



SCUOLA INTERNAZIONALE SUPERIORE DI STUDI AVANZATI

SISSA Digital Library

GFN: A graph feedforward network for resolution-invariant reduced operator learning in multifidelity applications

*Original*

GFN: A graph feedforward network for resolution-invariant reduced operator learning in multifidelity applications / Morrison, Oisín M.; Pichi, Federico; Hesthaven, Jan S.. - In: COMPUTER METHODS IN APPLIED MECHANICS AND ENGINEERING. - ISSN 0045-7825. - 432:(2024). [10.1016/j.cma.2024.117458]

*Availability:*

This version is available at: 20.500.11767/142750 since: 2024-11-08T17:10:31Z

*Publisher:*

*Published*

DOI:10.1016/j.cma.2024.117458

*Terms of use:*

Testo definito dall'ateneo relativo alle clausole di concessione d'uso

*Publisher copyright*  
Elsevier

This version is available for education and non-commercial purposes.

note finali coverpage

(Article begins on next page)

# GFN: A GRAPH FEEDFORWARD NETWORK FOR RESOLUTION-INVARIANT REDUCED OPERATOR LEARNING IN MULTIFIDELITY APPLICATIONS

OISÍN M. MORRISON<sup>1</sup> , FEDERICO PICHI<sup>1,2</sup> , AND JAN S. HESTHAVEN<sup>1</sup> 

**ABSTRACT.** This work presents a novel resolution-invariant model order reduction strategy for multifidelity applications. We base our architecture on a novel neural network layer developed in this work, the graph feedforward network, which extends the concept of feedforward networks to graph-structured data by creating a direct link between the weights of a neural network and the nodes of a mesh, enhancing the interpretability of the network. We exploit the method's capability of training and testing on different mesh sizes in an autoencoder-based reduction strategy for parametrised partial differential equations. We show that this extension comes with provable guarantees on the performance via error bounds. The capabilities of the proposed methodology are tested on three challenging benchmarks, including advection-dominated phenomena and problems with a high-dimensional parameter space. The method results in a more lightweight and highly flexible strategy when compared to state-of-the-art models, while showing excellent generalisation performance in both single fidelity and multifidelity scenarios.

**Keywords:** *graph neural networks, model order reduction, operator learning, resolution invariance, multifidelity surrogate modelling, parametrised PDEs.*

**Code availability:** <https://github.com/Oisin-M/GFN>

## 1. INTRODUCTION AND MOTIVATION

Traditional numerical solvers for the high-fidelity approximation of partial differential equations (PDEs) are computationally prohibitive in real-time and many-query contexts, underpinning the need for quicker evaluations of the numerical solution of PDEs. Reduced order models (ROMs) have arisen as a means to address this issue for parametrised PDEs, accelerating the process via the creation of efficient computational models [10], with fewer degrees of freedom.

Model order reduction (MOR) is therefore concerned with the creation of offline-online surrogate models that are cheaper whilst maintaining high levels of accuracy. One such popular approach is the reduced basis method, characterised by the creation of a reduced space, typically obtained via proper orthogonal decomposition (POD) or greedy methods [46, 79]. Such methods are generally linear, rendering them inefficient for problems exhibiting a slow decay in the Kolmogorov  $n$ -width [36, 73]. To combat this, a range of nonlinear ROM techniques have been developed, with a notable subclass of such methods being machine learning-based approaches. In particular, much promise has been shown in incorporating autoencoder-based architectures into the ROM context [33, 62, 77, 88] due to their feasibility as a nonlinear extension of principal component analysis [57], displaying superior compression capabilities compared to linear methods especially when dealing with problems with slow Kolmogorov  $n$ -width decay [36, 73].

Such autoencoder-based MOR strategies typically benefit from being non-intrusive approaches, removing the requirement for any knowledge of the generation procedure for the training data, therefore leading to possible exploitation of experimental data [30, 31, 59]. Moreover, contrarily to physics-informed neural networks (PINNs) [83] or sparse identification of nonlinear dynamics (SINDy) [14], these are usually not physics-based methods, and hence render it possible to learn the solution's behaviour even without prior knowledge.

<sup>1</sup> CHAIR OF COMPUTATIONAL MATHEMATICS AND SIMULATION SCIENCE, ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE, 1015 LAUSANNE, SWITZERLAND

<sup>2</sup> MATHLAB, MATHEMATICS AREA, SISSA, VIA BONOMEA 265, I-34136 TRIESTE, ITALY.

In this direction, some approaches seek to recover the operator mappings, also known as operator inference, on some fixed discretised domain [56, 75, 78]. Instead of approximating the solution at some fixed locations, advanced neural operators (NOs) go further by directly approximating the solution operator itself [13, 55]. As a result, NOs benefit from the desirable property of being independent from a fixed discretisation, possibly achieving super- and sub-resolution [55, 64, 86, 94].

The development of so-called *resolution-invariant* methods, i.e. methods without a dependence on a fixed discretisation, can be of the utmost importance when dealing with experimental data. This is especially the case in climate modelling, where information depends on both weather stations and satellites, the number and locations of which can vary greatly [25, 26, 72]. These considerations have lead to similar and complementary approaches including the aforementioned NOs, graph neural networks (GNNs) and multifidelity reduction strategies.

Multifidelity ROMs are typically focused on leveraging cheaper computational data, known as low-fidelity data, in addition to more expensive high-fidelity data, to build an efficient offline MOR strategy. Low-fidelity may be interpreted as simplification of the governing equations, low-order approximations, or scarce information, but here we focus on the case of coarser discretisations, aiming to obtain a ROM capable of learning from data on multiple fidelities. State-of-the-art approaches for ROM [19, 39, 52, 70], however, are not generally capable of learning from any arbitrary resolution, but rather from a prespecified fixed set of possible discretisations.

On the other hand, GNNs are capable of leveraging data from graphs of arbitrary cardinality, while preserving the structure of the problem by enforcing geometric priors into the model. A large number of such works (reviewed in [66, 101, 102]) have been focused on the generalisation of pooling, unpooling and convolutional layers to graphs, achieving excellent results across a variety of domains including drug development [93], traffic prediction [49], finance [97], climate [60] and fluid mechanics [9, 44]. Moreover, being designed for unstructured data, GNNs have an advantage over many current state-of-the-art NOs such as Fourier neural operators [64] and convolutional neural operators [86] which require data on structured grids.

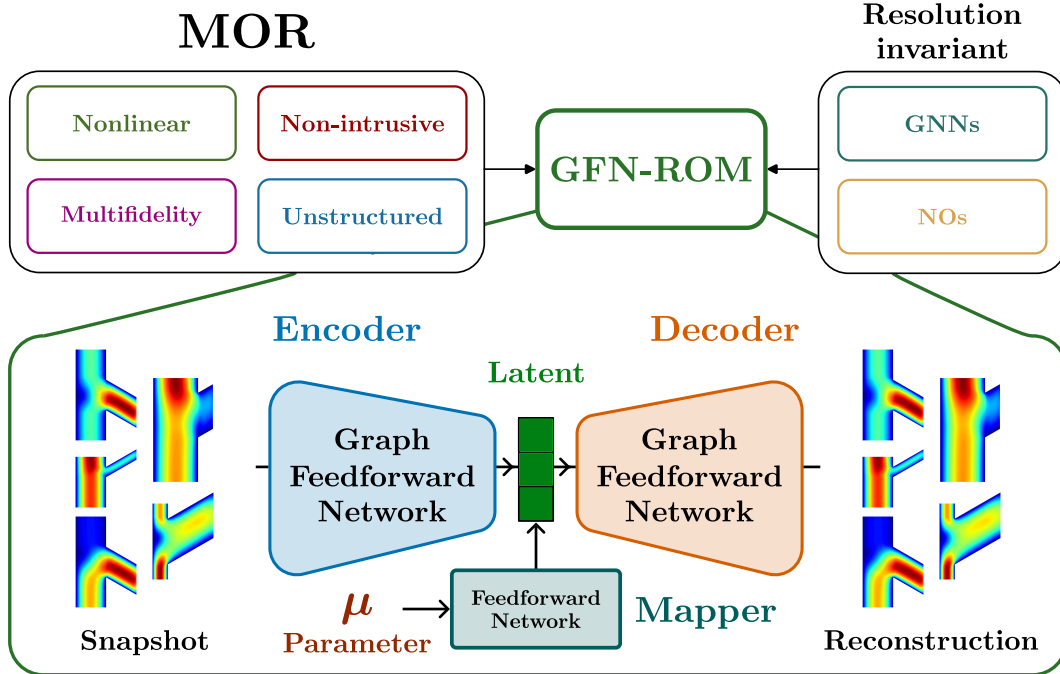


FIGURE 1. GFN-ROM is a nonlinear non-intrusive multifidelity ROM capable of dealing with unstructured data, interplaying between MOR and resolution-invariant techniques.

**Contributions.** In this work, we make the following contributions:

- (i) We introduce a novel neural network layer called the *graph feedforward network* (GFN) which generalises the feedforward layer for data on arbitrary discretisations.
- (ii) We introduce *GFN-ROM*, a new autoencoder-based ROM approach for multifidelity data using GFNs.
- (iii) We provide bounds on the super- and sub-resolution errors using GFNs and GFN-ROM.
- (iv) We demonstrate the remarkable generalisation properties of GFN-ROM through a collection of challenging problems.

To the best of our knowledge, GFN-ROM is the only graph-based resolution-invariant ROM. By associating weights and biases to nodes in a mesh, GFN-ROM also allows for better interpretability compared to approaches such as DL-ROM or GCA-ROM. Furthermore, we show that GFN-ROM is closely related to the standard definitions of NOs and GNNs, by retaining important properties of both concepts (see Figure 1).

The remainder of the paper is organised as follows. Section 2 presents the standard intrusive and non-intrusive framework for MOR. Section 3 defines the challenges in multifidelity applications, while Section 4 introduces resolution invariant methods. The core of the manuscript is presented in Section 5 where the GFN is introduced to leverage data on any discretisation. Section 6 and Section 7 present the exploitation of GFN in the ROM context, and how to perform adaptive multifidelity training. Finally, Section 8 shows the performance of the proposed methodology on complex MOR benchmarks, and Section 9 gives some conclusions and highlights future perspectives.

## 2. REDUCED ORDER MODELS

Full order models are designed to solve PDEs via high-fidelity systems of equations. Such systems involve a large number of degrees of freedom  $N_h$ , which makes their numerical solution costly to compute. In a parametrised PDE context, this is particularly prohibitive since one aims at recovering solutions in real-time for a number of different physical or geometrical configurations, giving rise to a parameter-dependent high-fidelity solution  $\mathbf{u}_h(\boldsymbol{\mu})$ . ROMs seek to mitigate this issue by building and subsequently solving a cheaper reduced system with a much lower number of degrees of freedom  $N \ll N_h$ , giving rise to the reduced order solutions  $\mathbf{u}_N(\boldsymbol{\mu})$ . High-fidelity solutions can then be approximated from the reduced order ones via some transform  $\mathbf{u}_h(\boldsymbol{\mu}) \approx \phi(\mathbf{u}_N(\boldsymbol{\mu}))$ , where  $\phi$  is a linear or nonlinear mapping dependent on the choice of ROM employed. Such methods can be constructed in an intrusive or non-intrusive fashion, depending on whether knowledge of the high-fidelity system is required for the computation of  $\mathbf{u}_N(\boldsymbol{\mu})$ .

**2.1. Projection-based intrusive methods.** Intrusive methods require knowledge of the high-fidelity system. For ease of illustration, we consider the case of a linear high-fidelity system:

$$(1) \quad \mathbf{A}_h(\boldsymbol{\mu})\mathbf{u}_h(\boldsymbol{\mu}) = \mathbf{f}_h(\boldsymbol{\mu}),$$

where  $\boldsymbol{\mu} \in \mathbb{R}^{N_\mu}$  is the parameter,  $\mathbf{A}_h \in \mathbb{R}^{N_h \times N_h}$  is the stiffness matrix,  $\mathbf{u}_h \in \mathbb{R}^{N_h \times N_{VF}}$  is the high-fidelity solution, and  $\mathbf{f}_h \in \mathbb{R}^{N_h \times N_{VF}}$  is the forcing term<sup>1</sup>.

A ROM aims at replacing Equation 1 with a cheaper reduced system. Linear MOR techniques express an approximation of the high-fidelity solution  $\tilde{\mathbf{u}}_h$  as a linear expansion over some chosen basis functions  $\{\boldsymbol{\psi}_i\}_{i=1}^N$  with  $\boldsymbol{\psi}_i \in \mathbb{R}^{N_h}$ . Introducing the matrix  $\mathbf{V} = [\boldsymbol{\psi}_1 | \dots | \boldsymbol{\psi}_N]$  one can concisely write  $\mathbf{u}_h \approx \mathbf{V}\mathbf{u}_N$ . POD is a popular choice for constructing a basis, extracting the principal components from a series of high-fidelity solutions for different parameter realisations, known as snapshots. POD provides the best rank  $N$  subspace for approximating the snapshots in a least-squares sense [69]. Once the set of basis functions has been selected, it remains to calculate the reduced coefficients  $\mathbf{u}_N$ . Intrusive projection-based MOR approaches exploit the knowledge of the high-fidelity system by imposing the projected residual of the reduced order solution onto  $\mathbf{V}$  is zero i.e. one solves for  $\mathbf{u}_N$  the system

$$(2) \quad \mathbf{V}^T(\mathbf{f}_h - \mathbf{A}_h\mathbf{V}\mathbf{u}_N) = \mathbf{0}.$$

---

<sup>1</sup>For ease of notation, in the following we drop the dependence on the parameters  $\boldsymbol{\mu}$ .



An efficient application in the many-query context requires Equation 2 to exhibit an affine parametric dependence in order to circumvent  $\mathbb{R}^{N_h}$ -dependent online operations. This is usually not fulfilled when dealing with complex and/or nonlinear problems.

**2.2. Non-intrusive methods.** When the high-fidelity system is not available, e.g. black-box or commercial solvers, intrusive MOR cannot be pursued and the reduced coefficients have to be recovered in a non-intrusive manner. For example, POD with interpolation (PODI) recovers the reduced coefficients from the projection of the snapshots onto the POD basis [15, 23]. A similar regression-based machine learning approach, POD-NN, is also possible by training a network to predict the reduced coefficients, with the projected coefficients used as the training dataset [7, 47].

Although non-intrusive, PODI, POD-NN and other such methods still represent linear approaches since they approximate the high-fidelity coefficients via a linear scheme  $\mathbf{u}_h \approx \mathbf{V} \mathbf{u}_N$ . This can be problematic, since many important physics phenomena exhibit a slow decay in the Kolmogorov  $n$ -width, making linear methods inefficient for such cases [36, 73]. As a result, nonlinear approaches have arisen which seek to avoid this limitation by instead prescribing the more general form of  $\mathbf{u}_h(\boldsymbol{\mu}) \approx \psi(\mathbf{u}_N(\boldsymbol{\mu}))$ , where  $\psi$  is a nonlinear mapping. A huge variety of nonlinear compression schemes have been explored for standard MOR: using local reduced bases to construct a piecewise linear scheme [3], considering nonlinear compression via kernel POD [24, 54], shifted POD [87] or registration methods [95].

Machine learning approaches are also possible, with the advantage of typically being cheap at inference. In this direction, Gaussian process regression (GPR) approaches have been heavily investigated [37–39, 52, 103]. In contrast to standard MOR, these approaches learn a probability distribution and are also advantageous in terms of uncertainty quantification [18]. However, GPR models are expensive with cubic scaling, and direct application of GPR to large datasets is infeasible, instead requiring techniques such as local approximate GPR [34, 35].

On the other hand, autoencoders have also arisen as a hugely popular means of nonlinear compression and do not suffer from such poor scaling. The method was originally introduced as a nonlinear generalisation of principal component analysis [57], making the approach a generalisation of POD-based methods. Recently, it has been used in many MOR applications with great success [32, 33, 45, 62, 71, 88], which makes it a hugely promising direction that we pursue in this work.

### 3. MULTIFIDELITY REDUCED ORDER MODELS

By removing any dependence on the generation procedure for the training data, non-intrusive ROMs can be used as black-box approaches, allowing e.g. to augment a small experimental dataset with simulated data [30, 31, 59]. In fact, even without embedding any physics into the ROM, it is still possible to learn the dynamics of the system. Nonetheless, standard approaches typically require all training data to be generated or measured on the same grid, and with the same accuracy. Whilst often assumed for algorithmic convenience, these requirements are highly undesirable for two key reasons: firstly, generating highly-fidelity data for training is expensive and secondly, experimental data is often multifidelity data.

**3.1. Generating high-fidelity data for training is expensive.** Several problems require fine discretisations to obtain accurate results and resolve multiscale systems. However, ROMs need to sample the parameter space well in order to make reasonable predictions. Thus, for standard MOR this involves the expensive computation of a large number of high-fidelity solutions. By allowing a ROM to train on data of different fidelities, one has the potential to greatly reduce the computational burden by computing only a small number of expensive (high-fidelity) simulations, and augmenting the data with a number of cheap (low-fidelity) simulations. This reflection has led to the development of a large number of multifidelity ROMs [19, 28, 39, 43, 52, 70, 82], aiming to alleviate the computational burden of generating training data.

Low-fidelity simulated data can be generated in a number of ways, either by reducing the accuracy of the computational method e.g. using  $\mathbb{P}1$  instead of  $\mathbb{P}2$  finite elements, considering a coarsened discretisation, or by simplifying the physics. The central task for multifidelity MOR is to determine

how to combine data arising from a number of different fidelities. A common assumption is that all fidelities are prespecified and separate models are built for each of the  $F$  different fidelities. A common approach has been to adopt a multifidelity GPR based approach for MOR [19, 39, 52, 70, 82], commonly known as cokriging [2, 53], expressing each of the  $F$  solution fidelities as a combination of  $F$  independent Gaussian processes. Similarly, multigrid approaches have also been heavily explored for parametric PDEs [5, 41–43, 68, 96], relying on a hierarchy of models, and iteratively computing corrections through a sequence of  $F$  coarser discretisations.

**3.2. Experimental data is often multifidelity data.** Other than synthetic data, ROMs are of utmost importance also when dealing with experimental data, e.g. coming from sensors which may move or break. This can lead to a high number of resolutions  $F$ , particularly for contexts such as climate modelling where data is dependent on weather stations, the number and locations of which can vary greatly [25, 26]. As a concrete example, the average number of stations used in the Global Precipitation Climatology Center [8] full-data product over Africa has varied from under 250 to over 3250 stations between 1901 and 2013 [25]. Even with a coarse yearly temporal resolution, this would still give a total of  $F = 112$  different resolutions.

This is particularly problematic when one considers multifidelity ROMs train a new model for each fidelity. Furthermore, in many cases it is not possible to upscale or downscale the model predictions to unseen fidelities (e.g. finer discretisation), or to leverage training data from other previously unseen fidelities either.

Thus, in the experimental context, a single model capable of training and being evaluated on complexity arbitrary discretisations is required as depicted in Figure 2, i.e. there should be no dependence in the model architecture on the discretisations of the training samples. The ability to evaluate and train with data on any discretisation is known as resolution invariance [55].

None of the MOR approaches discussed thus far are resolution-invariant, underscoring the need for the development of a new, resolution-invariant ROM.

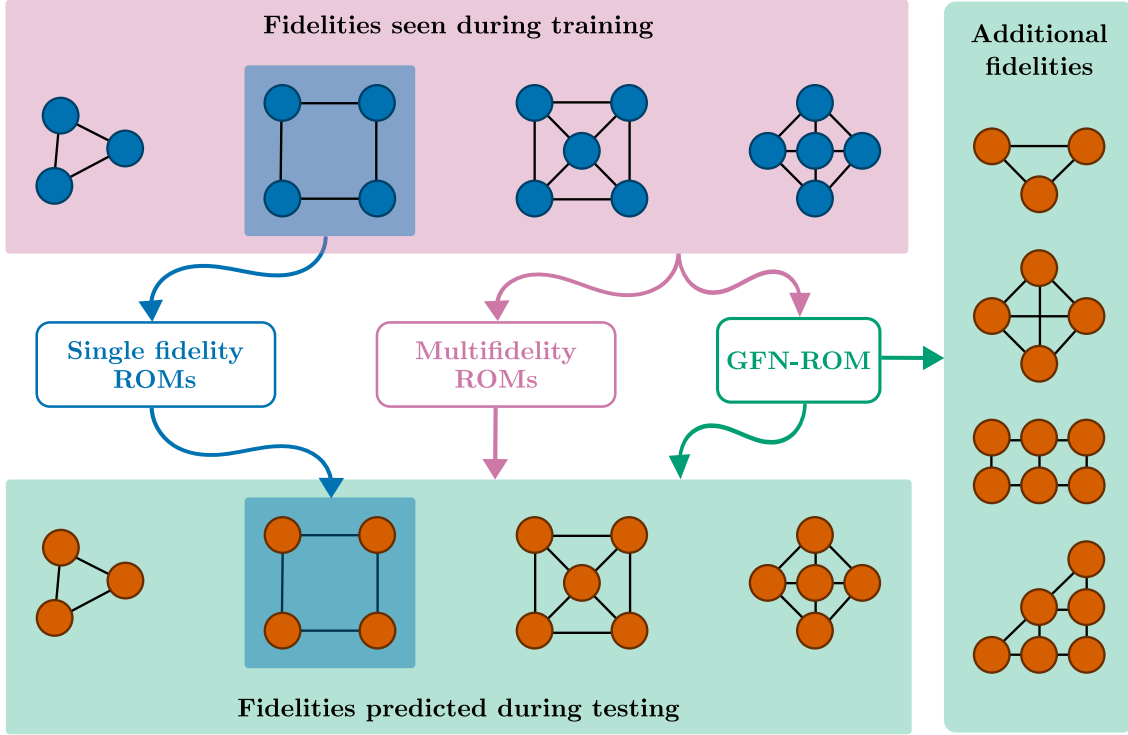


FIGURE 2. GFN-ROM as a resolution-invariant ROM, capable of handling data from any discretisation, both in training and in testing modes.

#### 4. RELATED RESOLUTION-INVARIANT METHODS

There exist a number of resolution-invariant approaches relevant for PDEs problems. Of particular interest are PINNs, NOs, and GNNs.

**4.1. Physics-informed neural networks.** Standard ROM approaches usually do not require any explicit positional information, simply relying on the snapshot matrix, thereby making resolution-invariant extensions not straightforward. However, already when considering the approximation of a single snapshot, it is possible to map directly from positional information to a PDE solution i.e.  $\mathbf{x} \mapsto \phi(\mathbf{x}) \approx u(\mathbf{x})$ , where  $\phi$  is a function to fit.

This is precisely the idea of PINNs [20, 81, 83], whereby a neural network  $\phi$  is trained via a physics-informed loss i.e. the loss is taken to be the residual of the network approximation for the governing PDE, and no training data is required. Whilst PINNs can perform well on several problems [4, 22, 50, 51, 80, 84, 85, 92], they suffer from a number of issues such as “spectral bias”, where they are incapable of learning functions with high frequencies [100], convergence discrepancies among their loss components [100] and stiff gradient flow dynamics [99]. Furthermore, their extension towards parametrised problems is in practice unsuitable due to the complexity of the training phase, rendering them not useful in the ROM context [65, 89]. For example, the POD-PINN approach trains a neural network to predict the reduced coefficients from the parameters by minimising the residual of the reduced problem, making it an intrusive approach requiring an efficient means of evaluating the residual of the reduced problem [17].

**4.2. Neural operators.** The inability of PINNs to learn several instances for parametric PDEs is due to the fact that such methods are not designed to directly model the solution operator. Unlike PINNs and standard machine learning methods which learn a map between evaluations of two functions, neural operators learn maps between the function spaces themselves [55]. By attempting to model explicitly the solution operator itself, neural operators remove the burdensome dependence of typical machine learning methods on the discretisation of the data, meaning they are inherently capable of achieving super- or sub-resolution [55, 64, 86, 94] and also typically benefit from desirable properties such as discretisation invariance [55]. Neural operators have shown impressive performance [12, 16, 40, 64, 86], even learning different physics at the same time [58, 74]. However, many state-of-the-art neural operators such as Fourier neural operators [64] and convolutional neural operators [86] require structured data as input. Additionally, such methods are designed for learning maps between two spatially dependent functions, and not from a global parameter to a PDE solution, as in the simplified yet more common MOR setting. With the exception of the general neural operator transformer [40], existing neural operators therefore do not take global parameters as input.

