all sensors.

Each filter $f \in \{1, \ldots, F\}$ associated with kernel size k is defined by a learnable kernel and a bias term. The 1D convolution operation of filter f over the influence line of sensor s produces a feature map

$$h_s^{(f)} = \sigma(x_s * w^{(f)} + b^{(f)}), \tag{18}$$

where $h_s^{(f)} \in \mathbb{R}^N$ is the feature map produced by filter f for the influence line of sensor s, the convolution operator is denoted as $*$, σ is the non-linear activation function, and $w^{(f)} \in \mathbb{R}^k$ and $b^{(f)} \in \mathbb{R}$ are the learnable kernel and bias of filter f. At each load position n, the output of the convolution is

$$h_s^{(f)}[n] = \sum_{i=0}^{k-1} w_i^{(f)} \, x_{s,\,n-i} + b^{(f)}, \tag{19}$$

where $h_s^{(f)}[n] \in \mathbb{R}$ is the output value at position n for sensor s after applying filter f, $w_i^{(f)} \in \mathbb{R}$ is the i-th weight of the kernel corresponding to filter f, and $x_{s,\,n-i} \in \mathbb{R}$ is the value of the influence line of sensor s at position $n - i$. To ensure that the convolution is well-defined at the boundaries of the input sequence, causal padding is applied such that $x_{s,\,n-i} = 0 \quad \text{if } n - i < 1$. This allows the output $h_s^{(f)}[n]$ to be computed for all $n \in 1, \ldots, N$, ensuring that the feature map has the same length as the original influence line.

Since each sensors influence line is processed independently, applying a single filter f with kernel size k produces a single feature map for each sensor

$$h_s^{(f)} \in \mathbb{R}^N, \quad \text{for } s = 1, \dots, S \tag{20}$$

By collecting the feature maps across all sensors, we obtain a 2D matrix for each filter

$$h^{(f)} = \begin{bmatrix} h_1^{(f)} & h_2^{(f)} & \dots & h_S^{(f)} \end{bmatrix} \in \mathbb{R}^{N \times S}, \tag{21}$$

where each column corresponds to the feature map of one sensor. After applying all F filters for kernel size k, we obtain a 3D tensor

$$h^{(k)} \in \mathbb{R}^{N \times S \times F} \tag{22}$$

where the third dimension stacks the feature maps across filters. Finally, for all kernel sizes $k \in \mathcal{K}$, the outputs $h^{(k)}$ are concatenated along the feature dimension

$$h = \text{concat}(h^{(k)} \; \forall \; k \in \mathcal{K}) \in \mathbb{R}^{N \times S \times (F|\mathcal{K}|)} \tag{23}$$

This tensor contains multiple learned features for each sensor and load position, extracted using multiple kernel sizes, allowing the network to capture patterns at different scales within the influence lines. Multiple convolutional blocks may be stacked sequentially to extract increasingly abstract features from the input data. To prepare the features for temporal processing by a recurrent layer, the sensor and feature dimensions are flattened into a single combined feature axis. This is done by reshaping the tensor to

$$h = (h_1, \dots, h_N) \in \mathbb{R}^{N \times D}, \tag{24}$$

where notation has been simplified by using $SF|\mathcal{K}| = D$. In this reshaped form, the load position dimension N is preserved, and for each load position the corresponding entry $h_n \in \mathbb{R}^D$ is a concatenation of all feature representations of all sensors.

This flattening operation ensures that the LSTM receives a standard input shape where sequential dependencies across load positions can be extracted while still retaining information from all sensors and all convolutional filters. The LSTM takes the sequence of feature vectors $(h_1, \dots, h_N)$ as an input, and updates its hidden state $\tilde{h}_n \in \mathbb{R}^U$ (with U the number of hidden units) and cell state $c_n \in \mathbb{R}^U$ at each load position $n \in \{1, \dots, N\}$ as

$$\begin{aligned} i_n &= \sigma\left(W_i h_n + R_i \tilde{h}_{n-1} + b_i\right) \\ f_n &= \sigma\left(W_f h_n + R_f \tilde{h}_{n-1} + b_f\right) \\ o_n &= \sigma\left(W_o h_n + R_o \tilde{h}_{n-1} + b_o\right) \\ \tilde{c}_n &= \tanh\left(W_c h_n + R_c \tilde{h}_{n-1} + b_c\right) \\ c_n &= f_n \odot c_{n-1} + i_n \odot \tilde{c}_n \\ \tilde{h}_n &= o_n \odot \tanh(c_n), \end{aligned} \tag{25}$$

where $h_n \in \mathbb{R}^D$ is the input feature vector at load position n, $\tilde{h}_n \in \mathbb{R}^U$ is the hidden state, $c_n \in \mathbb{R}^U$ is the cell state, $i_n, f_n, o_n \in \mathbb{R}^U$ are the input, forget, and output gates, respectively, $\tilde{c}_n \in \mathbb{R}^U$ is the candidate cell state, σ is the sigmoid activation function, $\odot$ denotes element-wise multiplication, $W \in \mathbb{R}^{U \times D}$ are input weight matrices, $R \in \mathbb{R}^{U \times U}$ are recurrent weight matrices, and $b \in \mathbb{R}^U$ are the bias vectors.

After the processing of the full sequence, the final hidden state $\tilde{h}_N \in \mathbb{R}^U$ serves as the output of the LSTM. This is then passed to a fully connected dense layer with a linear activation function to reduce the size of the summary vector to the desired dimension, i.e.,

$$\tilde{x} = W_{\text{out}} \, \tilde{h}_N + b_{\text{out}} \in \mathbb{R}^{d_s}, \tag{26}$$

where $W_{\text{out}} \in \mathbb{R}^{d_s \times U}$ and $b_{\text{out}} \in \mathbb{R}^{d_s}$ are a trainable weight matrix and bias vector.

