Data-Centric Engineering (2025), 6: e52
doi:10.1017/dce.2025.10030

CAMBRIDGE UNIVERSITY PRESS

RESEARCH ARTICLE

Projection-based model-order reduction via graph autoencoders suited for unstructured meshes

Liam Magargal[1], Parisa Khodabakhshi[1], Steven Rodriguez[2], Justin Jaworski[3] and John Michopoulos[4]

[1]Department of Mechanical Engineering and Mechanics, Lehigh University, Bethlehem, PA, USA
[2]Computational Multiphysics Systems Laboratory, US Naval Research Laboratory, Washington, DC, USA
[3]Kevin T. Crofton Department of Aerospace and Ocean Engineering, Virginia Tech, Blacksburg, VA, USA
[4]Principal Scientist for Material Innovation (Retired), US Naval Research Laboratory, Washington, DC, USA
Corresponding author: Parisa Khodabakhshi; Email: pak322@lehigh.edu

Received: 23 April 2025; **Revised:** 11 September 2025; **Accepted:** 19 October 2025

Keywords: deep least-squares Petrov–Galerkin; geometric deep learning; graph autoencoders; projection-based model-order reduction; unstructured mesh

Abstract

This paper presents the development of a graph autoencoder architecture capable of performing projection-based model-order reduction (PMOR) using a nonlinear manifold least-squares Petrov–Galerkin (LSPG) projection scheme. The architecture is particularly useful for advection-dominated flows modeled by unstructured meshes, as it provides a robust nonlinear mapping that can be leveraged in a PMOR setting. The presented graph autoencoder is constructed with a two-part process that consists of (1) generating a hierarchy of reduced graphs to emulate the compressive abilities of convolutional neural networks (CNNs) and (2) training a message passing operation at each step in the hierarchy of reduced graphs to emulate the filtering process of a CNN. The resulting framework provides improved flexibility over traditional CNN-based autoencoders because it is readily extendable to unstructured meshes. We provide an analysis of the interpretability of the graph autoencoder's latent state variables, where we find that the Jacobian of the decoder for the proposed graph autoencoder provides interpretable mode shapes akin to traditional proper orthogonal decomposition modes. To highlight the capabilities of the proposed framework, which is named geometric deep least-squares Petrov–Galerkin (GD-LSPG), we benchmark the method on a one-dimensional Burgers' model with a structured mesh and demonstrate the flexibility of GD-LSPG by deploying it on two test cases for two-dimensional Euler equations that use an unstructured mesh. The proposed framework is more flexible than using a traditional CNN-based autoencoder and provides considerable improvement in accuracy for very low-dimensional latent spaces in comparison with traditional affine projections.

Impact Statement

This work presents a novel graph autoencoder architecture designed for nonlinear dimensionality reduction of advection-dominated systems, resulting in very low-dimensional, yet interpretable, representations of the high-dimensional system. We demonstrate the autoencoder's applicability to a projection-based model-order reduction framework. The graph autoencoder's natural ability to represent unstructured meshes offers greater flexibility compared to convolutional neural network-based autoencoders, which are inherently suited for structured meshes.

This research article was awarded Open Materials badge for transparent practices. See the Data Availability Statement for details.

1. Introduction

Methods in computational mechanics aim to simulate complex physical phenomena via numerical methods. Specifically, approximate solutions are sought by spatially and temporally discretizing the governing equations of a physical system (LeVeque, 2002; Mazumder, 2015). In many engineering applications, spatial and temporal resolution must be refined to obtain a sufficiently detailed solution, ultimately resulting in extremely high-dimensional discretized systems. Dimensional compression aims to embed such high-dimensional discretized systems into low-dimensional representations, while retaining essential information.

Projection-based model-order reduction (PMOR) is a class of approximation methods that aims to reduce the computational cost associated with many-query tasks in computational mechanics, such as design optimization, uncertainty quantification, real-time rendering, etc., while preserving sufficient accuracy of the quantities of interest (Benner et al., 2015). PMOR achieves cost savings by projecting the original high-dimensional computational model (known in this context as the full-order model, or FOM in short) onto a precomputed low-dimensional latent space, which is computed using data recovered from the FOM in an offline stage. In the online stage, the projected form of the governing equations can be used to compute low-dimensional solutions, thereby reducing the operation count complexity and achieving cost savings.

In this study, our primary interest is on the dimensionality reduction capabilities of PMOR approaches for methods in computational mechanics that employ unstructured meshes. Two such methods are the finite volume method (FVM) and the finite element method (FEM), which are widely used in the realms of science and engineering for their ability to conduct high-fidelity simulations of complex physical phenomena (LeVeque, 2002; Reddy, 2005; Mazumder, 2015). Spatial discretization of a domain is typically achieved using one of two main mesh types: structured and unstructured meshes. Structured meshes employ a periodic, grid-like structure to discretize the domain. Conversely, unstructured meshes do not require a grid-like structure and allow mesh components to be arbitrarily ordered (Bern and Plassmann, 2000; Mazumder, 2015). This departure from a grid-like structure often makes spatial discretization of physical domains more convenient, allowing for the generation of a higher quality mesh with favorable features. The advantage of unstructured meshes over structured meshes becomes more prominent in domains with complex geometries.

Traditionally, PMOR methods rely on projecting the solution onto a low-dimensional latent space via a subspace approximation method, such as proper orthogonal decomposition (POD) (Sirovich, 1987), rational interpolation (Baur et al, 2011), or balanced truncation (Moore, 1981; Rezaian and Duraisamy, 2023). Although affine latent spaces have been leveraged extensively to achieve cost savings for a wide variety of linear and nonlinear models (Lieu et al., 2006; Bui-Thanh et al., 2008; Wentland et al., 2023), PMOR procedures employing affine solution manifolds often fail to accurately model advection-dominated solutions. Such solutions often exhibit features, such as sharp gradients, moving shocks and boundaries, and bulk motion, which will be smoothed out using affine solution manifolds. It is well known that these models exhibit a slowly decaying Kolmogorov n-width. The Kolmogorov n-width serves as a measure for the error introduced by approximating the solution manifold of a partial differential equation (PDE) with a linear trial manifold of dimension n (Welper, 2017; Ahmed and San, 2020; Peherstorfer, 2020, 2022; Franco et al., 2023). When the decay of the Kolmogorov n-width is slow, the affine latent space used to approximate the solution must be constructed with a high dimension, leading to marginal model reduction. As a result, a great amount of effort has been made to develop reduced-order models (ROMs) for advection-dominated flows, such as adaptive reduced basis schemes (Drohmann et al., 2011; Ohlberger and Rave, 2013), segmentation of the domain into multiple reduced-order bases (Dihlmann et al., 2011; Amsallem et al., 2012; Geelen and Willcox, 2022), quadratic manifolds (Barnett and Farhat, 2022; Geelen et al., 2023), modified POD bases (Welper, 2017; Nair and Balajewicz, 2019), and neural-network augmented latent spaces (Fresca and Manzoni, 2022; Barnett et al., 2023). None of these methods, however, have direct knowledge of the geometric structure and topology of the spatially discretized domain. Additionally, many are still fundamentally based on affine latent spaces. In this work,

our graph autoencoder aims to leverage knowledge of the geometric structure and topology of the spatially discretized domain to construct a robust nonlinear mapping from the high-dimensional space to the low-dimensional latent space.

Achieving cost-savings via PMOR for linear systems often comes directly from dimensional compression alone (Antoulas, 2005; Benner et al. 2017). However, for nonlinear dynamical systems, projection of the nonlinear term often has an associated operation count complexity that scales directly with the dimension of the FOM, which often results in minimal (if any) cost savings. As a remedy, some studies employ hyper-reduction methods, where the nonlinear terms are computed for a selection of sample points and used to update the corresponding low-dimensional states. Such methods include the discrete empirical interpolation method (Barrault et al., 2004; Chaturantabut and Sorensen, 2010; Drmac and Gugercin, 2016), Gauss-Newton with approximated tensors method (Carlberg et al., 2011, 2013), energy conserving sampling and weighing method (Farhat et al., 2015), and projection tree reduced-order modeling (PTROM) for non-local methods with dense adjacency matrices (Rodriguez et al., 2022). Alternatively, operator inference is often used to learn low-dimensional operators of a nonlinear equation of polynomial form from a regression problem (Peherstorfer and Willcox, 2016; Benner et al., 2020; McQuarrie et al., 2021). Furthermore, some methods have coupled the operator inference framework with a lifting transformation suited for nonlinear problems with general nonlinearity by introducing a change of variables to obtain a polynomial form of the model equations (Qian et al., 2020; Khodabakhshi and Willcox, 2022).

Recently, machine learning has been adopted to overcome the limitations of traditional model reduction when applied to advection-dominated flows with slowly decaying Kolmogorov n-widths. Historically, autoencoders have been developed to compress and reconstruct input information, such as images (Bank et al., 2023), but recently autoencoders have been leveraged in engineering applications. Specifically, the model reduction community has used autoencoders to identify a nonlinear mapping between the high-dimensional system and a low-dimensional latent space (Wiewel et al., 2019; Hasegawa et al., 2020) from state solution data alone. Once an autoencoder is trained, the mapping is leveraged to perform online time integration using one of two main classes of approaches. The first class involves learning the low-dimensional dynamics without knowledge of the dynamics of the high-dimensional system. Some studies train a neural network to approximate the low-dimensional update at each time step (Kim et al., 2019; Wiewel et al., 2019; Fresca et al., 2021; Maulik et al., 2021; Dutta et al., 2022), while others aim to obtain a low-dimensional system of semi-discrete ordinary differential equations (ODEs) (Fries et al., 2022; He et al., 2023). The second class, and the approach that we adopt in this paper, aims to project the governing equations of the semi-discrete high-dimensional dynamical system onto the low-dimensional latent space using the autoencoder, thereby embedding the physics into the resulting ROM (Kashima, 2016; Hartman and Mestha, 2017; Lee and Carlberg, 2020, 2021; Kim et al., 2022).

A common method used across both classes of machine learning-based ROMs is the convolutional neural network (CNN), which is used to construct low-dimensional solution manifolds (Kim et al., 2019; Wiewel et al., 2019; Hasegawa et al., 2020). Because CNNs require inputs to be constructed as a grid, the direct application of CNNs to unstructured meshes for the purpose of model reduction is currently untenable. As a result, a common workaround is to interpolate the unstructured mesh solution onto a structured mesh that can be provided to a CNN (Fresca et al., 2021). In recent years, graph neural networks (GNNs) have been developed to extract information of interest to the user from sets of unstructured and relational data (Battaglia et al., 2018; Zhou et al., 2020), making them an appropriate method to generate low-dimensional embeddings of models that use unstructured meshes. While dimensionality reduction and compression using GNNs has been achieved by graph U-nets (Gao and Ji, 2019) and multiscale graph autoencoders (Barwey et al., 2023), such approaches do not have a decoder that maps directly from the latent state vector to the approximate solution. Instead, they require knowledge of the encoder and cannot be directly used in PMOR. Alternatively, graph autoencoders have been used for fully data-driven model reduction (Gruber et al., 2022; Pichi et al., 2024), but such approaches do not include any knowledge of the governing equations.

The main focus of this study is to investigate the abilities of graph autoencoders to perform dimensionality reduction upon unstructured mesh solutions. Toward this end, we present a hierarchical

graph autoencoder architecture tailored to generate nonlinear mappings to very low-dimensional latent spaces. Once the low-dimensional latent space is obtained, we embed knowledge of the governing equations into the latent space representation to perform time integration by leveraging a nonlinear manifold least-squares Petrov–Galerkin (LSPG) projection. Traditionally, a nonlinear manifold LSPG projection that leverages a CNN-based autoencoder is referred to as deep LSPG (dLSPG; Lee and Carlberg, 2020, 2021). In this study, instead, we use a hierarchical graph autoencoder architecture, and the resulting method is named geometric deep LSPG (GD-LSPG). The proposed method is capable of performing PMOR when deploying unstructured meshes and is particularly useful in modeling the highly nonlinear behavior found in advection-dominated flows, while achieving significant dimensionality reduction. We analyze GD-LSPG from two different perspectives. First, we assess GD-LSPG's ability to generate low-dimensional solutions to parameter sets not seen during training. Second, we investigate the interpretability of the nonlinear manifold LSPG scheme. From this perspective, we discover highly interpretable mode shapes from the Jacobian of the decoder for the graph autoencoder, which can be directly related to the POD modes for a classical POD-LSPG projection (Carlberg et al., 2011, 2013). Furthermore, we find that the Jacobian of the decoder is closely related to saliency maps (Simonyan et al., 2013), a method commonly used in image classification to identify features in the image that are most indicative of the image's classification. Ultimately, the proposed method is capable of accurately modeling highly nonlinear behavior found in advection-driven problems while providing interpretable mode shapes to the user.

The paper is organized in the following manner. Section 2 describes the background and preliminaries of the GD-LSPG framework, which includes the FOM and its corresponding residual minimization scheme, a general formulation of performing nonlinear dimension reduction via autoencoders, and a brief overview of graph theory. Section 3 presents the graph autoencoder deployed in GD-LSPG. Section 4 presents the nonlinear manifold LSPG projection and analyzes the interpretability of the graph autoencoder's latent state variables by relating the Jacobian of the decoder for the graph autoencoder to the POD modes used in POD-LSPG. Section 5 includes a set of numerical experiments to investigate the capabilities of our proposed method. Specifically, we apply GD-LSPG to the benchmark one-dimensional (1D) Burgers' model using a structured mesh and to two test cases that use an unstructured mesh to solve the two-dimensional (2D) Euler equations. Namely, the first test case models a setup for a Riemann problem, while the second models a bow shock generated by flow past a cylinder. Finally, Section 6 presents conclusions and discusses avenues for future work.

2. Background and preliminaries

This section lays the foundation for the introduction of the GD-LSPG method by providing a formulation of the FOM and summarizing preliminaries related to autoencoders and graph theory. Specifically, Section 2.1 introduces the first-order PDE and residual-minimizing time integration scheme on which we develop the GD-LSPG method. Section 2.2 provides a general introduction to performing PMOR with an autoencoder, along with some of the current limitations of autoencoders in the literature. Finally, Section 2.3 presents the basics of graph theory to the reader.

2.1. Full-order model

Consider a system of $n_q \in \mathbb{N}$ PDEs where n_q depends on the number of state variables. Using a mesh to spatially discretize the physical domain into $N_c \in \mathbb{N}$ points, the semi-discrete system of the FOM is described by a system of time-continuous ODEs:

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{dt}} = \mathbf{f}(\mathbf{x}, t; \boldsymbol{\mu}), \quad \mathbf{x}(0; \boldsymbol{\mu}) = \mathbf{x}^0(\boldsymbol{\mu}), \tag{2.1}$$

where $\mathbf{x} \in \mathbb{R}^N$ is the semi-discrete state vector, $N = n_q N_c$ denotes the dimension of the FOM, $\boldsymbol{\mu} \in \mathcal{D}$ denotes the parameters, and $\mathbf{f} : \mathbb{R}^N \times \left(0, T_f\right] \times \mathcal{D} \to \mathbb{R}^N$ is the semi-discretized velocity function.

To approximate the time evolution of the state vector, $\mathbf{x}$, from the system of ODEs, we use the general form in Eq. (2.2),

$$\mathbf{r}:(\boldsymbol{\xi};\boldsymbol{\mu})\mapsto\alpha_0\boldsymbol{\xi}+\sum_{i=1}^{\tau}\alpha_i\mathbf{x}^{n+1-i}+\beta_0\mathbf{f}(\boldsymbol{\xi},t_{n+1};\boldsymbol{\mu})+\sum_{i=1}^{\tau}\beta_i\mathbf{f}\left(\mathbf{x}^{n+1-i},t_{n+1-i};\boldsymbol{\mu}\right), \tag{2.2}$$

in which the value of the state vector, $\mathbf{x}^{n+1}$, at time step $(n+1)\in\mathbb{N}$ is determined by minimizing the time-discrete residual $\mathbf{r}:\mathbb{R}^N\times\mathcal{D}\to\mathbb{R}^N$ given the state vector at a number of previous time steps. In Eq. (2.2), $\alpha_i\in\mathbb{R}$ and $\beta_i\in\mathbb{R}, i=0,1,\ldots,\tau$, are constants defined by the time integration scheme, $\boldsymbol{\xi}\in\mathbb{R}^N$ is the sought-after solution of the minimization scheme for the state vector at the $(n+1)^{\text{th}}$ time step, the superscript $n+1-i$ denotes the value of the variable at time step $n+1-i\in\mathbb{N}$, where the time step size $\Delta t\in\mathbb{R}_+$ is chosen to be fixed, that is, $t_i=i\Delta t$, and $\tau\in\mathbb{N}$ is the number of time steps associated with the time integration scheme. We note that the time integration scheme is implicit in cases where $\beta_0\neq 0$. The state vector at the $(n+1)^{\text{th}}$ time step, $\mathbf{x}^{n+1}$, is defined as the solution of the minimization problem,

$$\mathbf{x}^{n+1}=\underset{\boldsymbol{\xi}\in\mathbb{R}^N}{\arg\min}\,\|\mathbf{r}(\boldsymbol{\xi};\boldsymbol{\mu})\|_2,\quad n=0,\cdots,N_t-1. \tag{2.3}$$

where $N_t\in\mathbb{N}$ denotes the total number of time steps. With an appropriate selection of coefficients α_i and β_i, the general formulation of Eq. (2.2) will cover linear multistep schemes, where specific examples are provided in Section 5.