**4.3. Graph neural networks.** PDEs are often posed on complex domains, giving rise to unstructured meshes when computing numerical solutions. Similarly, experimental data is also rarely present on structured grids. As a result, training data for MOR is best considered as graph-based data, necessitating methods that can handle such unstructured meshes. GNNs have arisen in recent years as powerful tools to learn on graphs, and have already celebrated remarkable successes in modelling complex physics problems [9, 44, 60]. These methods aim to enforce some geometric structure to a problem by embedding geometric priors into the model. Much work (reviewed in [66, 101, 102]) has been dedicated to the the generalisation of pooling, unpooling and convolutional layers for graph-based data. GNNs are also not without limitations, with known issues for many graph convolutions such as reduced expressive power [21] and oversmoothing [48, 63, 91]. In the MOR context, one seeks a map from a global parameter to a PDE solution, which can be represented as a graph. In GNN parlance, this represents a graph unpooling operation. However, currently there exist very few unpooling methods. In the major GNN libraries in Python, namely TF-GNN [27], PyG [29] and DGL [98], there currently exists only one single graph unpooling layer, the  $k$ -nn interpolation layer, which is non-trainable and therefore clearly unsuitable for MOR.

## 5. GRAPH FEEDFORWARD NETWORK (GFN)

As we discussed before, machine learning approaches for MOR usually exploit neural network layers which are not suitable for dealing with multifidelity data. Specifically, in the GNN context there is a lack of suitable unpooling layers for mapping from an input vector to an output graph, allowing for multi-resolution approaches. Given that feedforward networks have been heavily used in MOR, regardless of their potential inconsistencies, in this work we present GFNs, a generalisation of feedforward networks capable of leveraging data on any discretisation.

Prior to presenting the method, we fix some notation and conventions for ease of reading, also providing their graphical illustrations in Figure 3.

- A mesh  $\mathcal{M}$  is an ordered collection of nodes (co-ordinates), without any duplicates<sup>2</sup>.
- $i_{\mathcal{M}}$  denotes an index associated to a node in  $\mathcal{M}$ , defined by the ordering of the mesh  $\mathcal{M}$ . Thus, the position of a node of index  $i_{\mathcal{M}}$  is given as  $\mathcal{M}[i_{\mathcal{M}}]$ .
- $i_{\mathcal{M}_o} \leftarrow j_{\mathcal{M}_n}$  means that the node of index  $i_{\mathcal{M}_o}$  in the mesh  $\mathcal{M}_o$  is the nearest neighbour of the node of index  $j_{\mathcal{M}_n}$  in the mesh  $\mathcal{M}_n$ , i.e.,  $i_{\mathcal{M}_o} = \operatorname{argmin}_{k_{\mathcal{M}_o}} |\mathcal{M}_n[j_{\mathcal{M}_n]} - \mathcal{M}_o[k_{\mathcal{M}_o}]|$ .
- $i_{\mathcal{M}_o} \rightarrow j_{\mathcal{M}_n}$  means that the node of index  $j_{\mathcal{M}_n}$  in the mesh  $\mathcal{M}_n$  is the nearest neighbour<sup>3</sup> of the node of index  $i_{\mathcal{M}_o}$  in the mesh  $\mathcal{M}_o$ , i.e.,  $j_{\mathcal{M}_n} = \operatorname{argmin}_{k_{\mathcal{M}_n}} |\mathcal{M}_n[k_{\mathcal{M}_n]} - \mathcal{M}_o[i_{\mathcal{M}_o}]|$ .
- $i_{\mathcal{M}_o} \leftrightarrow j_{\mathcal{M}_n}$  means that either  $i_{\mathcal{M}_o} \leftarrow j_{\mathcal{M}_n}$  or  $i_{\mathcal{M}_o} \rightarrow j_{\mathcal{M}_n}$ .
- $i_{\mathcal{M}_o} \longleftrightarrow j_{\mathcal{M}_n}$  means that both  $i_{\mathcal{M}_o} \leftarrow j_{\mathcal{M}_n}$  and  $i_{\mathcal{M}_o} \rightarrow j_{\mathcal{M}_n}$ .

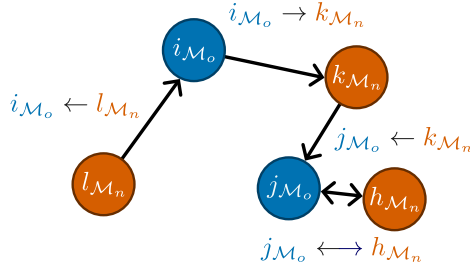


FIGURE 3. Illustration of the arrow notations used in this work.

**5.1. Feedforward networks for multifidelity data.** Numerous state-of-the-art ROMs such as GCA-ROM [77] and DL-ROM [33] employ autoencoder-based approaches for MOR. In its simplest possible form, a standard feedforward network for an autoencoder task on an unstructured mesh  $\mathcal{M}_o$  with  $|\mathcal{M}_o| = N_o$  nodes consists of a single-layer encoder and single-layer decoder defined as

$$(3) \quad \operatorname{enc}(\mathbf{u}_{\mathcal{M}_o})_i = \sigma \left( \sum_{j_{\mathcal{M}_o}=1}^{N_o} W_{ij_{\mathcal{M}_o}}^e u_{j_{\mathcal{M}_o}} + b_i^e \right), \quad \forall i = 1, \dots, L,$$

$$(4) \quad \operatorname{dec}(\mathbf{z})_{i_{\mathcal{M}_o}} = \sum_{j=1}^L W_{i_{\mathcal{M}_o}j}^d z_j + b_{i_{\mathcal{M}_o}}^d, \quad \forall i_{\mathcal{M}_o} = 1, \dots, N_o,$$

where  $\mathbf{u}_{\mathcal{M}_o} \in \mathbb{R}^{N_o}$  is an input sample on the mesh  $\mathcal{M}_o$ ,  $\mathbf{z} \in \mathbb{R}^L$  is a latent vector,  $\sigma$  is an activation function,  $\mathbf{W}^e \in \mathbb{R}^{L \times N_o}$  and  $\mathbf{b}^e \in \mathbb{R}^L$  are the encoder weights and biases, respectively, and  $\mathbf{W}^d \in \mathbb{R}^{N_o \times L}$  and  $\mathbf{b}^d \in \mathbb{R}^{N_o}$  are the decoder weights and biases, respectively. The reconstruction loss to optimise is simply given as  $\|\operatorname{dec}(\operatorname{enc}(\mathbf{u}_{\mathcal{M}_o})) - \mathbf{u}_{\mathcal{M}_o}\|_2^2$ , thus defining the machine learning task. We note that the encoder and decoder weights  $W_{ij_{\mathcal{M}_o}}^e$ ,  $W_{i_{\mathcal{M}_o}j}^d$  and the decoder bias  $b_{i_{\mathcal{M}_o}}^d$  are all associated to a node  $i_{\mathcal{M}_o}$ . Consequently, we can regard these weights and biases as belonging to a node in the

<sup>2</sup>We represent  $\mathcal{M}$  as a set, whose ordering is arbitrary and not important here, provided it is fixed for a given set.

<sup>3</sup>Note that we could write  $i_{\mathcal{M}_o} \rightarrow j_{\mathcal{M}_n}$  as  $j_{\mathcal{M}_n} \leftarrow i_{\mathcal{M}_o}$  and likewise  $i_{\mathcal{M}_o} \leftarrow j_{\mathcal{M}_n}$  as  $j_{\mathcal{M}_n} \rightarrow i_{\mathcal{M}_o}$ . However, for readability we adopt the convention that nodes pertaining to  $\mathcal{M}_o$  are given on the left side of the expression.

mesh  $\mathcal{M}_o$ , meaning that each has a spatial location given by the node's position. An illustration of this is shown for an autoencoder in Figure 4.

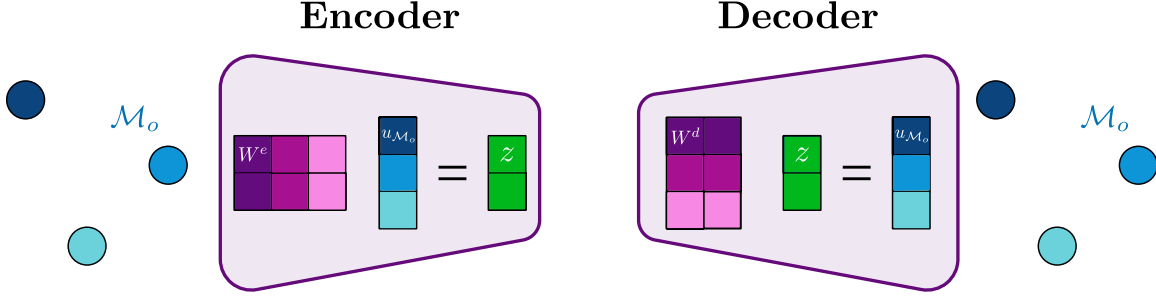


FIGURE 4. Single-layer graph feedforward autoencoder architecture. As shown via shading, the columns of the encoder weight matrix and the rows of the decoder weight matrix are associated to the individual nodes in the mesh  $\mathcal{M}_o$ .

Now, we suppose that we have a trained autoencoder defined above performing well on the unstructured mesh  $\mathcal{M}_o$ , and we wish to use this autoencoder on a new similar mesh  $\mathcal{M}_n$  with  $|\mathcal{M}_n| = N_n$  nodes. This is not possible for standard feedforward network, even if it is intuitively clear that one should be able to leverage the information from the previous model. As a result, we introduce the notion of a GFN and use the nearest neighbour information between the meshes  $\mathcal{M}_o$  and  $\mathcal{M}_n$  to generalise the autoencoder, with provable guarantees and desirable properties. Specifically, GFNs create a means of transforming weights and biases of the encoder and decoder ( $\mathbf{W}^e$ ,  $\mathbf{b}^e$ ,  $\mathbf{W}^d$  and  $\mathbf{W}^d$ ) for the training mesh  $\mathcal{M}_o$  to new weights and biases ( $\tilde{\mathbf{W}}^e$ ,  $\tilde{\mathbf{b}}^e$ ,  $\tilde{\mathbf{W}}^d$  and  $\tilde{\mathbf{W}}^d$ ) for the new mesh  $\mathcal{M}_n$ , allowing for the evaluation of a feedforward network on multifidelity data. To achieve this transform from  $\mathcal{M}_o$  to  $\mathcal{M}_n$ , which we denote coherently as  $\mathcal{M}_o \rightarrow \mathcal{M}_n$ , while retaining good performance, one can leverage the similarity of proximate nodes as illustrated in Figure 5.

Mathematically, the GFN transforms for  $\mathcal{M}_o \rightarrow \mathcal{M}_n$  are defined as

$$\begin{aligned}
 \tilde{W}_{ij\mathcal{M}_n}^e &= \sum_{\forall k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \leftrightarrow j_{\mathcal{M}_n}} \frac{W_{ik_{\mathcal{M}_o}}^e}{|\{h_{\mathcal{M}_n} \text{ s.t. } k_{\mathcal{M}_o} \leftrightarrow h_{\mathcal{M}_n}\}|}, \\
 \tilde{b}_i^e &= b_i^e, \\
 \tilde{W}_{i\mathcal{M}_n j}^d &= \text{mean}_{\forall k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \leftrightarrow i_{\mathcal{M}_n}} W_{k_{\mathcal{M}_o} j}^d, \\
 \tilde{b}_{i\mathcal{M}_n}^d &= \text{mean}_{\forall k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \leftrightarrow i_{\mathcal{M}_n}} b_{k_{\mathcal{M}_o}}^d.
 \end{aligned}
 \tag{5}$$

Note that trivially  $\tilde{b}_i^e = b_i^e$  since the encoder bias is not associated with the graph, but rather with the latent vector. These transforms require nearest neighbour computations for all nodes in  $\mathcal{M}_o$  and  $\mathcal{M}_n$ , leading to algorithm with time complexity of  $O(N_o \log N_n + N_n \log N_o)$ , using the  $k$ -d tree method for nearest neighbour computations [11]. The predictions on the new mesh  $\mathcal{M}_n$  then read analogously to before as

$$\text{enc}(\mathbf{u}_{\mathcal{M}_n})_i = \sigma \left( \sum_{j_{\mathcal{M}_n}=1}^{N_n} \tilde{W}_{ij\mathcal{M}_n}^e u_{j_{\mathcal{M}_n}} + \tilde{b}_i^e \right), \quad \forall i = 1, \dots, L,
 \tag{6}$$

$$\text{dec}(\mathbf{z})_{i_{\mathcal{M}_n}} = \sum_{j=1}^L \tilde{W}_{i_{\mathcal{M}_n} j}^d z_j + \tilde{b}_{i_{\mathcal{M}_n}}^d, \quad \forall i_{\mathcal{M}_n} = 1, \dots, N_n,
 \tag{7}$$

where  $\mathbf{u}_{\mathcal{M}_n} \in \mathbb{R}^{N_n}$  is now a new input sample on the mesh  $\mathcal{M}_n$  and the new weights and biases  $\tilde{\mathbf{W}}^e \in \mathbb{R}^{L \times N_n}$ ,  $\tilde{\mathbf{b}}^e \in \mathbb{R}^L$ ,  $\tilde{\mathbf{W}}^d \in \mathbb{R}^{N_n \times L}$  and  $\tilde{\mathbf{b}}^d \in \mathbb{R}^{N_n}$  are given in Equation 5. Note that the transformed weights are now associated to each of the nodes on the new mesh  $\mathcal{M}_n$ .

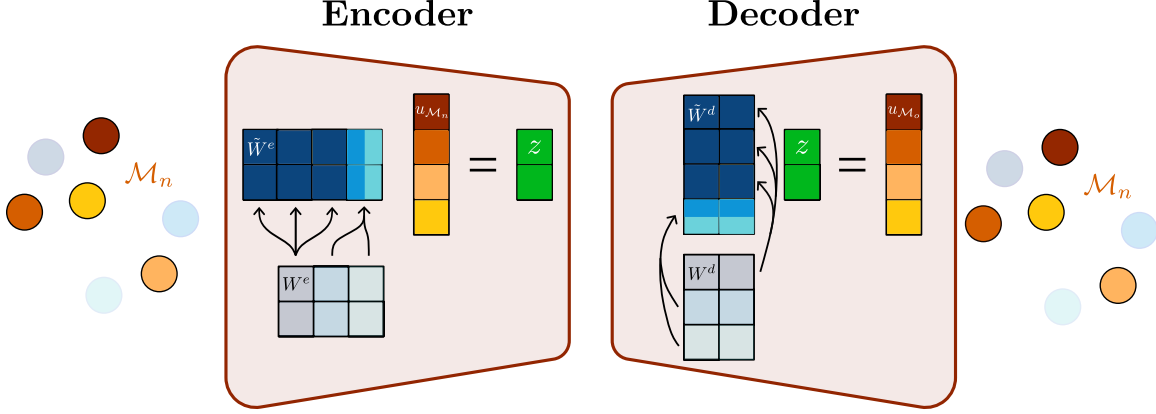


FIGURE 5. GFN approach to transform weights between meshes. Light blue shades denote the original mesh with its associated  $\mathbf{W}^e$  and  $\mathbf{W}^d$ . The arrows depict how the new weight matrices (in darker blue shades)  $\tilde{\mathbf{W}}^e$  and  $\tilde{\mathbf{W}}^d$  can be created from the original weights and used for prediction on the new mesh.

To keep track of meshes and transforms, we denote the presented GFN approach of transforming weights from a mesh  $\mathcal{M}_o$  to a new mesh  $\mathcal{M}_n$  as

$$(8) \quad \tilde{\mathbf{W}}^e, \tilde{\mathbf{W}}^d, \tilde{\mathbf{b}}^d = \text{GFN}^{\mathcal{M}_o \rightarrow \mathcal{M}_n}(\mathbf{W}^e, \mathbf{W}^d, \mathbf{b}^d).$$

Similarly, we also write  $\text{enc}^{\mathcal{M}_o \rightarrow \mathcal{M}_n}$  and  $\text{dec}^{\mathcal{M}_o \rightarrow \mathcal{M}_n}$  to track the applied transforms.

For autoencoders with multiple hidden layers, the GFN approach trivially generalises and only needs to be applied to the first encoder layer and the last decoder layer, since these are the only layers directly interacting with the changing meshes (for further details see Appendix A.1).

This approach of transforming via GFN thus defines a means of allowing an autoencoder, which has been trained on single fidelity data using standard feedforward networks only, to be evaluated on multifidelity data.

**5.2. Algorithmic details.** In this section, we highlight two notable instances of the possible transforms  $\mathcal{M}_o \rightarrow \mathcal{M}_n$  which lead to computational savings: the *expansive* and the *agglomerative* cases. Demonstrative examples of agglomerative, expansive, neither agglomerative nor expansive and both agglomerative and expansive cases are shown in Figure 6.

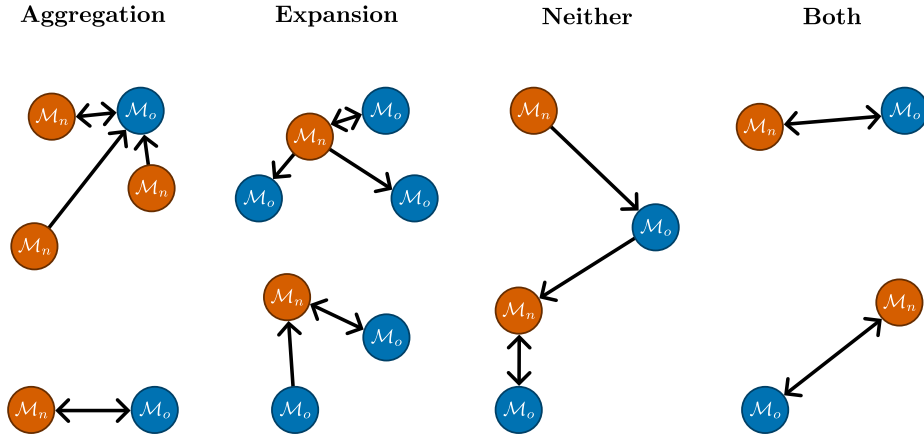


FIGURE 6. Examples of the different types of possible transforms.

5.2.1. *Expansive case.* The nearest neighbour of each node  $i_{\mathcal{M}_o}$  in  $\mathcal{M}_n$  has  $i_{\mathcal{M}_o}$  as its nearest neighbour in  $\mathcal{M}_o$ , i.e., the nearest neighbour of the nearest neighbour of each node  $i_{\mathcal{M}_o}$  is itself, following our notation:

$$(9) \quad \forall i_{\mathcal{M}_o}, \quad i_{\mathcal{M}_o} \rightarrow j_{\mathcal{M}_n} \implies i_{\mathcal{M}_o} \leftarrow j_{\mathcal{M}_n}.$$

In such case, the transform can be simplified (see the proof in Appendix A.2) as:

$$(10) \quad \begin{aligned} \tilde{W}_{ij_{\mathcal{M}_n}}^e &= \frac{1}{|\{h_{\mathcal{M}_n} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow h_{\mathcal{M}_n}\}|} W_{ik_{\mathcal{M}_o}}^e, & \text{where } k_{\mathcal{M}_o} \leftarrow j_{\mathcal{M}_n}, \\ \tilde{W}_{i_{\mathcal{M}_n}j}^d &= W_{k_{\mathcal{M}_o}j}^d, & \text{where } k_{\mathcal{M}_o} \leftarrow i_{\mathcal{M}_n}, \\ \tilde{b}_{i_{\mathcal{M}_n}}^d &= b_{k_{\mathcal{M}_o}}^d, & \text{where } k_{\mathcal{M}_o} \leftarrow i_{\mathcal{M}_n}. \end{aligned}$$

We note that the absence of rightward arrows means we are only required to find nearest neighbours of nodes in the mesh  $\mathcal{M}_n$  and not those in  $\mathcal{M}_o$ , reducing the algorithmic time complexity to  $O(N_n \log N_o)$ . The implementation of the expansive transformation with this complexity is shown in Algorithm 1.

---

**Algorithm 1** Simplified GFN transform in the expansive case, as given in Equation 10.

---

```

function EXPAND( $\mathbf{W}^e \in \mathbb{R}^{L \times N_o}$ ,  $\mathbf{W}^d \in \mathbb{R}^{N_o \times L}$ ,  $\mathbf{b}^d \in \mathbb{R}^{N_o}$ ,  $\mathcal{M}_o, \mathcal{M}_n$ )
  Initialise  $\tilde{\mathbf{W}}^e \in \mathbb{R}^{L \times N_n}$ ,  $\tilde{\mathbf{W}}^d \in \mathbb{R}^{N_n \times L}$ ,  $\tilde{\mathbf{b}}^d \in \mathbb{R}^{N_n}$ 
  counts  $\leftarrow \mathbf{0} \in \mathbb{R}^{N_o}$ 
  for  $i_{\mathcal{M}_n} = 1$  to  $N_n$  do
     $j_{\mathcal{M}_o} \leftarrow \text{nearest-neighbour}^{\mathcal{M}_o}(\mathcal{M}_n[i_{\mathcal{M}_n}])$ 
    counts[ $j_{\mathcal{M}_o}$ ]  $\leftarrow$  counts[ $j_{\mathcal{M}_o}$ ] + 1
  end for
  for  $i_{\mathcal{M}_n} = 1$  to  $N_n$  do
     $j_{\mathcal{M}_o} \leftarrow \text{nearest-neighbour}^{\mathcal{M}_o}(\mathcal{M}_n[i_{\mathcal{M}_n}])$ 
     $\tilde{\mathbf{W}}^e[:, i_{\mathcal{M}_n}] \leftarrow \mathbf{W}^e[:, j_{\mathcal{M}_o}] / \text{counts}[j_{\mathcal{M}_o}]$ 
     $\tilde{\mathbf{W}}^d[i_{\mathcal{M}_n}, :] \leftarrow \mathbf{W}^d[j_{\mathcal{M}_o}, :]$ 
     $\tilde{\mathbf{b}}^d[i_{\mathcal{M}_n}] \leftarrow \mathbf{b}^d[j_{\mathcal{M}_o}]$ 
  end for
  return  $\tilde{\mathbf{W}}^e, \tilde{\mathbf{W}}^d, \tilde{\mathbf{b}}^d$ 
end function

```

---

5.2.2. *Agglomerative case.* The nearest neighbour of each node  $j_{\mathcal{M}_n}$  in  $\mathcal{M}_o$  has  $j_{\mathcal{M}_n}$  as its nearest neighbour in  $\mathcal{M}_n$ , i.e., the nearest neighbour of the nearest neighbour of each node  $j_{\mathcal{M}_n}$  is itself, following our notation:

$$(11) \quad \forall j_{\mathcal{M}_n}, \quad i_{\mathcal{M}_o} \leftarrow j_{\mathcal{M}_n} \implies i_{\mathcal{M}_o} \rightarrow j_{\mathcal{M}_n}.$$

In such case, the transform can be simplified (see the proof in Appendix A.3) as:

$$(12) \quad \begin{aligned} \tilde{W}_{ij_{\mathcal{M}_n}}^e &= \sum_{\forall k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \rightarrow j_{\mathcal{M}_n}} W_{ik_{\mathcal{M}_o}}^e, \\ \tilde{W}_{i_{\mathcal{M}_n}j}^d &= \text{mean}_{\forall k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \rightarrow i_{\mathcal{M}_n}} W_{k_{\mathcal{M}_o}j}^d, \\ \tilde{b}_{i_{\mathcal{M}_n}}^d &= \text{mean}_{\forall k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \rightarrow i_{\mathcal{M}_n}} b_{k_{\mathcal{M}_o}}^d. \end{aligned}$$

We note that the absence of leftward arrows means we are only required to find nearest neighbours of nodes in the mesh  $\mathcal{M}_o$  and not those in  $\mathcal{M}_n$ , reducing the algorithmic time complexity to  $O(N_o \log N_n)$ . The implementation of the expansive transformation with this complexity is shown in Algorithm 2.