Appendix G.3 2D Convolutional Neural Network

The 2D-CNN summary network processes influence line data using a series of convolutional blocks, followed by a global average pooling layer. Before applying 2D convolutions, the input is reshaped into a tensor suitable for 2D image processing. Specifically, a channel dimension is added, resulting in

$$x_{\text{img}} \in \mathbb{R}^{S \times N \times 1} \tag{27}$$

The reshaped input is interpreted as a grayscale image with height S, width N, and 1 input channel.

The reformatted input is passed to a 2D convolutional layer, which applies a set of learnable filters to this input tensor. Each filter is a tensor

$$w^{(f)} \in \mathbb{R}^{k_S \times k_N \times C_{\text{in}}}, \tag{28}$$

where k_S and k_N are the kernel sizes along the sensor and load position axes, respectively, and C_{in} is the number of input channels. Initially, $C_{\text{in}} = 1$. The convolution operation slides each filter across the sensor and load axes of the input, computing a weighted sum at each spatial location. For a given filter f, the output value at location $(s, n) \in \{1, \dots, S\} \times \{1, \dots, N\}$ is given by

$$h^{(f)}[s, n] = \sum_{i=0}^{k_S-1} \sum_{j=0}^{k_N-1} \sum_{c=0}^{C_{\text{in}}-1} w^{(f)}_{i,j,c} \, x_{\text{img},\, s+i,\, n+j,\, c} + b^{(f)}, \tag{29}$$

where $b^{(f)} \in \mathbb{R}$ is the bias term associated with filter f, and zero-padding is applied if necessary to ensure that the convolution is defined near the boundaries. The result of applying this operation across the entire spatial domain is a feature map $h^{(f)} \in \mathbb{R}^{S \times N}$. All filters operate independently, and their outputs are stacked to form the full output tensor of the convolutional layer, i.e.,

$$h \in \mathbb{R}^{S \times N \times C_{\text{out}}}, \tag{30}$$

where C_{out} is the number of filters in the layer. A non-linear activation function is applied element-wise to this output, which introduces non-linearity into the model, enabling it to represent complex mappings from input to output, i.e.,

$$h_{\text{activated}} = \sigma(h), \tag{31}$$

where σ is the nonlinear activation function. This process of convolution followed by activation transforms the input into a richer set of local features, learned from spatially localized patterns across sensors and load positions. By stacking multiple such convolutional blocks the network progressively builds more abstract representations, capable of capturing complex patterns in the influence line data.

To reduce the output of the convolutional layers into a fixed-size feature vector while retaining the most essential information, a global average pooling operation is applied. This operation computes the mean value of each feature map across the entire spatial domain. That is, for each channel $c \in \{1, \dots, C^*\}$, the pooled output is

$$z_c = \frac{1}{S^* N^*} \sum_{s=1}^{S^*} \sum_{n=1}^{N^*} h[s, n, c], \tag{32}$$

resulting in a vector $z \in \mathbb{R}^{C^*}$. The final summary vector is produced by a dense output layer, according to

$$\tilde{x} = \sigma(W_{\text{out}} z + b_{\text{out}}), \tag{33}$$

with $W_{\text{out}} \in \mathbb{R}^{d_s \times C^*}$ and $b_{\text{out}} \in \mathbb{R}^{d_s}$, resulting in the desired summary vector $\tilde{x} \in \mathbb{R}^{d_s}$. All operations are vectorized across the batch dimension.

Appendix H. Summary Network Hyperparameters

Table H.1: Hyperparameters of the Summary Network Architectures.

Model	Hyperparameter	Value / Description
Dense	Activation	ReLU
	Layer 1 Weights	$W_1 \in \mathbb{R}^{256\times(SN)}$
	Layer 2 Weights	$W_2 \in \mathbb{R}^{128\times256}$
	Layer 3 Weights	$W_3 \in \mathbb{R}^{d_s\times128}$
	Dropout rate	0.1 (after each hidden layer)
Sequential	Activation	ReLU
	Conv Layers	2
	Padding	same
	Filters	$F = 32$
	Kernel sizes	$k \in \{1, 2\}$
	Conv weights	$W_{\text{conv}}^{(f)} \in \mathbb{R}^k,\ \forall f = 1, \ldots, F$
	Conv biases	$b_{\text{conv}}^{(f)} \in \mathbb{R},\ \forall f = 1, \ldots, F$
	LSTM units	$U = 128$
	Dense output	$\mathbb{R}^{d_s\times U}$
2D-CNN	Activation	ReLU
	Conv Layers	4
	Conv filters	{8, 16, 32, 64}
	Kernel size	3×3
	Strides	{(2,2), (2,2), (2,2), (1,1)}
	Padding	same
	Global Avg. Pooling	After final Conv2D layer
	Dense Layer 1 & 2	128, d_s
GSNN	Activation (Message)	ReLU
	Output dim (Message)	50
	Connectivity dictionary	Provided externally
	Message passing weights	$W_{\text{self}}, W_{\text{msg}} \in \mathbb{R}^{F\times50}$
	Bias vector	$b \in \mathbb{R}^{50}$
	Activation (Output)	ReLU
	Dense weights (Output)	$W_{\text{out}} \in \mathbb{R}^{d_s\times(50S)}$
	Dense biases (Output)	$b_{\text{out}} \in \mathbb{R}^{d_s}$