2.2. Nonlinear dimension reduction via autoencoders

A wide variety of nonlinear mappings have been adopted in the literature in recent years to obtain a low-dimensional latent space for PMOR on nonlinear problems. The focus of this study is on the use of autoencoders to approximate a mapping between the high-dimensional system and the low-dimensional latent space (Lee and Carlberg, 2020, 2021; Barnett and Farhat, 2022; Chen et al., 2022; Eivazi et al., 2022; Pan et al., 2023). Autoencoders are a class of deep learning architecture in which the basic idea is to perform dimensional compression on a dataset down to a latent space with an encoder, $\mathbf{Enc}:\mathbf{x}\mapsto\hat{\mathbf{x}}$ with $\mathbf{Enc}:\mathbb{R}^N\to\mathbb{R}^M$, and to reconstruct the dataset by decoding the latent space with a decoder, $\mathbf{Dec}:\hat{\mathbf{x}}\mapsto\tilde{\mathbf{x}}$ with $\mathbf{Dec}:\mathbb{R}^M\to\mathbb{R}^N$, where $M\ll N$. The former is a nonlinear mapping from the high-dimensional state vector, $\mathbf{x}$, to the low-dimensional latent representation, $\hat{\mathbf{x}}$, and the latter is a nonlinear mapping from the low-dimensional embedding to the reconstructed high-dimensional state vector, $\tilde{\mathbf{x}}$.

The encoder and decoder are constructed by a series of layers in which each layer applies a set of predefined functions to the output of the previous layer. The nonlinearity associated with the mapping is introduced through an appropriate selection of functions. General forms of the encoder and decoder, consisting of $n_h\in\mathbb{N}$ and $n_g\in\mathbb{N}$ layers, respectively, are,

$$\mathbf{Enc}:(\mathbf{x};\theta)\mapsto\mathbf{h}_{n_h}(\,\cdot\,;\boldsymbol{\Theta}_{n_h})\circ\mathbf{h}_{n_h-1}(\,\cdot\,;\boldsymbol{\Theta}_{n_h-1})\circ\ldots\circ\mathbf{h}_2(\,\cdot\,;\boldsymbol{\Theta}_2)\circ\mathbf{h}_1(\mathbf{x};\boldsymbol{\Theta}_1), \tag{2.4}$$

$$\mathbf{Dec}:(\hat{\mathbf{x}};\omega)\mapsto\mathbf{g}_{n_g}(\,\cdot\,;\boldsymbol{\Omega}_{n_g})\circ\mathbf{g}_{n_g-1}(\,\cdot\,;\boldsymbol{\Omega}_{n_g-1})\circ\ldots\circ\mathbf{g}_2(\,\cdot\,;\boldsymbol{\Omega}_2)\circ\mathbf{g}_1(\hat{\mathbf{x}};\boldsymbol{\Omega}_1), \tag{2.5}$$

where $\mathbf{h}_i(\,\cdot\,;\boldsymbol{\Theta}_i), i=1,\ldots,n_h$ and $\mathbf{g}_i(\,\cdot\,;\boldsymbol{\Omega}_i), i=1,\ldots,n_g$ denote the function(s) acting on the input of the i^{th} layer of the encoder and decoder networks, respectively, (or equivalently the output of the corresponding $(i-1)^{\text{th}}$ layer). As will be explained later in Sections 3.2 and 3.3, some layers encompass a number of functions, depending on their objective, which will collectively form $\mathbf{h}_i(\,\cdot\,;\boldsymbol{\Theta}_i)$ or $\mathbf{g}_i(\,\cdot\,;\boldsymbol{\Omega}_i)$. In Eq. (2.4) and Eq. (2.5), $\boldsymbol{\Theta}_i, i=1,\ldots,n_h$ and $\boldsymbol{\Omega}_i, i=1,\ldots,n_g$, denote the weights and biases of the i^{th} layer of the encoder and decoder networks, respectively. The set of all the weights and biases of the autoencoder, that is, $\theta:=\{\boldsymbol{\Theta}_1,\ldots,\boldsymbol{\Theta}_{n_h}\}$ and $\omega:=\{\boldsymbol{\Omega}_1,\ldots,\boldsymbol{\Omega}_{n_g}\}$, are trained to minimize an appropriately defined error norm between the input to the encoder and the output of the decoder. In this manuscript, we use an equal number of layers for the encoder and decoder, that is, $n_h=n_g=n_\ell$.

Due to their remarkable ability to filter grid-based information, in an extensive amount of literature on autoencoder-based PMOR (Kim et al., 2019; Wiewel et al., 2019; Fukami et al., 2020; Hasegawa et al.,

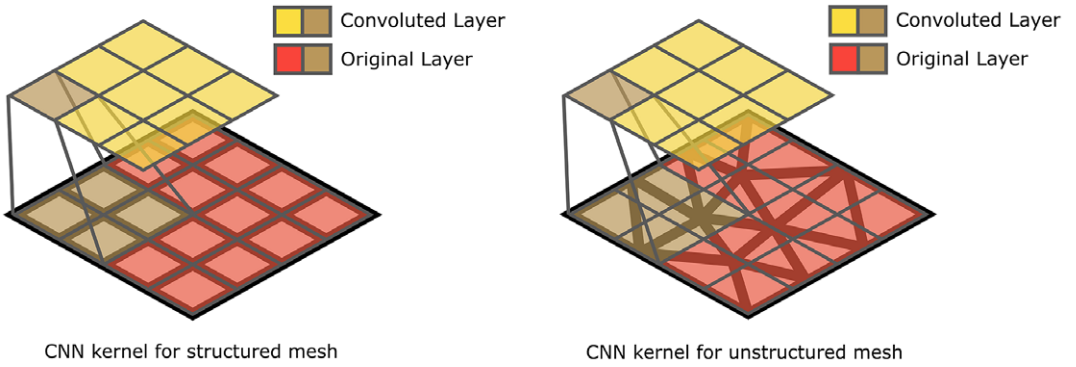

Figure 1. *Visualization of CNN kernel attempting to perform dimensional compression upon a structured mesh (left) and an unstructured mesh (right). Note that the nature of a structured mesh enables direct implementation into a CNN kernel for dimensionality reduction. Specifically, a structured mesh is essentially a pixel-based image commonly found in the CNN literature. Alternatively, the unstructured mesh is not readily formulated as a pixel-based image, meaning that the CNN kernel cannot be immediately applied to the unstructured mesh.*

2020; Lee and Carlberg, 2020, 2021), CNNs have been heavily relied upon as the backbone for the development of autoencoder architectures. However, since CNNs were primarily developed to analyze pixel-based images, they are dependent upon the inputs being a structured grid. As a result, PMOR methods leveraging CNNs are not readily applicable to unstructured meshes (see Figure 1).

A common approach to enable the use of CNNs on unstructured meshes is to interpolate the unstructured mesh onto a structured grid, thereby creating a pixel-based representation suitable for deployment in CNN-based autoencoders (Fresca et al., 2021). As will be demonstrated in Section 5, this strategy often fails to have strong generalization performance. Our proposed method overcomes the need for structured meshes for the sake of autoencoder-based PMOR such that both structured and unstructured meshes can be inputs to the proposed autoencoder. Because GNNs have been developed for non-Euclidean data, such as unstructured meshes, we aim to leverage GNNs to perform dimensionality reduction (see Figure 2). Given that unstructured meshes are commonly used in engineering applications, our approach can be widely extended to applications with arbitrary topology. Our proposed architecture follows an outline that is similar to graph U-nets (Gao and Ji, 2019), multiscale graph autoencoders (Barwey et al 2023), and graph convolutional autoencoders for parameterized PDEs (Pichi et al., 2024), wherein a hierarchy of graphs is generated, each with fewer nodes than the previous level.

2.3. Graph theory

Extensive reading on graph theory can be found in the works of Hamilton (2020) and Battaglia et al (2018), but a brief overview is provided in this section to provide sufficient background for our graph autoencoder architecture. A graph is a tuple $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where $\mathcal{V}$ denotes the node set, $|\mathcal{V}|$ denotes the number of nodes in the graph, and $\mathcal{E}$ denotes the edge set, which is chosen to represent user-prescribed relationships between the nodes in the node set. Depending on the application, the graph (and the associated node and edge sets) can be used to represent a wide variety of concepts. For example, molecules can be modeled as a graph by representing atoms as nodes and bonds as edges (Bongini et al., 2021), while social networks can be modeled as a graph by representing people as nodes and friendships as edges (Newman et al., 2002).

The adjacency matrix, $\mathbf{A} = \left[a_{ij}\right] \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$, is another way to represent the edge set of a graph. Consider the case where the nodes are indexed by a number, $i = 1, \cdots, |\mathcal{V}|$. If nodes i and j in the graph are connected via an edge, that is, if for $i,j \in \mathcal{V}$, we have $(i,j) \in \mathcal{E}$, the corresponding entry in the adjacency matrix is $a_{ij} = 1$. Otherwise, we have $a_{ij} = 0$. In this manuscript, we consider exclusively undirected graphs,

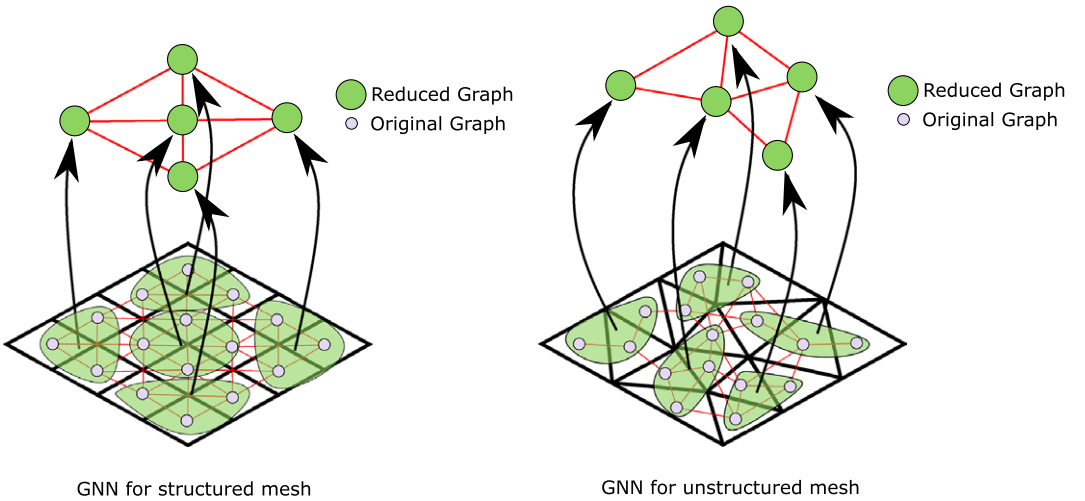

Figure 2. *Visualization of graph autoencoder performing dimensional compression upon a structured mesh (left) and an unstructured mesh (right). Note that, unlike CNNs, the GNN framework readily accepts both structured and unstructured meshes by modeling collocation points as graph nodes and prescribing edges between the collocation points close to each other in the domain.*

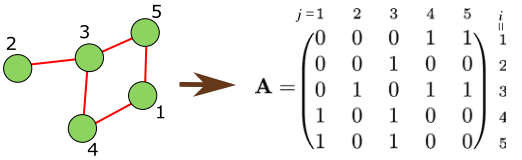

Figure 3. *Formation of the adjacency matrix from a given graph.*

meaning for any edge in the graph, $(i,j) \in \mathcal{E}$, we also have $(j,i) \in \mathcal{E}$. With this formulation, our adjacency matrix will be symmetric. A visualization of the construction of the adjacency matrix for a given graph is found in Figure 3.

A node feature matrix, $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times N_F}$, can be utilized to prescribe feature information to the node set of the graph, where the i^{th} row of X denotes the node feature vector of the i^{th} node in the graph, and $N_F \in \mathbb{N}$ denotes the number of features prescribed to each node.

3. Dimension reduction via graph autoencoders

In this section, we develop the specifics of the graph autoencoder used in GD-LSPG. Upon spatial discretization of the physical domain, each collocation point (e.g., either a cell center in the FVM mesh or a mesh node in the FEM) is represented by a node. We take the node set $\mathcal{V}$ to represent the collocation points in the discretized domain, that is, $|\mathcal{V}| = N_c$. To emulate the manner in which CNNs filter information from neighboring grid points in the spatial discretization, we take the edge set, $\mathcal{E}$, to connect the node representation of collocation points within a user-defined radius of each other, that is,

$$\mathcal{E} = \mathbf{Radius\_Graph}(\mathbf{Pos}, r) = \left\{ \forall (j,k) : \quad j,k \in \mathcal{V}, \quad \|\mathbf{Pos}_j - \mathbf{Pos}_k\| \leq r \right\}, \tag{3.1}$$

where $\mathbf{Pos} \in \mathbb{R}^{N_c \times n_d}$ is the matrix denoting the spatial positions of the node-representation of the collocation points in the discretization. Row j of the matrix (i.e., $\mathbf{Pos}_j$) denotes the position of node $j \in \mathcal{V}$, $n_d \in \mathbb{N}$ denotes the spatial dimensionality of the modeled problem, j and k denote the indices of the corresponding nodes in the node set, $r \in \mathbb{R}$ denotes the user-defined radius, and $\|\cdot\| : \mathbb{R}^{n_d} \to \mathbb{R}_+$ denotes

the Euclidean norm. The adjacency matrix is used to represent the edge set in a matrix format. Alternatively, one could choose to employ an edge set defined by a fixed number of nearest neighbors or an edge set defined by the neighboring collocation points in the mesh. We choose here to define the edge set by the **Radius_Graph** function (Eq. 3.1), as it allows for user control over the number of edges prescribed to each node and is a notion that is not lost under the hierarchical compression used in the graph autoencoder described in Section 3.1 (unlike an edge set defined by neighboring collocation points in the mesh), does not prescribe unnecessary edges or skewed graph topology near boundaries (unlike a fixed number of nearest neighbors), and ensures that the graph is undirected (a requirement for spectral clustering).

The feature matrix, $\mathbf{X} \in \mathbb{R}^{N_c \times n_q}$ is a matrix with the number of rows equal to the number of collocation points in the discretization, that is, N_c, and the number of columns equal to the number of state variables in the governing PDE, n_q. In other words, the feature matrix $\mathbf{X}$ is the matrix version of the state vector $\mathbf{x} \in \mathbb{R}^N$ (with $N = n_q N_c$) introduced in Section 2. As a result, the formulation has a direct mapping between the state vector $\mathbf{x}$ and the node feature matrix $\mathbf{X}$ (**Matricize** : $\mathbf{x} \mapsto \mathbf{X}$, with **Matricize** : $\mathbb{R}^{N_c n_q} \rightarrow \mathbb{R}^{N_c \times n_q}$) and a direct mapping between the node feature matrix $\mathbf{X}$ and the state vector $\mathbf{x}$ (**Vectorize** : $\mathbf{X} \mapsto \mathbf{x}$, with **Vectorize** : $\mathbb{R}^{N_c \times n_q} \rightarrow \mathbb{R}^{N_c n_q}$).

Once a graph representation of a solution state is constructed, it can be encoded to a latent representation with a graph autoencoder following the general form of Eqs. (2.4) and (2.5). In the subsequent sections, we present the graph autoencoder used in the GD-LSPG framework and the specifics of the architecture of the encoder and the decoder. First, Section 3.1 presents a hierarchical spectral clustering algorithm used by the autoencoder to generate a hierarchy of reduced graphs to emulate the compressive abilities of CNNs. Next, Section 3.2 details the encoder architecture and its deployment of the hierarchy of reduced graphs to create a low-dimensional embedding of the input graph. Then, Section 3.3 details the decoder architecture and its deployment of the hierarchy of reduced graphs in reverse order to reconstruct the original input graph from its latent representation. In our graph autoencoder, we include an additional layer with no trainable parameters for preprocessing and postprocessing in the encoder (Section 3.2.1) and the decoder (Section 3.3.3), respectively. Finally, Section 3.4 presents the training strategy deployed to optimize the training parameters of the encoder and decoder. Figure 4 provides a visual representation of the graph autoencoder deployed in GD-LSPG, with $n_\ell = 3$ for demonstration purposes.