---

**Algorithm 2** Simplified GFN transform in the agglomerative case, as given in Equation 12.

---

```

function AGGLOMERATE( $\mathbf{W}^e \in \mathbb{R}^{L \times N_o}$ ,  $\mathbf{W}^d \in \mathbb{R}^{N_o \times L}$ ,  $\mathbf{b}^d \in \mathbb{R}^{N_o}$ ,  $\mathcal{M}_o, \mathcal{M}_n$ )
  Initialise  $\tilde{\mathbf{W}}^e \in \mathbb{R}^{L \times N_n}$ ,  $\tilde{\mathbf{W}}^d \in \mathbb{R}^{N_n \times L}$ ,  $\tilde{\mathbf{b}}^d \in \mathbb{R}^{N_n}$ 
  counts  $\leftarrow \mathbf{0} \in \mathbb{R}^{N_n}$ 
  for  $i_{\mathcal{M}_o} = 1$  to  $N_o$  do
     $j_{\mathcal{M}_n} \leftarrow \text{nearest-neighbour}^{\mathcal{M}_n}(\mathcal{M}_o[i_{\mathcal{M}_o}])$ 
    counts[ $j_{\mathcal{M}_n}$ ]  $\leftarrow$  counts[ $j_{\mathcal{M}_n}$ ] + 1
     $\tilde{\mathbf{W}}^e[:, j_{\mathcal{M}_n}] \leftarrow \mathbf{W}^e[:, i_{\mathcal{M}_o}]$ 
     $\tilde{\mathbf{W}}^d[j_{\mathcal{M}_n}, :] \leftarrow ((\text{counts}[j_{\mathcal{M}_n}] - 1) \tilde{\mathbf{W}}^d[j_{\mathcal{M}_n}, :] + \mathbf{W}^d[i_{\mathcal{M}_o}, :]) / \text{counts}[j_{\mathcal{M}_n}]$ 
     $\tilde{\mathbf{b}}^d[j_{\mathcal{M}_n}] \leftarrow ((\text{counts}[j_{\mathcal{M}_n}] - 1) \tilde{\mathbf{b}}^d[j_{\mathcal{M}_n}] + \mathbf{b}^d[i_{\mathcal{M}_o}]) / \text{counts}[j_{\mathcal{M}_n}]$ 
  end for
  return  $\tilde{\mathbf{W}}^e, \tilde{\mathbf{W}}^d, \tilde{\mathbf{b}}^d$ 
end function

```

---

5.2.3. *Computational advantages.* For hierarchical meshes, it follows that if  $\mathcal{M}_o \subseteq \mathcal{M}_n$  then the transform  $\mathcal{M}_o \rightarrow \mathcal{M}_n$  is expansive, whereas if  $\mathcal{M}_n \subseteq \mathcal{M}_o$  then the transform  $\mathcal{M}_o \rightarrow \mathcal{M}_n$  is agglomerative. Moreover, if the transform is both expansive and agglomerative then it must hold that  $\forall i_{\mathcal{M}_o}, i_{\mathcal{M}_o} \longleftrightarrow j_{\mathcal{M}_n}$  and  $\forall j_{\mathcal{M}_n}, i_{\mathcal{M}_o} \longleftrightarrow j_{\mathcal{M}_n}$ , i.e. there is a one-to-one correspondence between nodes in the original mesh  $\mathcal{M}_o$  and in the new mesh  $\mathcal{M}_n$ , and the GFN transforms reduce to nearest neighbour interpolation of the weights.

However, for general meshes a transform can be both, neither or one of expansive and agglomerative. Despite this, notably any general GFN transform can be expressed as a succession of an expansive update and an agglomerative update, (proof in Appendix A.5) i.e. Equation 5 is equivalent to defining the new mesh

$$(13) \quad \mathcal{M} = \mathcal{M}_o \cup \{\mathcal{M}_n[i_{\mathcal{M}_n}] \text{ s.t. } j_{\mathcal{M}_o} \leftarrow i_{\mathcal{M}_n} \text{ but not } j_{\mathcal{M}_o} \rightarrow i_{\mathcal{M}_n}\},$$

and computing the transform in two steps, as

$$(14) \quad \left. \begin{aligned} \hat{W}_{ij_{\mathcal{M}}}^e &= \frac{1}{|\{h_{\mathcal{M}} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow h_{\mathcal{M}}\}|} W_{ik_{\mathcal{M}_o}}^e, & \text{where } k_{\mathcal{M}_o} \leftarrow j_{\mathcal{M}}, \\ \hat{W}_{i_{\mathcal{M}}j}^d &= W_{k_{\mathcal{M}_o}j}^d, & \text{where } k_{\mathcal{M}_o} \leftarrow i_{\mathcal{M}}, \\ \hat{b}_{i_{\mathcal{M}}}^d &= b_{k_{\mathcal{M}_o}}^d, & \text{where } k_{\mathcal{M}_o} \leftarrow i_{\mathcal{M}}, \end{aligned} \right\} \text{Expansion step}$$

followed by

$$(15) \quad \left. \begin{aligned} \tilde{W}_{ij_{\mathcal{M}_n}}^e &= \sum_{\forall k_{\mathcal{M}} \text{ s.t. } k_{\mathcal{M}} \rightarrow j_{\mathcal{M}_n}} \hat{W}_{ik_{\mathcal{M}}}^e, \\ \tilde{W}_{i_{\mathcal{M}_n}j}^d &= \text{mean}_{\forall k_{\mathcal{M}} \text{ s.t. } k_{\mathcal{M}} \rightarrow i_{\mathcal{M}_n}} \hat{W}_{k_{\mathcal{M}}j}^d, \\ \tilde{b}_{i_{\mathcal{M}_n}}^d &= \text{mean}_{\forall k_{\mathcal{M}} \text{ s.t. } k_{\mathcal{M}} \rightarrow i_{\mathcal{M}_n}} \hat{b}_{k_{\mathcal{M}}}^d. \end{aligned} \right\} \text{Agglomerative step}$$

The general GFN algorithm can thus be easily expressed as in Algorithm 3, with a demonstrative illustration of this interpretation in Figure 7.

## 6. GRAPH FEEDFORWARD NETWORK REDUCED ORDER MODEL (GFN-ROM)

Having defined GFNs as a means of evaluating trained autoencoders on multifidelity data, we leverage them to propose a reduced order model capable of being evaluated on arbitrary meshes. As depicted in Figure 8, the proposed GFN-ROM architecture consists of two components: an autoencoder and a mapper. The autoencoder is based on GFNs, while the mapper can be any neural network model which maps the fixed-size vector parameters directly to the fixed-size latent representation. Notice that, within this parametric context, the mapper does not need to be defined using GFNs since it does not deal with graph-structured data.



**Algorithm 3** General GFN transform leveraging an expansion step followed by an agglomerative update, as given in Equation 5.

---

```

function GFN( $\mathbf{W}^e \in \mathbb{R}^{L \times N_o}$ ,  $\mathbf{W}^d \in \mathbb{R}^{N_n \times L}$ ,  $\mathbf{b}^d \in \mathbb{R}^{N_o}$ ,  $\mathcal{M}_o, \mathcal{M}_n$ )
   $\mathcal{M} = \mathcal{M}_o \cup \{\mathcal{M}_n [i_{\mathcal{M}_n}] \text{ s.t. } j_{\mathcal{M}_o} \leftarrow i_{\mathcal{M}_n} \text{ but not } j_{\mathcal{M}_o} \rightarrow i_{\mathcal{M}_n}\}$ 
   $\hat{\mathbf{W}}^e, \hat{\mathbf{W}}^d, \hat{\mathbf{b}}^d \leftarrow \text{EXPAND}(\mathbf{W}^e, \mathbf{W}^d, \mathbf{b}^d, \mathcal{M}_o, \mathcal{M})$ 
   $\tilde{\mathbf{W}}^e, \tilde{\mathbf{W}}^d, \tilde{\mathbf{b}}^d \leftarrow \text{AGGLOMERATE}(\hat{\mathbf{W}}^e, \hat{\mathbf{W}}^d, \hat{\mathbf{b}}^d, \mathcal{M}, \mathcal{M}_n)$ 
  return  $\tilde{\mathbf{W}}^e, \tilde{\mathbf{W}}^d, \tilde{\mathbf{b}}^d$ 
end function

```

---

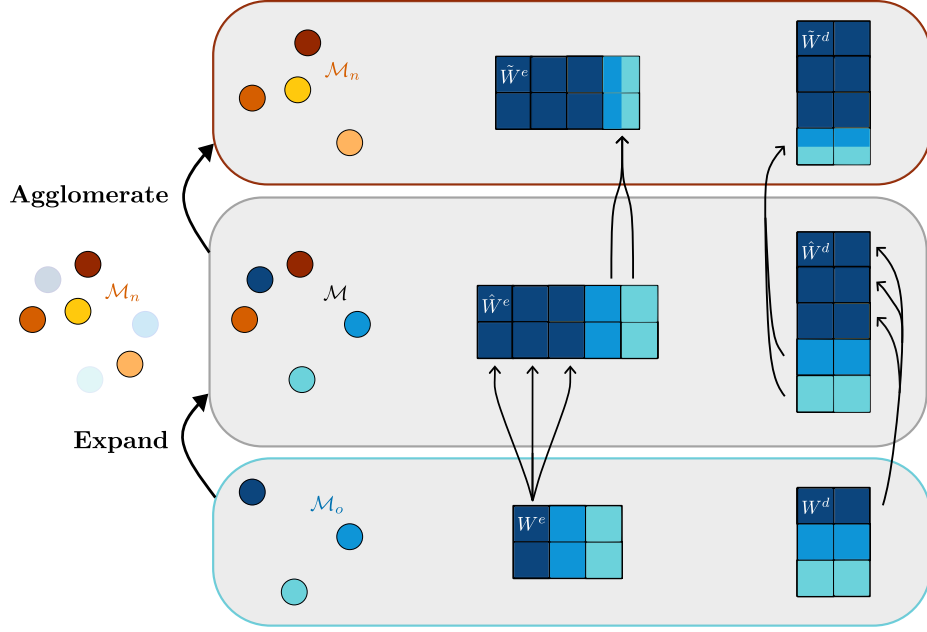


FIGURE 7. The GFN transform can be thought about as applying an expansive update followed by an agglomerative update.

GFN-ROM is inspired by recent state-of-the-art autoencoder-based models such as DL-ROM [33], which uses standard convolutions, and GCA-ROM [77], which uses graph convolutions. In fact, GFN-ROM can be regarded as an extension of such methods. Like both models, we train the GFN-ROM model based on a weighted sum of two losses: the reconstruction loss and the mapper loss. Given a GFN-ROM model with weights defined on a mesh  $\mathcal{M}_o$ , the per-example loss functions are defined for the  $t$ th training sample  $(\boldsymbol{\mu}_t, \mathbf{u}_{\mathcal{M}_n^t}(\boldsymbol{\mu}_t))$ , defined on the  $t$ -th mesh  $\mathcal{M}_n^t$  and for the  $t$ -th parameter value  $\boldsymbol{\mu}_t$ , as

$$(16) \quad \mathcal{L}_{\text{recon}}[\boldsymbol{\mu}_t, \mathbf{u}_{\mathcal{M}_n^t}(\boldsymbol{\mu}_t)] = \frac{1}{|\mathcal{M}_n^t|} \|\text{dec}^{\mathcal{M}_o \rightarrow \mathcal{M}_n^t}(\text{enc}^{\mathcal{M}_o \rightarrow \mathcal{M}_n^t}(\mathbf{u}_{\mathcal{M}_n^t}(\boldsymbol{\mu}_t))) - \mathbf{u}_{\mathcal{M}_n^t}(\boldsymbol{\mu}_t)\|_2^2,$$

$$(17) \quad \mathcal{L}_{\text{map}}[\boldsymbol{\mu}_t, \mathbf{u}_{\mathcal{M}_n^t}(\boldsymbol{\mu}_t)] = \frac{1}{L} \|\text{enc}^{\mathcal{M}_o \rightarrow \mathcal{M}_n^t}(\mathbf{u}_{\mathcal{M}_n^t}(\boldsymbol{\mu}_t)) - \text{map}(\boldsymbol{\mu}_t)\|_2^2.$$

The overall loss is given as a weighted mean of these losses over a set  $\{(\boldsymbol{\mu}_t, \mathbf{u}_{\mathcal{M}_n^t}(\boldsymbol{\mu}_t))\}_{t=1}^T$  of  $T$  training examples

$$(18) \quad \mathcal{L} = \frac{1}{T} \sum_{t=1}^T \frac{|\mathcal{M}_n^t|}{\sum_{s=1}^T |\mathcal{M}_n^s|} (\mathcal{L}_{\text{recon}}[\boldsymbol{\mu}_t, \mathbf{u}_{\mathcal{M}_n^t}(\boldsymbol{\mu}_t)] + \omega \mathcal{L}_{\text{map}}[\boldsymbol{\mu}_t, \mathbf{u}_{\mathcal{M}_n^t}(\boldsymbol{\mu}_t)]),$$

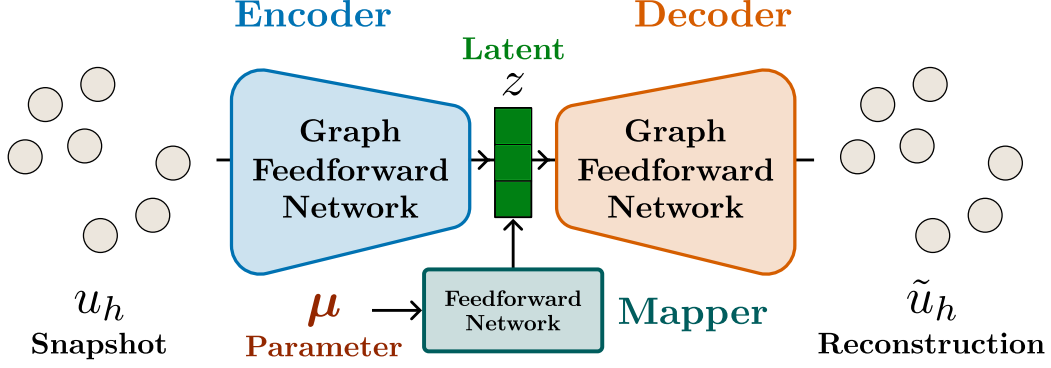


FIGURE 8. GFN-ROM architecture, consisting of an autoencoder and a mapper. During inference, predictions can be computed by decoding the mapper output.

where we have introduced the mapper weight hyperparameter  $\omega$ . We note that at inference, the encoder is no longer required and, for a parameter  $\mu_t$  and mesh  $\mathcal{M}_n^t$ , a GFN-ROM model with weights defined on a mesh  $\mathcal{M}_o$  predicts  $\text{dec}^{\mathcal{M}_o \rightarrow \mathcal{M}_n^t}(\text{map}(\mu_t))$ .

**6.1. Main features of the architecture.** The proposed GFN-ROM architecture benefits from a number of desirable properties that we discuss in the following sections, which make it an attractive tool for applications with multifidelity data.

**6.1.1. Self-consistency.** The GFN transforms are self-consistent approaches, meaning that a series of two transforms from  $\mathcal{M}_o$  to  $\mathcal{M}_n$  and then back to  $\mathcal{M}_o$  recovers exactly the original weights  $\mathbf{W}^e, \mathbf{W}^d, \mathbf{b}^d$  provided that the transform  $\mathcal{M}_o$  to  $\mathcal{M}_n$  is expansive, i.e.

$$\mathbf{W}^e, \mathbf{W}^d, \mathbf{b}^d = \text{GFN}^{\mathcal{M}_n \rightarrow \mathcal{M}_o}(\text{GFN}^{\mathcal{M}_o \rightarrow \mathcal{M}_n}(\mathbf{W}^e, \mathbf{W}^d, \mathbf{b}^d)).$$

The self-consistency, a proof of which is given in Appendix A.4, is a marked advantage in comparison to any other interpolation methods such as Kriging, linear regression, spline, and  $k$ NN (with the exception of the case  $k = 1$ ). This means that no information is lost during the expansive transform and the original weights can be directly recovered, unlike for other interpolation-based approaches.

**6.1.2. Bound on super- and sub-resolution error of GFN-ROM.** GFN-ROM is a resolution-invariant architecture, meaning one is often interested in evaluating the model on a finer discretisation (super-resolution) or coarser discretisation (sub-resolution) compared to a reference resolution. A particularly attractive feature of the architecture is that one can bound the performance on any testing fidelity given certain information. Specifically, given a bound  $\delta$  on the difference between the features of nearest neighbours, and a bound  $\tau$  on the performance of GFN-ROM on the known mesh  $\mathcal{M}_o$ , i.e.<sup>4</sup>

$$(19) \quad \forall i_{\mathcal{M}_o}, \quad i_{\mathcal{M}_o} \leftarrow \rightarrow j_{\mathcal{M}_n} \implies |u(x_{i_{\mathcal{M}_o}}) - u(x_{j_{\mathcal{M}_n}})| \leq \delta,$$

$$(20) \quad \forall i_{\mathcal{M}_o}, \quad |u(x_{i_{\mathcal{M}_o}}) - \text{dec}^{\mathcal{M}_o \rightarrow \mathcal{M}_o}(\text{map}(\mu))_{i_{\mathcal{M}_o}}| \leq \tau,$$

one can show that the performance of GFN-ROM on the new mesh  $\mathcal{M}_n$  can be bounded as:

$$(21) \quad \forall i_{\mathcal{M}_n}, \quad |u(x_{i_{\mathcal{M}_n}}) - \text{dec}^{\mathcal{M}_o \rightarrow \mathcal{M}_n}(\text{map}(\mu))_{i_{\mathcal{M}_n}}| \leq \tau + \delta.$$

A proof is given in Appendix A.6. We note that there is no dependence at all on the weights and biases, only on the mesh similarities and the GFN-ROM performance on the reference resolution.

<sup>4</sup>We use the notation  $u(x_{i_{\mathcal{M}_o}})$ , equivalent to  $u_{i_{\mathcal{M}_o}}$ , in order to emphasise the dependence of  $u$  on spatial position.

6.1.3. *Bound on super- and sub-resolution error of the GFN-ROM mapper.* Given the same bound  $\delta$  as in Equation 19, and a bound  $\alpha$  on the performance of the mapper on the known mesh  $\mathcal{M}_o$ , i.e.

$$(22) \quad \forall i, \quad |\text{map}(\boldsymbol{\mu})_i - \text{enc}^{\mathcal{M}_o \rightarrow \mathcal{M}_o}(\mathbf{u}_{\mathcal{M}_o})_i| \leq \alpha,$$

one can show that the performance of the mapper on the new mesh  $\mathcal{M}_n$  can be bounded as:

$$(23) \quad \forall i, \quad |\text{map}(\boldsymbol{\mu})_i - \text{enc}^{\mathcal{M}_o \rightarrow \mathcal{M}_n}(\mathbf{u}_{\mathcal{M}_n})_i| \leq \alpha + \delta C^{P+1} \|\mathbf{W}^e\|_\infty.$$

A proof is given in Appendix A.7.

6.1.4. *Bound on super- and sub-resolution error of the GFN-ROM autoencoder.* Given the same bound  $\delta$  as in Equation 19, and a bound  $\beta$  on the performance of the autoencoder on the known mesh  $\mathcal{M}_o$ , i.e.

$$(24) \quad \forall i_{\mathcal{M}_o}, \quad |u(x_{i_{\mathcal{M}_o}}) - \text{dec}^{\mathcal{M}_o \rightarrow \mathcal{M}_o}(\text{enc}^{\mathcal{M}_o \rightarrow \mathcal{M}_o}(\mathbf{u}_{\mathcal{M}_o}))_{i_{\mathcal{M}_o}}| \leq \beta,$$

one can show that the performance of the autoencoder on the new mesh  $\mathcal{M}_n$  can be bounded as:

$$(25) \quad \forall i_{\mathcal{M}_n}, \quad |u(x_{i_{\mathcal{M}_n}}) - \text{dec}^{\mathcal{M}_o \rightarrow \mathcal{M}_n}(\text{enc}^{\mathcal{M}_o \rightarrow \mathcal{M}_n}(\mathbf{u}_{\mathcal{M}_n}))_{i_{\mathcal{M}_n}}| \leq \beta + \delta + \delta C^{Q+1} \|\mathbf{W}^d\|_\infty \|\mathbf{W}^e\|_\infty.$$

A proof is given in Appendix A.8. The dependence on the norm of the weights in the bound suggests that regularisation is important to allow for good generalisation for the autoencoder.

6.2. **Link between GFN-ROM and novel architectures.** Following the discussion in Section 4.2 and Section 4.3, we discuss here how GFN-ROM is connected with state-of-the-art resolution invariant approaches.

6.2.1. *GNNs and GFN-ROM.* Concerning graph neural networks, we can interpret a GFN encoder layer and a GFN decoder layer as two different types of GNN: namely, a graph pooling layer and a graph unpooling layer, respectively. We can consider the input to the autoencoder, which is some PDE solution  $u(x, \boldsymbol{\mu})$  evaluated on some discretized domain, as a fully connected graph. The GFN encoder layer is then tasked with computing a vector representation summarising the input graph, i.e. a graph pooling task. The GFN decoder layer has precisely the opposite task of reconstructing a graph given a summarised representation i.e. graph unpooling. At inference, the overall GFN-ROM architecture can thus be thought of as a graph unpooling model, where the PDE solution is identified by the parameters  $\boldsymbol{\mu}$ .

6.2.2. *NOs and GFN-ROM.* A parameterised PDE with a parameter  $a \in \mathcal{A}$  and solution  $u \in \mathcal{U}$ , where  $\mathcal{A}$  and  $\mathcal{U}$  are Banach spaces, defines a mapping  $\mathcal{G}^\dagger : \mathcal{A} \rightarrow \mathcal{U}$ , i.e.  $u(x) = \mathcal{G}^\dagger[a](x)$ . A neural operator aims to directly approximate this mapping  $\mathcal{G}^\dagger$  by means of an operator  $\mathcal{G}_\theta \approx \mathcal{G}^\dagger$  with trainable parameters  $\theta$ . A single-layer neural operator can be written as

$$G_\theta = \mathcal{Q} \circ \sigma(W + \mathcal{K} + b) \circ \mathcal{P},$$

where  $\mathcal{P}$ ,  $\mathcal{Q}$  are the local lifting and projection mappings,  $W$  is a local linear operator (matrix),  $\mathcal{K}$  is an integral kernel operator,  $b$  is a bias function and  $\sigma$  is a pointwise activation function. We refer to Kovachki *et al.* for further details [55]. Considering a simplified neural operator with  $W = 0$  and no lifting or projection mappings, we obtain

$$(26) \quad G_\theta[v](y) = \sigma \left( \int_D \kappa(x, y) v(x) dx + b(y) \right),$$

where  $\kappa$  is a kernel function and  $D$  is the domain of integration. Usually, one does not have complete access to the function  $v(x)$ , but only to its evaluations at certain points, meaning this integral has to be numerically computed. Supposing we are given function evaluations on a mesh  $\mathcal{M}$ , the numerical approximation of the integral reads as

$$(27) \quad G_\theta[v](y) \approx \sigma \left( \sum_{i_{\mathcal{M}}} h_{i_{\mathcal{M}}} \kappa(x_{i_{\mathcal{M}}}, y) v(x_{i_{\mathcal{M}}}) + b(y) \right),$$

where  $h_{i_{\mathcal{M}}}$  is a factor depending on the integration method.

GFN-ROM is closely related to NOs. In fact, in the expansive case, GFN-ROM can be recast as a particular instance of NOs, since we can directly obtain both the encoder and decoder. To make the link with GFN-ROM, we start with the mesh  $\mathcal{M}_o$ , the associated weights and biases ( $\mathbf{W}^e$ ,  $\mathbf{b}^e$ ,  $\mathbf{W}^d$ ,  $\mathbf{b}^d$ ), and choose a particular kernel  $\kappa$ .

Starting from Equation 26, discarding the  $y$ -dependence and taking  $D = \Omega$ ,  $b = b_i^e$ , and  $\kappa(x) = W_{ik_{\mathcal{M}_o}}^e / |B_{k_{\mathcal{M}_o}}|$  where  $k_{\mathcal{M}_o} \leftarrow x$ , and  $B_{k_{\mathcal{M}_o}} = \{x \in \Omega \mid k_{\mathcal{M}_o} \leftarrow x\}$ , one can write<sup>5</sup>

$$\begin{aligned}
 G_\theta[u] &= \sigma \left( \int_D \kappa(x) u(x) dx + b \right), \\
 &= \sigma \left( \int_\Omega \frac{W_{ik_{\mathcal{M}_o}}^e}{|B_{k_{\mathcal{M}_o}}|} u(x) dx + b_i^e \right), \quad k_{\mathcal{M}_o} \leftarrow x, \\
 &= \sigma \left( \sum_{k_{\mathcal{M}_o}=1}^{N_o} W_{ik_{\mathcal{M}_o}}^e \frac{1}{|B_{k_{\mathcal{M}_o}}|} \int_{B_{k_{\mathcal{M}_o}}} u(x) dx + b_i^e \right).
 \end{aligned}
 \tag{28}$$

Approximating the integral with observations of  $u(x)$  on a mesh  $\mathcal{M}_n$ , with  $k_{\mathcal{M}_o} \leftarrow j_{\mathcal{M}_n}$ , gives the formula for the GFN-ROM encoder in the expansive case

$$\begin{aligned}
 G_\theta[u] &= \sigma \left( \sum_{k_{\mathcal{M}_o}=1}^{N_o} W_{ik_{\mathcal{M}_o}}^e \text{mean}_{\forall j_{\mathcal{M}_n} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow j_{\mathcal{M}_n}} u_{j_{\mathcal{M}_n}} + b_i^e \right), \\
 &= \sigma \left( \sum_{j_{\mathcal{M}_n}=1}^{N_n} \frac{W_{ik_{\mathcal{M}_o}}^e}{|\{h_{\mathcal{M}_n} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow h_{\mathcal{M}_n}\}|} u_{j_{\mathcal{M}_n}} + b_i^e \right), \quad \text{where } k_{\mathcal{M}_o} \leftarrow j_{\mathcal{M}_n}, \\
 &= \text{enc}^{\mathcal{M}_o \rightarrow \mathcal{M}_n}(\mathbf{u}_{\mathcal{M}_n})_i.
 \end{aligned}
 \tag{29}$$

Similarly, the decoder can be obtained from Equation 26 without any  $x$ -dependence. Taking  $\tilde{\kappa} = |D|\kappa$ , setting  $\sigma$  to the identity function,  $\tilde{\kappa}(y) = \mathbf{W}_{k_{\mathcal{M}_o},:}^d$  where  $k_{\mathcal{M}_o} \leftarrow y$ , and  $b(y) = b_{k_{\mathcal{M}_o}}^d$  where  $k_{\mathcal{M}_o} \leftarrow y$ , one obtains

$$\begin{aligned}
 G_\theta[\mathbf{z}](y) &= \sigma \left( \int_D \kappa(y) \mathbf{z} dx + b(y) \right), \\
 &= \sigma(\tilde{\kappa}(y) \mathbf{z} + b(y)), \\
 &= \sum_{j=1}^L W_{k_{\mathcal{M}_o}j}^d z_j + b_{k_{\mathcal{M}_o}}^d, \quad k_{\mathcal{M}_o} \leftarrow y.
 \end{aligned}
 \tag{30}$$

Considering  $y = \mathcal{M}_n[i_{\mathcal{M}_n}]$ , we recover the GFN-ROM decoder in the expansive case as

$$\begin{aligned}
 G_\theta[\mathbf{z}](y) &= \sum_{j=1}^L W_{k_{\mathcal{M}_o}j}^d z_j + b_{k_{\mathcal{M}_o}}^d, \\
 &= \text{dec}^{\mathcal{M}_o \rightarrow \mathcal{M}_n}(\mathbf{z})_{i_{\mathcal{M}_n}}.
 \end{aligned}
 \tag{31}$$

Like the decoder, the GFN-ROM model at inference in the expansive case can be thought of as a neural operator, but with the mapper acting as the lifting function. Thus, in such settings, GFN-ROM encoder and decoder are very closely linked to resolution-invariant neural operators with a choice of piecewise constant kernel function.