Appendix I. Example Training Loss Plot

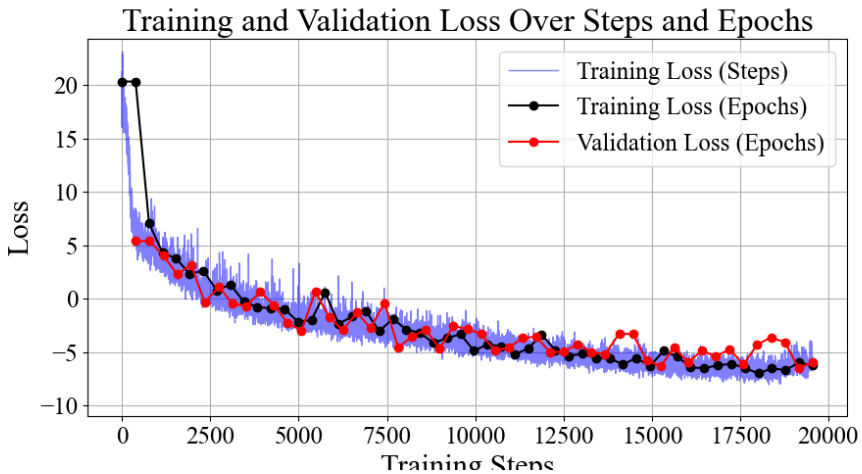


Figure I.1: Typical training loss over epochs plot.

Appendix J. Application to Real-Life Scenario

This appendix contains the first steps towards the application of the method described in this work to the real-life bridge demonstrator. The real demonstrator is equipped with 6 vertical displacement sensors and 8 cable force sensors.

Appendix J.1 Data Collection and Processing

All sensors are activated, and the self-driving vehicle is turned on. Once the car has fully crossed the bridge, the sensors are deactivated. The resulting measurements are shown in Figure J.1.

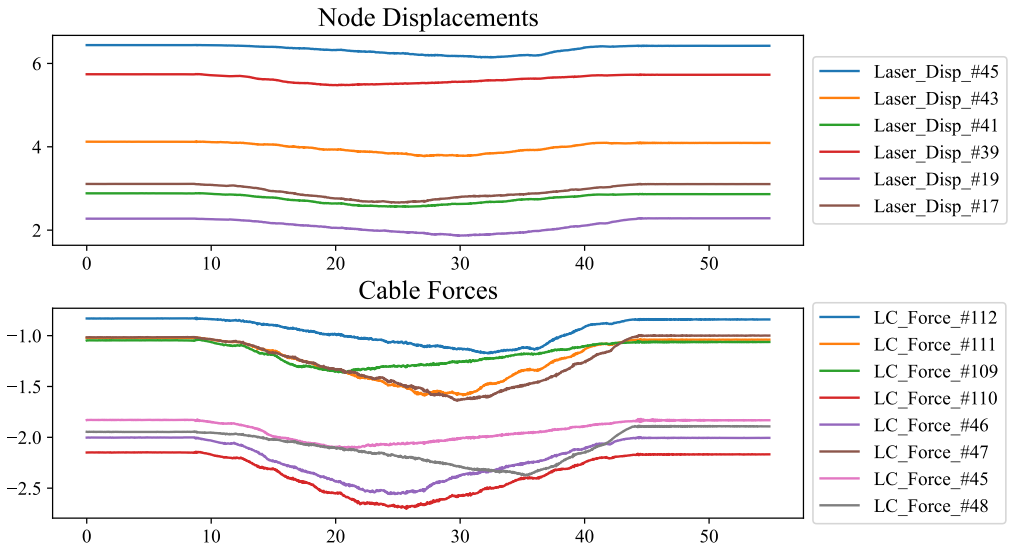


Figure J.1: Original measurements.

The first step is to automatically detect the timestamps at which the car drives onto the bridge and exits the bridge. By cropping off the measurements taken outside of this time frame, the resulting measurement resembles an influence line. To automatically detect the start and end of the experiment, we first filter the signals. The result is shown in Figure J.2.

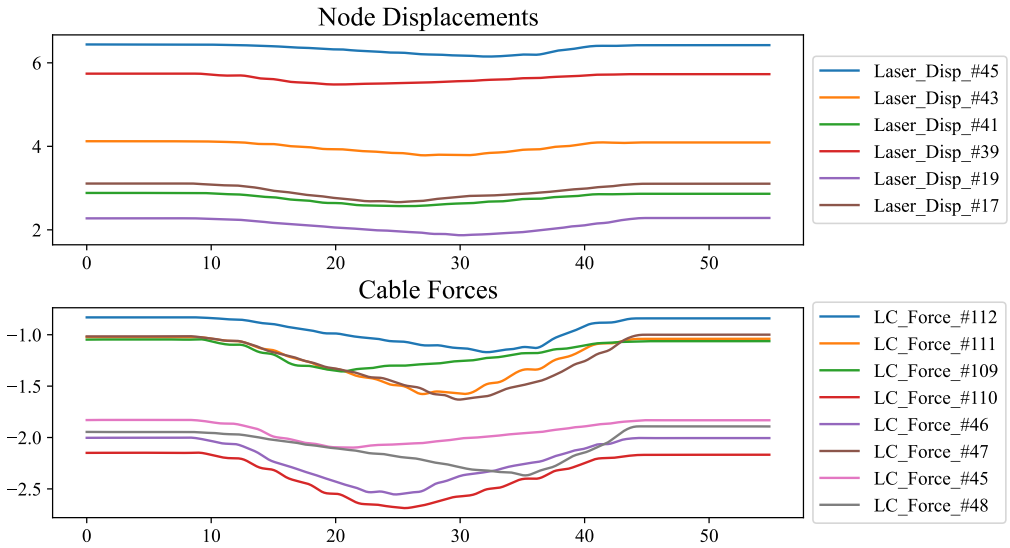


Figure J.2: Filtered measurements (for determining the cutoff timestamps).