3.1. Generating a hierarchy of reduced graphs with spectral clustering

To compute a hierarchy of reduced graphs for the autoencoder used in GD-LSPG, at each level in the hierarchy, we aim to partition the graph into a pre-defined number of non-overlapping sets of strongly connected nodes. We then use the partitions to aggregate each cluster of nodes together into a single node at the next layer of the hierarchy, thereby reducing the number of nodes in the graph and the total dimension of the graph. In this study, we have chosen spectral clustering for two reasons. First, spectral clustering leverages knowledge of graph topology to compute cluster assignments, thereby considering the physical domain's inherent geometry during clustering. Second, its implementation is straightforward because it relies on the same edge sets used during message passing at each level of the hierarchy of reduced graphs. The interested reader is directed to Von Luxburg (2007) for further discussion on the advantages of spectral clustering for graph-structured data.

We consider the case where the encoder and the decoder each have $n_\ell \in \mathbb{N}$ layers. As will be presented in the subsequent sections, the encoder and decoder both have a fully connected/multi-layer perceptron (MLP) layer along with $n_\ell - 1$ layers with compressed graphs. Therefore, in this section, we aim to produce a hierarchy of reduced graphs composed of $n_\ell - 1$ reduced graphs that result in a hierarchy of n_ℓ graphs, including the input graph of the discretized FOM (i.e., graph 0). The graphs in the encoder and the decoder will have the same topology but with reverse ordering. This means that the i^{th} graph in the hierarchy of the graphs of the encoder, $i = 0, \cdots, n_\ell - 1$, will be equivalent to the $(n_\ell - i - 1)^{\text{th}}$ graph of the decoder (refer to Figure 4). In other words, the first graph of the decoder is the $(n_\ell - 1)^{\text{th}}$ (final) graph of the encoder, and the final graph of the decoder is the zeroth (original) graph of the encoder. Hence, we

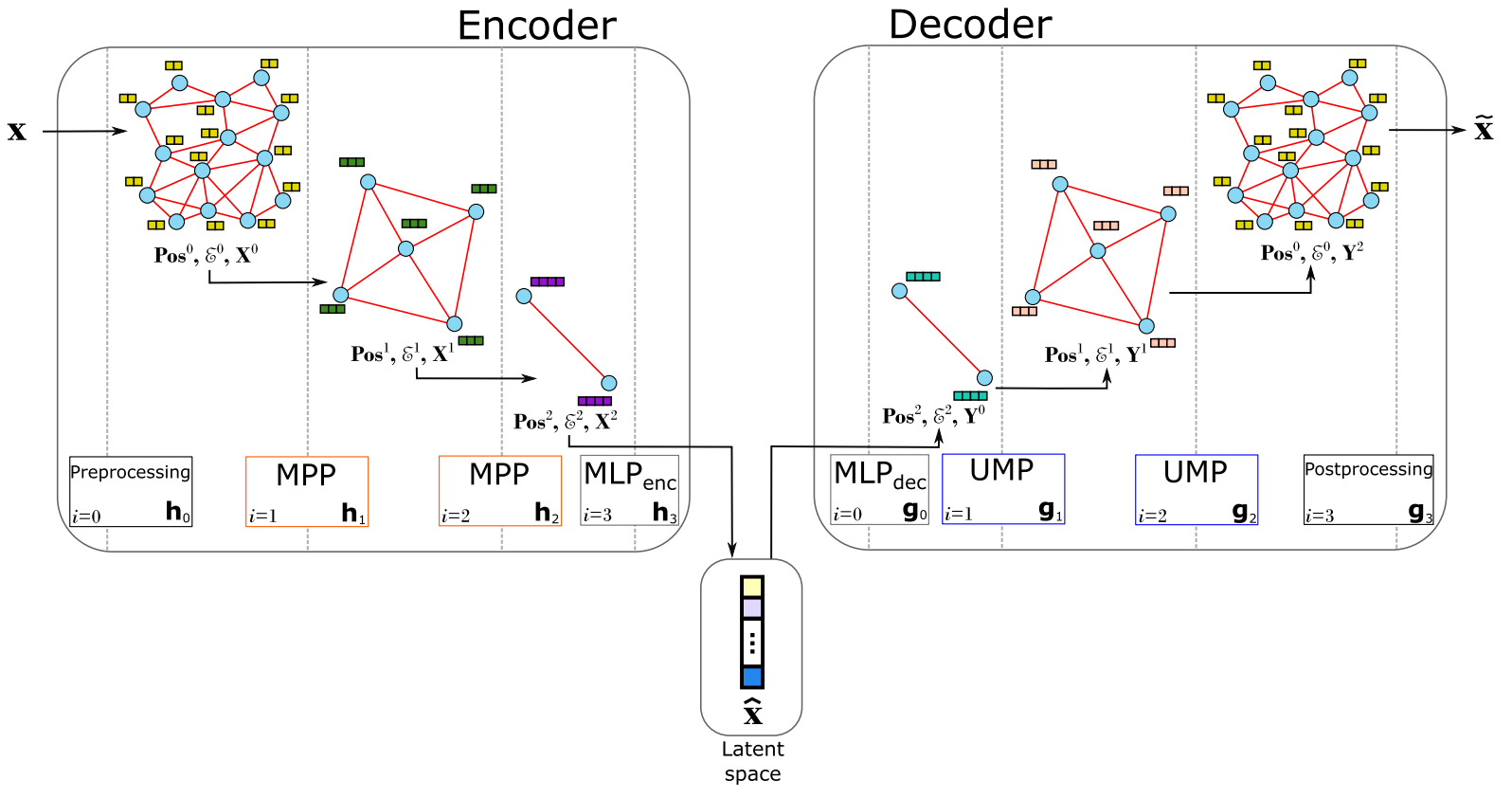

Figure 4. *Graph autoencoder architecture deployed in GD-LSPG. Vertical dotted lines represent the separation between layers in the hierarchy of reduced graphs. Boxes over the vertical dotted lines at the bottom of the figure represent specific layers of the autoencoder described in this section. The encoder is the trained mapping between the high-dimensional state* $\mathbf{x}$ *and the low-dimensional latent space* $\hat{\mathbf{x}}$*. The encoder is comprised of a preprocessing layer to model the input state vector as a graph, followed by a series of trained message passing and pooling (MPP) layers to reduce the number of nodes in the graph, then a flattening and fully connected/multilayer perceptron (MLP) layer. The resulting low-dimensional state vector is sent to the decoder, which is the trained mapping between the low-dimensional latent space* $\hat{\mathbf{x}}$ *and the reconstructed high-dimensional state* $\tilde{\mathbf{x}}$*. The decoder is comprised of a fully connected/MLP layer, followed by a series of unpooling and message passing (UMP) layers to increase the number of nodes in the graph, and then a postprocessing layer to prepare the output for deployment in the time integration scheme. Note that the superscript* i *in* $\mathbf{Pos}^i$ *and* $\mathcal{E}^i$ *represents the graph number in the hierarchy of graphs with* $i=0$ *denoting the original graph representing the discretized mesh.*

focus on building the hierarchy of the graphs for the encoder. This task can be achieved by minimizing the number of "broken" edges in the graph topology of the $(i-1)^{\text{th}}$ layer to form the clusters for the i^{th} layer's graph. The graph representation of the FOM, $\mathcal{G}^0 = (\mathcal{V}^0, \mathcal{E}^0)$ are given from Eq. (3.1) using the discretized mesh of the FOM. In addition, the number of nodes in the layers $i = 1, \cdots, n_\ell - 1$, that is, $\{|\mathcal{V}^1|, |\mathcal{V}^2|, \cdots, |\mathcal{V}^{n_\ell - 1}|\}$, along with the radius used in Eq. (3.1), that is, $\{r^0, r^1, \ldots, r^{n_{\ell-1}}\}$, are hyperparameters prescribed *a priori*, dictating the amount of reduction performed and the number of edges at each layer in the hierarchy. These hyperparameters in our graph autoencoder framework can be viewed to be equivalent to the kernel size and stride used to construct a CNN-based autoencoder.

At layer $i \in \{1, \cdots, n_\ell - 1\}$ of the encoder, we aim to reduce the number of nodes from $|\mathcal{V}^{i-1}|$ in layer $i-1$ to $|\mathcal{V}^i|$ in layer i, with $|\mathcal{V}^i| < |\mathcal{V}^{i-1}|$, by clustering nodes that are strongly connected. This action is carried out by forming $|\mathcal{V}^i|$ clusters, that is, $\mathcal{A}_1^{i-1}, \mathcal{A}_2^{i-1}, \cdots, \mathcal{A}_{|\mathcal{V}^i|}^{i-1}$ with the following conditions,

$$\begin{cases} |\mathcal{A}_j^{i-1}| \geq 1, & j = 1, \cdots, |\mathcal{V}^i| \\ \mathcal{A}_j^{i-1} \subset \mathcal{V}^{i-1}, & j = 1, \cdots, |\mathcal{V}^i| \\ \mathcal{A}_j^{i-1} \cap \mathcal{A}_k^{i-1} = \emptyset, & j \neq k, j, k = 1, \cdots, |\mathcal{V}^i| \\ \mathcal{A}_1^{i-1} \cup \mathcal{A}_2^{i-1} \cup \cdots \cup \mathcal{A}_{|\mathcal{V}^i|}^{i-1} = \mathcal{V}^{i-1}, & \end{cases} \tag{3.2}$$

which ensures that all clusters are a nonempty subsets of the node set of layer $i-1$, the intersection of any two distinct clusters is the empty set, and the union of all clusters is equal to the node set of layer $i-1$. In Eq. (3.2), $|\cdot|$ denotes the cardinality of the set. To define clusters, $\mathcal{A}_1^{i-1}, \mathcal{A}_2^{i-1}, \cdots, \mathcal{A}_{|\mathcal{V}^i|}^{i-1}$, we minimize the **RatioCut** function (Wei and Cheng, 1989),

$$\textbf{RatioCut} : (\mathcal{V}^{i-1}, \mathcal{E}^{i-1}) \mapsto \frac{1}{2} \sum_{k=1}^{|\mathcal{V}^i|} \frac{|\forall (u,v) \in \mathcal{E}^{i-1} : u \in \mathcal{A}_k^{i-1}, v \in \overline{\mathcal{A}}_k^{i-1}|}{|\mathcal{A}_k^{i-1}|}, \tag{3.3}$$

that tends to measure the number of broken edges for the given cluster choice, where $\mathcal{E}^{i-1}$ denotes the edge set of the graph at the $(i-1)^{\text{th}}$ level in the hierarchy, (u,v) represents any existing edge in $\mathcal{E}^{i-1}$ connecting nodes u and v, $\mathcal{A}_k^{i-1} \subset \mathcal{V}^{i-1}$ denotes a subset of nodes in the graph at the $(i-1)^{\text{th}}$ level in the hierarchy and $\overline{\mathcal{A}}_k^{i-1} = \mathcal{V}^{i-1} \backslash \mathcal{A}_k^{i-1}$ denotes the complement of the set $\mathcal{A}_k^{i-1}$ at the same level. Minimizing the **RatioCut** from Eq. (3.3) results in clusters of locally connected nodes with relatively equal sizes (Hamilton, 2020).

The number of distinct ways we can choose $|\mathcal{V}^i|$ clusters from $|\mathcal{V}^{i-1}|$ nodes while satisfying conditions of Eq. (3.2) is determined from the Stirling number of the second kind (Rennie and Dobson, 1969), $S(n,k) = \frac{1}{k!} \sum_{j=0}^{k} (-1)^j \binom{k}{j} (k-j)^n$ with $n = |\mathcal{V}^{i-1}|$ and $k = |\mathcal{V}^i|$ resulting in an NP-hard minimization problem (Von Luxburg, 2007; Hamilton, 2020). As discussed in Von Luxburg (2007), spectral clustering introduces a relaxation on the minimization problem to eliminate its discrete nature. The departure from a discrete set allows the user to perform an eigenvalue analysis on the graph to generate the clusters. Specifically, spectral clustering groups nodes together based on their spectral features defined by the eigenvectors of the Laplacian using any one of a wide variety of standard clustering techniques. In this study, $K-$means clustering (MacQueen, 1967) is chosen as it is commonly used for this application in the literature (Von Luxburg, 2007; Hamilton, 2020). The algorithm for building the hierarchy of reduced graphs is summarized in Algorithm 1.

The method of spectral clustering from Hamilton (2020) is leveraged in this study and makes up steps $1-4$ of Algorithm 1, where $\mathbf{Pos}^i \in \mathbb{R}^{|\mathcal{V}^i| \times n_d}$ denotes the matrix of spatial coordinates for the graph at the i^{th} level in the hierarchy, $\mathbf{A}^i$ is the adjacency matrix of the i^{th} layer in the hierarchy generated by the edge set, $\mathcal{E}^i$, of layer i, previously defined in Eq. (3.1), $r^i \in \mathbb{R}_+$ denotes a user-prescribed radius to be used in Eq. (3.1), $\mathbf{S}^i \in \mathbb{R}^{|\mathcal{V}^{i+1}| \times |\mathcal{V}^i|}$ denotes the assignment matrix of the i^{th} layer of the hierarchy which is used to assign each node in layer i to a cluster in layer $i+1$, and thus a portion of a single node at the layer $i+1$ in

the hierarchy. The assignment matrix (defined to act as an arithmetic mean on the features of the nodes in each cluster) is used to cluster and decrease the number of nodes in the graph at each step in the hierarchy of reduced graphs. In Algorithm 1, $\mathbf{D}^i \in \mathbb{R}^{|\mathcal{V}^i| \times |\mathcal{V}^i|}$ is the diagonal degree matrix representing the number of edges connected to each node in the i^{th} layer of the hierarchy, $\mathbf{L}^i = \mathbf{D}^i - \mathbf{A}^i$ is the Laplacian of the graph associated with the i^{th} layer in the hierarchy, $\mathbf{B}^i \in \mathbb{R}^{|\mathcal{V}^i| \times \left(|\mathcal{V}^{i+1}|-1\right)}$ denotes the spectral node feature matrix formed by the eigenvectors associated with the $|\mathcal{V}^{i+1}|-1$ smallest eigenvalues of $\mathbf{L}^i$, excluding the smallest. According to Von Luxburg (2007), the smallest eigenvalue of the unnormalized Laplacian is simply zero and can therefore be neglected. As dimensional compression is performed in the hierarchy of reduced graphs, the nodes of the graphs deeper in the hierarchy tend to become closer together due to the nature of the positions of each node being computed based on the arithmetic mean of the positions of their corresponding cluster in the previous layer. To avoid the natural accumulation of the nodes to a smaller spatial domain, a rescaling operator, that is, $\mathbf{Rescale}: \mathbb{R}^{|\mathcal{V}^{i+1}| \times n_d} \longrightarrow \mathbb{R}^{|\mathcal{V}^{i+1}| \times n_d}$ is used at each layer to rescale $\mathbf{Pos}^{i+1}$ such that the maximum and minimum values of the coordinates match that of the previous layer in the hierarchy. This algorithm is visually represented in Figure 5.