---

<sup>5</sup>We abuse notation to write  $k_{\mathcal{M}_o} \leftarrow x$  since  $x$  is a position and not an index, but the meaning is analogous to before, i.e.  $k_{\mathcal{M}_o} = \text{argmin}_{h_{\mathcal{M}_o}} |\mathcal{M}_o[h_{\mathcal{M}_o}] - x|$ .

**6.2.3. ROM-related properties.** In addition to the above proofs, the GFN-ROM architecture benefits from a number of typical properties important for reduced order models. Namely, it is purely data-driven and non-intrusive, not requiring any knowledge of the generation process for the training data. It is furthermore a nonlinear method, leveraging an autoencoder for nonlinear compression. The method is suitable for unstructured meshes, which is vital in order to deal with general PDEs, where the presence of complex domains typically lead to unstructured meshes. Most importantly, the method is a completely multifidelity approach and is capable of inference on arbitrary meshes.

## 7. ADAPTIVE MULTIFIDELITY TRAINING

The GFN method proposes a means of transferring weights from one graph  $\mathcal{M}_o$  to a new graph  $\mathcal{M}_n$  as  $\tilde{\mathbf{W}}^e, \tilde{\mathbf{W}}^d, \tilde{\mathbf{b}}^d = \text{GFN}^{\mathcal{M}_o \rightarrow \mathcal{M}_n}(\mathbf{W}^e, \mathbf{W}^d, \mathbf{b}^d)$ , allowing one to evaluate a feedforward-based approach on new meshes. Whilst this is already a major advantage, it does not prescribe how to effectively train in a multifidelity setting.

In this section, we discuss means of training in multifidelity applications. We aim at learning from a series of  $T$  meshes  $\mathcal{M}_n^1, \dots, \mathcal{M}_n^T$  by training the GFN-ROM model with weights and biases corresponding to a fixed or an adaptive mesh  $\mathcal{M}_o$ . We describe below the two approaches, the difference between which is illustrated in Figure 9.

**7.1. Fixed mesh.** The GFN transforms are entirely differentiable, meaning that for weights and biases  $\mathbf{W}^e$ ,  $\mathbf{W}^d$  and  $\mathbf{b}^d$  associated on a fixed mesh  $\mathcal{M}_o$ , it is possible to compute gradients  $\nabla_{\mathbf{W}^e} \mathcal{L}$ ,  $\nabla_{\mathbf{W}^d} \mathcal{L}$  and  $\nabla_{\mathbf{b}^d} \mathcal{L}$  when training on new meshes. As a result, it is in fact possible to use the standard gradient-based optimisation methods to learn the optimal weights  $\mathbf{W}^e$ ,  $\mathbf{W}^d$  and  $\mathbf{b}^d$ , which are always associated to the mesh  $\mathcal{M}_o$ . If one considers the mesh  $\mathcal{M}_o$  as being a suitable sampling of the domain, this approach therefore suffices as a means of training the model in a multifidelity setting. However, the choice of mesh  $\mathcal{M}_o$  is extremely important for the performance of the model. The fixed nature of  $\mathcal{M}_o$  is also undesirable since, like interpolation approaches onto fixed grids, it does not allow the model to adapt the mesh  $\mathcal{M}_o$  to better match the training data it sees.

**7.2. Adaptive mesh.** GFN-ROM differs from interpolation onto a fixed grid since it is possible to transform weights and biases from a coarse mesh into weights and biases for a finer mesh. Thus, one can always overwrite the old weights and biases corresponding to the old mesh with new weights and biases of the new mesh, and then optimise the new weights. As a result, the approach leads to a number of advantages in allowing the architecture to adapt its number of parameters during training. That is, instead of having a single fixed mesh  $\mathcal{M}_o$  associated to the fixed weights and biases for the model, we replace it by a master mesh  $\mathcal{M}^t$  which can change during each training sample  $t = 1, \dots, T$ . Importantly, the sequence of master meshes during training must be hierarchically increasing i.e.  $\mathcal{M}^0 \subseteq \dots \subseteq \mathcal{M}^T$ , since we do not wish to lose information at any point.

A simple scheme for adaptive training could be setting  $\mathcal{M}^t = \mathcal{M}^{t-1} \cup \mathcal{M}_n^t$ . However, this approach is suboptimal since it necessarily adds every node seen during training to the master mesh. This means that the master mesh can become prohibitively large, and result in a worse time complexity of  $O((N_o + N_n) \log N_o + (N_o + N_n) \log N_n)$  for the transform  $\mathcal{M}_o \rightarrow \mathcal{M}_n$ .

Instead, a better approach is possible without incurring the computational issues of the previous algorithm. As discussed in subsubsection 5.2.3 and summarised in Algorithm 3, the GFN transforms can be decomposed into an expansive and agglomerative part via the choice of a suitable intermediate mesh given in Equation 13. This intermediate mesh is already computed during the GFN transforms, meaning it does not represent an extra operation. Furthermore, the intermediate mesh is larger than the original mesh. Therefore, it is possible to use such mesh to develop a computationally efficient adaptive approach i.e.,  $\mathcal{M}^t = \mathcal{M}^{t-1} \cup \{\mathcal{M}_n^t [i_{\mathcal{M}_n^t}] \text{ s.t. } j_{\mathcal{M}^{t-1}} \leftarrow i_{\mathcal{M}_n^t} \text{ but not } j_{\mathcal{M}^{t-1}} \rightarrow i_{\mathcal{M}_n^t}\}$ .

For a transform  $\mathcal{M}_o \rightarrow \mathcal{M}_n$ , we retain the time complexity of the non-adaptive algorithm. Furthermore, this approach only adds nodes to the master mesh when the master mesh undersamples<sup>6</sup>. We note that because the weights and biases are changing shape along with the master mesh during

<sup>6</sup>Undersampling here refers purely to nearest-neighbour information i.e., areas where a node in the master mesh has multiple nearest neighbours are considered undersampled.

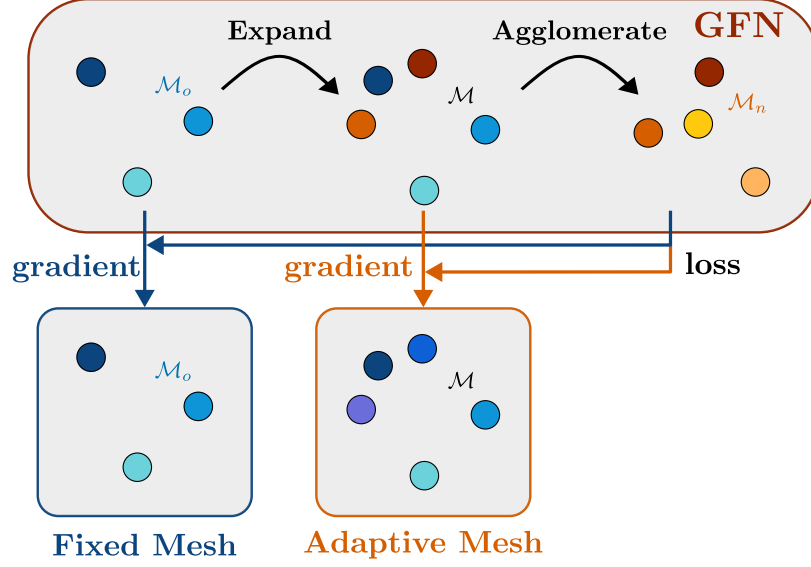


FIGURE 9. Training with fixed or adaptive mesh methods when given a starting mesh  $\mathcal{M}_o$  and new mesh  $\mathcal{M}_n$ . The fixed mesh approach updates the original mesh weights, whilst the adaptive mesh approach updates weights on a new mesh  $\mathcal{M}$ , created from information from both meshes.

training for adaptive methods, one cannot use Adam, RMSProp or any momentum-based optimisation methods for updates<sup>7</sup>.

The training loop for GFN-ROM is given in Algorithm 4, illustrating the difference between the adaptive and fixed mesh modes.

---

**Algorithm 4** Training algorithm for GFN-ROM, showing both adaptive and fixed mesh approaches.

---

**Require:**  $T$  training samples of parameter and solution pairs i.e.  $\{(\mu_t, \mathbf{u}_{\mathcal{M}_n^t}(\mu_t))\}_{t=1}^T$   
 Initialise a mesh  $\mathcal{M}_o$  with associated GFN weights  $\mathbf{W}^e, \mathbf{W}^d, \mathbf{b}^d$   
 Initialise  $\theta$ : all parameters needed for GFN-ROM excluding  $\mathbf{W}^e, \mathbf{W}^d, \mathbf{b}^d$   
**for**  $t \leftarrow 1, \dots, T$  **do**  
    $\mathcal{M} = \mathcal{M}_o \cup \{\mathcal{M}_n^t[i_{\mathcal{M}_n^t}] \text{ s.t. } j_{\mathcal{M}_o} \leftarrow i_{\mathcal{M}_n^t} \text{ but not } j_{\mathcal{M}_o} \rightarrow i_{\mathcal{M}_n^t}\}$   
   **if** adaptive **then** ▷ adaptive mode  
      $\mathbf{W}^e, \mathbf{W}^d, \mathbf{b}^d \leftarrow \text{EXPAND}(\mathbf{W}^e, \mathbf{W}^d, \mathbf{b}^d, \mathcal{M}_o, \mathcal{M})$   
      $\mathcal{M}_o \leftarrow \mathcal{M}$   
      $\tilde{\mathbf{W}}^e, \tilde{\mathbf{W}}^d, \tilde{\mathbf{b}}^d \leftarrow \text{AGGLOMERATE}(\mathbf{W}^e, \mathbf{W}^d, \mathbf{b}^d, \mathcal{M}_o, \mathcal{M}_n)$   
   **else** ▷ fixed mode  
      $\hat{\mathbf{W}}^e, \hat{\mathbf{W}}^d, \hat{\mathbf{b}}^d \leftarrow \text{EXPAND}(\mathbf{W}^e, \mathbf{W}^d, \mathbf{b}^d, \mathcal{M}_o, \mathcal{M})$   
      $\tilde{\mathbf{W}}^e, \tilde{\mathbf{W}}^d, \tilde{\mathbf{b}}^d \leftarrow \text{AGGLOMERATE}(\hat{\mathbf{W}}^e, \hat{\mathbf{W}}^d, \hat{\mathbf{b}}^d, \mathcal{M}, \mathcal{M}_n)$   
   **end if**  
   Compute loss  $\mathcal{L}(\mu_t, \mathbf{u}_{\mathcal{M}_n^t}; \tilde{\mathbf{W}}^e, \tilde{\mathbf{W}}^d, \tilde{\mathbf{b}}^d, \theta)$   
   Update parameters  $\mathbf{W}^e, \mathbf{W}^d, \mathbf{b}^d, \theta$  using gradients  $\nabla_{\mathbf{W}^e} \mathcal{L}, \nabla_{\mathbf{W}^d} \mathcal{L}, \nabla_{\mathbf{b}^d} \mathcal{L}, \nabla_{\theta} \mathcal{L}$   
**end for**

---

<sup>7</sup>In practice, we often know all of the training fidelities we shall see during training. In such cases, the final mesh  $\mathcal{M}^T$  can be precomputed and the model trained as for a fixed mesh.

## 8. NUMERICAL SIMULATIONS

Having introduced the GFN-ROM architecture, providing guarantees on super- and sub-resolution errors, we examine its performance on three challenging benchmarks for MOR. We compare the results of the methodology to intrusive and non-intrusive ROMs: namely, POD-Galerkin (POD-G) and POD with projection (representing the best possible linear method), POD-NN [47, 76] and GCA-ROM [77]. For fairness, we strive to make the MOR baselines for the different models as comparable as possible by adopting similar architectural and hyperparameter choices between methods, as shown in Appendix B.1. Importantly, we do not fine tune the architecture or seek the best possible initialisation for each specific problem. As such, we stress that these results do not represent the best possible performance, but illustrate that GFN-ROM is able to directly handle several challenging benchmarks with no specific fine tuning. As a general rule, we have chosen a latent/reduced dimension  $N, L = \lfloor 1.5 \times N_\mu \rfloor$ , and the small-data regime with a train/test split of 30/70.

Each of the problems examined here are selected to showcase a different physical or computational complexity. The first two benchmarks have been previously investigated with GCA-ROM [77], which we use as a baseline for nonlinear reduction – namely the Graetz problem and an advection-dominated problem. Furthermore, we examine a Stokes flow benchmark with a 7-dimensional parameter space to illustrate the model’s capability of dealing with high-dimensional physical and geometric parameter spaces, and the low data regime with sparse samplings of the parameter space. Training data is generated via finite element methods using the RBniCS package [90], which is built on top of the FEniCS package [1, 67]. For each problem, we consider four meshes in order to test the multifidelity performance of our architecture<sup>8</sup>.

The results for single fidelity experiments across our baselines are shown in Table 1.

	POD-G	POD-Proj	POD-NN	GCA-ROM	GFN-ROM			
	Large	Large	Large	Large	Large	Medium	Small	Tiny
Graetz	3.44	3.44	3.54	0.74	1.02	0.88	0.98	1.28
Advection	27.09	25.66	30.58	4.73	4.73	4.48	7.22	12.35
Stokes	2.45	2.45	2.94	4.63	4.24	4.12	4.44	5.55

TABLE 1. Mean relative errors (%) evaluated on large mesh.

Since GFN-ROM is capable of being trained on multifidelity data, we investigate its generalisation ability across different resolutions. In Table 2, we report the change in performance of GFN-ROM when one substitutes half of the training data with cheaper, lower-fidelity data.

	GFN-ROM					
	Large & Medium	Large & Small	Large & Tiny	Medium & Small	Medium & Tiny	Small & Tiny
Graetz	0.96 (+0.06)	0.98 (+0.04)	1.40 (-0.37)	4.44 (-3.56)	1.03 (-0.15)	1.09 (-0.11)
Advection	5.02 (-0.30)	5.35 (-0.62)	5.63 (-0.91)	5.22 (-0.73)	6.27 (-1.79)	8.77 (-1.55)
Stokes	3.63 (+0.61)	4.94 (-0.70)	4.44 (-0.19)	4.51 (-0.39)	5.56 (-1.45)	5.65 (-1.21)

TABLE 2. Mean relative errors (%) evaluated on large mesh, and the change when training only with the finer of the two in Table 1. A positive change indicates reduced error and thus better performance.

GFN-ROM is a more lightweight and flexible method compared to existing state-of-the-art ROMs. We show the method’s computational efficiency with respect to GCA-ROM in Table 3. GFN-ROM offers large savings in terms of training time (up to 63x reduction) and in terms of the number of trainable parameters (up to 25x reduction). All simulations have been performed on a workstation equipped with an Nvidia Quadro RTX 4000 GPU.

<sup>8</sup>The meshes are generated from the largest mesh by using the moving front subsampling algorithm [61].

		GFN-ROM				GCA-ROM
		Large	Medium	Small	Tiny	Large
Graetz	Training time (s)	119	46	31	26	600
	Trainable parameters	2 898 761	911 004	311 910	115 821	2 898 795
	Mesh nodes	7205	2248 (31%)	754 (10%)	265 (4%)	7205
Advection	Training time (s)	124	53	34	23	408.9
	Trainable parameters	3 538 757	1 110 702	387 298	140 282	3 538 791
	Mesh nodes	8801	2746 (31%)	942 (11%)	326 (4%)	8801
Stokes	Training time (s)	248	90	45	31	1949
	Trainable parameters	2 827 589	905 596	316 126	118 032	2 827 623
	Mesh nodes	7019	2226 (32%)	756 (11%)	262 (4%)	7019

TABLE 3. Computational efficiency of GFN-ROM in comparison to GCA-ROM, and its scaling w.r.t. the mesh size.

**8.1. Graetz problem.** The Graetz problem represents a common benchmark in MOR, combining a forced heat diffusion with horizontal heat advection in a parametrised geometry. Specifically, we study a steady-state Graetz problem on the parametrised geometry  $\Omega(\mu_1) = \Omega^1 \cup \Omega^2(\mu_1) \in \mathbb{R}^2$ , with  $\Omega^1 = [0, 1] \times [0, 1]$  and  $\Omega^2(\mu_1) = [1, 1 + \mu_1] \times [0, 1]$  illustrated in Figure 10a, and discretised by four meshes as summarised in Table 3.

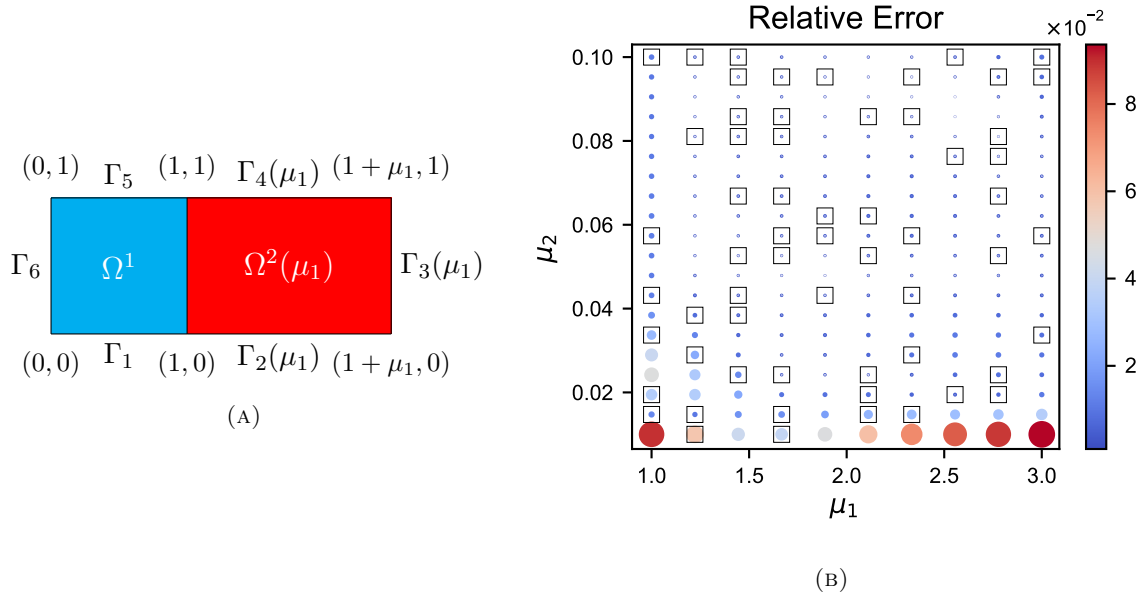


FIGURE 10. (A) The parameterised domain for the Graetz problem. (B) Mean relative errors for GFN-ROM trained on the large mesh for the Graetz problem. Squares indicate the parameter realisations seen during training.

Homogeneous Dirichlet conditions are imposed on  $\Gamma_D = \Gamma_1 \cup \Gamma_5 \cup \Gamma_6$ , whilst a constant Dirichlet boundary condition is imposed on  $\Gamma_G = \Gamma_2(\mu_1) \cup \Gamma_4(\mu_1)$  as  $u(\boldsymbol{\mu})|_{\Gamma_G} = 1$ . The remaining boundary  $\Gamma_3(\mu_1)$  is given a homogeneous von Neumann boundary condition. The weak formulation of the Graetz problem reads as follows: given  $\boldsymbol{\mu} \in \mathbb{P}$ , find  $u(\boldsymbol{\mu}) \in \mathbb{U}(\mu_1)$  s.t.

$$(32) \quad \mu_2 \int_{\Omega(\mu_1)} \nabla u \cdot \nabla v \, d\Omega + \int_{\Omega(\mu_1)} y(1-y) \frac{\partial u}{\partial x} v \, d\Omega = 0, \quad \forall v \in \mathbb{V},$$



where  $\mathbb{U}(\mu_1) = \{v \in H^1(\Omega(\mu_1)) : v|_{\Gamma_D} = 0, v|_{\Gamma_G} = 1\}$  and  $\mathbb{V}(\mu_1) = \{v \in H^1(\Omega(\mu_1)) : v|_{\Gamma_D \cup \Gamma_G} = 0\}$ . Note that due to the parametrised domain, these function spaces are dependent on the geometric parameter. This problem is parametrised by  $\boldsymbol{\mu} = (\mu_1, \mu_2)$ , where  $\mu_1$  is a geometrical parameter governing the length of  $\Omega^2$ , and  $\mu_2$  is a physical parameter, namely the diffusivity coefficient. We consider a dataset with 200 snapshots computed on a uniform grid of  $\mathbb{P}$ , with 10 and 20 equispaced values for  $\mu_1 \in [1, 3]$  and  $\mu_2 \in [0.01, 0.1]$ , respectively.

Training and testing the performance on GFN-ROM on the largest mesh, we achieve a mean relative error of 1.02% over the full dataset. As illustrated in Figure 10b, we can see GFN-ROM is well able to capture the overall solution. Note that this performance is very good, particularly considering only 60 parameter realisations are seen for training, and the model learns a latent representation of size  $L = 3$ . Interestingly, for the Graetz problem we see that errors are much higher for small values of  $\mu_2$  i.e. in the low diffusivity region, where the advection dominates. Moreover, few training samples are representing this regime, making it more difficult for GFN-ROM to learn how to model the region.

Nonetheless, the overall performance on GFN-ROM is extremely good on this benchmark, achieving over a 3x reduction in mean relative error compared to linear methods (see Table 1), and comparable performance to GCA-ROM.

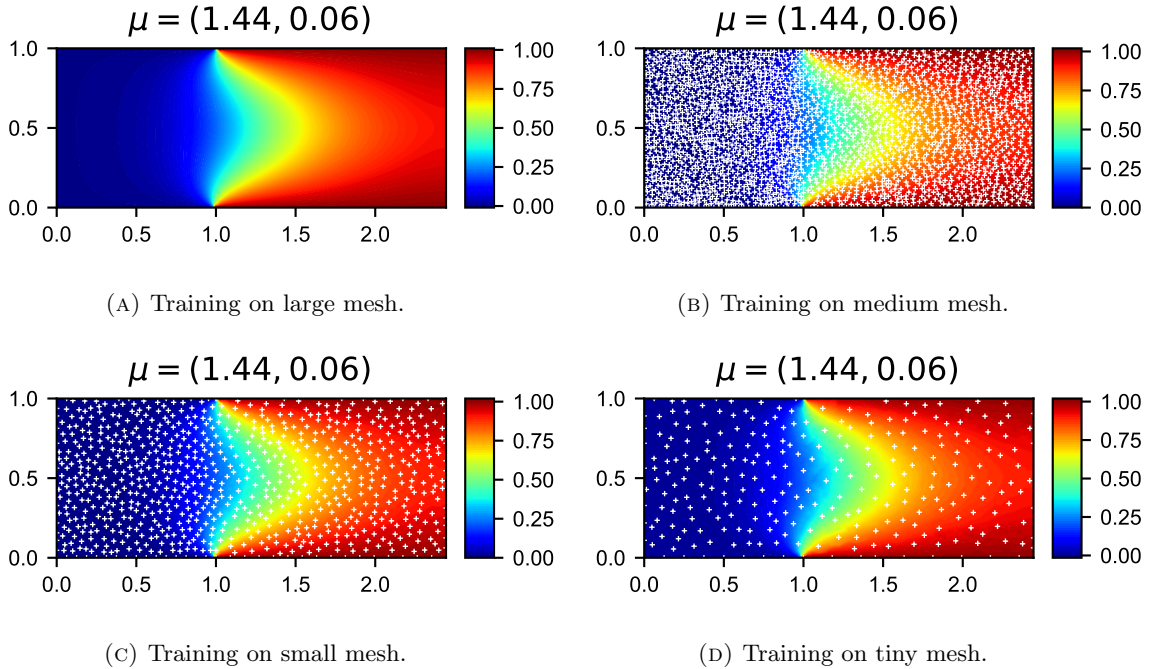


FIGURE 11. GFN-ROM solution fields for the Graetz problem when trained on (A) large mesh, (B) medium mesh, (C) small mesh, and (D) tiny mesh. The mesh points are shown by white pluses.