Next, the time stamps at which the car enters and exits the bridge were determined, using the deviations of sensor signals from their initial steady state, combined with stabilization criteria at the end, to identify the start and cutoff indices per measurement. These indices were then averaged across sensors (after removing outliers) and mapped to the corresponding adjusted time stamps for plotting. Results are shown in Figure J.3.

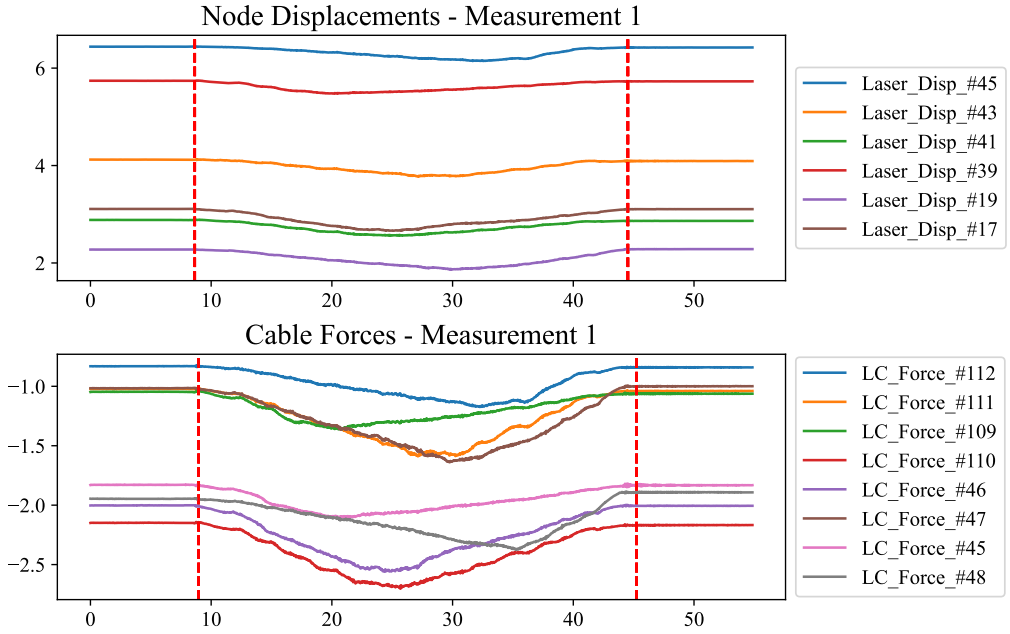


Figure J.3: Original measurements with cutoff timestamps.

Next, the bias was removed by subtracting the mean sensor value recorded before the car entered the bridge. For each measurement, the mean up to the detected no-movement index was computed per sensor and subtracted from the entire signal. This ensured that all signals were zero-centered at the start of the experiment. Results are shown in Figure J.4.

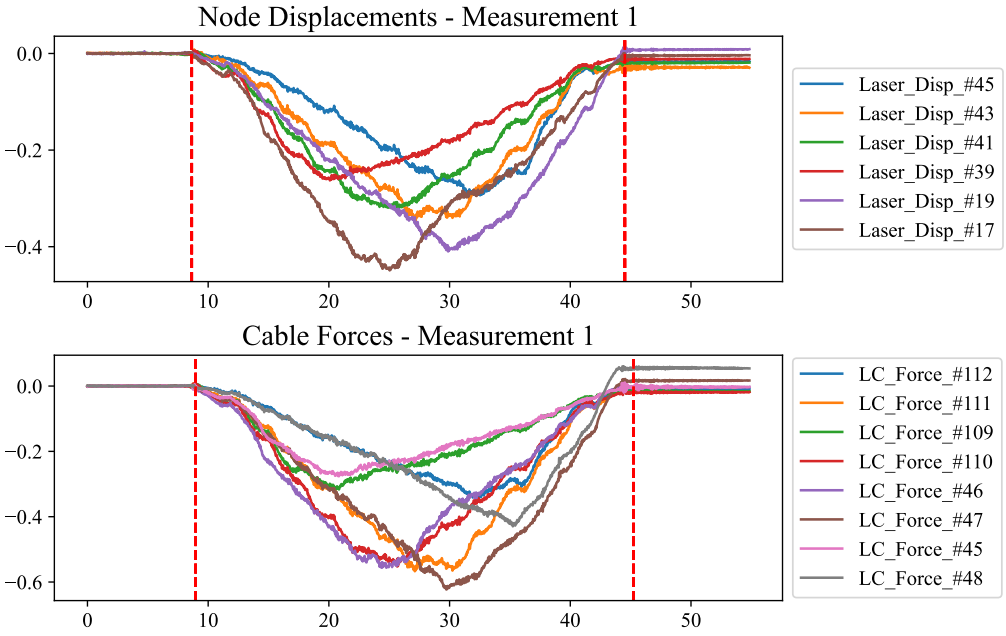


Figure J.4: Unbiased measurements with cutoff timestamps.

Next, a filter was applied to the measurement data, removing the worst sensor noise. The results can be found in Figure J.5.