The construction of the hierarchy of reduced graphs is performed in the offline stage. While the hierarchy of graphs will be utilized in the encoder and decoder, the architecture of the encoder and

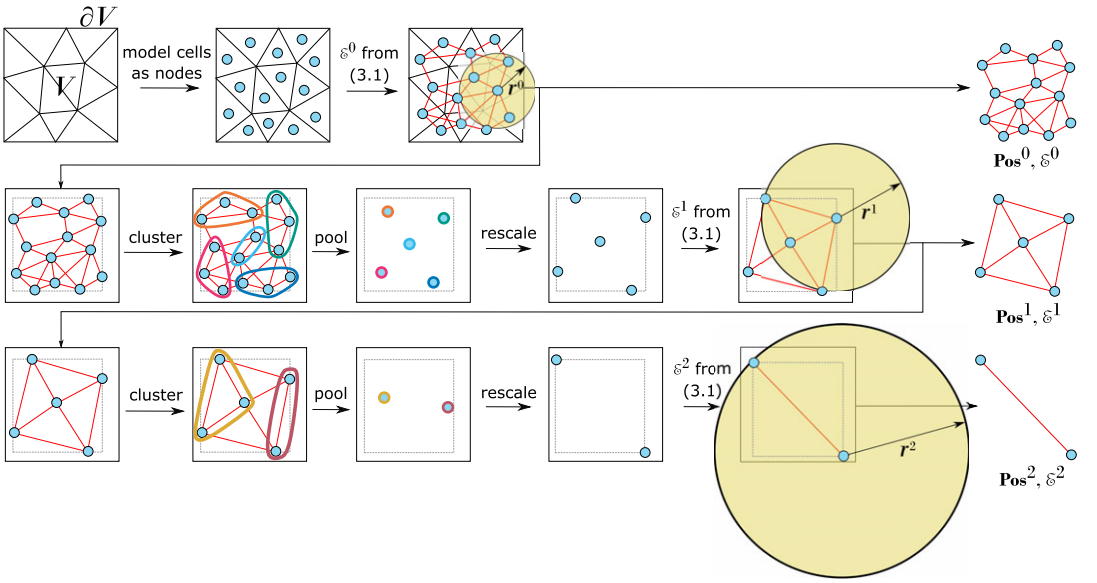

Figure 5. *A visual representation of Algorithm 1 for generating a hierarchy of reduced meshes. In this figure, it is assumed that the collocation points are cells in the FVM discretization, however the same principles apply for discretization used in various other methods in computational mechanics. The input mesh (top left, generated for a domain V and its boundary ∂V represented by the solid black box) is modeled as the layer "0" graph in which the collocation points of the discretized domain form the node set $\mathcal{V}^0$, and the edge set $\mathcal{E}^0$ (and subsequently the associated adjacency matrix $\mathbf{A}^0$) are determined from Eq. (3.1) by determining the nodes that fall within r^0 distance of each other. At layer i ($i = 0, \cdots, n_\ell - 2$), nodes are partitioned into clusters using a spectral clustering algorithm to achieve a dimensionally reduced graph from that of layer i. The nodal positions of the graph of layer $i+1$, obtained from the arithmetic mean position of node clusters from layer i, are rescaled to ensure that the maximum and minimum coordinates of the nodes in graph layer $i+1$ are equal to those of layer i, where the maximum and minimum values of the x and y coordinates are represented by the grey dotted box. Finally, the edge set of the reduced graph of layer $i+1$, that is, $\mathcal{E}^{i+1}$, is determined from Eq. (3.1).*

decoder does not influence the spectral clustering step. The graph autoencoder must use only the exact hierarchy of graph topologies seen during training, that is, if the original mesh is changed or if the hyperparameters of the hierarchy of reduced graphs are modified, the graph autoencoder must be retrained.

Algorithm 1: Hierarchical spectral clustering for graph reduction

Inputs: $\mathbf{Pos}^0, n_\ell, |\mathcal{V}^1|, \ldots, |\mathcal{V}^{n_\ell-1}|, r^0, \ldots, r^{n_\ell-1}$
Outputs: $\mathcal{E}^0, \ldots, \mathcal{E}^{n_\ell-1}, \mathbf{S}^0, \ldots, \mathbf{S}^{n_\ell-2}, \mathbf{Pos}^1, \ldots, \mathbf{Pos}^{n_\ell-1}$
Initialize $i \leftarrow 0$
Initialize $\mathcal{E}^0 \leftarrow \mathbf{Radius\_Graph}\left(\mathbf{Pos}^0, r^0\right)$;
while $i < n_\ell - 1$ **do**
1. Compute the adjacency matrix, $\mathbf{A}^i$, from $\mathcal{E}^i$
2. Compute the graph Laplacian, $\mathbf{L}^i \leftarrow \mathbf{D}^i - \mathbf{A}^i$
3. Form the spectral node feature matrix, $\mathbf{B}^i \in \mathbb{R}^{|\mathcal{V}^i| \times \left(|\mathcal{V}^{i+1}|-1\right)}$, with the eigenvectors associated with the $|\mathcal{V}^{i+1}| - 1$ smallest eigenvalues of $\mathbf{L}^i$ (excluding the smallest) as its columns;
4. Obtain node clusters $\mathcal{A}^i_1, \mathcal{A}^i_2, \ldots, \mathcal{A}^i_{|\mathcal{V}^{i+1}|}$ by performing $K-$means clustering on the spectral node features (i.e., the rows of $\mathbf{B}^i$)
5. Generate the assignment matrix, $\mathbf{S}^i \in \mathbb{R}^{|\mathcal{V}^{i+1}| \times |\mathcal{V}^i|}$, such that $\mathbf{S}^i_{jk} = \frac{1}{|\mathcal{A}^i_j|}$ if $k \in \mathcal{A}^i_j$, and $\mathbf{S}^i_{jk} = 0$, otherwise.
6. $\mathbf{Pos}^{i+1} \leftarrow \mathbf{Rescale}\left(\mathbf{S}^i \mathbf{Pos}^i\right)$
7. Compute the edge set for layer $i+1$ based on nearest neighbors within the radius r^{i+1}, $\mathcal{E}^{i+1} \leftarrow \mathbf{Radius\_Graph}\left(\mathbf{Pos}^{i+1}, r^{i+1}\right)$
8. $i \leftarrow i + 1$

end

3.2. Encoder architecture

The encoder architecture deploys the hierarchy of reduced graphs computed via the procedure from Section 3.1. The encoder consists of layers $i = 0, \ldots, n_\ell$. The zeroth layer ($i = 0$), outlined in Section 3.2.1, is a preprocessing layer that tailors the input data x to the form suited for the graph autoencoder. Layers $i = 1, \ldots, n_\ell - 1$, outlined in Section 3.2.2, leverage the hierarchy of reduced graphs from Section 3.1 to perform message passing and pooling (MPP) operations that reduce the dimension of the system. The final layer of the encoder ($i = n_\ell$), as outlined in Section 3.2.3, utilizes an MLP to arrive at the low-dimensional embedding $\hat{x}$.

3.2.1. Preprocessing – layer 0

The preprocessing layer of the encoder ($i = 0$) encompasses two operators, **Matricize** and **Scale**, acting on the input vector $\mathbf{x}$. For a FOM with n_q state variables, the **Matricize** operator is used to convert $\mathbf{x} \in \mathbb{R}^{n_q N_c}$ to the node feature matrix $\mathbf{X} \in \mathbb{R}^{N_c \times n_q}$ in which each column of the matrix represents the nodal values of one state variable. If the FOM consists of only one state variable (i.e., $n_q = 1$), $\mathbf{X} = \mathbf{x}$, and the **Matricize** operator will be the identity operator. As defined in Eq. (3.4), the **Scale** operator acts on the resulting node feature matrix to improve the numerical stability of training, as is commonly performed in the literature (Lee and Carlberg, 2020, 2021),

$$\mathbf{Scale} : \mathbf{X}^0_{ij} \mapsto \frac{\mathbf{X}^0_{ij} - \mathcal{X}^{min}_j}{\mathcal{X}^{max}_j - \mathcal{X}^{min}_j}, \quad i = 1, \ldots, N_c, j = 1, \ldots, n_q \tag{3.4}$$

where $\mathbf{Scale} : \mathbb{R} \rightarrow [0, 1]$ is an element-wise scaling operator acting on the elements of $\mathbf{X}$, and $\mathcal{X}^{max}_j$, $\mathcal{X}^{min}_j \in \mathbb{R}$ denote the maximum and minimum values, respectively, of the j^{th} feature (i.e., j^{th} column of

matrix $\mathbf{X}$) in the solution states used to train the autoencoder, which are determined and stored before training begins. The resulting form of the preprocessing layer is

$$\mathbf{h}_0 : (\mathbf{x}; \boldsymbol{\Theta}_0) \mapsto \mathbf{Scale}(\cdot) \circ \mathbf{Matricize}(\mathbf{x}), \tag{3.5}$$

where $\mathbf{h}_0 : \mathbb{R}^{N_c n_q} \to \mathbb{R}^{N_c \times n_q}$, and $\boldsymbol{\Theta}_0 = \emptyset$ is the empty set, as the preprocessing layer does not have trainable weights and biases.

3.2.2. Message passing and pooling (MPP) – layers 1, …, $n_\ell - 1$

The MPP layer consists of two processes, where each relies upon the hierarchy of reduced graphs computed in Section 3.1. The first operation is a message passing operation, wherein nodes connected by an edge exchange information with each other to obtain information about nearby nodes. The optimal information exchange is obtained from training the autoencoder. In the encoder, the message passing operation in layer i increases the number of features associated with each node from $N_F^{i-1} \in \mathbb{N}$ to $N_F^i \in \mathbb{N}$. We take our message passing operation to be a mean aggregation SAGEConv from Hamilton et al. (2017), which applies updates to each node based on the arithmetic mean of its neighbors' features, that is,

$$\mathbf{MP}_{enc}^i : \left(\mathbf{X}^{i-1}; \boldsymbol{\Theta}_i\right) \mapsto \sigma\left(\mathbf{X}_j^{i-1}\mathbf{W}_1^i + \left(\text{mean}_{n \in \mathcal{K}^{i-1}(j)} \mathbf{X}_n^{i-1}\right)\mathbf{W}_2^i\right), \quad j = 1, \ldots, |\mathcal{V}^{i-1}|, \tag{3.6}$$

with $\mathbf{MP}_{enc}^i : \mathbb{R}^{|\mathcal{V}^{i-1}| \times N_F^{i-1}} \times \mathbb{R}^{N_F^{i-1} \times N_F^i} \times \mathbb{R}^{N_F^{i-1} \times N_F^i} \to \mathbb{R}^{|\mathcal{V}^{i-1}| \times N_F^i}$, where $\mathbf{X}^{i-1} \in \mathbb{R}^{|\mathcal{V}^{i-1}| \times N_F^{i-1}}$ denotes the input node feature matrix to the i^{th} layer, the subscripts j and n denote the j^{th} and n^{th} rows of $\mathbf{X}^{i-1}$, $\mathbf{W}_1^i$, $\mathbf{W}_2^i \in \mathbb{R}^{N_F^{i-1} \times N_F^i}$ denote the weights with $\boldsymbol{\Theta}_i = \left\{\mathbf{W}_1^i, \mathbf{W}_2^i\right\}$ denoting the set of weights for the i^{th} MPP layer, $\mathcal{K}^{i-1}(j)$ denotes the set of nodes connected to node j based on the adjacency matrix $\mathbf{A}^{i-1}$, where $j \in \mathbb{N}$ denotes the j^{th} node in the graph at layer $i-1$, and $\sigma : \mathbb{R} \to \mathbb{R}$ denotes the element-wise activation function, chosen here to be the exponential linear unit (ELU) due to its continuously differentiable property (Clevert et al., 2016). The SAGEConv function described in Eq. (3.6) includes a loop over all nodes $j \in \mathcal{V}^{i-1}$, where for each node, the j^{th} row of the output $\overline{\mathbf{X}}^{i-1}$ of the message passing operation is calculated. The output of Eq. (3.6) has the same number of rows as its input, $\mathbf{X}^{i-1}$, but can have a different number of features (i.e., N_F^i is not necessarily equal to N_F^{i-1}).

The next step of the MPP layer is a pooling operation. In the pooling operation, the assignment matrices from Section 3.1 are used to reduce the number of nodes in a graph. By construction, the assignment matrices are equivalent to an arithmetic mean operation. As a result, we use them to compute the arithmetic mean feature vector of each cluster to get $\mathbf{X}^i$, that is,

$$\mathbf{Pool}^i : \left(\overline{\mathbf{X}}^{i-1}\right) \mapsto \mathbf{S}^{i-1}\overline{\mathbf{X}}^{i-1}, \tag{3.7}$$

with $\mathbf{Pool}^i : \mathbb{R}^{|\mathcal{V}^{i-1}| \times N_F^i} \to \mathbb{R}^{|\mathcal{V}^i| \times N_F^i}$, where $\overline{\mathbf{X}}^{i-1} \in \mathbb{R}^{|\mathcal{V}^{i-1}| \times N_F^i}$ denotes the output of the message passing operation, $\mathbf{MP}_{enc}^i$, and $\mathbf{S}^{i-1} \in \mathbb{R}^{|\mathcal{V}^i| \times |\mathcal{V}^{i-1}|}$ is the assignment matrix precomputed by the spectral clustering algorithm in Section 3.1. The full MPP layer takes the form,

$$\mathbf{h}_i : \left(\mathbf{X}^{i-1}; \boldsymbol{\Theta}_i\right) \mapsto \mathbf{Pool}^i(\cdot) \circ \mathbf{MP}_{enc}^i\left(\mathbf{X}^{i-1}; \boldsymbol{\Theta}_i\right), \tag{3.8}$$

with $\mathbf{h}_i : \mathbb{R}^{|\mathcal{V}^{i-1}| \times N_F^{i-1}} \times \mathbb{R}^{N_F^{i-1} \times N_F^i} \times \mathbb{R}^{N_F^{i-1} \times N_F^i} \to \mathbb{R}^{|\mathcal{V}^i| \times N_F^i}$. Hence, the MPP layer, as visually represented in Figure 6, decreases the number of nodes in a given graph and increases the number of features associated with each node.

3.2.3. Fully connected layer: compression – layer n_ℓ

In the final layer of the encoder ($i = n_\ell$), we first flatten the input matrix $\mathbf{X}^{n_\ell - 1} \in \mathbb{R}^{|\mathcal{V}^{n_\ell - 1}| \times N_F^{n_\ell - 1}}$ to a vector-representation, that is, $\mathbf{Flatten} : \mathbf{X}^{n_\ell - 1} \mapsto \overline{\mathbf{x}}^{n_\ell - 1}$, where $\overline{\mathbf{x}}^{n_\ell - 1} \in \mathbb{R}^{|\mathcal{V}^{n_\ell - 1}| N_F^{n_\ell - 1}}$. Here, we note that the **Flatten** operator is similar to the **Vectorize** operator, but with dimensions different than the node feature matrix of

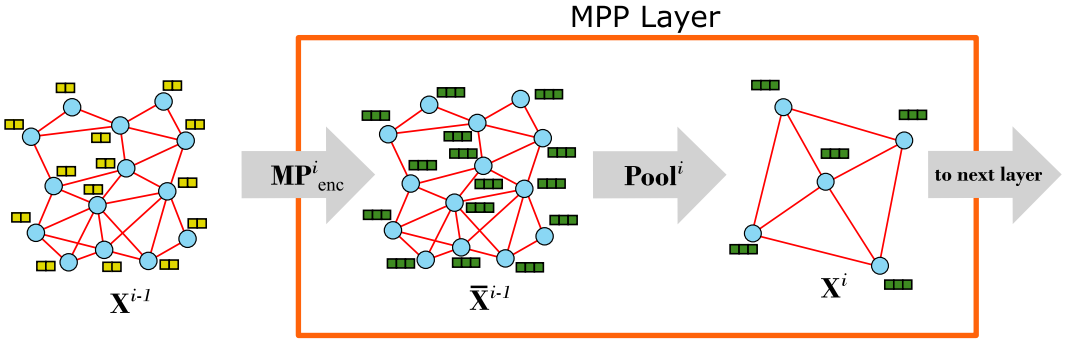

Figure 6. *MPP layer used in the encoder of our graph autoencoder. The layer accepts a graph as input and performs message passing to exchange information between locally connected nodes. Next, the graph nodes are pooled together based on their clusters from the hierarchical spectral clustering algorithm. This pooling operation reduces the number of nodes in the graph to perform dimensional compression.*

the full-order solution. Next, a fully connected/MLP layer is applied to the flattened state to compress it to a low-dimensional vector representation, that is,

$$\mathbf{MLP}_{enc} : \left(\overline{\mathbf{x}}^{n_\ell-1}; \mathbf{\Theta}_{n_\ell}\right) \mapsto \mathbf{W}^{n_\ell} \overline{\mathbf{x}}^{n_\ell-1}, \tag{3.9}$$

with $\mathbf{MLP}_{enc} : \mathbb{R}^{|\mathcal{V}^{n_\ell-1}|N_F^{n_\ell-1}} \times \mathbb{R}^{M \times |\mathcal{V}^{n_\ell-1}|N_F^{n_\ell-1}} \to \mathbb{R}^{M}$, where $\mathbf{W}^{n_\ell} \in \mathbb{R}^{M \times |\mathcal{V}^{n_\ell-1}|N_F^{n_\ell-1}}$ denote the weights with $\mathbf{\Theta}_{n_\ell} = \{\mathbf{W}^{n_\ell}\}$. Note that no activation function is applied to the output, as the inclusion of an activation function was empirically found to be prone to vanishing gradients during time integration when performed in the manner outlined in Section 4. Furthermore, no bias term is included, as it was empirically found to be unnecessary. The final layer of the encoder architecture takes the form

$$\mathbf{h}_{n_\ell} : \left(\mathbf{X}^{n_\ell-1}; \mathbf{\Theta}_{n_\ell}\right) \mapsto \mathbf{MLP}_{enc}(\cdot; \mathbf{\Theta}_{n_\ell}) \circ \mathbf{Flatten}\left(\mathbf{X}^{n_\ell-1}\right), \tag{3.10}$$

with $\mathbf{h}_{n_\ell} : \mathbb{R}^{|\mathcal{V}^{n_\ell-1}| \times N_F^{n_\ell-1}} \times \mathbb{R}^{M \times |\mathcal{V}^{n_\ell-1}|N_F^{n_\ell-1}} \to \mathbb{R}^{M}$. The output of this layer, $\hat{\mathbf{x}} \in \mathbb{R}^{M}$, is the low-dimensional latent representation of the solution state.