Moreover, we can further improve the performance of GFN-ROM by leveraging its resolution invariance. In fact, when training on a coarser representation, e.g. the medium mesh, the mean relative error is reduced to 0.88%, illustrating the new opportunities arising from GFN-ROM's ability to perform super-resolution. Even when training on the coarsest (tiny) discretisation, the mean relative error increases only slightly by 0.26% to 1.28% despite this representing only 4% of the training data. In Figure 11, we show the predictions of the four different GFN-ROM models trained on the four different meshes for a parameter realisation  $\boldsymbol{\mu} = (1.44, 0.06)$  not seen during training. The model predictions are qualitatively indistinguishable, showing that GFN-ROM is capable of

extracting enough information even on cheaper computational data without incurring any penalty on performance. Even further generalisations are possible thanks to the possibility of training on mixed-fidelity data. GFN-ROM shows excellent generalisation properties and performance is minorly impacted when substituting high-fidelity data for lower-fidelity data, as seen in Table 2. For example, the performance even improves from 1.02% when training solely on the large mesh to 0.96% when substituting half of the large mesh dataset with cheaper data from the medium mesh (representing a 35% reduction in training data).

**8.2. Advection-dominated problem.** Advection-dominated problems, due to the slow decay of the Kolmogorov  $n$ -width [36, 73], represent an important benchmarking case in the ROM context to compare linear and nonlinear reduction strategies. We consider a fixed domain  $\Omega = [0, 1] \times [0, 1] \in \mathbb{R}^2$ , illustrated in Figure 12a with homogeneous Dirichlet boundary conditions. We generate four meshes summarised in Table 3.

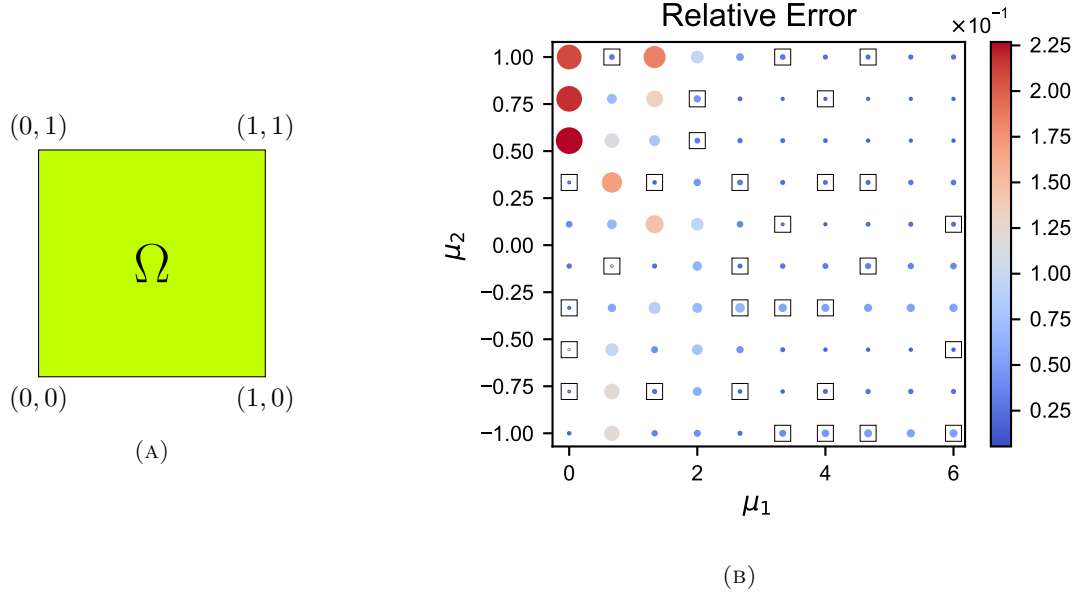


FIGURE 12. (A) The parameterised domain for the advection problem. (B) Mean relative errors for GFN-ROM trained on the large mesh for the advection problem. Squares indicate the parameter realisations seen during training.

The weak formulation of the advection problem reads as follows: given  $\boldsymbol{\mu} \in \mathbb{P}$ , find  $u(\boldsymbol{\mu}) \in \mathbb{V}$  s.t.

$$(33) \quad D(\mu_1) \int_{\Omega} \nabla u \cdot \nabla v \, d\Omega + \int_{\Omega} (\boldsymbol{\beta} \cdot \nabla u) v \, d\Omega = \int_{\Omega} v \, d\Omega, \quad \forall v \in \mathbb{V} = H_0^1(\Omega).$$

A total of 100 snapshots are generated for this problem on a uniform grid of  $\mathbb{P}$ , with 10 and 10 equispaced values for  $\mu_1 \in [0, 6]$  and  $\mu_2 \in [-1, 1]$ , respectively. The first parameter  $\mu_1$  determines the diffusion coefficient via  $D(\mu_1) = 10^{-\mu_1}$  and the second parameter  $\mu_2$  determines the strength and sign of the transport  $\boldsymbol{\beta}(\mu_2) = (\mu_2, \mu_2)$ .

As expected, POD-based techniques perform poorly due to their linear nature on this benchmark, with POD-G only achieving a mean relative error of 27.09% over the full dataset. GFN-ROM's nonlinear nature allows it to achieve a far superior mean relative error of only 4.73% when training on the large mesh, which matches GCA-ROM's performance. We show the relative errors over the parameter space in Figure 12b. We stress that this performance is possible even in the low data regime with only 30 training examples and a small latent representation of size  $L = 3$ . Contrarily to the Graetz problem, it can be seen that poorer performances occur for lower Péclet numbers

( $Pe \approx |\frac{\mu_2}{10^{-\mu_1}}|$ ) compared to higher Péclet numbers i.e. the model performs better when transport dominates. This is likely due to the fact that the model training data is skewed towards examples where the advection in the opposite direction dominates, making it less adept at predicting solutions where diffusion has a stronger impact and the  $\mu_2$  is positive.

On this benchmark, GFN-ROM's resolution invariant nature allows for training on cheaper meshes, and outperforming GCA-ROM, achieving a mean relative error of 4.48% when training on the medium mesh (69% reduction in training data). In Figure 13, we show the predictions of the GFN-ROM models trained on the four meshes for a parameter  $\mu = (3.33, -0.78)$  not seen during training. Qualitatively, we can see that all models but tiny are capable of learning well-enough the solution dynamics. The tradeoff to consider is of course given by the performance of large models and the computational efficiency of the smaller ones. Similarly to the Graetz benchmark, we also observed good generalisation for the multifidelity cases, as shown in Table 2.

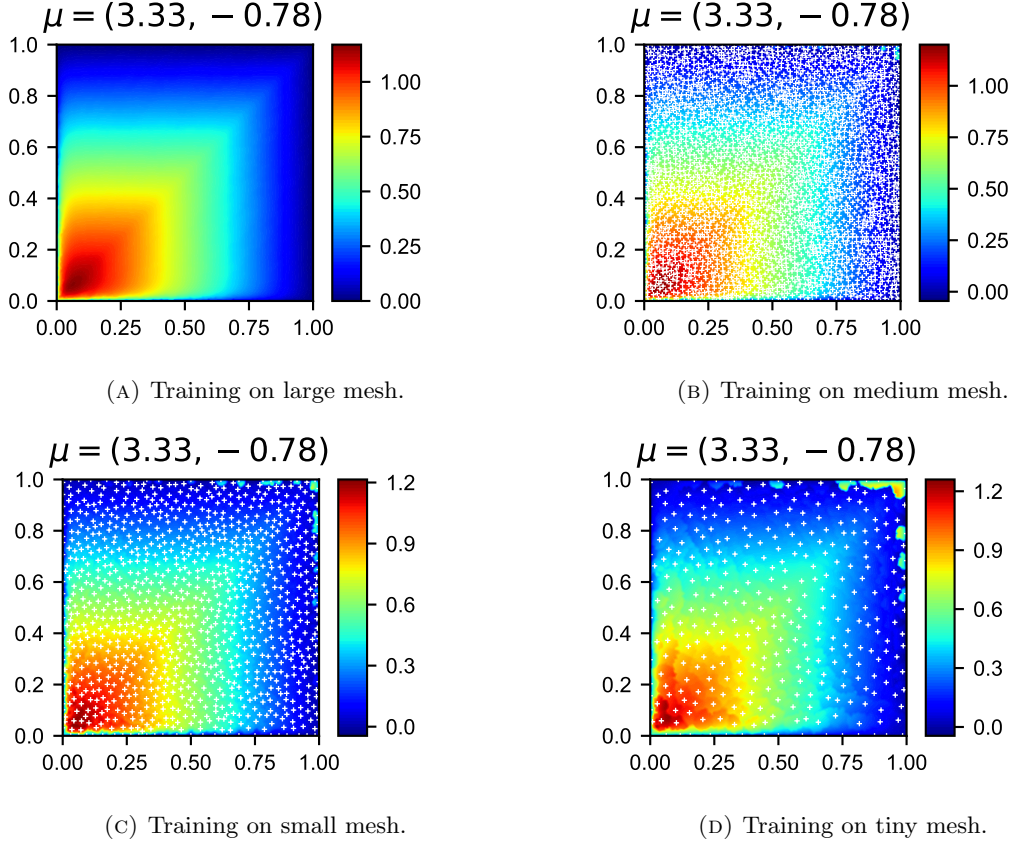


FIGURE 13. GFN-ROM solution fields for the advection problem when trained on (A) large mesh, (B) medium mesh, (C) small mesh, and (D) tiny mesh. The mesh points are shown by white pluses.

**8.3. Stokes problem.** We now consider a more complex benchmark in computational fluid dynamics, modelling a steady-state Stokes flow problem with high-dimensional physical/geometric parameter space and sparse sampling of it. This setting makes the problem particularly challenging to learn, and an important test case for the ability of a ROM to deal with the low-data regime. For simplicity, we consider the case of a viscosity of fixed value 1, but we allow for a variable forcing term  $\mathbf{f} = (0, \mu_6)$ , governed by a physical parameter  $\mu_6$ . We also consider a total of 6 geometrical parameters  $\mu_{\text{geo}} = (\mu_0, \mu_1, \mu_2, \mu_3, \mu_4, \mu_5)$ , to deform the domain depicted in Figure 14. We generate four

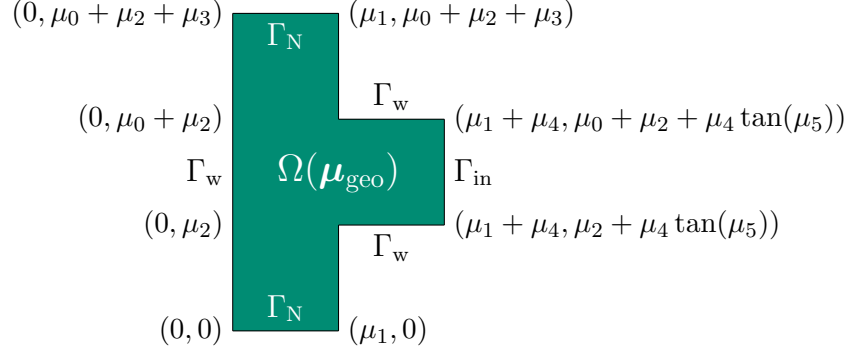


FIGURE 14. The domain for the Stokes problem.

meshes summarised in Table 3. As boundary conditions, the inlet  $\Gamma_{in}$  is given a Dirichlet boundary condition of  $\mathbf{u}_{in} = [4(y-1)(2-y), 0]$ , the walls  $\Gamma_w$  are given zero Dirichlet boundary conditions and the outlet  $\Gamma_N$  is left as a homogeneous Neumann boundary condition. A weak formulation for this problem reads: for a given parameter  $\boldsymbol{\mu} \in \mathbb{P}$ , find  $\mathbf{u}(\boldsymbol{\mu}) \in \mathbb{U}$ ,  $p \in \mathbb{M}$  s.t.

$$(34) \quad \begin{cases} \int_{\Omega} \nabla \mathbf{u} : \nabla \mathbf{v} \, d\Omega - \int_{\Omega} p \nabla \cdot \mathbf{v} \, d\Omega = \int_{\Omega} \mathbf{f} \cdot \mathbf{v} \, d\Omega, & \forall \mathbf{v} \in \mathbb{V}, \\ \int_{\Omega} q \nabla \cdot \mathbf{u} \, d\Omega = 0, & \forall q \in \mathbb{M}, \end{cases}$$

where the parameter dependent function spaces for the velocity are defined as  $\mathbb{V}(\boldsymbol{\mu}_{geo}) = \{\mathbf{v} \in (H^1(\Omega(\boldsymbol{\mu}_{geo})))^2 : \mathbf{v}|_{\Gamma_w \cup \Gamma_{in}} = \mathbf{0}\}$  and  $\mathbb{U}(\boldsymbol{\mu}_{geo}) = \{\mathbf{v} \in (H^1(\Omega(\boldsymbol{\mu}_{geo})))^2 : \mathbf{v}|_{\Gamma_w} = \mathbf{0}, \mathbf{v}|_{\Gamma_{in}} = \mathbf{u}_{in}\}$ , and the function space for the pressure is defined as  $\mathbb{M}(\boldsymbol{\mu}_{geo}) = L^2(\Omega(\boldsymbol{\mu}_{geo}))$ . The problem uses a mixed finite element discretisation with the velocity and pressure as solution variables, meaning that the inf-sup condition is necessary for the well posedness of this problem and as a result the supremiser operator  $T^\mu : \mathbb{M} \rightarrow \mathbb{V}$  is used (see [6] for more details).

We consider a sparse sampling with solutions computed only for the following parameter choices:  $\mu_0, \mu_1, \mu_2, \mu_3, \mu_4 \in \{0.5, 1.5\}$ ,  $\mu_5 \in \{-\frac{\pi}{6}, \frac{\pi}{6}\}$ ,  $\mu_6 \in \{-10, -8, -6, -4, -2, 0, 2, 4, 6, 8, 10\}$ , leading to a total of 704 high-fidelity solutions, and the field of interest is the magnitude of the the velocity  $|\mathbf{u}|$ .

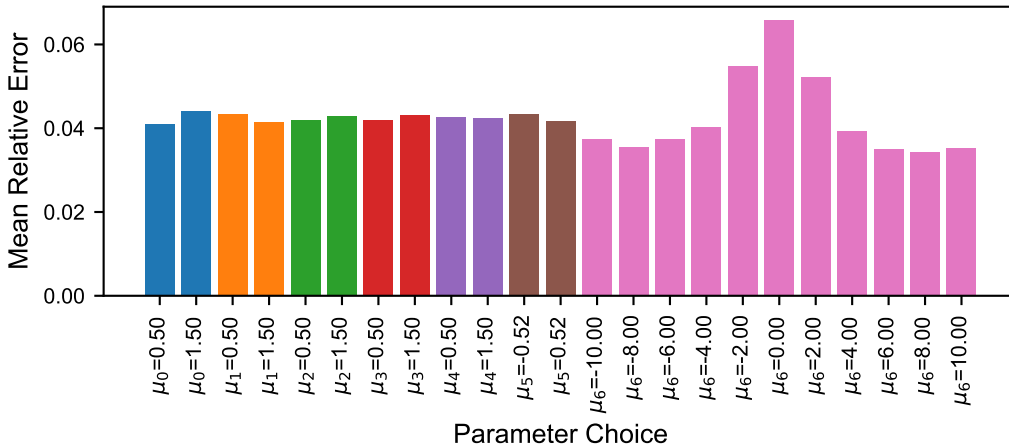


FIGURE 15. Histogram of the mean relative error for GFN-ROM trained on the large mesh for the Stokes problem.

Training on the large mesh, GFN-ROM achieves a mean relative error of 4.24%, outperforming GCA-ROM. We stress that this performance is possible whilst remaining in the low data regime, undersampling the parameter space. Despite this, the model is well able to capture the solution. As shown in Figure 15, model performance is found to be very similar on average for the possible geometrical parameters, while, as expected, the results are more sensitive to the physical parameter  $\mu_6$ , with slightly worse approximation for the case  $\mu_6 = 0$  where the forcing term vanishes.

GFN-ROM shows even further performance improvements by training on cheaper computational meshes, achieving a mean relative error of 4.12% for the medium mesh (representing a 68% reduction in the number of nodes). Even when training on the coarsest discretisation with just 4% of the data, performance only increases to 5.55% and is still able to capture the behaviour with respect to the physical and geometric parameters not seen during training, respectively in Figure 16 and Figure 17.

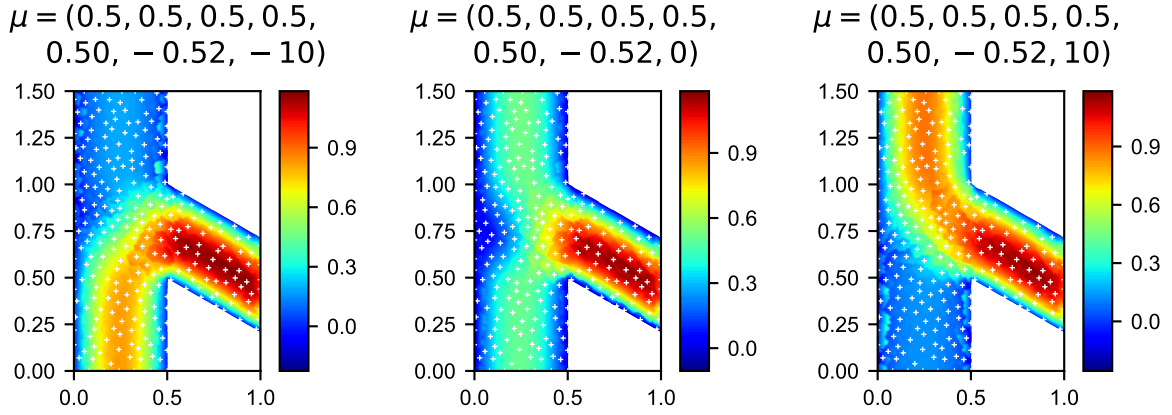


FIGURE 16. Super-resolution via GFN-ROM when training on the tiny mesh and evaluating on large mesh for variations in the physical parameter  $\mu_6$  showcasing downward, none and upward forcings.

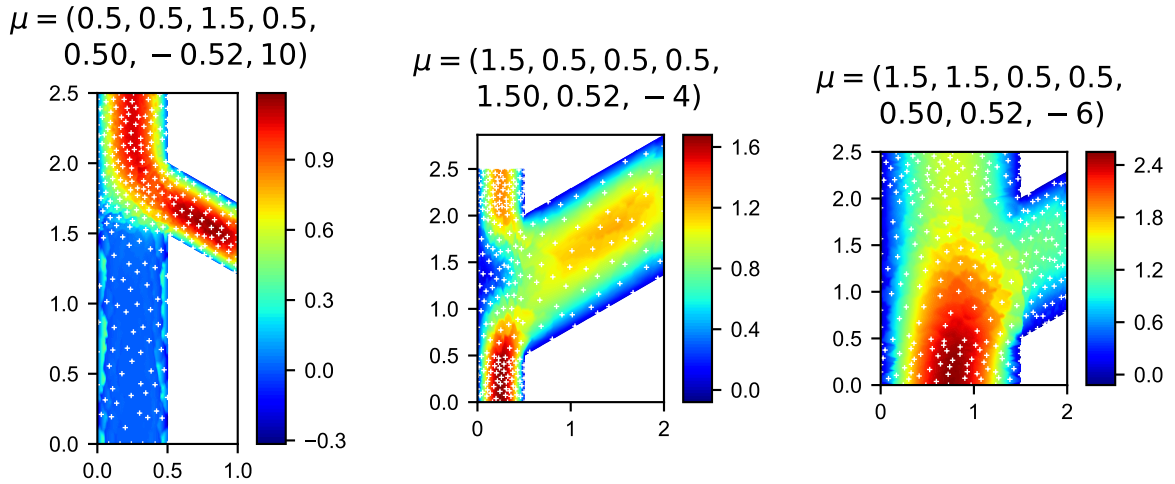


FIGURE 17. Super-resolution via GFN-ROM when training on the tiny mesh and evaluating on large mesh for variations in the geometrical parameters.

Once again, the complex parametric variability and the scarce amount of information (training on 211 parameter realisations for the 7-dimensional parameter space) do not compromise the generalisation capability of the methodology.

We also see excellent results for multifidelity training. In general, performances are only minorly affected when substituting half of the data for cheaper data, underlining the possibility of training on cheaper data whilst retaining excellent performance. It is even possible to achieve a better performance of 3.63% by replacing 50% of data on the large mesh with cheaper data on the medium mesh (representing a 34% reduction in training data), as detailed in Table 2.

## 9. CONCLUSION

In this work, we presented a novel graph-based resolution-invariant ROM, as a powerful tool to investigate multifidelity applications in the MOR context. We showed that it is possible to apply feedforward networks for graphical data by means of the graph feedforward network, allowing for the extension of many single-fidelity architectures to the multifidelity setting. Furthermore, we showed that this extension comes with provable guarantees on the performance in terms of error bounds.

Beyond such theoretical guarantees, we have also demonstrated that our architecture performs well in practice on three challenging benchmarks involving parametrised domains, advection-dominated problems and high-dimensional parameter spaces. Across all benchmarks, GFN-ROM achieved satisfactory performances, especially considering we worked within a low-data regime. Indeed, like GCA-ROM, our architecture has been trained on only 30% of the dataset, and has been exploited to learn a small latent representation. GFN-ROM achieved comparable or better performance than GCA-ROM whilst being more flexible, interpretable and lightweight, and applicable for multifidelity data.

We showed that GFN-ROM can perform super-resolution, and demonstrated that within the proposed architecture, it is possible to substitute high-fidelity data for lower-fidelity data without incurring performance penalties. In practice, we showed that often performance can even be improved when training on cheaper computational data. Our work thus highlighted the importance of multifidelity ROMs, showing that computational savings can be made by training on smaller meshes without deterioration in performance.

Future work will investigate further possibilities with graph feedforward networks, including tackling time-dependent problems, problems with spatially dependent parameters, enforcing more smoothness in GFN-ROM predictions, forcing greater dependence on local connections and automating finding the best training mesh (adapting the number of model parameters during training). We also plan to investigate multimodal modelling by using information from the Fourier domain, and the generalisation of convolutional neural operators to unstructured grids.

## ACKNOWLEDGMENT

FP acknowledges the “GO for IT” program within the CRUI fund for the project “Reduced order method for nonlinear PDEs enhanced by machine learning”. This work has been conducted within the research activities of the consortium iNEST (Interconnected North-East Innovation Ecosystem), Piano Nazionale di Ripresa e Resilienza (PNRR) – Missione 4 Componente 2, Investimento 1.5 – D.D. 1058 23/06/2022, ECS00000043, supported by the European Union’s NextGenerationEU program.

## REFERENCES

- [1] M. Alnæs, J. Blechta, J. Hake, A. Johansson, B. Kehlet, A. Logg, C. Richardson, J. Ring, M. E. Rognes, and G. N. Wells. The FEniCS project version 1.5. *Archive of Numerical Software*, 3(100), 2015.
- [2] M. A. Alvarez, L. Rosasco, N. D. Lawrence, et al. Kernels for vector-valued functions: A review. *Foundations and Trends® in Machine Learning*, 4(3):195–266, 2012.