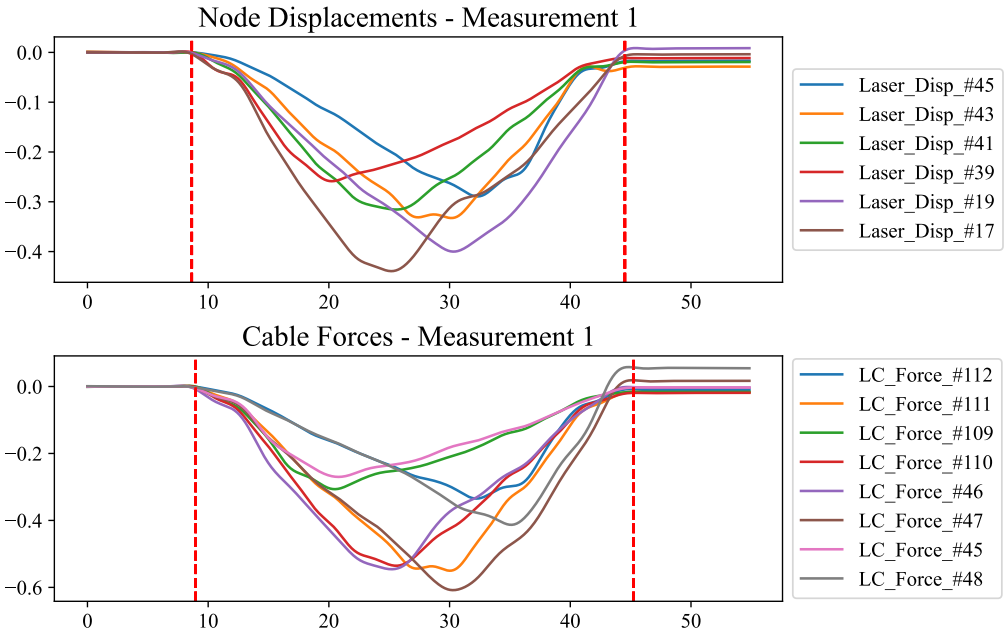


Figure J.5: Unbiased, filtered measurements with cutoff timestamps.

Finally, the values outside of the relevant time stamps were removed from the data, yielding influence line-like data for each sensor. The results can be found in Figure J.6.

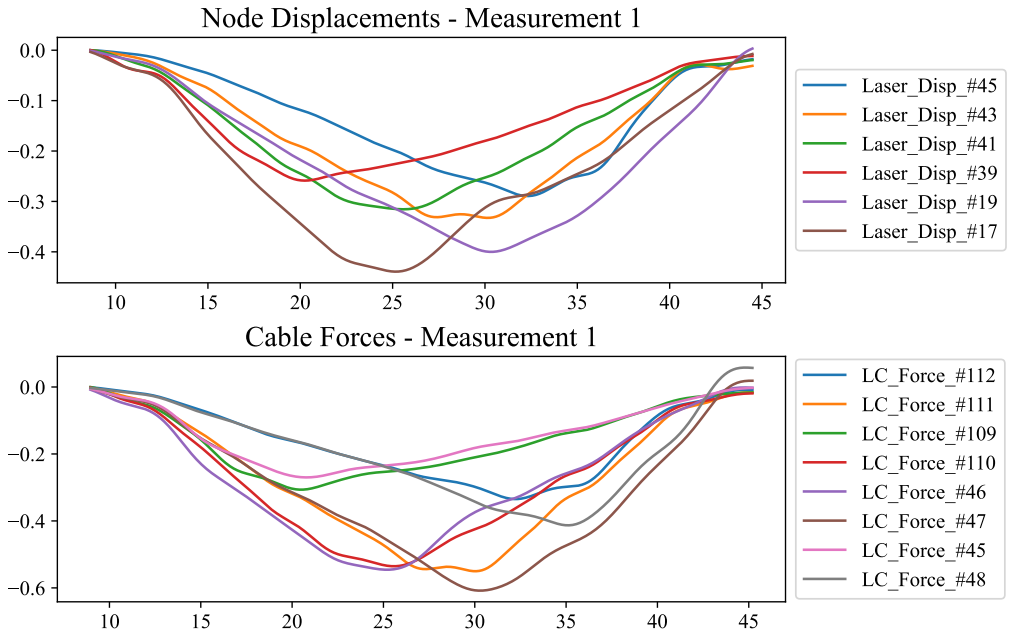


Figure J.6: Processed measurements emulating influence lines.

Appendix J.2 Model Misspecification

A FEM model was created to generate training simulations for the real-life scenario. This FEM was based on the same information used in the synthetic scenarios covered in this work, with a number of tweaks to ensure it closer emulates reality. The base stiffness of the materials was identified, as displayed in Figure J.7.