3.3. Decoder architecture

Much like the encoder, the decoder architecture deploys the hierarchy of reduced graphs from Section 3.1. The decoder consists of layers $i = 0, \ldots, n_\ell$. The zeroth layer ($i = 0$), outlined in Section 3.3.1, utilizes an MLP to reconstruct a small graph from the low-dimensional latent representation, $\hat{\mathbf{x}}$. Layers $i = 1, \ldots, n_\ell - 1$, outlined in Section 3.3.2, leverage the hierarchy of reduced graphs from Section 3.1 in reverse order to perform unpooling and message passing (UMP). The final layer of the decoder ($i = n_\ell$), outlined in Section 3.3.3, is a postprocessing layer that restructures the output graph into a state vector for deployment in the time integration scheme.

3.3.1. Fully connected layer: expansion – layer 0

The zeroth layer of the decoder ($i = 0$) entails two functions. It first applies a fully connected/MLP layer to the latent representation,

$$\mathbf{MLP}_{dec} : (\hat{\mathbf{x}}; \mathbf{\Omega}_0) \mapsto \sigma\left(\mathcal{W}^0 \hat{\mathbf{x}}\right), \tag{3.11}$$

with $\mathbf{MLP}_{dec} : \mathbb{R}^{M} \times \mathbb{R}^{|\mathcal{V}^{n_\ell-1}|N_F^{n_\ell-1} \times M} \to \mathbb{R}^{|\mathcal{V}^{n_\ell-1}|N_F^{n_\ell-1}}$, where $\mathcal{W}^0 \in \mathbb{R}^{|\mathcal{V}^{n_\ell-1}|N_F^{n_\ell-1} \times M}$ denote the weights of the MLP layer of the decoder, with $\mathbf{\Omega}_0 = \{\mathcal{W}^0\}$, and $\sigma : \mathbb{R} \to \mathbb{R}$ denotes the element-wise activation function. Additionally, note that, much like in the encoder, the fully connected/MLP layer does not have a bias, as it was empirically noticed to be unnecessary. An unflattening operator is then applied to the output of the fully connected layer, $\overline{\mathbf{y}}^0$, to generate a node feature matrix corresponding to the $n_\ell - 1$ graph in the

hierarchy of reduced graphs, $\mathbf{Unflatten}: \overline{\mathbf{y}}^0 \mapsto \mathbf{Y}^0$, with $\mathbf{Unflatten}: \mathbb{R}^{|\mathcal{V}^{n_\ell-1}|N_F^{n_\ell-1}} \to \mathbb{R}^{|\mathcal{V}^{n_\ell-1}| \times N_F^{n_\ell-1}}$, where $\overline{\mathbf{y}}^0 \in \mathbb{R}^{|\mathcal{V}^{n_\ell-1}|N_F^{n_\ell-1}}$ denotes the output of $\mathbf{MLP}_{dec}$ and $\mathbf{Y}^0 \in \mathbb{R}^{|\mathcal{V}^{n_\ell-1}| \times N_F^{n_\ell-1}}$. We note that the unflattening operator is similar to the **Matricize** operator introduced previously but applied to a vector with a size different from the full-order state vector. Ultimately, the first layer of the decoder takes the form,

$$\mathbf{g}_0 : (\hat{\mathbf{x}}; \mathbf{\Omega}_0) \mapsto \mathbf{Unflatten}(\cdot) \circ \mathbf{MLP}_{dec}(\hat{\mathbf{x}}; \mathbf{\Omega}_0), \tag{3.12}$$

where $\mathbf{g}_0 : \mathbb{R}^M \times \mathbb{R}^{|\mathcal{V}^{n_\ell-1}|N_F^{n_\ell-1} \times M} \to \mathbb{R}^{|\mathcal{V}^{n_\ell-1}| \times N_F^{n_\ell-1}}$.

3.3.2. Unpooling and message passing (UMP) – layers 1, …, $n_\ell - 1$

The next layers in the decoder architecture ($i = 1, \ldots, n_\ell - 1$) consist of UMP layers. The first step in a UMP layer is to perform an unpooling operation, wherein nodes are reintroduced to the graph, and their feature vectors are interpolated. In layer i of the decoder with $i = 1, \cdots, n_\ell - 1$, the unpooling operation receives the graph of layer $n_\ell - i$ as an input and outputs the graph of layer $n_\ell - i - 1$ in the hierarchy of $n_\ell - 1$ graphs of the encoder. For example, in Figure 4 with $n_\ell = 3$, the input and output to the second layer of the decoder ($i = 2$) are the graphs of 1st layer ($n_\ell - i = 1$) and the zeroth layer ($n_\ell - i - 1 = 0$) of the hierarchy of graphs in the encoder, respectively. For ease of notation, we introduce $\hat{i} = n_\ell - i$ as a counter used to denote the hierarchy of reduced graphs in the opposite order as the encoder. In the unpooling operation of layer i of the decoder, a node's features of graph $\hat{i}$ are interpolated using the k-nearest neighbors of the node features of graph $\hat{i} - 1$,

$$\mathbf{Unpool}^i : \mathbf{Y}^{i-1} \mapsto \frac{\sum_{n \in \mathcal{N}^{\hat{i}-1}(j)} \mathbf{w}\left(\mathbf{Pos}_j^{\hat{i}-1}, \mathbf{Pos}_n^{\hat{i}}\right) \mathbf{Y}_n^{i-1}}{\sum_{n \in \mathcal{N}^{\hat{i}-1}(j)} \mathbf{w}\left(\mathbf{Pos}_j^{\hat{i}-1}, \mathbf{Pos}_n^{\hat{i}}\right)}, \quad j = 1, \cdots, |\mathcal{V}^{\hat{i}-1}| \tag{3.13}$$

where,

$$\mathbf{w} : \left(\mathbf{Pos}_j^{\hat{i}-1}, \mathbf{Pos}_n^{\hat{i}}\right) \mapsto \frac{1}{\|\mathbf{Pos}_j^{\hat{i}-1} - \mathbf{Pos}_n^{\hat{i}}\|}, \tag{3.14}$$

with $\mathbf{Unpool}^i : \mathbb{R}^{|\mathcal{V}^{\hat{i}}| \times N_F^{\hat{i}}} \to \mathbb{R}^{|\mathcal{V}^{\hat{i}-1}| \times N_F^{\hat{i}}}$, $\mathcal{N}^{\hat{i}-1}(j)$ is the $k-$nearest neighbors in $\mathcal{V}^{\hat{i}}$ of the j^{th} node in $\mathcal{V}^{\hat{i}-1}$, with $k \in \mathbb{N}$ denoting the number of nearest neighbors used for interpolation. $\mathbf{Pos}_j^{\hat{i}-1} \in \mathbb{R}^{n_d}$ is the spatial position of the j^{th} node at the $(\hat{i}-1)^{\text{th}}$ layer of the hierarchy of reduced graphs, $\mathbf{Pos}_n^{\hat{i}} \in \mathbb{R}^{n_d}$ is the spatial position of the n^{th} node in the $\hat{i}^{\text{th}}$ layer in the hierarchy of reduced graphs, $\mathbf{w} : \mathbb{R}^{n_d} \times \mathbb{R}^{n_d} \to \mathbb{R}_+$ denotes the spatial interpolation function, and $\|\cdot\| : \mathbb{R}^{n_d} \to \mathbb{R}_+$ denotes the Euclidean norm. Much like the SAGEConv function Eq. (3.6), the unpooling of Eq. (3.13) is performed by looping over all nodes, $j \in \mathcal{V}^{\hat{i}-1}$, to compute the rows $j = 1, \cdots, |\mathcal{V}^{\hat{i}-1}|$ of the output of the unpooling operation, $\overline{\mathbf{Y}}^{i-1} \in \mathbb{R}^{|\mathcal{V}^{\hat{i}-1}| \times N_F^{\hat{i}}}$. Next, a message passing operation is applied to the outputs of the unpooling operation,

$$\mathbf{MP}_{dec}^i : \left(\overline{\mathbf{Y}}^{i-1}; \mathbf{\Omega}_i\right) \mapsto \sigma\left(\overline{\mathbf{Y}}_j^{i-1} W_1^i + \left(\text{mean}_{n \in \mathcal{K}^{\hat{i}-1}(j)} \overline{\mathbf{Y}}_n^{i-1}\right) W_2^i\right), \quad j = 1, \cdots, |\mathcal{V}^{\hat{i}-1}|, \tag{3.15}$$

with $\mathbf{MP}_{dec}^i : \mathbb{R}^{|\mathcal{V}^{\hat{i}-1}| \times N_F^{\hat{i}}} \times \mathbb{R}^{N_F^{\hat{i}} \times N_F^{\hat{i}-1}} \times \mathbb{R}^{N_F^{\hat{i}} \times N_F^{\hat{i}-1}} \to \mathbb{R}^{|\mathcal{V}^{\hat{i}-1}| \times N_F^{\hat{i}-1}}$, where $W_1^i, W_2^i \in \mathbb{R}^{N_F^{\hat{i}} \times N_F^{\hat{i}-1}}$ denote the weights with $\mathbf{\Omega}_i = \left\{W_1^i, W_2^i\right\}$ denoting the set of weights for the i^{th} UMP layer, $\mathcal{K}^{\hat{i}-1}(j)$ denotes the set of nodes connected to node j in the graph of $(\hat{i}-1)^{\text{th}}$ layer based on the adjacency matrix $\mathbf{A}^{\hat{i}-1}$, where the subscripts j and n denote the j^{th} and n^{th} nodes, respectively, and $\sigma : \mathbb{R} \to \mathbb{R}$ denotes the element-wise activation function. According to Eq. (3.15), the output of $\mathbf{MP}_{dec}^i$ is determined in a row-wise manner. Ultimately, the UMP layer takes the form,

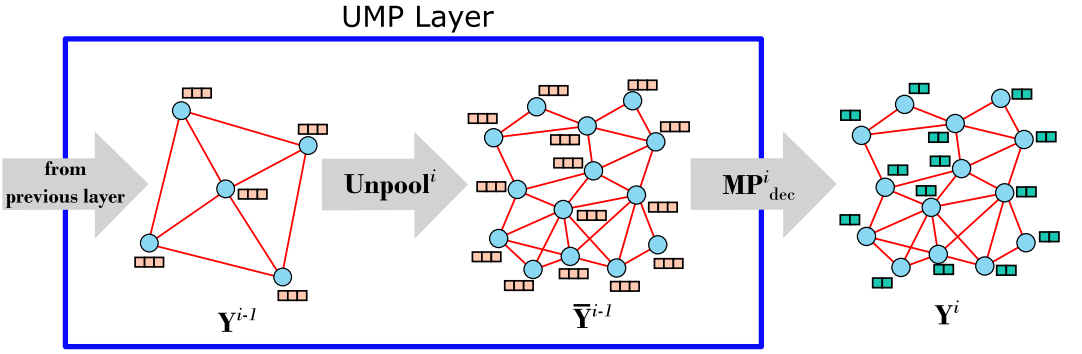

Figure 7. *UMP layer used in the decoder of our graph autoencoder. The layer accepts a graph as an input and performs an unpooling operation to re-introduce nodes into the graph, thus increasing the dimension of the graph. Next, message passing is performed on the unpooled graph to exchange information between locally connected nodes.*

$$\mathbf{g}_i : \left(\mathbf{Y}^{i-1}; \boldsymbol{\Omega}_i\right) \mapsto \mathbf{MP}^i_{dec}(\cdot; \boldsymbol{\Omega}_i) \circ \mathbf{Unpool}^i\left(\mathbf{Y}^{i-1}\right), \tag{3.16}$$

with $\mathbf{g}_i : \mathbb{R}^{|\mathcal{V}^{\hat{i}}| \times N_F^{\hat{i}}} \times \mathbb{R}^{N_F^{\hat{i}} \times N_F^{\hat{i}-1}} \times \mathbb{R}^{N_F^{\hat{i}} \times N_F^{\hat{i}-1}} \longrightarrow \mathbb{R}^{|\mathcal{V}^{\hat{i}-1}| \times N_F^{\hat{i}-1}}$. Hence, the UMP layer increases the number of nodes in a given graph and decreases the number of features associated with each node, and it gives the node feature matrix Y^i as the output. The UMP layer is visually represented in Figure 7.

3.3.3. Postprocessing – layern_ℓ

To represent the output of the decoder as a state vector for appropriate deployment in the time integration scheme, a postprocessing step is applied as the final layer ($i = n_\ell$). First, the **InvScale** operator is applied to invert the original **Scale** operation,

$$\mathbf{InvScale} : \mathbf{Y}^{n_\ell - 1}_{ij} \mapsto \mathbf{Y}^{n_\ell - 1}_{ij}\left(\mathcal{X}^{max}_j - \mathcal{X}^{min}_j\right) + \mathcal{X}^{min}_j, \tag{3.17}$$

where $\mathbf{InvScale} : \mathbb{R} \longrightarrow \mathbb{R}$ is an element-wise scaling operator. Next, a **Vectorize** operator is applied to reshape the output of the decoder to a state vector. Ultimately, the postprocessing step takes the form,

$$\mathbf{g}_{n_\ell} : \left(\mathbf{Y}^{n_\ell - 1}; \boldsymbol{\Omega}_{n_\ell}\right) \mapsto \mathbf{Vectorize}(\cdot) \circ \mathbf{InvScale}\left(\mathbf{Y}^{n_\ell - 1}\right), \tag{3.18}$$

where $\mathbf{g}_{n_\ell} : \mathbb{R}^{N_c \times n_q} \longrightarrow \mathbb{R}^{N_c n_q}$, and $\boldsymbol{\Omega}_{n_\ell} = \emptyset$ is the empty set, as there are no trainable parameters in the postprocessing layer. The output of the decoder $\tilde{\mathbf{x}} \in \mathbb{R}^N$ is a reconstruction of the original state vector $\mathbf{x}$.

3.4. Training the autoencoder

The components of the autoencoder that require training are the message passing operations and fully connected/MLP layers with $\theta = \{\boldsymbol{\Theta}_1, \boldsymbol{\Theta}_2, \cdots, \boldsymbol{\Theta}_{n_\ell}\}$, $\omega = \{\boldsymbol{\Omega}_0, \boldsymbol{\Omega}_1, \cdots, \boldsymbol{\Omega}_{n_{\ell-1}}\}$ as trainable parameters. To train these, we adopt the same loss function as Lee and Carlberg (2020, 2021), which is the L^2-norm of the reconstructed solution state,

$$\mathcal{L} = \sum_{i=1}^{N_{\text{train}}} \left\| \mathbf{x}^i - \mathbf{Dec}(\cdot) \circ \mathbf{Enc}\left(\mathbf{x}^i\right) \right\|_2^2, \tag{3.19}$$

where $\mathbf{x}^i \in \mathbb{R}^N$ is the i^{th} solution state in the training set and $N_{\text{train}} \in \mathbb{N}$ denotes the total number of training solution states generated by the FOM.

4. Projection scheme and interpretability

Time-stepping for ROMs using autoencoders has been achieved by a variety of methods, including training neural networks to compute time updates (Kim et al., 2019; Regazzoni et al., 2019; Fresca et al., 2021; Maulik et al., 2021; Fresca and Manzoni, 2022), identifying low-dimensional systems of ODEs (Fries et al., 2022; He et al., 2023), and projecting the semi-discretized equations of the FOM onto a nonlinear manifold (Lee and Carlberg, 2020, 2021; Chen et al., 2022; Kim et al., 2022). To examine the autoencoder's ability to embed the physics of the FOM, we choose to perform time integration using the time-discrete residual-minimizing LSPG projection (Carlberg et al., 2011, 2017; Lee and Carlberg, 2020, 2021). Specifically, GD-LSPG leverages the graph autoencoder to project the governing equations onto a low-dimensional latent space, thus performing time integration on the latent state variables.

4.1. Least-squares Petrov–Galerkin projection

In this section, we summarize the main points from the literature to provide sufficient background on the LSPG projection and how it is leveraged in this scope of work and direct the reader to Carlberg et al (2011, 2013, 2017) and Lee and Carlberg (2020, 2021)for further background and properties of the LSPG projection.