- [3] D. Amsallem, M. J. Zahr, and C. Farhat. Nonlinear model order reduction based on local reduced-order bases. *International Journal for Numerical Methods in Engineering*, 92(10):891–916, 2012.
- [4] A. Arzani and S. T. Dawson. Data-driven cardiovascular flow modelling: examples and opportunities. *Journal of the Royal Society Interface*, 18(175):20200802, 2021.
- [5] J. Ballani, D. Kressner, and M. D. Peters. Multilevel tensor approximation of PDEs with random data. *Stochastics and Partial Differential Equations: Analysis and Computations*, 5(3):400–427, 2017.
- [6] F. Ballarin, A. Manzoni, A. Quarteroni, and G. Rozza. Supremizer stabilization of POD–Galerkin approximation of parametrized steady incompressible Navier–Stokes equations. *International Journal for Numerical Methods in Engineering*, 102(5):1136–1161, 2015.
- [7] J. L. Barnett, C. Farhat, and Y. Maday. Neural-network-augmented projection-based model order reduction for mitigating the Kolmogorov barrier to reducibility of CFD models. *arXiv preprint arXiv:2212.08939*, 2022.
- [8] A. Becker, P. Finger, A. Meyer-Christoffer, B. Rudolf, K. Schamm, U. Schneider, and M. Ziese. A description of the global land-surface precipitation data products of the global precipitation climatology centre with sample applications including centennial (trend) analysis from 1901–present. *Earth System Science Data*, 5(1):71–99, 2013.
- [9] F. D. A. Belbute-Peres, T. Economou, and Z. Kolter. Combining differentiable PDE solvers and graph neural networks for fluid flow prediction. In *International Conference on Machine Learning*, pages 2402–2411. PMLR, 2020.
- [10] P. Benner, M. Ohlberger, A. Cohen, and K. Willcox. *Model reduction and approximation: Theory and algorithms*. SIAM, 2017.
- [11] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.
- [12] B. Bonev, T. Kurth, C. Hundt, J. Pathak, M. Baust, K. Kashinath, and A. Anandkumar. Spherical Fourier neural operators: Learning stable dynamics on the sphere. In *International Conference on Machine Learning*, pages 2806–2823. PMLR, 2023.
- [13] N. Boullé and A. Townsend. *A Mathematical Guide to Operator Learning*, 2023.
- [14] S. L. Brunton, J. L. Proctor, and J. N. Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the national academy of sciences*, 113(15):3932–3937, 2016.
- [15] T. Bui-Thanh, M. Damodaran, and K. Willcox. Proper orthogonal decomposition extensions for parametric applications in compressible aerodynamics. In *21st AIAA Applied Aerodynamics Conference*, page 4213, 2003.
- [16] S. Cao. Choose a transformer: Fourier or Galerkin. *Advances in Neural Information Processing Systems*, 34:24924–24940, 2021.
- [17] W. Chen, Q. Wang, J. S. Hesthaven, and C. Zhang. Physics-informed machine learning for reduced-order modeling of nonlinear problems. *Journal of Computational Physics*, 446:110666, 2021.
- [18] L. Cici, S. Fresca, M. Guo, A. Manzoni, and P. Zunino. Uncertainty quantification for nonlinear solid mechanics using reduced order models with Gaussian process regression. *arXiv preprint arXiv:2302.08216*, 2023.
- [19] P. Conti, M. Guo, A. Manzoni, and J. S. Hesthaven. Multi-fidelity surrogate modeling using long short-term memory networks. *Computer Methods in Applied Mechanics and Engineering*, 404:115811, 2023.
- [20] S. Cuomo, V. S. Di Cola, F. Giampaolo, G. Rozza, M. Raissi, and F. Piccialli. Scientific machine learning through physics-informed neural networks: Where we are and what’s next. *Journal of Scientific Computing*, 92(3):88, 2022.
- [21] T. Danel, P. Spurek, J. Tabor, M. Śmieja, L. Struski, A. Słowik, and Ł. Maziarka. Spatial graph convolutional networks. In *Neural Information Processing: 27th International Conference*,



- ICONIP 2020, Bangkok, Thailand, November 18–22, 2020, Proceedings, Part V*, pages 668–675. Springer, 2020.
- [22] T. De Ryck, A. D. Jagtap, and S. Mishra. Error estimates for physics informed neural networks approximating the navier-stokes equations. *arXiv preprint arXiv:2203.09346*, 2022.
  - [23] N. Demo, M. Tezzele, and G. Rozza. A non-intrusive approach for the reconstruction of POD modal coefficients through active subspaces. *Comptes Rendus Mécanique*, 347(11):873–881, 2019.
  - [24] P. Díez, A. Muixí, S. Zlotnik, and A. García-González. Nonlinear dimensionality reduction for parametric problems: A kernel proper orthogonal decomposition. *International Journal for Numerical Methods in Engineering*, 122(24):7306–7327, 2021.
  - [25] T. Dinku. Challenges with availability and quality of climate data in Africa. In *Extreme hydrology and climate variability*, pages 71–80. Elsevier, 2019.
  - [26] T. Dinku, M. C. Thomson, R. Cousin, J. del Corral, P. Ceccato, J. Hansen, and S. J. Connor. Enhancing national climate services (ENACTS) for development in Africa. *Climate and Development*, 10(7):664–672, 2018.
  - [27] O. Ferludin, A. Eigenwillig, M. Blais, D. Zelle, J. Pfeifer, A. Sanchez-Gonzalez, W. L. S. Li, S. Abu-El-Haija, P. Battaglia, N. Bulut, J. Halcrow, F. M. G. de Almeida, P. Gonnet, L. Jiang, P. Kothari, S. Lattanzi, A. Linhares, B. Mayer, V. Mirrokni, J. Palowitch, M. Paradkar, J. She, A. Tsitsulin, K. Villela, L. Wang, D. Wong, and B. Perozzi. TF-GNN: graph neural networks in tensorflow. *CoRR*, abs/2207.03522, 2023.
  - [28] G. M. Fernández-Godino. Review of multi-fidelity models. *Advances in Computational Science and Engineering*, 1(4):351–400, 2023.
  - [29] M. Fey and J. E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.
  - [30] K. Fidkowski. Quantifying uncertainties in radiation hydrodynamics models. *Ann Arbor*, 1001:48109, 2014.
  - [31] A. I. Forrester. Black-box calibration for complex-system simulation. *Philosophical Transactions of The Royal Society A: Mathematical, Physical and Engineering Sciences*, 368(1924):3567–3579, 2010.
  - [32] N. R. Franco, S. Fresca, F. Tombari, and A. Manzoni. Deep learning-based surrogate models for parametrized PDEs: Handling geometric variability through graph neural networks. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 33(12):123121, 2023.
  - [33] S. Fresca, L. Dede’, and A. Manzoni. A comprehensive deep learning-based approach to reduced order modeling of nonlinear time-dependent parametrized PDEs. *Journal of Scientific Computing*, 87:1–36, 2021.
  - [34] R. B. Gramacy. laGP: large-scale spatial modeling via local approximate Gaussian processes in R. *Journal of Statistical Software*, 72:1–46, 2016.
  - [35] R. B. Gramacy. *Surrogates: Gaussian process modeling, design, and optimization for the applied sciences*. CRC press, 2020.
  - [36] C. Greif and K. Urban. Decay of the Kolmogorov n-width for wave problems. *Applied Mathematics Letters*, 96:216–222, 2019.
  - [37] M. Guo and J. S. Hesthaven. Reduced order modeling for nonlinear structural analysis using Gaussian process regression. *Computer methods in applied mechanics and engineering*, 341:807–826, 2018.
  - [38] M. Guo and J. S. Hesthaven. Data-driven reduced order modeling for time-dependent problems. *Computer methods in applied mechanics and engineering*, 345:75–99, 2019.
  - [39] M. Guo, A. Manzoni, M. Amendt, P. Conti, and J. S. Hesthaven. Multi-fidelity regression using artificial neural networks: Efficient approximation of parameter-dependent output quantities. *Computer Methods in Applied Mechanics and Engineering*, 389:114378, 2022.
  - [40] Z. Hao, Z. Wang, H. Su, C. Ying, Y. Dong, S. Liu, Z. Cheng, J. Song, and J. Zhu. GNOT: A general neural operator transformer for operator learning. In *International Conference on Machine Learning*, pages 12556–12569. PMLR, 2023.



- [41] H. Harbrecht, M. Peters, and M. Siebenmorgen. Multilevel accelerated quadrature for PDEs with log-normally distributed diffusion coefficient. *SIAM/ASA Journal on Uncertainty Quantification*, 4(1):520–551, 2016.
- [42] C. Heiß, I. Gühring, and M. Eigel. A neural multilevel method for high-dimensional parametric PDEs. In *The Symbiosis of Deep Learning and Differential Equations*, 2021.
- [43] C. Heiß, I. Gühring, and M. Eigel. Multilevel CNNs for parametric PDEs. *arXiv preprint arXiv:2304.00388*, 2023.
- [44] Q. Hernández, A. Badías, F. Chinesta, and E. Cueto. Thermodynamics-informed graph neural networks. *arXiv preprint arXiv:2203.01874*, 2022.
- [45] Q. Hernandez, A. Badias, D. Gonzalez, F. Chinesta, and E. Cueto. Deep learning of thermodynamics-aware reduced-order models from data. *Computer Methods in Applied Mechanics and Engineering*, 379:113763, 2021.
- [46] J. S. Hesthaven, G. Rozza, and B. Stamm. *Certified Reduced Basis Methods for Parametrized Partial Differential Equations*. SpringerBriefs in Mathematics. Springer International Publishing, 2015.
- [47] J. S. Hesthaven and S. Ubbiali. Non-intrusive reduced order modeling of nonlinear problems using neural networks. *Journal of Computational Physics*, 363:55–78, 2018.
- [48] W. Huang, Y. Rong, T. Xu, F. Sun, and J. Huang. Tackling over-smoothing for general graph convolutional networks. *arXiv preprint arXiv:2008.09864*, 2020.
- [49] W. Jiang and J. Luo. Graph neural network for traffic forecasting: A survey. *Expert Systems with Applications*, page 117921, 2022.
- [50] X. Jin, S. Cai, H. Li, and G. E. Karniadakis. Nsfnets (navier-stokes flow nets): Physics-informed neural networks for the incompressible navier-stokes equations. *Journal of Computational Physics*, 426:109951, 2021.
- [51] A. Kashefi and T. Mukerji. Physics-informed pointnet: A deep learning solver for steady-state incompressible flows and thermal fields on multiple sets of irregular geometries. *arXiv preprint arXiv:2202.05476*, 2022.
- [52] M. Kast, M. Guo, and J. S. Hesthaven. A non-intrusive multifidelity method for the reduced order modeling of nonlinear problems. *Computer Methods in Applied Mechanics and Engineering*, 364:112947, 2020.
- [53] M. C. Kennedy and A. O’Hagan. Predicting the output from a complex computer code when fast approximations are available. *Biometrika*, 87(1):1–13, 2000.
- [54] M. Khamlich, F. Pichi, and G. Rozza. Optimal Transport-inspired Deep Learning Framework for Slow-Decaying Problems: Exploiting Sinkhorn Loss and Wasserstein Kernel, 2023.
- [55] N. Kovachki, Z. Li, B. Liu, K. Azizzadenesheli, K. Bhattacharya, A. Stuart, and A. Anandkumar. Neural operator: Learning maps between function spaces with applications to pdes. *Journal of Machine Learning Research*, 24(89):1–97, 2023.
- [56] B. Kramer, B. Peherstorfer, and K. E. Willcox. Learning nonlinear reduced models from data with operator inference. *Annual Review of Fluid Mechanics*, 56, 2024.
- [57] M. A. Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal*, 37(2):233–243, 1991.
- [58] T. Kurth, S. Subramanian, P. Harrington, J. Pathak, M. Mardani, D. Hall, A. Miele, K. Kashinath, and A. Anandkumar. FourCastNet: Accelerating global high-resolution weather forecasting using adaptive Fourier neural operators. In *Proceedings of the Platform for Advanced Scientific Computing Conference*, pages 1–11, 2023.
- [59] Y. Kuya, K. Takeda, X. Zhang, and A. I. Forrester. Multifidelity surrogate modeling of experimental and computational aerodynamic data sets. *AIAA journal*, 49(2):289–298, 2011.
- [60] R. Lam, A. Sanchez-Gonzalez, M. Willson, P. Wirnsberger, M. Fortunato, F. Alet, S. Ravuri, T. Ewalds, Z. Eaton-Rosen, W. Hu, et al. Learning skillful medium-range global weather forecasting. *Science*, page eadi2336, 2023.
- [61] A. P. Lawrence, M. E. Nielsen, and B. Fornberg. Node subsampling for multilevel meshfree elliptic PDE solvers. *arXiv preprint arXiv:2303.09080*, 2023.

- [62] K. Lee and K. T. Carlberg. Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders. *Journal of Computational Physics*, 404:108973, 2020.
- [63] Q. Li, Z. Han, and X.-M. Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.
- [64] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020.
- [65] Z. Li, H. Zheng, N. Kovachki, D. Jin, H. Chen, B. Liu, K. Azizzadenesheli, and A. Anandkumar. Physics-informed neural operator for learning partial differential equations. *arXiv preprint arXiv:2111.03794*, 2021.
- [66] C. Liu, Y. Zhan, J. Wu, C. Li, B. Du, W. Hu, T. Liu, and D. Tao. Graph pooling for graph neural networks: Progress, challenges, and opportunities. *arXiv preprint arXiv:2204.07321*, 2022.
- [67] A. Logg, K.-A. Mardal, and G. Wells, editors. *Automated Solution of Differential Equations by the Finite Element Method: The FEniCS Book*, volume 84 of *Lecture Notes in Computational Science and Engineering*. Springer, Berlin, Heidelberg, 2012.
- [68] K. O. Lye, S. Mishra, and R. Molinaro. A multi-level procedure for enhancing accuracy of machine learning algorithms. *European Journal of Applied Mathematics*, 32(3):436–469, 2021.
- [69] A. Manzoni, F. Negri, and A. Quarteroni. Dimensionality reduction of parameter-dependent problems through proper orthogonal decomposition. *Annals of Mathematical Sciences and Applications*, 1(2):341–377, 2016.
- [70] X. Meng and G. E. Karniadakis. A composite neural network that learns from multi-fidelity data: Application to function approximation and inverse PDE problems. *Journal of Computational Physics*, 401:109020, 2020.
- [71] B. Moya, A. Badias, D. Gonzalez, F. Chinesta, and E. Cueto. Physics perception in sloshing scenes with guaranteed thermodynamic consistency. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(2):2136–2150, 2022.
- [72] S. K. Mukkavilli, D. S. Civitarese, J. Schmude, J. Jakubik, A. Jones, N. Nguyen, C. Phillips, S. Roy, S. Singh, C. Watson, et al. AI foundation models for weather and climate: Applications, design, and implementation. *arXiv preprint arXiv:2309.10808*, 2023.
- [73] M. Ohlberger and S. Rave. Reduced basis methods: Success, limitations and future challenges. *arXiv preprint arXiv:1511.02021*, 2015.
- [74] J. Pathak, S. Subramanian, P. Harrington, S. Raja, A. Chattopadhyay, M. Mardani, T. Kurth, D. Hall, Z. Li, K. Azizzadenesheli, et al. FourCastNet: A global data-driven high-resolution weather model using adaptive Fourier neural operators. *arXiv preprint arXiv:2202.11214*, 2022.
- [75] B. Peherstorfer and K. Willcox. Data-driven operator inference for nonintrusive projection-based model reduction. *Computer Methods in Applied Mechanics and Engineering*, 306:196–215, 2016.
- [76] F. Pichi, F. Ballarin, G. Rozza, and J. S. Hesthaven. An artificial neural network approach to bifurcating phenomena in computational fluid dynamics. *Computers & Fluids*, 254:105813, 2023.
- [77] F. Pichi, B. Moya, and J. S. Hesthaven. A graph convolutional autoencoder approach to model order reduction for parametrized PDEs. *arXiv preprint arXiv:2305.08573*, 2023.
- [78] E. Qian, I.-G. Farcas, and K. Willcox. Reduced operator inference for nonlinear partial differential equations. *SIAM Journal on Scientific Computing*, 44(4):A1934–A1959, 2022.
- [79] A. Quarteroni, A. Manzoni, and F. Negri. *Reduced basis methods for partial differential equations: An introduction*, volume 92. Springer, 2015.
- [80] M. Raissi, H. Babaei, and P. Givi. Deep learning of turbulent scalar mixing. *Physical Review Fluids*, 4(12):124501, 2019.

- [81] M. Raissi and G. E. Karniadakis. Hidden physics models: Machine learning of nonlinear partial differential equations. *Journal of Computational Physics*, 357:125–141, 2018.
- [82] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Inferring solutions of differential equations using noisy multi-fidelity data. *Journal of Computational Physics*, 335:736–746, 2017.
- [83] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.
- [84] M. Raissi, Z. Wang, M. S. Triantafyllou, and G. E. Karniadakis. Deep learning of vortex-induced vibrations. *Journal of Fluid Mechanics*, 861:119–137, 2019.
- [85] M. Raissi, A. Yazdani, and G. E. Karniadakis. Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. *Science*, 367(6481):1026–1030, 2020.
- [86] B. Raonić, R. Molinaro, T. Rohner, S. Mishra, and E. de Bezenac. Convolutional neural operators. *arXiv preprint arXiv:2302.01178*, 2023.
- [87] J. Reiss, P. Schulze, J. Sesterhenn, and V. Mehrmann. The shifted proper orthogonal decomposition: A mode decomposition for multiple transport phenomena. *SIAM Journal on Scientific Computing*, 40(3):A1322–A1344, 2018.
- [88] F. Romor, G. Stabile, and G. Rozza. Non-linear manifold reduced-order models with convolutional autoencoders and reduced over-collocation method. *Journal of Scientific Computing*, 94(3):74, 2023.
- [89] S. G. Rosofsky, H. Al Majed, and E. Huerta. Applications of physics informed neural operators. *Machine Learning: Science and Technology*, 4(2):025022, 2023.
- [90] G. Rozza, F. Ballarin, L. Scandurra, and F. Pichi. *Real Time Reduced Order Computational Mechanics: Parametric PDEs Worked Out Problems*, volume 5 of *SISSA Springer Series*. Springer Nature Switzerland, Cham, 2024.
- [91] T. K. Rusch, M. M. Bronstein, and S. Mishra. A survey on oversmoothing in graph neural networks. *arXiv preprint arXiv:2303.10993*, 2023.
- [92] L. Sun, H. Gao, S. Pan, and J.-X. Wang. Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data. *Computer Methods in Applied Mechanics and Engineering*, 361:112732, 2020.
- [93] M. Sun, S. Zhao, C. Gilvary, O. Elemento, J. Zhou, and F. Wang. Graph convolutional networks for computational drug development and discovery. *Briefings in Bioinformatics*, 21(3):919–935, 2020.
- [94] Y. Sun, C. Moya, G. Lin, and M. Yue. DeepGraphONet: A deep graph operator network to learn and zero-shot transfer the dynamic response of networked systems. *arXiv preprint arXiv:2209.10622*, 2022.
- [95] T. Taddei. A registration method for model order reduction: data compression and geometry reduction. *SIAM Journal on Scientific Computing*, 42(2):A997–A1027, 2020.
- [96] A. L. Teckentrup, P. Jantsch, C. G. Webster, and M. Gunzburger. A multilevel stochastic collocation method for partial differential equations with random input data. *SIAM/ASA Journal on Uncertainty Quantification*, 3(1):1046–1074, 2015.
- [97] J. Wang, S. Zhang, Y. Xiao, and R. Song. A review on graph neural network methods in financial applications. *arXiv preprint arXiv:2111.15367*, 2021.
- [98] M. Wang, D. Zheng, Z. Ye, Q. Gan, M. Li, X. Song, J. Zhou, C. Ma, L. Yu, Y. Gai, T. Xiao, T. He, G. Karypis, J. Li, and Z. Zhang. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315*, 2019.
- [99] S. Wang, Y. Teng, and P. Perdikaris. Understanding and mitigating gradient flow pathologies in physics-informed neural networks. *SIAM Journal on Scientific Computing*, 43(5):A3055–A3081, 2021.
- [100] S. Wang, X. Yu, and P. Perdikaris. When and why pinns fail to train: A neural tangent kernel perspective. *Journal of Computational Physics*, 449:110768, 2022.

- [101] S. Zhang, H. Tong, J. Xu, and R. Maciejewski. Graph convolutional networks: Algorithms, applications and open challenges. In *Computational Data and Social Networks: 7th International Conference, CSoNet 2018, Shanghai, China, December 18–20, 2018, Proceedings 7*, pages 79–91. Springer, 2018.
- [102] S. Zhang, H. Tong, J. Xu, and R. Maciejewski. Graph convolutional networks: A comprehensive review. *Computational Social Networks*, 6(1):1–23, 2019.
- [103] Z. Zhang, M. Guo, and J. S. Hesthaven. Model order reduction for large-scale structures with local nonlinearities. *Computer Methods in Applied Mechanics and Engineering*, 353:491–515, 2019.

## APPENDIX A. PROOFS

**A.1. Extension to multiple layers.** For ease of notation we define each of the outputs of the  $\text{GFN}^{\mathcal{M}_o \rightarrow \mathcal{M}_n}$  function as

$$\text{GFN}_{W^e}^{\mathcal{M}_o \rightarrow \mathcal{M}_n}(\mathbf{W}^e), \text{GFN}_{W^d}^{\mathcal{M}_o \rightarrow \mathcal{M}_n}(\mathbf{W}^d), \text{GFN}_{b^d}^{\mathcal{M}_o \rightarrow \mathcal{M}_n}(\mathbf{b}^d) = \text{GFN}^{\mathcal{M}_o \rightarrow \mathcal{M}_n}(\mathbf{W}^e, \mathbf{W}^d, \mathbf{b}^d).$$

The single-layer GFN autoencoder given in Equation 6 and Equation 7 can then be rewritten as

$$\begin{aligned} \text{enc}^{\mathcal{M}_o \rightarrow \mathcal{M}_n}(\mathbf{u}_{\mathcal{M}_n})_i &= \sigma \left( \sum_{j_{\mathcal{M}_n}=1}^{N_n} \text{GFN}_{W^e}^{\mathcal{M}_o \rightarrow \mathcal{M}_n}(\mathbf{W}^e)_{ij_{\mathcal{M}_n}} u_{j_{\mathcal{M}_n}} + b_i^e \right), \quad \forall i = 1, \dots, L, \\ \text{dec}^{\mathcal{M}_o \rightarrow \mathcal{M}_n}(\mathbf{z})_{i_{\mathcal{M}_n}} &= \sum_{j=1}^L \text{GFN}_{W^d}^{\mathcal{M}_o \rightarrow \mathcal{M}_n}(\mathbf{W}^d)_{i_{\mathcal{M}_n}j} z_j + b_{i_{\mathcal{M}_n}}^d, \quad \forall i_{\mathcal{M}_n} = 1, \dots, N_n. \end{aligned}$$

This can be extended to  $Q + 1$  hidden layers as

$$\begin{aligned} \text{enc}_0^{\mathcal{M}_o \rightarrow \mathcal{M}_n}(\mathbf{x})_i &= \sigma \left( \sum_{j_{\mathcal{M}_n}=1}^{N_n} \text{GFN}_{W^e}^{\mathcal{M}_o \rightarrow \mathcal{M}_n}(\mathbf{W}^e)_{ij_{\mathcal{M}_n}} x_{j_{\mathcal{M}_n}} + b_i^e \right), \\ &\quad \forall i = 1, \dots, L_1, \\ \text{enc}_{P \dots 1}(\mathbf{x})_i &= \sigma \left( \sum_{l_P=1}^{L_P} W_{i,l_P}^{(P)} \sigma \left( \dots \sigma \left( \sum_{l_1=1}^{L_1} W_{l_2,l_1}^{(1)} x_{l_1} + b_{l_2}^{(1)} \right) + \dots \right) + b_i^{(P)} \right), \\ &\quad \forall i = 1, \dots, L_{P+1}, \\ \text{dec}_{Q \dots P+1}(\mathbf{x})_i &= \sigma \left( \sum_{l_Q=1}^{L_Q} W_{i,l_Q}^{(Q)} \sigma \left( \dots \sigma \left( \sum_{l_{P+1}=1}^{L_{P+1}} W_{l_{P+2},l_{P+1}}^{(P+1)} x_{l_{P+1}} + b_{l_{P+2}}^{(P+1)} \right) + \dots \right) + b_i^{(Q)} \right), \\ &\quad \forall i = 1, \dots, L_{Q+1}, \\ \text{dec}_{Q+1}^{\mathcal{M}_o \rightarrow \mathcal{M}_n}(\mathbf{x})_{i_{\mathcal{M}_n}} &= \sum_{j=1}^{L_{Q+1}} \text{GFN}_{W^d}^{\mathcal{M}_o \rightarrow \mathcal{M}_n}(\mathbf{W}^d)_{i_{\mathcal{M}_n}j} x_j + \text{GFN}_{b^d}^{\mathcal{M}_o \rightarrow \mathcal{M}_n}(\mathbf{b}^d)_{i_{\mathcal{M}_n}}, \\ &\quad \forall i_{\mathcal{M}_n} = 1, \dots, N_n, \end{aligned}$$

where  $0 \leq P \leq Q$ ,  $\mathcal{M}_n$  is a new mesh,  $L_i$  represents the output sizes for the  $(i + 1)$ -th network layer with  $L_{P+1} = L$  and  $L_{Q+2} = N_n$ ,  $\text{enc}^{\mathcal{M}_o \rightarrow \mathcal{M}_n}(\mathbf{x})_i = \text{enc}_{P \dots 1}(\text{enc}_0^{\mathcal{M}_o \rightarrow \mathcal{M}_n}(\mathbf{x}))_i, \forall i = 1, \dots, L$  is the encoded latent representation of an input  $\mathbf{x} \in \mathbb{R}^{N_n}$  on mesh  $\mathcal{M}_n$  and  $\text{dec}^{\mathcal{M}_o \rightarrow \mathcal{M}_n}(\mathbf{z})_{i_{\mathcal{M}_n}} = \text{dec}_{Q+1}^{\mathcal{M}_o \rightarrow \mathcal{M}_n}(\text{dec}_{Q \dots P+1}(\mathbf{z}))_{i_{\mathcal{M}_n}}, \forall i_{\mathcal{M}_n} = 1, \dots, N_n$  is the reconstructed solution on  $\mathcal{M}_n$  from a given latent representation  $\mathbf{z} \in \mathbb{R}^L$ .