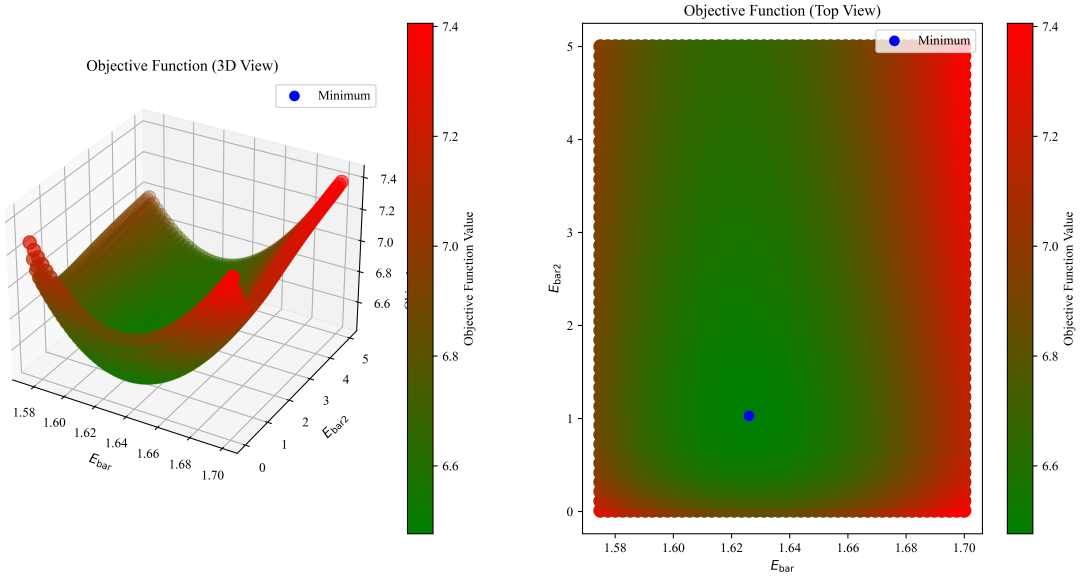


Figure J.7: Identification of Suitable Base Stiffnesses

However, the linear FEM was not able to capture the non-linear and irregular behaviour of the real bridge. Therefore, a neural network was trained to correct any residuals, attempting to close the gap between the modeled and simulated responses. The results can be seen in Figure J.8, in which the simulated responses (dotted lines) and the measured responses (solid lines) are plotted for various damage scenarios.

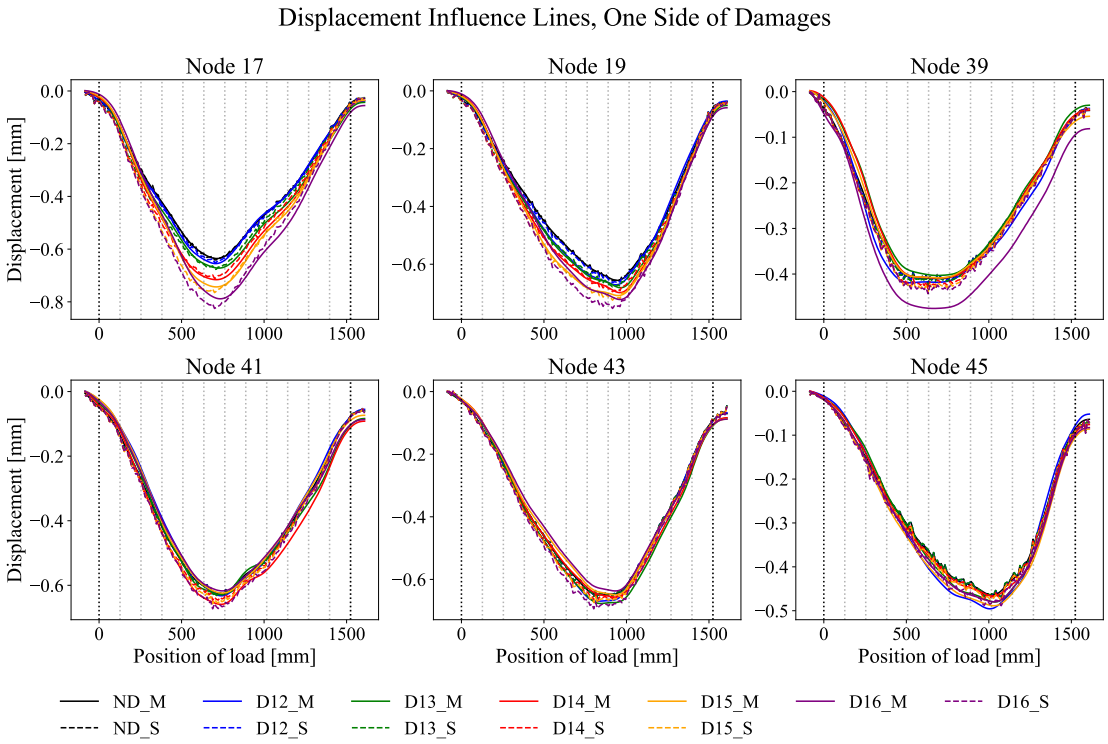


Figure J.8: Measured vs Simulated Responses of the Bridge Demonstrator

Unfortunately, the match between the real data generation process and the simulator was not close enough to perform proper damage detection using BayesFlow. This highlights the critical importance of accurate modeling. While the neural network correction helped to reduce systematic residuals, it could not fully compensate for structural dynamics and complex boundary effects that were absent from the FEM model. As a result, the inference task became ill-posed, underscoring the need for either more expressive physics-based models or hybrid approaches that integrate richer data-driven corrections.

References

[1] X. W. Ye et al. "Statistical analysis of stress spectra for fatigue life assessment of steel bridges with structural health monitoring data". In: *Engineering Structures* 45 (Dec. 2012), pp. 166–176. ISSN: 01410296. DOI: 10.1016/j.engstruct.2012.06.016.

[2] Lu Deng, Wangchen Yan, and Lei Nie. "A simple corrosion fatigue design method for bridges considering the coupled corrosion-overloading effect". In: *Engineering Structures* 178 (Jan. 2019), pp. 309–317. ISSN: 18737323. DOI: 10.1016/j.engstruct.2018.10.028.

[3] Luca Capacci, Fabio Biondini, and Dan M. Frangopol. "Resilience of aging structures and infrastructure systems with emphasis on seismic resilience of bridges and road networks: Review". In: *Resilient Cities and Structures* 1.2 (June 2022), pp. 23–41. ISSN: 27727416. DOI: 10.1016/j.rcns.2022.05.001.