To illustrate, we set the initial conditions of the low-dimensional state vector to be the encoding of the initial conditions of the high-dimensional system, that is, $\hat{\mathbf{x}}(0;\boldsymbol{\mu}) = \mathbf{Enc}(\mathbf{x}(0;\boldsymbol{\mu}))$, and approximate the full-order state vector of the solution of the system, (Eq. 2.1), to be,

$$\tilde{\mathbf{x}}(t;\boldsymbol{\mu}) = \mathbf{Dec}(\hat{\mathbf{x}}(t;\boldsymbol{\mu})), \tag{4.1}$$

where $\tilde{\mathbf{x}} : \mathbb{R}_+ \times \mathcal{D} \to \mathbb{R}^N$ denotes the predicted solution state. As is performed across the literature (Carlberg et al., 2011; Lee and Carlberg, 2020, 2021), we next substitute Eq. (4.1) into Eq. (2.3) and project the residual onto a test basis, $\Psi \in \mathbb{R}^{N \times M}$, to prevent the system from becoming overdetermined. Ultimately, we arrive at the following minimization problem:

$$\hat{\mathbf{x}}(t;\boldsymbol{\mu}) = \underset{\hat{\boldsymbol{\xi}} \in \mathbb{R}^M}{\arg\min} \left\| \left(\Psi\left(\hat{\boldsymbol{\xi}};\boldsymbol{\mu}\right)\right)^T \mathbf{r}\left(\mathbf{Dec}\left(\hat{\boldsymbol{\xi}}(t;\boldsymbol{\mu})\right)\right) \right\|_2^2, \tag{4.2}$$

where $\hat{\boldsymbol{\xi}} \in \mathbb{R}^M$ is the sought-after low-dimensional solution state at time t. Under this approximation, the boundary conditions are not explicitly preserved. However, boundary conditions are promoted implicitly by the chosen time-discrete residual minimizing projection scheme. Taking the test basis, $\Psi : \mathbb{R}^M \times \mathcal{D} \to \mathbb{R}^{N \times M}$, to be,

$$\Psi : \left(\hat{\boldsymbol{\xi}};\boldsymbol{\mu}\right) \mapsto \left(\left.\frac{\partial \mathbf{r}}{\partial \mathbf{x}}\right|_{\mathbf{Dec}\left(\hat{\boldsymbol{\xi}}(t;\boldsymbol{\mu})\right)} \right) \left(\left.\frac{\mathrm{d}\mathbf{Dec}}{\mathrm{d}\hat{\boldsymbol{\xi}}}\right|_{\hat{\boldsymbol{\xi}}(t;\boldsymbol{\mu})} \right), \tag{4.3}$$

a nonlinear manifold LSPG projection is obtained (with $\left.\frac{\mathrm{d}\mathbf{Dec}}{\mathrm{d}\hat{\boldsymbol{\xi}}}\right|_{\hat{\boldsymbol{\xi}}(t;\boldsymbol{\mu})}$ denoting the Jacobian of the decoder), and an iterative Newton solver can be used to minimize (Eq. 4.2). Such a solver takes the form,

$$\left(\Psi\left(\hat{\mathbf{x}}^{n(j)};\boldsymbol{\mu}\right)\right)^T \Psi\left(\hat{\mathbf{x}}^{n(j)};\boldsymbol{\mu}\right)\left(\hat{\mathbf{x}}^{n(j+1)} - \hat{\mathbf{x}}^{n(j)}\right) = -\beta^{(j)} \left(\Psi\left(\hat{\mathbf{x}}^{n(j)};\boldsymbol{\mu}\right)\right)^T \mathbf{r}\left(\mathbf{Dec}\left(\hat{\mathbf{x}}^{n(j)}\right);\boldsymbol{\mu}\right), \tag{4.4}$$

where the superscript $n(j)$ denotes the j^{th} iteration of n^{th} time step, $\beta^{(j)} \in \mathbb{R}_+$ is the step size chosen to satisfy Wolfe conditions (Nocedal and Wright, 1999). An initial guess at each time step is chosen to be $\hat{\mathbf{x}}^{n(0)} = \hat{\mathbf{x}}^{n-1}$, where $\hat{\mathbf{x}}^{n-1}$ denotes the converged solution from the previous time step, $n-1$. The solution is updated iteratively until the L^2-norm of the reduced-state residual for the current iteration falls below a user-prescribed fraction of that of the initial guess at the time step $n = 1$ (which is the encoded initial condition, $\hat{\mathbf{x}}^0$),

$$\text{Convergence criterion} : \frac{\left\| \hat{\mathbf{r}}\left(\hat{\mathbf{x}}^{n(j)};\boldsymbol{\mu}\right)\right\|_2}{\left\| \hat{\mathbf{r}}\left(\hat{\mathbf{x}}^{1(0)};\boldsymbol{\mu}\right)\right\|_2} \leq \kappa, \tag{4.5}$$

where $\kappa \in [0,1]$ is the user-defined tolerance, and the reduced state residual $\hat{r}$ is obtained from the projection of the residual,

$$\hat{\mathbf{r}} : \left(\hat{\boldsymbol{\xi}};\boldsymbol{\mu}\right) \mapsto \left(\Psi\left(\hat{\boldsymbol{\xi}};\boldsymbol{\mu}\right)\right)^T \mathbf{r}\left(\mathbf{Dec}\left(\hat{\boldsymbol{\xi}}\right);\boldsymbol{\mu}\right). \tag{4.6}$$

It is important to note that LSPG projections of this form require knowledge of the FOM solver and direct access to the residual and its Jacobian, meaning GD-LSPG can only be performed for methods in computational mechanics where these information are available.

4.2. Interpretability of the latent state vector

In this section, we build from the LSPG projection to provide further insight into the interpretability of the proposed graph autoencoder. Although there is no unified definition for interpretability in scientific machine learning, in our study, we take it to mean, "the ability to explain or present in understandable terms to a human," (the definition from Doshi-Velez and Kim, (2017)). It is known that the latent space for autoencoders is commonly difficult to interpret due to the entanglement of the latent state variables (Eivazi et al., 2022; Kang et al., 2022). More broadly, in recent years, interpretability has emerged as a major focus area in the field of scientific machine learning (Baker et al., 2019).

Here, we demonstrate that the Jacobian of the decoder used in the nonlinear manifold LSPG projection scheme (Section 4.1) bears a direct analogy to the POD modes. To illustrate this relationship, consider the case where the decoder is an affine POD projection (see Appendix B),

$$\tilde{\mathbf{x}}(t;\boldsymbol{\mu}) = \mathbf{Dec}(\hat{\mathbf{x}}(t;\boldsymbol{\mu})) := \boldsymbol{\Phi}\hat{\mathbf{x}}(t;\boldsymbol{\mu}), \tag{4.7}$$

where $\boldsymbol{\Phi} \in \mathbb{R}^{N \times M}$ denotes the POD modes, it can be shown that the Jacobian of the decoder is simply the POD modes themselves, that is,

$$\left.\frac{\mathrm{d}\mathbf{Dec}}{\mathrm{d}\hat{\mathbf{x}}}\right|_{\hat{\mathbf{x}}(t;\boldsymbol{\mu})} = \boldsymbol{\Phi}. \tag{4.8}$$

In this case, substituting Eq. (4.8) into Eq. (4.3) yields a classical POD-LSPG projection. Building on this intuition, we interpret the Jacobian of the decoder in the graph autoencoder in the same manner. While the POD modes are time-invariant and independent of the latent state vector, $\hat{\mathbf{x}}(t;\boldsymbol{\mu})$, the Jacobian of the decoder of the graph autoencoder depends explicitly on $\hat{\mathbf{x}}(t;\boldsymbol{\mu})$, indicating that the corresponding modes evolve over time. Further discussion on the interpretability of the graph autoencoder is provided in Section 5 through numerical examples.

We can further relate our perspective for investigating interpretability to common strategies found in the literature, typically, classified as either *global* or *local* interpretability (Doshi-Velez and Kim, 2017). Local interpretability seeks to identify the reason for a specific prediction, whereas global interpretability seeks to identify the trends in a model's behavior for all predictions. Through this lens, analyzing the Jacobian of the decoder for the graph autoencoder can be viewed as a form of local interpretability. In particular, this approach closely parallels saliency maps (Simonyan et al., 2013), originally developed for image classification problems. Saliency maps aim to find the features in the input that are most predictive of the output by employing a first-order Taylor expansion.

5. Numerical experiments

We evaluate the efficiency and accuracy of the GD-LSPG through a series of test problems. First, to provide a baseline for comparison to the rest of the literature, we use a commonly studied 1D Burgers' model using a structured mesh (Rewieński, 2003; Lee and Carlberg, 2020, 2021; Barnett et al., 2023). This allows us to benchmark the accuracy of GD-LSPG with PMOR methods that deploy CNN-based autoencoders. Second, we deploy GD-LSPG to a model solving the 2D Euler equations for two different settings. The first setting uses an unstructured mesh to solve a setup for the Riemann problem (Kurganov

and Tadmor, 2002; Liska and Wendroff, 2003). Despite using an unstructured mesh, the domain is square and regular, meaning we can interpolate the unstructured mesh solution onto a structured mesh solution that can be deployed to a CNN-based autoencoder. In this setting, we demonstrate that the graph autoencoder generalizes better than interpolating to a structured mesh and using a CNN-based autoencoder. The second setting demonstrates the versatility of GD-LSPG for a problem with a more complicated geometry. In the second setting, we further evaluate the ability of the graph autoencoder when presented with noisy training data. All autoencoders are trained with PyTorch (Paszke et al., 2019) and PyTorch-Geometric (Fey and Lenssen, 2019). A detailed description of the employed autoencoder architectures and choice of hyperparameters for all examples are provided in Appendix A. To train the models and therefore minimize (Eq. 3.19), the Adam optimizer (Kingma and Ba, 2014) is deployed to perform stochastic gradient descent with an adaptive learning rate. In this study, we use reconstruction and state prediction errors as the primary performance metrics to assess accuracy. The reconstruction error is used to assess the ROM's ability to reconstruct a precomputed full-order solution, and it is defined as

$$\text{autoencoder reconstruction error} = \frac{\sqrt{\sum_{n=1}^{N_t} \|\mathbf{x}^n(\boldsymbol{\mu}) - \mathbf{Dec} \circ \mathbf{Enc}(\mathbf{x}^n(\boldsymbol{\mu}))\|_2^2}}{\sqrt{\sum_{n=1}^{N_t} \|\mathbf{x}^n(\boldsymbol{\mu})\|_2^2}}, \tag{5.1}$$

where $\mathbf{x}^n(\boldsymbol{\mu})$ is the full-order solution at the n^{th} time step. On the other hand, the POD reconstruction error is evaluated from

$$\text{POD reconstruction error} = \frac{\sqrt{\sum_{n=1}^{N_t} \left\| \left(\mathbf{I} - \boldsymbol{\Phi}\boldsymbol{\Phi}^T\right)\mathbf{x}^n(\boldsymbol{\mu}) \right\|_2^2}}{\sqrt{\sum_{n=1}^{N_t} \|\mathbf{x}^n(\boldsymbol{\mu})\|_2^2}}, \tag{5.2}$$

where $\boldsymbol{\Phi} \in \mathbb{R}^{N \times M}$ is the matrix of reduced basis vectors from an affine POD approximation constructed based on the method of snapshots (see Appendix B).

The state prediction error is used to assess the accuracy of the ROM obtained from different methods in predicting the full-order solution,

$$\text{state prediction error} = \frac{\sqrt{\sum_{n=1}^{N_t} \|\mathbf{x}^n(\boldsymbol{\mu}) - \tilde{\mathbf{x}}^n(\boldsymbol{\mu})\|_2^2}}{\sqrt{\sum_{n=1}^{N_t} \|\mathbf{x}^n(\boldsymbol{\mu})\|_2^2}}. \tag{5.3}$$

In all numerical examples, both dLSPG and GD-LSPG employ functorch's automatic differentiation (He and Zou, 2021) to obtain the Jacobian of the decoder.

5.1. One-dimensional Burgers' equation

To benchmark GD-LSPG with dLSPG and POD-LSPG, we use a 1D Burgers' model on a structured mesh. Due to its close relationship to the Navier–Stokes equations (Chan et al., 2010) and advection-driven behavior, the 1D Burgers' model is commonly chosen as a test case in the literature (Lee and Carlberg, 2020, 2021; Chen et al., 2022; Geelen and Willcox, 2022; Barnett et al., 2023). In this study, we choose the numerical experiment originally found in Rewieński (2003) with the added parameterization from Lee and Carlberg (2020). The governing equation is,

$$\begin{aligned} \frac{\partial w(x,t;\boldsymbol{\mu})}{\partial t} + \frac{\partial f(w(x,t;\boldsymbol{\mu}))}{\partial x} &= 0.02e^{\mu_2 x}, \quad \forall x \in (0,L), \forall t \in (0,T], \\ w(0,t;\boldsymbol{\mu}) &= \mu_1, \quad \forall t \in (0,T], \\ w(x,0;\boldsymbol{\mu}) &= 1, \quad \forall x \in (0,L), \end{aligned} \tag{5.4}$$

where $f(w) = 0.5w^2$, $x \in \mathbb{R}$ denotes spatial position, $t \in \mathbb{R}_+$ denotes time, $L \in \mathbb{R}$ denotes the length of the 1D physical domain, and $T \in \mathbb{R}_+$ denotes the final time. We utilize the FVM by dividing the spatial domain into 256 equally sized cells over a domain of length $L = 100$, lending to a structured finite volume

mesh. A backward-Euler time-integration scheme is employed, which corresponds to the cell-wise equations at the i^{th} cell,

$$\mathbf{f}_i : (\boldsymbol{\xi}, t; \boldsymbol{\mu}) \mapsto \frac{1}{2\Delta x}\left(\left(w_{i-1}^{n+1}\right)^2 - \left(w_i^{n+1}\right)^2\right) - 0.02e^{\mu_2 x_i}, \tag{5.5}$$

with $\alpha_0 = 1$, $\alpha_1 = -1$, $\beta_0 = -\Delta t$, $\beta_1 = 0$, and $\tau = 1$ when written in the form of Eq. (2.2). Note that since Burgers' equation involves only one state variable, that is, is a scalar variable. In Eq. (5.5), $\Delta x \in \mathbb{R}_+$ is the length of each cell in the uniform 1D mesh, $x_i \in \mathbb{R}$ is the coordinate f the center of the i^{th} cell in the mesh, and $\boldsymbol{\xi} = \left(w_1^{n+1}, w_2^{n+1}, \ldots, w_{N_c}^{n+1}\right)^T$ is the sought-after state solution at the $(n+1)^{\text{th}}$ time step. The time integration scheme uses a constant time step size $\Delta t = .07$ and a final time $T = 35$ for a total of 501 time steps per solution including the initial value. To train both the CNN-based autoencoder and the graph autoencoder, as well as obtain an affine POD basis, the solution to the FOM is computed for a total of 80 parameter scenarios with the parameters $\boldsymbol{\mu} = \left(\mu_1 = 4.25 + \left(\frac{1.25}{9}\right)\mathrm{i}, \mu_2 = .015 + \left(\frac{.015}{7}\right)\mathrm{j}\right)$, for $\mathrm{i} = 0, \ldots, 9$ and $\mathrm{j} = 0, \ldots, 7$. Once trained, the autoencoders are deployed in an online setting to perform time integration for their respective ROMs.

To generate the POD-LSPG solution, we set the tolerance κ in Eq. (4.5) to be 10^{-4}, whereas, the tolerance is set to 10^{-3} for the CNN-based dLSPG and GD-LSPG. The step size, $\beta^{(j)}$, for POD-LSPG is set to 1.0. Alternatively, in dLSPG and GD-LSPG, an adaptive step size strategy is adopted. For dLSPG, we begin with a step size of 1.0 and reduce by 5% every five iterations that convergence is not achieved. Likewise, in GD-LSPG, we begin with a step size of 0.5 and reduce by 10% every 10 iterations if convergence is not achieved.