**A.2. Simplified expansive equations.** We consider the GFN transforms  $\mathcal{M}_o \rightarrow \mathcal{M}_n$  and show that they can be simplified under the expansive condition given in Equation 9.

First of all, we show that the condition implies

$$\forall i_{\mathcal{M}_n}, \quad \{k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow \neg i_{\mathcal{M}_n}\} = \{k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow i_{\mathcal{M}_n}\},$$

which is a set containing a unique element  $k_{\mathcal{M}_o}$  defined as  $k_{\mathcal{M}_o} \leftarrow i_{\mathcal{M}_n}$ .

*Proof:* We have to show that  $\{k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow \neg i_{\mathcal{M}_n}\} = \{k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow i_{\mathcal{M}_n}\} \cup \{k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \rightarrow i_{\mathcal{M}_n}\} = \{k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow i_{\mathcal{M}_n}\}$ . If  $\{k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \rightarrow i_{\mathcal{M}_n}\} = \emptyset$ , this is trivially the case. Therefore, in the following we consider only the case where it is not.

Trivially,  $|\{k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow i_{\mathcal{M}_n}\}| = 1$  since a node  $i_{\mathcal{M}_n}$  can only have one nearest neighbour. In fact, additionally we can show it is also the case that  $|\{k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \rightarrow i_{\mathcal{M}_n}\}| = 1$ , which can be proven by contradiction. Suppose that  $|\{k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \rightarrow i_{\mathcal{M}_n}\}| = R \neq 1$ . This means that there exist multiple distinct nodes  $\{k_{\mathcal{M}_o}^r\}_{r=1}^R$ , each satisfying  $k_{\mathcal{M}_o}^r \rightarrow i_{\mathcal{M}_n}$ . Via the expansive condition in Equation 9, it follows that each of these nodes must also satisfy  $k_{\mathcal{M}_o}^r \leftarrow i_{\mathcal{M}_n}$ . This is not possible since  $i_{\mathcal{M}_n}$  can only have a single nearest neighbour, meaning therefore  $|\{k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \rightarrow i_{\mathcal{M}_n}\}| = 1$ .

We note that the single node in the set  $|\{k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \rightarrow i_{\mathcal{M}_n}\}|$  must also satisfy  $k_{\mathcal{M}_o} \leftarrow i_{\mathcal{M}_n}$  due to Equation 9. Therefore, it is in fact the exact same node as the node in the set  $|\{k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow i_{\mathcal{M}_n}\}|$ , thus concluding the proof.

Secondly, we show that for a given  $k_{\mathcal{M}_o}$

$$|\{h_{\mathcal{M}_n} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow h_{\mathcal{M}_n}\}| = |\{h_{\mathcal{M}_n} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow h_{\mathcal{M}_n}\}|.$$

*Proof:* Due to Equation 9, it is the case that  $\{h_{\mathcal{M}_n} \text{ s.t. } k_{\mathcal{M}_o} \rightarrow h_{\mathcal{M}_n}\} \subseteq \{h_{\mathcal{M}_n} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow h_{\mathcal{M}_n}\}$ , concluding the proof.

The updates can therefore be simplified to

$$\begin{aligned} \tilde{W}_{ij_{\mathcal{M}_n}}^e &= \frac{1}{|\{h_{\mathcal{M}_n} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow h_{\mathcal{M}_n}\}|} W_{ik_{\mathcal{M}_o}}^e, & \text{where } k_{\mathcal{M}_o} \leftarrow j_{\mathcal{M}_n}, \\ \tilde{W}_{i_{\mathcal{M}_n}j}^d &= W_{k_{\mathcal{M}_o}j}^d, & \text{where } k_{\mathcal{M}_o} \leftarrow i_{\mathcal{M}_n}, \\ \tilde{b}_{i_{\mathcal{M}_n}}^d &= b_{k_{\mathcal{M}_o}}^d, & \text{where } k_{\mathcal{M}_o} \leftarrow i_{\mathcal{M}_n}, \end{aligned}$$

which therefore concludes the proof.

**A.3. Simplified agglomerative equations.** We consider the GFN transforms  $\mathcal{M}_o \rightarrow \mathcal{M}_n$  and show that they can be simplified under the agglomerative condition given in Equation 11.

First of all, we show that the condition implies

$$\forall j_{\mathcal{M}_n}, \quad \{k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow j_{\mathcal{M}_n}\} = \{k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \rightarrow j_{\mathcal{M}_n}\}.$$

*Proof:* Due to Equation 11, it is the case that  $\{k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow j_{\mathcal{M}_n}\} \subseteq \{k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \rightarrow j_{\mathcal{M}_n}\}$ , concluding the proof.

Secondly, we show that for a given  $k_{\mathcal{M}_o}$

$$|\{h_{\mathcal{M}_n} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow h_{\mathcal{M}_n}\}| = 1.$$

*Proof:* We want to show that

$$|\{h_{\mathcal{M}_n} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow h_{\mathcal{M}_n}\}| = |\{h_{\mathcal{M}_n} \text{ s.t. } k_{\mathcal{M}_o} \rightarrow h_{\mathcal{M}_n}\} \cup \{h_{\mathcal{M}_n} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow h_{\mathcal{M}_n}\}| = 1.$$

If  $\{h_{\mathcal{M}_n} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow h_{\mathcal{M}_n}\} = \emptyset$ , this is trivially the case since  $k_{\mathcal{M}_o}$  only has one nearest neighbour i.e.  $|\{h_{\mathcal{M}_n} \text{ s.t. } k_{\mathcal{M}_o} \rightarrow h_{\mathcal{M}_n}\}| = 1$ . Therefore, in the following we consider only the case where it is not.

With this, we can show it is also the case that  $|\{h_{\mathcal{M}_n} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow h_{\mathcal{M}_n}\}| = 1$ , which can be proven by contradiction. Suppose that  $|\{h_{\mathcal{M}_n} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow h_{\mathcal{M}_n}\}| = R \neq 1$ . This means that there exist multiple distinct nodes  $\{h_{\mathcal{M}_n}^r\}_{r=1}^R$ , each satisfying  $k_{\mathcal{M}_o} \leftarrow h_{\mathcal{M}_n}^r$ . Via the agglomerative condition in Equation 11, it follows that each of these nodes must also satisfy  $k_{\mathcal{M}_o} \rightarrow h_{\mathcal{M}_n}^r$ . This is not possible since  $k_{\mathcal{M}_o}$  can only have a single nearest neighbour, meaning therefore  $|\{h_{\mathcal{M}_n} \text{ s.t. } k_{\mathcal{M}_o} \rightarrow h_{\mathcal{M}_n}\}| = 1$ .

We note that the single node in the set  $|\{h_{\mathcal{M}_n} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow h_{\mathcal{M}_n}\}|$  must also satisfy  $k_{\mathcal{M}_o} \rightarrow h_{\mathcal{M}_n}$  due to Equation 11. Therefore, it is in fact the exact same node as the node in the set  $|\{h_{\mathcal{M}_n} \text{ s.t. } k_{\mathcal{M}_o} \rightarrow h_{\mathcal{M}_n}\}|$ , thus concluding the proof.

The updates can therefore be simplified to

$$\begin{aligned} \tilde{W}_{ij_{\mathcal{M}_n}}^e &= \sum_{\forall k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \rightarrow j_{\mathcal{M}_n}} W_{ik_{\mathcal{M}_o}}^e, \\ \tilde{W}_{i_{\mathcal{M}_n}j}^d &= \text{mean}_{\forall k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \rightarrow i_{\mathcal{M}_n}} W_{k_{\mathcal{M}_o}j}^d, \\ \tilde{b}_{i_{\mathcal{M}_n}}^d &= \text{mean}_{\forall k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \rightarrow i_{\mathcal{M}_n}} b_{k_{\mathcal{M}_o}}^d, \end{aligned}$$

which therefore concludes the proof.

**A.4. Self-consistency.** We consider a sequence of two transforms, firstly an expansive transform  $\mathcal{M}_o \rightarrow \mathcal{M}_n$ , and then secondly a transform  $\mathcal{M}_n \rightarrow \mathcal{M}_o$ . Note that if  $\mathcal{M}_o \rightarrow \mathcal{M}_n$  is expansive, then it must follow by definition that  $\mathcal{M}_n \rightarrow \mathcal{M}_o$  is agglomerative. The first transform  $\mathcal{M}_o \rightarrow \mathcal{M}_n$  is expansive. Following Equation 9, this gives the first update as

$$\begin{aligned}\hat{W}_{ij_{\mathcal{M}_n}}^e &= \frac{1}{|\{h_{\mathcal{M}_n} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow h_{\mathcal{M}_n}\}|} W_{ik_{\mathcal{M}_o}}^e, & \text{where } k_{\mathcal{M}_o} \leftarrow j_{\mathcal{M}_n}, \\ \hat{W}_{i_{\mathcal{M}_n}j}^d &= W_{k_{\mathcal{M}_o}j}^d, & \text{where } k_{\mathcal{M}_o} \leftarrow i_{\mathcal{M}_n}, \\ \hat{b}_{i_{\mathcal{M}_n}}^d &= b_{k_{\mathcal{M}_o}}^d, & \text{where } k_{\mathcal{M}_o} \leftarrow i_{\mathcal{M}_n}.\end{aligned}$$

The second transform  $\mathcal{M}_o \leftarrow \mathcal{M}_n$  is agglomerative (note that the transform is *not*  $\mathcal{M}_o \rightarrow \mathcal{M}_n$ ). Following Equation 11, this gives the second update as

$$\begin{aligned}\tilde{W}_{ij_{\mathcal{M}_o}}^e &= \sum_{\forall k_{\mathcal{M}_n} \text{ s.t. } j_{\mathcal{M}_o} \leftarrow k_{\mathcal{M}_n}} \hat{W}_{ik_{\mathcal{M}_n}}^e, \\ \tilde{W}_{i_{\mathcal{M}_o}j}^d &= \text{mean}_{\forall k_{\mathcal{M}_n} \text{ s.t. } i_{\mathcal{M}_o} \leftarrow k_{\mathcal{M}_n}} \hat{W}_{k_{\mathcal{M}_n}j}^d, \\ \tilde{b}_{i_{\mathcal{M}_o}}^d &= \text{mean}_{\forall k_{\mathcal{M}_n} \text{ s.t. } i_{\mathcal{M}_o} \leftarrow k_{\mathcal{M}_n}} b_{k_{\mathcal{M}_n}}^d.\end{aligned}$$

Subbing everything in gives

$$\begin{aligned}\tilde{W}_{ij_{\mathcal{M}_o}}^e &= \sum_{\forall k_{\mathcal{M}_n} \text{ s.t. } j_{\mathcal{M}_o} \leftarrow k_{\mathcal{M}_n}} \frac{1}{|\{h_{\mathcal{M}_n} \text{ s.t. } l_{\mathcal{M}_o} \leftarrow h_{\mathcal{M}_n}\}|} W_{il_{\mathcal{M}_o}}^e, & \text{where } l_{\mathcal{M}_o} \leftarrow k_{\mathcal{M}_n}, \\ &= \sum_{\forall k_{\mathcal{M}_n} \text{ s.t. } j_{\mathcal{M}_o} \leftarrow k_{\mathcal{M}_n}} \frac{1}{|\{h_{\mathcal{M}_n} \text{ s.t. } j_{\mathcal{M}_o} \leftarrow h_{\mathcal{M}_n}\}|} W_{ij_{\mathcal{M}_o}}^e, \\ &= W_{ij_{\mathcal{M}_o}}^e, \\ \tilde{W}_{i_{\mathcal{M}_o}j}^d &= \text{mean}_{\forall k_{\mathcal{M}_n} \text{ s.t. } i_{\mathcal{M}_o} \leftarrow k_{\mathcal{M}_n}} W_{l_{\mathcal{M}_o}j}^d, & \text{where } l_{\mathcal{M}_o} \leftarrow k_{\mathcal{M}_n}, \\ &= \text{mean}_{\forall k_{\mathcal{M}_n} \text{ s.t. } i_{\mathcal{M}_o} \leftarrow k_{\mathcal{M}_n}} W_{i_{\mathcal{M}_o}j}^d, \\ &= W_{i_{\mathcal{M}_o}j}^d, \\ \tilde{b}_{i_{\mathcal{M}_o}}^d &= \text{mean}_{\forall k_{\mathcal{M}_n} \text{ s.t. } i_{\mathcal{M}_o} \leftarrow k_{\mathcal{M}_n}} b_{l_{\mathcal{M}_o}}^d, & \text{where } l_{\mathcal{M}_o} \leftarrow k_{\mathcal{M}_n}, \\ &= \text{mean}_{\forall k_{\mathcal{M}_n} \text{ s.t. } i_{\mathcal{M}_o} \leftarrow k_{\mathcal{M}_n}} b_{i_{\mathcal{M}_o}}^d, \\ &= b_{i_{\mathcal{M}_o}}^d.\end{aligned}$$

This concludes the proof.

**A.5. General GFN transform as a composition of an expansive and agglomerative transform.** Splitting up the summations, the general GFN transforms given in Equation 5 can be rewritten as

$$\begin{aligned}\tilde{W}_{ij_{\mathcal{M}_n}}^e &= \left[ \sum_{\forall k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \rightarrow j_{\mathcal{M}_n}} + \sum_{\substack{\forall k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow j_{\mathcal{M}_n} \\ \text{but not } k_{\mathcal{M}_o} \rightarrow j_{\mathcal{M}_n}}} \right] \frac{W_{ik_{\mathcal{M}_o}}^e}{|\{h_{\mathcal{M}_n} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow/\rightarrow h_{\mathcal{M}_n}\}|}, \\ \tilde{W}_{i_{\mathcal{M}_n}j}^d &= \frac{1}{|\{k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow/\rightarrow i_{\mathcal{M}_n}\}|} \left[ \sum_{\forall k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \rightarrow i_{\mathcal{M}_n}} + \sum_{\substack{\forall k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow i_{\mathcal{M}_n} \\ \text{but not } k_{\mathcal{M}_o} \rightarrow i_{\mathcal{M}_n}}} \right] W_{k_{\mathcal{M}_o}j}^d, \\ \tilde{b}_{i_{\mathcal{M}_n}}^d &= \frac{1}{|\{k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow/\rightarrow i_{\mathcal{M}_n}\}|} \left[ \sum_{\forall k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \rightarrow i_{\mathcal{M}_n}} + \sum_{\substack{\forall k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow i_{\mathcal{M}_n} \\ \text{but not } k_{\mathcal{M}_o} \rightarrow i_{\mathcal{M}_n}}} \right] b_{k_{\mathcal{M}_o}}^d.\end{aligned}$$

By defining an auxiliary mesh

$$\mathcal{M} = \mathcal{M}_o \cup \{\mathcal{M}_n [i_{\mathcal{M}_n}] \text{ s.t. } j_{\mathcal{M}_o} \leftarrow i_{\mathcal{M}_n} \text{ but not } j_{\mathcal{M}_o} \rightarrow i_{\mathcal{M}_n}\},$$

we can convert the summations to

$$\begin{aligned}\sum_{\forall k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \rightarrow i_{\mathcal{M}_n}} &\rightarrow \sum_{\substack{\forall l_{\mathcal{M}} \text{ s.t. } l_{\mathcal{M}} \rightarrow i_{\mathcal{M}_n} \\ \text{and } l_{\mathcal{M}} \in \mathcal{M}_o}}, \quad k_{\mathcal{M}_o} \leftarrow l_{\mathcal{M}}, \\ \sum_{\substack{\forall k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow i_{\mathcal{M}_n} \\ \text{but not } k_{\mathcal{M}_o} \rightarrow i_{\mathcal{M}_n}}} &\rightarrow \sum_{\substack{\forall l_{\mathcal{M}} \text{ s.t. } l_{\mathcal{M}} \rightarrow i_{\mathcal{M}_n} \\ \text{and } l_{\mathcal{M}} \in \mathcal{M}_n}}, \quad k_{\mathcal{M}_o} \leftarrow l_{\mathcal{M}}.\end{aligned}$$

Subbing this in, this gives an updated expression for the general GFN transforms of

$$\begin{aligned}\tilde{W}_{ij_{\mathcal{M}_n}}^e &= \left[ \sum_{\substack{\forall l_{\mathcal{M}} \text{ s.t. } l_{\mathcal{M}} \rightarrow j_{\mathcal{M}_n} \\ \text{and } l_{\mathcal{M}} \in \mathcal{M}_o}} + \sum_{\substack{\forall l_{\mathcal{M}} \text{ s.t. } l_{\mathcal{M}} \rightarrow j_{\mathcal{M}_n} \\ \text{and } l_{\mathcal{M}} \in \mathcal{M}_n}} \right] \frac{W_{ik_{\mathcal{M}_o}}^e}{|\{h_{\mathcal{M}_n} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow/\rightarrow h_{\mathcal{M}_n}\}|}, \quad k_{\mathcal{M}_o} \leftarrow l_{\mathcal{M}}, \\ \tilde{W}_{i_{\mathcal{M}_n}j}^d &= \frac{1}{|\{g_{\mathcal{M}_o} \text{ s.t. } g_{\mathcal{M}_o} \leftarrow/\rightarrow i_{\mathcal{M}_n}\}|} \left[ \sum_{\substack{\forall l_{\mathcal{M}} \text{ s.t. } l_{\mathcal{M}} \rightarrow i_{\mathcal{M}_n} \\ \text{and } l_{\mathcal{M}} \in \mathcal{M}_o}} + \sum_{\substack{\forall l_{\mathcal{M}} \text{ s.t. } l_{\mathcal{M}} \rightarrow i_{\mathcal{M}_n} \\ \text{and } l_{\mathcal{M}} \in \mathcal{M}_n}} \right] W_{k_{\mathcal{M}_o}j}^d, \quad k_{\mathcal{M}_o} \leftarrow l_{\mathcal{M}}, \\ \tilde{b}_{i_{\mathcal{M}_n}}^d &= \frac{1}{|\{g_{\mathcal{M}_o} \text{ s.t. } g_{\mathcal{M}_o} \leftarrow/\rightarrow i_{\mathcal{M}_n}\}|} \left[ \sum_{\substack{\forall l_{\mathcal{M}} \text{ s.t. } l_{\mathcal{M}} \rightarrow i_{\mathcal{M}_n} \\ \text{and } l_{\mathcal{M}} \in \mathcal{M}_o}} + \sum_{\substack{\forall l_{\mathcal{M}_n} \text{ s.t. } l_{\mathcal{M}} \rightarrow i_{\mathcal{M}_n} \\ \text{and } l_{\mathcal{M}} \in \mathcal{M}_n}} \right] b_{i_{\mathcal{M}_n}}^d, \quad k_{\mathcal{M}_o} \leftarrow l_{\mathcal{M}}.\end{aligned}$$

Since it is the case by construction of  $\mathcal{M}$  that  $|\{h_{\mathcal{M}_n} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow/\rightarrow h_{\mathcal{M}_n}\}| = |\{h_{\mathcal{M}} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow h_{\mathcal{M}}\}|$  and  $|\{g_{\mathcal{M}_o} \text{ s.t. } g_{\mathcal{M}_o} \leftarrow/\rightarrow i_{\mathcal{M}_n}\}| = |\{l_{\mathcal{M}} \text{ s.t. } l_{\mathcal{M}} \rightarrow i_{\mathcal{M}_n}\}|$ , we can further rewrite these expressions as

$$\begin{aligned}\tilde{W}_{ij_{\mathcal{M}_n}}^e &= \sum_{\forall l_{\mathcal{M}} \text{ s.t. } l_{\mathcal{M}} \rightarrow j_{\mathcal{M}_n}} \frac{W_{ik_{\mathcal{M}_o}}^e}{|\{h_{\mathcal{M}} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow h_{\mathcal{M}}\}|}, \quad k_{\mathcal{M}_o} \leftarrow l_{\mathcal{M}}, \\ \tilde{W}_{i_{\mathcal{M}_n}j}^d &= \text{mean}_{\forall l_{\mathcal{M}} \text{ s.t. } l_{\mathcal{M}} \rightarrow i_{\mathcal{M}_n}} W_{k_{\mathcal{M}_o}j}^d, \quad k_{\mathcal{M}_o} \leftarrow l_{\mathcal{M}}, \\ \tilde{b}_{i_{\mathcal{M}_n}}^d &= \text{mean}_{\forall l_{\mathcal{M}} \text{ s.t. } l_{\mathcal{M}} \rightarrow i_{\mathcal{M}_n}} b_{k_{\mathcal{M}_o}}^d, \quad k_{\mathcal{M}_o} \leftarrow l_{\mathcal{M}}.\end{aligned}$$



We can finally compute the transform in two steps, as

$$\left. \begin{aligned} \hat{W}_{ij_{\mathcal{M}}}^e &= \frac{1}{|\{h_{\mathcal{M}} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow h_{\mathcal{M}}\}|} W_{ik_{\mathcal{M}_o}}^e, & \text{where } k_{\mathcal{M}_o} \leftarrow j_{\mathcal{M}}, \\ \hat{W}_{i_{\mathcal{M}}j}^d &= W_{k_{\mathcal{M}_o}j}^d, & \text{where } k_{\mathcal{M}_o} \leftarrow i_{\mathcal{M}}, \\ \hat{b}_{i_{\mathcal{M}}}^d &= b_{k_{\mathcal{M}_o}}^d, & \text{where } k_{\mathcal{M}_o} \leftarrow i_{\mathcal{M}}, \end{aligned} \right\} \text{Expansion step}$$

followed by

$$\left. \begin{aligned} \tilde{W}_{ij_{\mathcal{M}_n}}^e &= \sum_{\forall k_{\mathcal{M}} \text{ s.t. } k_{\mathcal{M}} \rightarrow j_{\mathcal{M}_n}} \hat{W}_{ik_{\mathcal{M}}}^e, \\ \tilde{W}_{i_{\mathcal{M}_n}j}^d &= \text{mean}_{\forall k_{\mathcal{M}} \text{ s.t. } k_{\mathcal{M}} \rightarrow i_{\mathcal{M}_n}} \hat{W}_{k_{\mathcal{M}}j}^d, \\ \tilde{b}_{i_{\mathcal{M}_n}}^d &= \text{mean}_{\forall k_{\mathcal{M}} \text{ s.t. } k_{\mathcal{M}} \rightarrow i_{\mathcal{M}_n}} \hat{b}_{k_{\mathcal{M}}}^d, \end{aligned} \right\} \text{Agglomerative step}$$

which concludes the proof.