[4] Hua-Peng Chen and Yi-Qing Ni. *Structural Health Monitoring of Large Civil Engineering Structures*. Wiley, Mar. 2018. ISBN: 9781119166641. DOI: 10.1002/9781119166641.

[5] Piervincenzo Rizzo and Alireza Enshaeian. *Challenges in bridge health monitoring: A review*. July 2021. DOI: 10.3390/s21134336.

[6] Funda Yavuz, Upul Attanayake, and Haluk Aktan. "Economic Impact on Surrounding Businesses due to Bridge Construction". In: *Procedia Computer Science*. Vol. 109. Elsevier B.V., 2017, pp. 108–115. DOI: 10.1016/j.procs.2017.05.301.

[7] George C. Lee et al. *A Study of U.S. Bridge Failures (1980-2012)*. Tech. rep. Buffalo State University, June 2013.

[8] Guglielmo Mattioli. *What caused the Genoa bridge collapse-and the end of an Italian national myth?* English. Feb. 2019. URL: https://www.theguardian.com/cities/2019/feb/26/what-caused-the-genoa-morandi-bridge-collapse-and-the-end-of-an-italian-national-myth.

[9] Andrés Martinez and Christos Lathourakis. "Bayesian model-based damage detection in engineering systems using Invertible Neural Networks". In: *11th European Workshop on Structural Health Monitoring, EWSHM 2024*. NDT.net, 2024. DOI: 10.58286/29809.

[10] Vahid Reza Gharehbaghi et al. *A Critical Review on Structural Health Monitoring: Definitions, Methods, and Perspectives*. June 2022. DOI: 10.1007/s11831-021-09665-9.

[11] Vincenzo Gattulli and Leonardo Chiaramonte. "Condition assesment by visual inspection for a bridge management system". In: *Computer-Aided Civil and Infrastructure Engineering* 20.2 (2005), pp. 95–107. ISSN: 10939687. DOI: 10.1111/j.1467-8667.2005.00379.x.

[12] Hong Nan Li et al. "State-of-the-art in structural health monitoring of large and complex civil infrastructures". In: *Journal of Civil Structural Health Monitoring* 6.1 (Feb. 2016), pp. 3–16. ISSN: 21905479. DOI: 10.1007/s13349-015-0108-9.

[13] Elise Kaartinen, Kyle Dunphy, and Ayan Sadhu. *LiDAR-Based Structural Health Monitoring: Applications in Civil Infrastructure Systems*. June 2022. DOI: 10.3390/s22124610.

[14] Majdi Flah et al. "Machine Learning Algorithms in Civil Structural Health Monitoring: A Systematic Review". In: *Archives of Computational Methods in Engineering* 28.4 (June 2021), pp. 2621–2643. ISSN: 18861784. DOI: 10.1007/s11831-020-09471-9.

[15] Alexandre Cury et al., eds. *Structural Health Monitoring Based on Data Science Techniques*. Vol. 21. Structural Integrity. Cham: Springer International Publishing, 2022. ISBN: 978-3-030-81715-2. DOI: 10.1007/978-3-030-81716-9. URL: https://link.springer.com/10.1007/978-3-030-81716-9.

[16] Francisco J. Carrión, Juan A. Quintana, and Saúl E. Crespo. "SHM of a stayed bridge during a structural failure, case study: the Rio Papaloapan Bridge". In: *Journal of Civil Structural Health Monitoring* 7.2 (Apr. 2017), pp. 139–151. ISSN: 21905479. DOI: 10.1007/s13349-017-0221-z.

[17] Mayank Mishra, Paulo B. Lourenço, and G. V. Ramana. *Structural health monitoring of civil engineering structures by using the internet of things: A review*. May 2022. DOI: 10.1016/j.jobe.2021.103954.

[18] Yingtao Liu and Subhadarshi Nayak. "Structural health monitoring: State of the art and perspectives". In: *JOM* 64.7 (July 2012), pp. 789–792. ISSN: 10474838. DOI: 10.1007/s11837-012-0370-9.

[19] Gaia Pianigiani and Joanna Berendt. *Poor Maintenance and Construction Flaws Are Cited in Italy Bridge Collapse*. Oct. 2020.

[20] A. Gallet et al. *Structural engineering from an inverse problems perspective*. 2022. DOI: 10.1098/rspa.2021.0526.

[21] Richard C. Aster, Brian Borchers, and Clifford H. Thurber. "Parameter Estimation and Inverse Problems - Introduction". In: *Parameter Estimation and Inverse Problems*. Elsevier, 2013, pp. 1–23. DOI: 10.1016/b978-0-12-385048-5.00001-x.

[22] Michael U. Gutmann et al. "Likelihood-free inference via classification". In: *Statistics and Computing* 28.2 (Mar. 2018), pp. 411–425. ISSN: 15731375. DOI: 10.1007/s11222-017-9738-6.

[23] Christopher Drovandi and David T. Frazier. "A comparison of likelihood-free methods with and without summary statistics". In: *Statistics and Computing* 32.3 (June 2022). ISSN: 15731375. DOI: 10.1007/s11222-022-10092-4.

[24] Kyle Cranmer, Johann Brehmer, and Gilles Louppe. "The frontier of simulation-based inference". In: *Proceedings of the National Academy of Sciences of the United States of America* 117.48 (Dec. 2020), pp. 30055–30062. ISSN: 10916490. DOI: 10.1073/pnas.1912789117.