Figure 8 depicts the solution state at various time steps for two test parameter set realizations not seen in the training set and two latent space dimensions of $M = 3, 10$. Additionally, the POD reconstruction errors

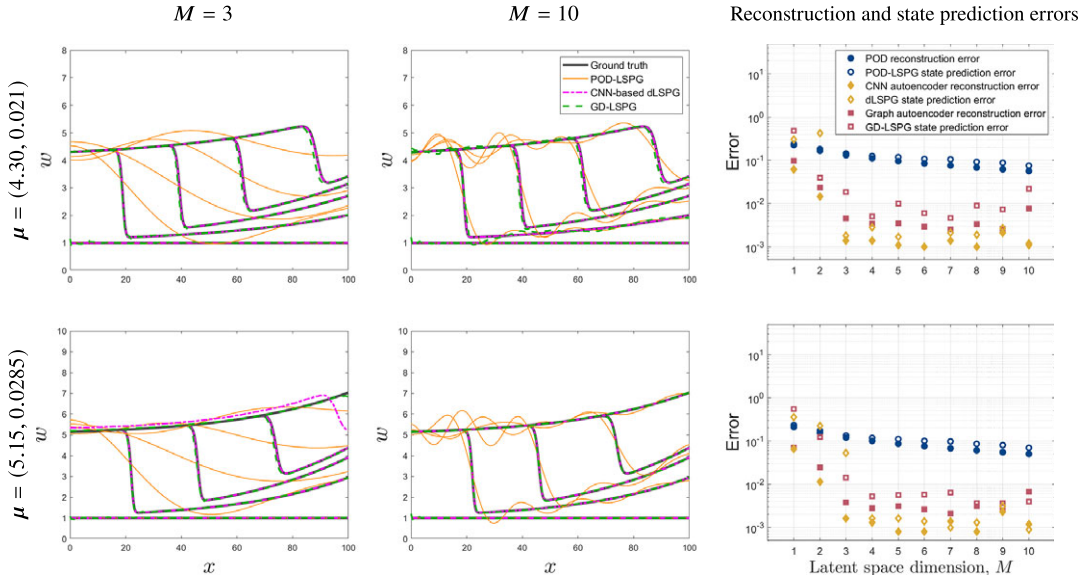

Figure 8. *The left two columns represent the state solution for Burgers' model Eq. (5.4) for time steps* $t = 0, 7, 14, 21,$ *and* 28 *(ordered from left to right) for two latent space dimensions (*$M = 3, 10$*, respectively), while the right column depicts the error metrics from Eq. (5.1) to Eq. (5.3) for various PMOR methods. Figures in the first row correspond to test parameters* $\boldsymbol{\mu} = (\mu_1 = 4.30, \mu_2 = 0.021)$*, while figures in the second row correspond to test parameters* $\boldsymbol{\mu} = (\mu_1 = 5.15, \mu_2 = 0.0285)$*. GD-LSPG and dLSPG both outperform POD-LSPG in predicting the highly nonlinear behavior of the Burgers' equation.*

from Eq. (5.2), and autoencoder reconstruction errors from Eq. (5.1) for the CNN-based autoencoder, inspired by Lee and Carlberg (2020, 2021), and the reconstruction errors from the graph autoencoder can be found in Figure 8. Using the state prediction error of Eq. (5.3), we also compare the performance of GD-LSPG to that of the traditional affine POD-LSPG (Carlberg et al., 2011, 2017), as well as dLSPG, which leverages a CNN-based autoencoder (Lee and Carlberg, 2020, 2021). We emphasize that the reconstruction errors for the graph autoencoder are more than an order of magnitude smaller than that of the affine POD approximation for latent space dimensions 3–10. Likewise, the state prediction errors of GD-LSPG are roughly an order of magnitude lower than that of POD-LSPG for latent space dimensions 3–10. This outcome is due to the fact that an affine subspace is not well-suited for such nonlinear problems. Benchmarking the graph autoencoder with the traditional CNN-based autoencoder, we find that the graph autoencoder's reconstruction errors and state prediction errors to be less than an order of magnitude greater than those of the CNN-based autoencoder for the vast majority of latent space dimensions. This comparison implies that, while GD-LSPG gains adaptability and is applicable to unstructured meshes, it does not perform as well as CNN-based dLSPG for the Burgers' model with a structured mesh. However, as noticed from the solution states provided in Figure 8, GD-LSPG is able to model the advection-dominated shock behavior in a manner similar to traditional CNN-based dLSPG, where traditional affine POD-LSPG tends to fail. We note that for $\boldsymbol{\mu} = (\mu_1 = 5.15, \mu_2 = 0.0285)$ the dLSPG solution using a latent space dimension $M = 3$ exhibits an erroneous solution when the shock approaches the right side of the domain, which persistently occurred for different adaptive step size strategies. It is worth noting that high errors were reported in Lee and Carlberg (2020) for the same latent space dimension without explicit elaboration on the main cause of such high error. However, for latent space dimensions 4–10, dLSPG outperforms GD-LSPG in terms of accuracy. Additionally, the errant solution was not noticed for the parameter set $\boldsymbol{\mu} = (\mu_1 = 4.30, \mu_2 = 0.021)$.

Next, we analyze the interpretability of the latent state vector through the Jacobian of the decoder (as presented in Section 4.2). We reiterate that the Jacobian of the decoder for the graph autoencoder can be interpreted in the same manner as the POD modes from a classical POD-LSPG scheme. Figure 9 presents the Jacobian of the decoder for the GD-LSPG solution to the test parameters $\boldsymbol{\mu} = (\mu_1 = 4.30, \mu_2 = 0.021)$ for $M = 3$ as well as the POD modes used to obtain the POD-LSPG solution. Whereas the POD modes remain fixed for all time steps, the Jacobian of the decoder for the graph autoencoder demonstrates highly interpretable time-varying mode shapes. It is evident that the latent state variables from the graph autoencoder capture information about the moving shock boundary.

Figure 10 depicts the difference between the ROM prediction of the full-order state vector and the FOM results with space and time. It can be seen that both dLSPG and GD-LSPG provide an improved ability to model the shock behavior of Eq. (5.4) over POD-LSPG. Additionally, we once again see that the dLSPG solution for $\boldsymbol{\mu} = (\mu_1 = 5.15, \mu_2 = 0.0285)$ at latent space dimension $M = 3$ struggles to converge when the shock approaches the right side of the domain. Finally, it is apparent that the main source of error for GD-LSPG is a slight phase lag between the ground truth location of the shock and GD-LSPG's prediction of the shock location. Hence, on a structured mesh, GD-LSPG provides an improvement over traditional affine POD-LSPG (Carlberg et al., 2011, 2013) in a manner comparable to that of dLSPG (Lee and Carlberg, 2020, 2021; Kim et al., 2022).

To assess the computational cost of the GD-LSPG method and compare it to POD-LSPG and dLSPG, we provide an analysis of the 1D Burgers' model. All operations in this section are performed in PyTorch using a single Intel® Xeon® Platinum 8358 CPU @ 2.60GHz ICE LAKE core. We generate the solution five times using each ROM and report the average time for each component of the ROM procedure (normalized by the average time of the FOM) in Figure 11. We present the time to get $\mathbf{r}^{n(k)}$ from Eq. (2.2) and evaluate its Jacobian, the time to get the Jacobian of the decoder, the time to check the convergence criterion (Eqs. 4.5–4.6), the time to decode to the high-dimensional space (Eq. 4.1), the time to compute Ψ, $\Psi^T\Psi$, and $\Psi^T\mathbf{r}^{n(k)}$, and the time to update the low-dimensional solution state (Eq. 4.4). The most time-consuming components of dLSPG and GD-LSPG are the time associated with computing the Jacobian of the decoder (which is not needed for the POD-LSPG approach) and the time associated with computing the high-dimensional residual and its Jacobian.

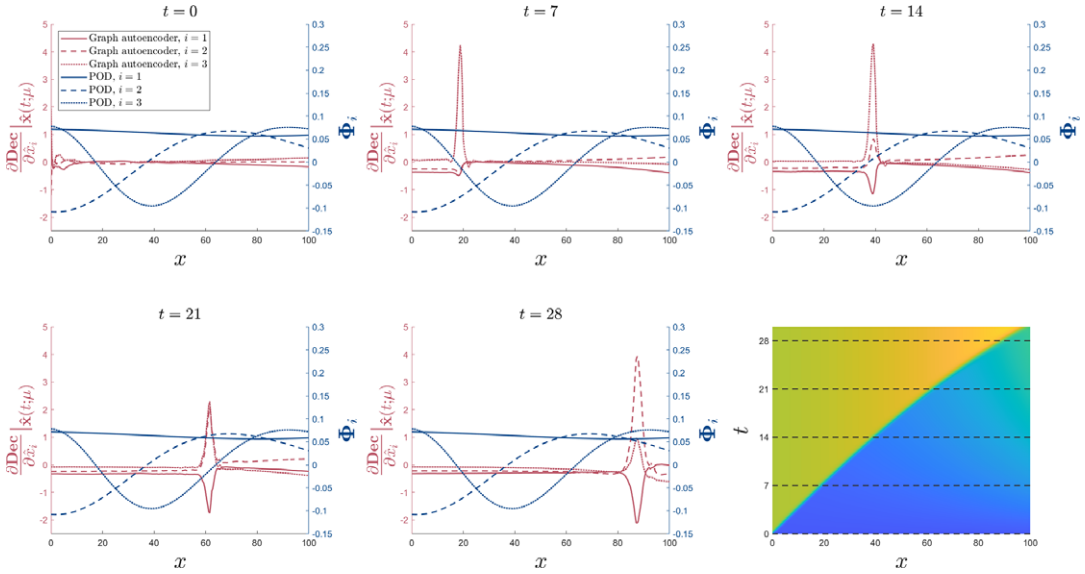

Figure 9. *Jacobian of the decoder of GD-LSPG with respect to the i^{th} latent state variable, $\frac{\partial \text{Dec}}{\partial \tilde{x}_i}\Big|_{\hat{\mathbf{x}}(t;\boldsymbol{\mu})}$, is shown alongside the POD modes, Φ_i, for a latent space dimension $M = 3$. Results are presented at time instances $t = 0, 7, 14, 21,$ and 28, for the test parameter set $\boldsymbol{\mu} = (\mu_1 = 4.30, \mu_2 = 0.021)$. Unlike the time-invariant and highly diffusive POD modes, the Jacobian of the decoder for the graph autoencoder captures critical information about the location of the moving shock boundary. In all subplots (except for the bottom right), the left axis corresponds to the Jacobian of the decoder for latent space variable i, while the right axis corresponds to the the i^{th} POD mode. All subplots (except for the bottom right) share the same legend as the one shown for $t = 0$. The bottom right figure displays the full-order solution with time and space, in which the horizontal lines highlight the selected time instances to highlight the match between the location of the moving shock boundary and the identified features by the Jacobian of the decoder.*

As noted by Farhat et al (2015), cost savings in ROMs employing LSPG projection can be achieved under three primary scenarios. In the first scenario, although setting up the linear system in Eq. (4.4) has an operation count that scales with the dimension of the FOM, cost savings can still be achieved if the computational cost of the FOM is dominated by the cost of inverting the Jacobian of the time-discrete residual, and if the cost to set up and minimize Eq. (4.2) is considerably lower than that of setting up and minimizing Eq. (2.3). In the second scenario, the ROM is sufficiently stable to be solved with a much larger time step than the FOM, thereby requiring fewer evaluations of Eq. (4.4) than Eq. (2.3). In the third (and most common) scenario, a hyper-reduction scheme is employed to sparsely sample terms in the residual to approximate the minimization of Eq. (4.2) with substantially fewer computations. POD-LSPG achieves approximately 20% cost savings for the latent space dimension $M = 3$, primarily due to satisfying the first scenario. However, this comes at the expense of reduced solution accuracy. Since dLSPG and GD-LSPG do not fall under any of the identified scenarios for cost savings, they do not achieve cost savings with respect to the FOM in this setting. However, the primary goal of this study is not computational efficiency, but rather to assess the extent of dimensionality reduction capabilities of GD-LSPG while accurately capturing the advection-dominated nonlinear behavior inherent in the numerical examples.

5.2. Two-dimensional Euler equations

In our second and third numerical experiments, we consider the FVM deployed to solve the two-dimensional Euler equations in two distinct settings. In our first setting, a Riemann problem setup for

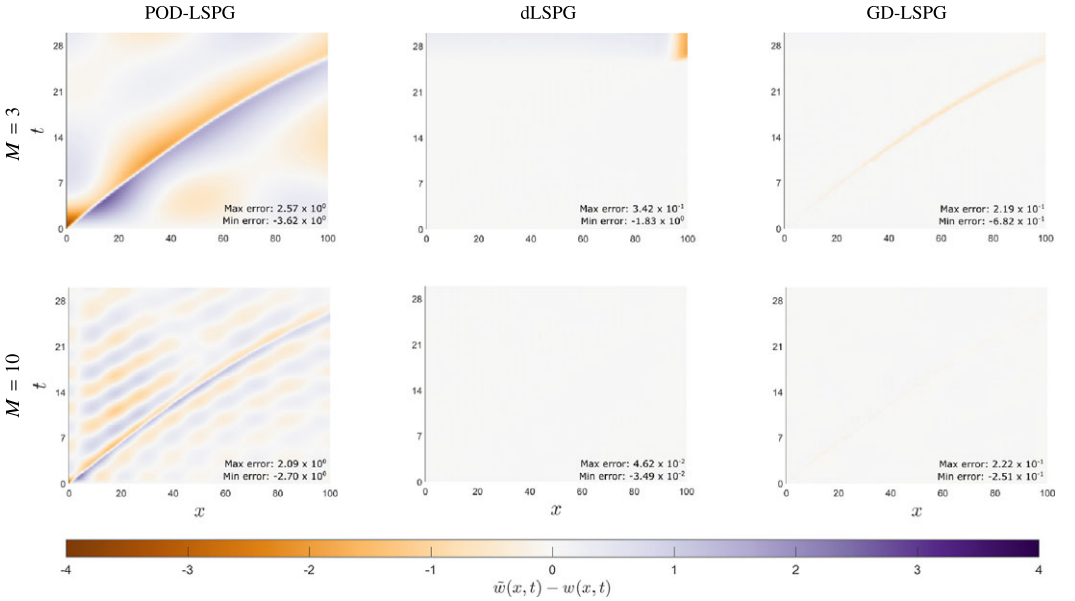

Figure 10. *Local error for dLSPG, GD-LSPG, and POD-LSPG for the parameter set* $\boldsymbol{\mu} = (\mu_1 = 5.15, \mu_2 = 0.0285)$. *The top and bottom rows correspond to solutions generated with latent space dimensions* $M = 3$ *and* $M = 10$, *respectively. The local error is simply taken to be* $\tilde{w}(x,t) - w(x,t)$, *or the difference between the predicted solution state and the ground truth solution state. The POD-LSPG solution introduces considerable error throughout the domain. Alternatively, dLSPG and GD-LSPG introduce lower-order localized errors, primarily around the shock. The slight phase difference between shocks in the predicted solution and the ground truth solution is the main error contributor.*

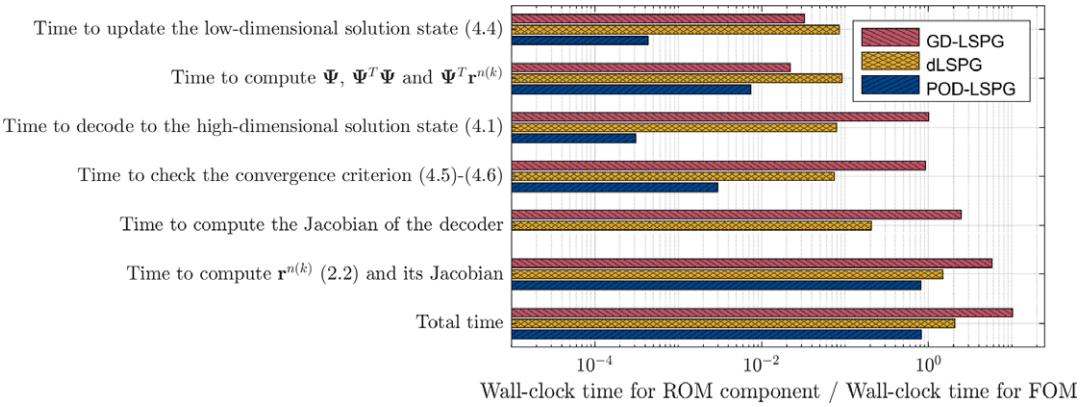

Figure 11. *Wall-clock time breakdown of individual components of GD-LSPG, dLSPG, and POD-LSPG with a latent space dimension* $M = 3$ *normalized to the wall-clock time associated with the FOM solution. Solutions were generated for the parameter set* $\boldsymbol{\mu} = (\mu_1 = 4.30, \mu_2 = 0.021)$. *The POD-LSPG approach does not have to compute the Jacobian of the decoder at each iteration. The breakdown reveals the most expensive components of the dLSPG and GD-LSPG methods include the time to get the high-dimensional residual, Jacobian of the high-dimensional residual, and Jacobian of the decoder. The wall-clock time for the FOM was 38.63 seconds.*

a square domain is solved using an unstructured mesh, which allows us to establish the benefits of our graph autoencoder architecture. In the second setting, we demonstrate the flexibility of our graph autoencoder architecture by applying it to a bow shock generated by flow past a cylinder problem on an irregular domain modeled with an unstructured mesh. To introduce the FVM solver used for these two experiments, we provide a brief overview of the important concepts in this section and encourage the reader to consult Roe (1981), Ren (2003), Nishikawa and Kitamura (2008), and Paardekooper (2017) for further reading on Riemann solvers for the Euler equations. We begin with the two-dimensional Euler equations in the form of hyperbolic PDEs:

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}}{\partial x} + \frac{\partial \mathbf{G}}{\partial y} = \mathbf{0}, \tag{5.6}$$

$$\mathbf{U} = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho E \end{bmatrix}, \quad \mathbf{F} = \begin{bmatrix} \rho u \\ \rho u^2 + P \\ \rho uv \\ \rho uH \end{bmatrix}, \quad \mathbf{G} = \begin{bmatrix} \rho v \\ \rho uv \\ \rho v^2 + P \\ \rho vH \end{bmatrix}, \tag{5.7}$$

where $\rho \in \mathbb{R}_+$ denotes density, $u \in \mathbb{R}$ and $v \in \mathbb{R}$ denote velocities in the x and y directions, respectively, $P \in \mathbb{R}_+$ denotes pressure, $E = \frac{1}{\gamma - 1}\frac{p}{\rho} + \frac{1}{2}(u^2 + v^2) \in \mathbb{R}_+$ and $H = \frac{\gamma}{\gamma - 1}\frac{p}{\rho} + \frac{1}{2}(u^2 + v^2) \in \mathbb{R}_+$ denote specific total energy and enthalpy, respectively, and $\gamma \in \mathbb{R}_+$ is the specific heat ratio. We integrate Eq. (5.6) over a control volume, Γ, and apply the divergence theorem to get the form,

$$\int_\Gamma \frac{\mathrm{d}}{dt}\mathbf{U} dV + \int_{\partial\Gamma} \mathbf{H} \cdot \hat{\mathbf{n}} dA = \mathbf{0}, \tag{5.8}$$

where $dA \in \mathbb{R}_+$, $dV \in \mathbb{R}_+$ and $\partial\Gamma$ denote the differential surface area, the differential volume, and the surface of the control volume, respectively, $\mathbf{H} = \mathbf{F}\hat{i} + \mathbf{G}\hat{j}$, $\hat{\mathbf{n}} = n_x\hat{i} + n_y\hat{j}$ denotes the outward facing unit normal vector from the control volume, where $\hat{i}$ and $\hat{j}$ denote the Cartesian unit vectors in x and y directions, respectively, and $n_x \in [-1, 1]$ and $n_y \in [-1, 1]$ denote the components of $\hat{n}$ decomposed in the x and y directions.