**A.6. Super- and sub-resolution error bound for GFN-ROM.** We seek a bound on the following quantity

$$\left| u(x_{i_{\mathcal{M}_n}}) - \text{dec}^{\mathcal{M}_o \rightarrow \mathcal{M}_n}(\text{map}(\boldsymbol{\mu}))_{i_{\mathcal{M}_n}} \right|, \forall i_{\mathcal{M}_n}.$$

We can rewrite the expression and bound it as

$$\begin{aligned} & \left| u(x_{i_{\mathcal{M}_n}}) - \text{dec}^{\mathcal{M}_o \rightarrow \mathcal{M}_n}(\text{map}(\boldsymbol{\mu}))_{i_{\mathcal{M}_n}} \right| \\ &= \left| u(x_{i_{\mathcal{M}_n}}) - \sum_{j=1}^L \tilde{W}_{i_{\mathcal{M}_n}j}^d \text{map}(\boldsymbol{\mu})_j + \tilde{b}_{i_{\mathcal{M}_n}}^d \right|, \\ &= \left| u(x_{i_{\mathcal{M}_n}}) - \sum_{j=1}^L \text{mean}_{\forall k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow \rightarrow i_{\mathcal{M}_n}} W_{k_{\mathcal{M}_o}j}^d \text{map}(\boldsymbol{\mu})_j + \text{mean}_{\forall k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow \rightarrow i_{\mathcal{M}_n}} b_{k_{\mathcal{M}_o}}^d \right|, \\ &= \left| u(x_{i_{\mathcal{M}_n}}) - \text{mean}_{\forall k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow \rightarrow i_{\mathcal{M}_n}} \left( \sum_{j=1}^L W_{k_{\mathcal{M}_o}j}^d \text{map}(\boldsymbol{\mu})_j + b_{k_{\mathcal{M}_o}}^d \right) \right| \\ &= \left| u(x_{i_{\mathcal{M}_n}}) - \text{mean}_{\forall k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow \rightarrow i_{\mathcal{M}_n}} \text{dec}^{\mathcal{M}_o \rightarrow \mathcal{M}_o}(\text{map}(\boldsymbol{\mu}))_{k_{\mathcal{M}_o}} \right|, \\ &= \left| \text{mean}_{\forall k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow \rightarrow i_{\mathcal{M}_n}} \left( u(x_{i_{\mathcal{M}_n}}) - u(x_{k_{\mathcal{M}_o}}) + u(x_{k_{\mathcal{M}_o}}) - \text{dec}^{\mathcal{M}_o \rightarrow \mathcal{M}_o}(\text{map}(\boldsymbol{\mu}))_{k_{\mathcal{M}_o}} \right) \right|, \\ &\leq \text{mean}_{\forall k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow \rightarrow i_{\mathcal{M}_n}} \left| u(x_{i_{\mathcal{M}_n}}) - u(x_{k_{\mathcal{M}_o}}) + u(x_{k_{\mathcal{M}_o}}) - \text{dec}^{\mathcal{M}_o \rightarrow \mathcal{M}_o}(\text{map}(\boldsymbol{\mu}))_{k_{\mathcal{M}_o}} \right|, \\ &\leq \text{mean}_{\forall k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow \rightarrow i_{\mathcal{M}_n}} \left( \left| u(x_{i_{\mathcal{M}_n}}) - u(x_{k_{\mathcal{M}_o}}) \right| + \left| u(x_{k_{\mathcal{M}_o}}) - \text{dec}^{\mathcal{M}_o \rightarrow \mathcal{M}_o}(\text{map}(\boldsymbol{\mu}))_{k_{\mathcal{M}_o}} \right| \right), \\ &\leq \text{mean}_{\forall k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow \rightarrow i_{\mathcal{M}_n}} (\delta + \tau), \\ &= \delta + \tau. \end{aligned}$$

This concludes the proof. For the multiple layer case, the result is also the same.

**A.7. Super- and sub-resolution error bound for GFN-ROM mapper.** We seek a bound on

$$(35) \quad \left| \text{map}(\boldsymbol{\mu})_i - \text{enc}^{\mathcal{M}_o \rightarrow \mathcal{M}_n}(\mathbf{u}_{\mathcal{M}_n})_i \right|, \forall i.$$

We can rewrite the expression as

$$\begin{aligned}
& \left| \text{map}(\boldsymbol{\mu})_i - \text{enc}^{\mathcal{M}_o \rightarrow \mathcal{M}_o}(\mathbf{u}_{\mathcal{M}_o})_i + \text{enc}^{\mathcal{M}_o \rightarrow \mathcal{M}_o}(\mathbf{u}_{\mathcal{M}_o})_i - \text{enc}^{\mathcal{M}_o \rightarrow \mathcal{M}_n}(\mathbf{u}_{\mathcal{M}_n})_i \right| \\
& \leq \left| \text{map}(\boldsymbol{\mu})_i - \text{enc}^{\mathcal{M}_o \rightarrow \mathcal{M}_o}(\mathbf{u}_{\mathcal{M}_o})_i \right| + \left| \text{enc}^{\mathcal{M}_o \rightarrow \mathcal{M}_o}(\mathbf{u}_{\mathcal{M}_o})_i - \text{enc}^{\mathcal{M}_o \rightarrow \mathcal{M}_n}(\mathbf{u}_{\mathcal{M}_n})_i \right|, \\
& \leq \alpha + \left| \text{enc}^{\mathcal{M}_o \rightarrow \mathcal{M}_o}(\mathbf{u}_{\mathcal{M}_o})_i - \text{enc}^{\mathcal{M}_o \rightarrow \mathcal{M}_n}(\mathbf{u}_{\mathcal{M}_n})_i \right|, \\
& = \alpha + \left| \sigma \left( \sum_{j_{\mathcal{M}_o}=1}^{N_o} W_{ij_{\mathcal{M}_o}}^e u(x_{j_{\mathcal{M}_o}}) + b_i^e \right) - \sigma \left( \sum_{k_{\mathcal{M}_n}=1}^{N_n} \tilde{W}_{ik_{\mathcal{M}_n}}^e u(x_{k_{\mathcal{M}_n}}) + \tilde{b}_i^e \right) \right|.
\end{aligned}$$

We assume a Lipschitz condition with Lipschitz constant  $C$  for the activation function  $\sigma$ , which allows us to continue bounding as

$$\begin{aligned}
& \leq \alpha + C \left| \sum_{j_{\mathcal{M}_o}=1}^{N_o} W_{ij_{\mathcal{M}_o}}^e u(x_{j_{\mathcal{M}_o}}) + b_i^e - \left( \sum_{k_{\mathcal{M}_n}=1}^{N_n} \tilde{W}_{ik_{\mathcal{M}_n}}^e u(x_{k_{\mathcal{M}_n}}) + \tilde{b}_i^e \right) \right|, \\
& = \alpha + C \left| \sum_{j_{\mathcal{M}_o}=1}^{N_o} W_{ij_{\mathcal{M}_o}}^e u(x_{j_{\mathcal{M}_o}}) - \sum_{k_{\mathcal{M}_n}=1}^{N_n} \tilde{W}_{ik_{\mathcal{M}_n}}^e u(x_{k_{\mathcal{M}_n}}) \right|, \\
& = \alpha + C \left| \sum_{j_{\mathcal{M}_o}=1}^{N_o} W_{ij_{\mathcal{M}_o}}^e u(x_{j_{\mathcal{M}_o}}) - \sum_{k_{\mathcal{M}_n}=1}^{N_n} \sum_{\forall m_{\mathcal{M}_o} \text{ s.t. } m_{\mathcal{M}_o} \leftrightarrow k_{\mathcal{M}_n}} \frac{W_{im_{\mathcal{M}_o}}^e}{|\{h_{\mathcal{M}_n} \text{ s.t. } m_{\mathcal{M}_o} \leftrightarrow h_{\mathcal{M}_n}\}|} u(x_{k_{\mathcal{M}_n}}) \right|, \\
& = \alpha + C \left| \sum_{j_{\mathcal{M}_o}=1}^{N_o} W_{ij_{\mathcal{M}_o}}^e \left( u(x_{j_{\mathcal{M}_o}}) - \sum_{\forall k_{\mathcal{M}_n} \text{ s.t. } j_{\mathcal{M}_o} \leftrightarrow k_{\mathcal{M}_n}} \frac{u(x_{k_{\mathcal{M}_n}})}{|\{h_{\mathcal{M}_n} \text{ s.t. } j_{\mathcal{M}_o} \leftrightarrow h_{\mathcal{M}_n}\}|} \right) \right|, \\
& = \alpha + C \left| \sum_{j_{\mathcal{M}_o}=1}^{N_o} \frac{W_{ij_{\mathcal{M}_o}}^e}{|\{h_{\mathcal{M}_n} \text{ s.t. } j_{\mathcal{M}_o} \leftrightarrow h_{\mathcal{M}_n}\}|} \sum_{\forall k_{\mathcal{M}_n} \text{ s.t. } j_{\mathcal{M}_o} \leftrightarrow k_{\mathcal{M}_n}} (u(x_{j_{\mathcal{M}_o}}) - u(x_{k_{\mathcal{M}_n}})) \right|, \\
& = \alpha + C \left| \sum_{j_{\mathcal{M}_o}=1}^{N_o} \frac{W_{ij_{\mathcal{M}_o}}^e}{|\{h_{\mathcal{M}_n} \text{ s.t. } j_{\mathcal{M}_o} \leftrightarrow h_{\mathcal{M}_n}\}|} \sum_{\forall k_{\mathcal{M}_n} \text{ s.t. } j_{\mathcal{M}_o} \leftrightarrow k_{\mathcal{M}_n}} \delta \right|, \\
& \leq \alpha + \delta C \sum_{j_{\mathcal{M}_o}=1}^{N_o} |W_{ij_{\mathcal{M}_o}}^e|.
\end{aligned}$$

To obtain a bound independent of  $i$ , one can bound as

$$\leq \alpha + \delta C \|\mathbf{W}^e\|_{\infty}.$$

This concludes the proof. For the multiple layer case, the result is

$$\begin{aligned}
& \leq \alpha + \delta C^{P+1} \sum_{l_P=1}^{L_P} |W_{i,l_P}^{(P)}| \sum_{l_{P-1}=1}^{L_{P-1}} |W_{l_P,l_{P-1}}^{(P-1)}| \cdots \sum_{l_1=1}^{L_1} |W_{l_2,l_1}^{(1)}| \sum_{j_{\mathcal{M}_o}=1}^{N_o} |W_{l_1,j_{\mathcal{M}_o}}^e|, \\
& \leq \alpha + \delta C^{P+1} \|\mathbf{W}^e\|_{\infty} \prod_{p=1}^P \|\mathbf{W}^{(p)}\|_{\infty}.
\end{aligned}$$

**A.8. Super- and sub-resolution error bound for GFN-ROM autoencoder.** We seek a bound on

$$\left| u(x_{i_{\mathcal{M}_n}}) - \text{dec}^{\mathcal{M}_o \rightarrow \mathcal{M}_n}(\text{enc}^{\mathcal{M}_o \rightarrow \mathcal{M}_n}(\mathbf{u}_{\mathcal{M}_n}))_{i_{\mathcal{M}_n}} \right|, \forall i_{\mathcal{M}_n}.$$

To begin bounding this, we can rewrite the expression as

$$\begin{aligned}
&= \left| u(x_{i_{\mathcal{M}_n}}) - \frac{\text{mean}}{\forall k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow \rightarrow i_{\mathcal{M}_n}} u(x_{k_{\mathcal{M}_o}}) \right. \\
&\quad + \frac{\text{mean}}{\forall k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow \rightarrow i_{\mathcal{M}_n}} u(x_{k_{\mathcal{M}_o}}) - \frac{\text{mean}}{\forall k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow \rightarrow i_{\mathcal{M}_n}} \text{dec}^{\mathcal{M}_o \rightarrow \mathcal{M}_o}(\text{enc}^{\mathcal{M}_o \rightarrow \mathcal{M}_o}(\mathbf{u}_{\mathcal{M}_o}))_{k_{\mathcal{M}_o}} \\
&\quad \left. + \frac{\text{mean}}{\forall k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow \rightarrow i_{\mathcal{M}_n}} \text{dec}^{\mathcal{M}_o \rightarrow \mathcal{M}_o}(\text{enc}^{\mathcal{M}_o \rightarrow \mathcal{M}_o}(\mathbf{u}_{\mathcal{M}_o}))_{k_{\mathcal{M}_o}} - \text{dec}^{\mathcal{M}_o \rightarrow \mathcal{M}_n}(\text{enc}^{\mathcal{M}_o \rightarrow \mathcal{M}_n}(\mathbf{u}_{\mathcal{M}_n}))_{i_{\mathcal{M}_n}} \right|, \\
&\leq \left| u(x_{i_{\mathcal{M}_n}}) - \frac{\text{mean}}{\forall k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow \rightarrow i_{\mathcal{M}_n}} u(x_{k_{\mathcal{M}_o}}) \right| \\
&\quad + \left| \frac{\text{mean}}{\forall k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow \rightarrow i_{\mathcal{M}_n}} u(x_{k_{\mathcal{M}_o}}) - \frac{\text{mean}}{\forall k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow \rightarrow i_{\mathcal{M}_n}} \text{dec}^{\mathcal{M}_o \rightarrow \mathcal{M}_o}(\text{enc}^{\mathcal{M}_o \rightarrow \mathcal{M}_o}(\mathbf{u}_{\mathcal{M}_o}))_{k_{\mathcal{M}_o}} \right| \\
&\quad + \left| \frac{\text{mean}}{\forall k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow \rightarrow i_{\mathcal{M}_n}} \text{dec}^{\mathcal{M}_o \rightarrow \mathcal{M}_o}(\text{enc}^{\mathcal{M}_o \rightarrow \mathcal{M}_o}(\mathbf{u}_{\mathcal{M}_o}))_{k_{\mathcal{M}_o}} - \text{dec}^{\mathcal{M}_o \rightarrow \mathcal{M}_n}(\text{enc}^{\mathcal{M}_o \rightarrow \mathcal{M}_n}(\mathbf{u}_{\mathcal{M}_n}))_{i_{\mathcal{M}_n}} \right|, \\
&\leq \delta + \beta + \left| \frac{\text{mean}}{\forall k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow \rightarrow i_{\mathcal{M}_n}} \text{dec}^{\mathcal{M}_o \rightarrow \mathcal{M}_o}(\text{enc}^{\mathcal{M}_o \rightarrow \mathcal{M}_o}(\mathbf{u}_{\mathcal{M}_o}))_{k_{\mathcal{M}_o}} - \text{dec}^{\mathcal{M}_o \rightarrow \mathcal{M}_n}(\text{enc}^{\mathcal{M}_o \rightarrow \mathcal{M}_n}(\mathbf{u}_{\mathcal{M}_n}))_{i_{\mathcal{M}_n}} \right|.
\end{aligned}$$

We consider the last term in more detail. We have

$$\begin{aligned}
&\left| \frac{\text{mean}}{\forall k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow \rightarrow i_{\mathcal{M}_n}} \text{dec}^{\mathcal{M}_o \rightarrow \mathcal{M}_o}(\text{enc}^{\mathcal{M}_o \rightarrow \mathcal{M}_o}(\mathbf{u}_{\mathcal{M}_o}))_{k_{\mathcal{M}_o}} - \text{dec}^{\mathcal{M}_o \rightarrow \mathcal{M}_n}(\text{enc}^{\mathcal{M}_o \rightarrow \mathcal{M}_n}(\mathbf{u}_{\mathcal{M}_n}))_{i_{\mathcal{M}_n}} \right| \\
&= \left| \frac{\text{mean}}{\forall k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow \rightarrow i_{\mathcal{M}_n}} \left( \sum_{j=1}^L W_{k_{\mathcal{M}_o} j}^d \text{enc}^{\mathcal{M}_o \rightarrow \mathcal{M}_o}(\mathbf{u}_{\mathcal{M}_o})_j + b_{k_{\mathcal{M}_o}}^d \right) \right. \\
&\quad \left. - \sum_{m=1}^L \tilde{W}_{i_{\mathcal{M}_n} m}^d \text{enc}^{\mathcal{M}_o \rightarrow \mathcal{M}_n}(\mathbf{u}_{\mathcal{M}_n})_m - \tilde{b}_{i_{\mathcal{M}_n}}^d \right|, \\
&= \left| \frac{\text{mean}}{\forall k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow \rightarrow i_{\mathcal{M}_n}} \left( \sum_{j=1}^L W_{k_{\mathcal{M}_o} j}^d \text{enc}^{\mathcal{M}_o \rightarrow \mathcal{M}_o}(\mathbf{u}_{\mathcal{M}_o})_j + b_{k_{\mathcal{M}_o}}^d \right) \right. \\
&\quad \left. - \sum_{m=1}^L \frac{\text{mean}}{\forall k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow \rightarrow i_{\mathcal{M}_n}} \left( W_{k_{\mathcal{M}_o} m}^d \right) \text{enc}^{\mathcal{M}_o \rightarrow \mathcal{M}_n}(\mathbf{u}_{\mathcal{M}_n})_m - \frac{\text{mean}}{\forall k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow \rightarrow i_{\mathcal{M}_n}} \left( b_{k_{\mathcal{M}_o}}^d \right) \right|, \\
&= \left| \frac{\text{mean}}{\forall k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow \rightarrow i_{\mathcal{M}_n}} \sum_{j=1}^L W_{k_{\mathcal{M}_o} j}^d \left( \text{enc}^{\mathcal{M}_o \rightarrow \mathcal{M}_o}(\mathbf{u}_{\mathcal{M}_o})_j - \text{enc}^{\mathcal{M}_o \rightarrow \mathcal{M}_n}(\mathbf{u}_{\mathcal{M}_n})_j \right) \right|, \\
&\leq \frac{\text{mean}}{\forall k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \leftarrow \rightarrow i_{\mathcal{M}_n}} \sum_{j=1}^L \left| W_{k_{\mathcal{M}_o} j}^d \right| \left| \text{enc}^{\mathcal{M}_o \rightarrow \mathcal{M}_o}(\mathbf{u}_{\mathcal{M}_o})_j - \text{enc}^{\mathcal{M}_o \rightarrow \mathcal{M}_n}(\mathbf{u}_{\mathcal{M}_n})_j \right|.
\end{aligned}$$

We assume a Lipschitz condition with Lipschitz constant  $C$  for the activation function  $\sigma$ , allowing us to bound

$$\begin{aligned}
& \left| \text{enc}^{\mathcal{M}_o \rightarrow \mathcal{M}_o}(\mathbf{u}_{\mathcal{M}_o})_i - \text{enc}^{\mathcal{M}_o \rightarrow \mathcal{M}_n}(\mathbf{u}_{\mathcal{M}_n})_i \right| \\
&= \left| \sigma \left( \sum_{j_{\mathcal{M}_o}=1}^{N_o} W_{ij_{\mathcal{M}_o}}^e u(x_{j_{\mathcal{M}_o}}) + b_i^e \right) - \sigma \left( \sum_{k_{\mathcal{M}_n}=1}^{N_n} \tilde{W}_{ik_{\mathcal{M}_n}}^e u(x_{k_{\mathcal{M}_n}}) + \tilde{b}_i^e \right) \right|, \\
&\leq C \left| \sum_{j_{\mathcal{M}_o}=1}^{N_o} W_{ij_{\mathcal{M}_o}}^e u(x_{j_{\mathcal{M}_o}}) + b_i^e - \left( \sum_{k_{\mathcal{M}_n}=1}^{N_n} \tilde{W}_{ik_{\mathcal{M}_n}}^e u(x_{k_{\mathcal{M}_n}}) + \tilde{b}_i^e \right) \right|, \\
&= C \left| \sum_{j_{\mathcal{M}_o}=1}^{N_o} W_{ij_{\mathcal{M}_o}}^e u(x_{j_{\mathcal{M}_o}}) - \sum_{k_{\mathcal{M}_n}=1}^{N_n} \tilde{W}_{ik_{\mathcal{M}_n}}^e u(x_{k_{\mathcal{M}_n}}) \right|, \\
&= C \left| \sum_{j_{\mathcal{M}_o}=1}^{N_o} W_{ij_{\mathcal{M}_o}}^e u(x_{j_{\mathcal{M}_o}}) - \sum_{k_{\mathcal{M}_n}=1}^{N_n} \sum_{\forall m_{\mathcal{M}_o} \text{ s.t. } m_{\mathcal{M}_o} \leftrightarrow k_{\mathcal{M}_n}} \frac{W_{im_{\mathcal{M}_o}}^e}{|\{h_{\mathcal{M}_n} \text{ s.t. } m_{\mathcal{M}_o} \leftrightarrow h_{\mathcal{M}_n}\}|} u(x_{k_{\mathcal{M}_n}}) \right|, \\
&= C \left| \sum_{j_{\mathcal{M}_o}=1}^{N_o} W_{ij_{\mathcal{M}_o}}^e \left( u(x_{j_{\mathcal{M}_o}}) - \sum_{\forall k_{\mathcal{M}_n} \text{ s.t. } j_{\mathcal{M}_o} \leftrightarrow k_{\mathcal{M}_n}} \frac{u(x_{k_{\mathcal{M}_n}})}{|\{h_{\mathcal{M}_n} \text{ s.t. } j_{\mathcal{M}_o} \leftrightarrow h_{\mathcal{M}_n}\}|} \right) \right|, \\
&= C \left| \sum_{j_{\mathcal{M}_o}=1}^{N_o} \frac{W_{ij_{\mathcal{M}_o}}^e}{|\{h_{\mathcal{M}_n} \text{ s.t. } j_{\mathcal{M}_o} \leftrightarrow h_{\mathcal{M}_n}\}|} \sum_{\forall k_{\mathcal{M}_n} \text{ s.t. } j_{\mathcal{M}_o} \leftrightarrow k_{\mathcal{M}_n}} (u(x_{j_{\mathcal{M}_o}}) - u(x_{k_{\mathcal{M}_n}})) \right|, \\
&= C \left| \sum_{j_{\mathcal{M}_o}=1}^{N_o} \frac{W_{ij_{\mathcal{M}_o}}^e}{|\{h_{\mathcal{M}_n} \text{ s.t. } j_{\mathcal{M}_o} \leftrightarrow h_{\mathcal{M}_n}\}|} \sum_{\forall k_{\mathcal{M}_n} \text{ s.t. } j_{\mathcal{M}_o} \leftrightarrow k_{\mathcal{M}_n}} \delta \right|, \\
&\leq \delta C \sum_{j_{\mathcal{M}_o}=1}^{N_o} |W_{ij_{\mathcal{M}_o}}^e|.
\end{aligned}$$

Substituting these expressions into the original bound, we obtain

$$\leq \delta + \beta + \delta C \text{mean}_{\forall k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \leftrightarrow i_{\mathcal{M}_n}} \sum_{j=1}^L |W_{k_{\mathcal{M}_o} j}^d| \sum_{m_{\mathcal{M}_o}=1}^{N_o} |W_{jm_{\mathcal{M}_o}}^e|.$$

To obtain a bound independent of  $i$ , one can bound as

$$\leq \delta + \beta + \delta C \|\mathbf{W}^d\|_\infty \|\mathbf{W}^e\|_\infty.$$

This concludes the proof. For the multiple layer case, the result is

$$\begin{aligned}
&\leq \beta + \delta + \delta C^{Q+1} \text{mean}_{\forall k_{\mathcal{M}_o} \text{ s.t. } k_{\mathcal{M}_o} \leftrightarrow i_{\mathcal{M}_n}} \sum_{j=1}^{LQ+1} |W_{k_{\mathcal{M}_o} j}^d| \sum_{l_Q=1}^{LQ} |W_{j, l_Q}^{(Q)}| \cdots \sum_{l_1=1}^{L_1} |W_{l_2, l_1}^{(1)}| \sum_{j_{\mathcal{M}_o}=1}^{N_o} |W_{j_{\mathcal{M}_o} l_1}^e|, \\
&= \beta + \delta + \delta C^{Q+1} \|\mathbf{W}^d\|_\infty \|\mathbf{W}^e\|_\infty \prod_{p=1}^Q \|\mathbf{W}^{(p)}\|_\infty.
\end{aligned}$$

## APPENDIX B. TRAINING DETAILS

## B.1. Network architecture and hyperparameter choices.

TABLE 4. Network architecture and hyperparameter choices for the MOR methods.

	POD-NN	GCA-ROM	GFN-ROM
Bottleneck size ( $N$ )	$\lfloor 1.5 \times N_\mu \rfloor$	$\lfloor 1.5 \times N_\mu \rfloor$	$\lfloor 1.5 \times N_\mu \rfloor$
Mapper weight ( $\omega$ )	-	10	10
Optimiser	Adam	Adam	Adam <sup>a</sup>
Learning rate	$10^{-3}$	$10^{-3}$	$10^{-3}$
L2 regularisation	$10^{-5}$	$10^{-5}$	$10^{-5}$
Training epochs	5000	5000	5000
Autoencoder sizes	-	$[\mathcal{M}], 200, N]$	$[\mathcal{M}], 200, N]$
Train/test split	30/70	30/70	30/70
Autoencoder activation	-	ELU <sup>b</sup>	Tanh <sup>b</sup>
Mapper sizes	$[N_\mu, 50, 50, 50, 50, N]$	$[N_\mu, 50, 50, 50, 50, N]$	$[N_\mu, 50, 50, 50, 50, N]$
Mapper activation	Tanh	Tanh <sup>c</sup>	Tanh

\* POD-G is also undertaken with  $N = \lfloor 1.5 \times N_\mu \rfloor$  degrees of freedom and a train/test split of 30/70.

<sup>a</sup> We employ the precomputed adaptive method so that we can use Adam.

<sup>b</sup> Activation not applied to the last decoder layer.

<sup>c</sup> Activation not applied to the last mapper layer.