[25] Mikael Sunnåker et al. "Approximate Bayesian Computation". In: *PLoS Computational Biology* 9.1 (2013). ISSN: 15537358. DOI: 10.1371/journal.pcbi.1002803.

[26] S. A. Sisson, Y. Fan, and M. A. Beaumont. "Overview of Approximate Bayesian Computation". In: (Feb. 2018). URL: http://arxiv.org/abs/1802.09720.

[27] Stefan T Radev et al. "BayesFlow: Amortized Bayesian Workflows With Neural Networks". In: (June 2023). URL: http://arxiv.org/abs/2306.16015.

[28] Stefan T Radev et al. *JANA: Jointly Amortized Neural Approximation of Complex Bayesian Models*. Tech. rep. 2023.

[29] Stefan T. Radev et al. "BayesFlow: Learning complex stochastic models with invertible neural networks". In: (Mar. 2020). URL: http://arxiv.org/abs/2003.06281.

[30] Zhiqiang Gong, Ping Zhong, and Weidong Hu. "Diversity in Machine Learning". In: *IEEE Access* 7 (2019), pp. 64323–64350. ISSN: 21693536. DOI: 10.1109/ACCESS.2019.2917620.

[31] Hyontai Sug. "Performance of Machine Learning Algorithms and Diversity in Data". In: *MATEC Web of Conferences*. Vol. 210. EDP Sciences, Oct. 2018. DOI: 10.1051/matecconf/201821004019.

[32] Song Cen et al. *Some advances in high-performance finite element methods*. Oct. 2019. DOI: 10.1108/EC-10-2018-0479.

[33] O C Zienkiewicz, R L Taylor, and J Z Zhu. *The Finite Element Method: Its Basis and Fundamentals Sixth edition*. Tech. rep. URL: www.cimne.upc.es.

[34] George Em Karniadakis et al. *Physics-informed machine learning*. June 2021. DOI: 10.1038/s42254-021-00314-5.

[35] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. "Density estimation using Real NVP". In: (Feb. 2017). URL: http://arxiv.org/abs/1605.08803.

[36] Lynton Ardizzone et al. "Guided Image Generation with Conditional Invertible Neural Networks". In: (July 2019). URL: http://arxiv.org/abs/1907.02392.

[37] Iain M. Johnstone and D. Michael Titterington. *Statistical challenges of high-dimensional data*. Nov. 2009. DOI: 10.1098/rsta.2009.0159.

[38] Naomi Altman and Martin Krzywinski. *The curse(s) of dimensionality this-month*. June 2018. DOI: 10.1038/s41592-018-0019-x.

[39] Dennis Prangle. "Summary Statistics in Approximate Bayesian Computation". In: (Dec. 2015). URL: http://arxiv.org/abs/1512.05633.

[40] Amir Globerson. *Sufficient Dimensionality Reduction Naftali Tishby*. Tech. rep. 2003, pp. 1307–1331.

[41] D. Goyal and B. S. Pabla. "The Vibration Monitoring Methods and Signal Processing Techniques for Structural Health Monitoring: A Review". In: *Archives of Computational Methods in Engineering* 23.4 (Dec. 2016), pp. 585–594. ISSN: 18861784. DOI: 10.1007/s11831-015-9145-0.

[42] M. I. Friswell. "Inverse methods in structural health monitoring". In: *Key Engineering Materials* 204-205 (2001), pp. 201–210. ISSN: 16629795. DOI: 10.4028/www.scientific.net/kem.204-205.201.

[43] Spilios D. Fassois and John S. Sakellariou. "Time-series methods for fault detection and identification in vibrating structures". In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 365.1851 (Feb. 2007), pp. 411–448. ISSN: 1364503X. DOI: 10.1098/rsta.2006.1929.

[44] Hervé Abdi and Lynne J. Williams. *Principal component analysis*. July 2010. DOI: 10.1002/wics.101.

[45] Sasan Karamizadeh et al. "An Overview of Principal Component Analysis". In: *Journal of Signal and Information Processing* 04.03 (2013), pp. 173–175. ISSN: 2159-4465. DOI: 10.4236/jsip.2013.43b031.

[46] Joseph Pateras, Pratip Rana, and Preetam Ghosh. *A Taxonomic Survey of Physics-Informed Machine Learning*. June 2023. DOI: 10.3390/app13126892.

[47] Zhongkai Hao et al. "Physics-Informed Machine Learning: A Survey on Problems, Methods and Applications". In: (Mar. 2023). URL: http://arxiv.org/abs/2211.08064.

[48] Chuizheng Meng et al. "When physics meets machine learning: a survey of physics-informed machine learning". In: *Machine Learning for Computational Science and Engineering* 1.1 (June 2025). ISSN: 3005-1428. DOI: 10.1007/s44379-025-00016-0.

[49] 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". In: *Journal of Computational Physics* 378 (Feb. 2019), pp. 686–707. ISSN: 10902716. DOI: 10.1016/j.jcp.2018.10.045.

[50] Stefan Bloemheuvel, Jurgen van den Hoogen, and Martin Atzmueller. "A computational framework for modeling complex sensor network data using graph signal processing and graph neural networks in structural health monitoring". In: *Applied Network Science* 6.1 (Dec. 2021). ISSN: 23648228. DOI: 10.1007/s41109-021-00438-8.

[51] Thomas Titscher et al. "Bayesian model calibration and damage detection for a digital twin of a bridge demonstrator". In: *Engineering Reports* 5.11 (Nov. 2023). ISSN: 25778196. DOI: 10.1002/eng2.12669.