An approximate solution for Eq. (5.8) is achieved by first spatially discretizing the domain, where the surface integral term is approximated by obtaining the numerical flux passing over the cell faces in the unstructured mesh. The numerical flux is computed using a Riemann solver designed to resolve the computationally difficult nature of the hyperbolic Euler equations. In this numerical experiment, we choose a Rotated Roe, Harten, Lax, and van Leer (R-RHLL) flux from Nishikawa and Kitamura (2008) coupled with a forward Euler time integration scheme to generate a time series solution. The resulting scheme for a single finite volume cell takes the form

$$\mathbf{U}_i^{n+1} = \mathbf{U}_i^n - \Delta t \sum_{j \in m(i)} \mathbf{\Pi}_{ij}\left(\mathbf{U}_i^n, \ \mathbf{U}_j^n\right), \tag{5.9}$$

where $\mathbf{U}_i^n, \mathbf{U}_j^n \in \mathbb{R}^4$ denote the state vector of the i^{th} and j^{th} cells at the n^{th} time step, respectively, $m(i)$ denotes the set of neighboring cells of the i^{th} cell (i.e., sharing an interface), $\mathbf{\Pi}_{ij} : \mathbb{R}^4 \times \mathbb{R}^4 \to \mathbb{R}^4$ denotes the function that computes the R-RHLL flux at the interface between the i^{th}–j^{th} cells (Nishikawa and Kitamura, 2008). Therefore, Eq. (5.9) can be written in the residual-minimization cell-wise form of Eq. (2.2) at the i^{th} cell with

$$\mathbf{f}_i : (\mathbf{U}^n, t^n; \boldsymbol{\mu}) \mapsto \sum_{j \in \mathcal{M}(i)} \mathbf{\Pi}_{ij}\left(\mathbf{U}_i^n, \ \mathbf{U}_j^n\right), \tag{5.10}$$

and $\alpha_0 = 1$, $\alpha_1 = -1$, $\beta_0 = 0$, $\beta_1 = \Delta t$, $\tau = 1$, and $\boldsymbol{\xi} = \left(\mathbf{U}_1^{n+1}, \mathbf{U}_2^{n+1}, \ldots, \mathbf{U}_{N_c}^{n+1}\right)^T$ when written in the form of Eq. (2.2). We note that the minimization problem associated with the residual of the FOM (Eq. 5.9) is solved explicitly using the forward Euler scheme. Consequently, the POD-LSPG solution is equivalent to the POD-Galerkin solution (see Theorem 4.2 in Lee and Carlberg (2020)). Additionally, when the forward

Euler scheme is deployed in GD-LSPG, minimizing the projection of the residual onto the low-dimensional latent space according to Eq. (4.6) is performed implicitly using the iterative solver described in Section 4. Our implementation of the R-RHLL flux uses a triangular mesh generated by Gmsh (Geuzaine and Remacle, 2009)and Numba's just-in-time compiler (Lam et al., 2015) to compile the code efficiently.

For the problems studied in this section, the ROMs are built to reconstruct the conserved state variables $(\rho, \rho u, \rho v, \rho E)$, as it allows for natural integration of Eqs. (5.6–5.10) into the projection scheme outlined in Section 4. Consequently, the reported reconstruction and state prediction errors are evaluated with respect to the conserved state variables. To compare the ROM solutions with those of FOM, we focus on pressure and density fields where the latter is shown by contours.

5.2.1. Riemann problem

Our first setting in which we deploy GD-LSPG for the 2D Euler equations is a Riemann problem setup. We solve Eqs. (5.6–5.7) on the domain $x \in [0,1]$, $y \in [0,1]$ with outflow boundary conditions that are computed via the fluxes of the cells along each boundary. The initial conditions are defined by dividing the domain into quadrants, where a different state is defined in each quadrant (see Figure 12a). In this experiment, we define the quadrants as a parameterized version of configuration G from Schulz-Rinne et al (1993) (or configuration 15 from Lax and Liu (1998), Kurganov and Tadmor (2002), and Liska and Wendroff (2003)), that is,

$$\begin{aligned} \rho_1 = 1.0, \quad \rho_3 = 0.8, \\ u_1 = u_3 = u_4 = \mu_u, \\ v_1 = v_2 = v_3 = \mu_v, \\ P_1 = 1.0, \quad P_2 = P_3 = P_4 = 0.4, \end{aligned} \tag{5.11}$$

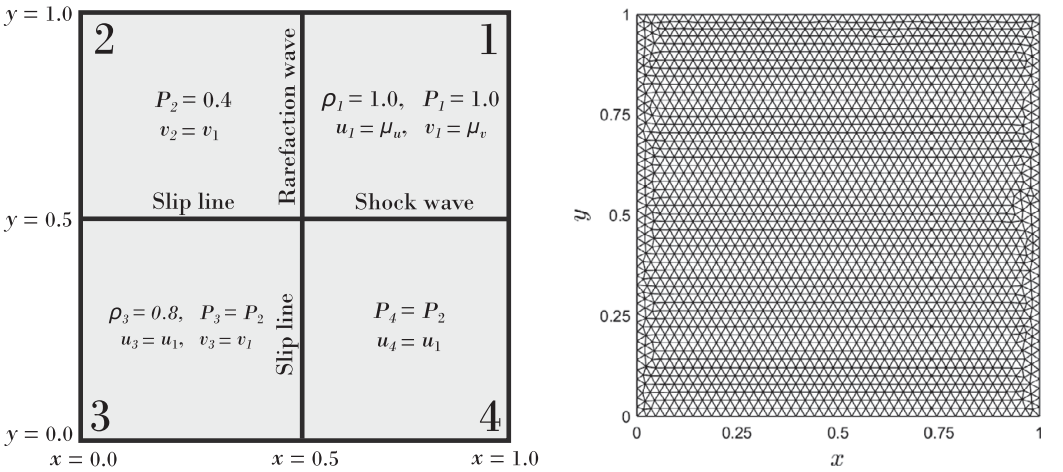

Figure 12. *(a) Setup for the parametric Euler equations to be solved by a Riemann solver. The quadrants have been numbered in the figure. The problem's parameters are taken to be the initial velocities in the x and y directions in the top right quadrant, that is, $\boldsymbol{\mu} = (\mu_u, \mu_v)$. Varying the initial velocity results in the shock wave and rarefaction wave propagating at different speeds and in different directions, resulting in an advection-driven flow. (b) The unstructured mesh used to solve 2D Euler equations for Riemann problem setup. Note that this unstructured finite volume mesh is not directly compatible with a CNN-based autoencoder, and will therefore require interpolation to perform dLSPG.*

where $\mu_u, \mu_v \in \mathbb{R}$ are the model's parameters, that is, $\boldsymbol{\mu} = (\mu_u, \mu_v)$, and the remaining variables $(\rho_2, \rho_4, u_2, v_4)$ are defined by the Rankine-Hugoniot relations and the relations for a polytropic gas. Specifically, the rarefaction wave yields the conditions,

$$\begin{aligned} \rho_2 &= \rho_1 \left(\frac{P_2}{P_1}\right)^{\frac{1}{\gamma}}, \\ u_2 &= u_1 + \frac{2\sqrt{\gamma}}{\gamma - 1}\left(\sqrt{\frac{P_2}{\rho_2}} - \sqrt{\frac{P_1}{\rho_1}}\right), \end{aligned} \tag{5.12}$$

and the shock wave yields the conditions,

$$\begin{aligned} \rho_4 &= \rho_1 \left(\frac{\frac{P_4}{P_1} + \frac{\gamma - 1}{\gamma + 1}}{1 + \frac{\gamma - 1}{\gamma + 1}\frac{P_4}{P_1}}\right), \\ v_4 &= v_1 + \sqrt{\frac{(P_4 - P_1)(\rho_4 - \rho_1)}{\rho_4 \rho_1}}. \end{aligned} \tag{5.13}$$

We generate an unstructured mesh with 4328 finite volume cells, where the mesh is presented in Figure 12b. Next, we perform a parametric study by varying the initial velocities in the top right quadrant via $\boldsymbol{\mu} = (\mu_u = -1.2 - 0.2i, \mu_v = -0.3 - 0.1j)$, with $i = 0, \ldots, 4$ and $j = 0, \ldots, 4$, resulting in solutions to 25 different parameter sets. We take $\Delta t = 0.001$ and $T_f = 0.3$ (which ensures that the shock and rarefaction waves remain in the domain for all parameter sets), therefore collecting 301 snapshots for each parameter set including the initial conditions. The solutions from the parametric study were used as training data to train the autoencoder. To generate the POD-LSPG solution, we set the tolerance κ in Eq. (4.5) to be 10^{-4}, whereas, the tolerance is set to 10^{-3} for the CNN-based dLSPG and GD-LSPG. The step size, $\beta^{(j)}$, for all three models is set to 1.0 at all time steps.

While this solution is modeled by an unstructured mesh, the domain is square and the discretization is mostly regular. As a result, we can interpolate the solution states on unstructured cell centers to a regular, structured counterpart mesh for direct application to a CNN-based autoencoder. In this study, we simply use a $k-$nearest neighbors interpolation with $k = 3$. Here, we establish the benefit of using GD-LSPG for the unstructured mesh as opposed to deploying the interpolated CNN-based autoencoder to dLSPG. To illustrate, we repeat the training for each autoencoder at a given latent space dimension five times, where the only variation between training processes is the random initialization of weights and biases and the random generation of mini-batches at each epoch.

Figure 13 presents the pressure field at $t = 0.3$ for two test parameter sets not seen during training, $\boldsymbol{\mu} = (\mu_u = -1.3, \mu_v = -0.65)$ and $\boldsymbol{\mu} = (\mu_u = -1.9, \mu_v = -0.35)$. Results are presented for the ground truth, POD-LSPG, and both the best- and worst-case predictions for interpolated dLSPG and GD-LSPG, where the best and worst cases correspond to the lowest and highest state predictions errors among the five independently trained autoencoders, respectively. As expected, GD-LSPG captures the moving shock behavior more accurately than the affine POD-LSPG solution and also provides more accurate pressure fields in the regions separated by the shock and rarefaction waves. For the best-case error, the interpolated dLSPG solution is slightly more accurate than the GD-LSPG solution. However, for the worst-case error, the GD-LSPG solution is considerably more accurate than the interpolated dLSPG solution. This implies that, within the setting studied, the GD-LSPG framework yields more consistent results than those obtained via the interpolated dLSPG framework.

Figure 14 reports reconstruction and state prediction errors for five independently trained autoencoders at each latent space dimension. We consider latent space dimensions of 1–10 compared to the FOM dimension of $4328 \times 4 = 17312$. Specifically, for the latent space dimensions 3 and 4, some of the interpolated CNN-based autoencoders generate solutions that perform considerably worse than the

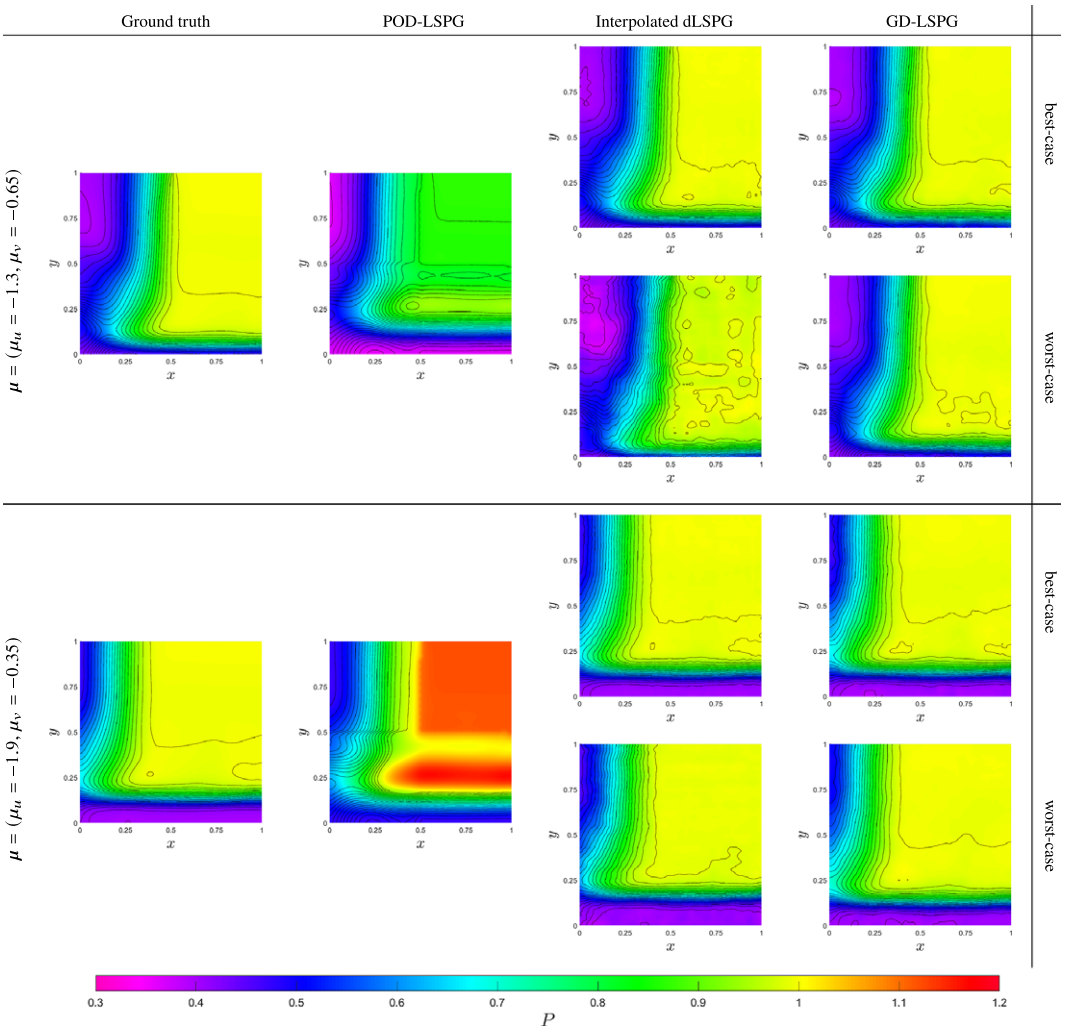

Figure 13. *Ground truth, POD-LSPG, and both best- and worst-case interpolated dLSPG and GD-LSPG pressure fields, denoted by color, and density fields, denoted by contours, for the 2D Euler equations. The results are shown at* $t = 0.3$ *for test parameter sets* $\boldsymbol{\mu} = (\mu_u = -1.3, \mu_v = -0.65)$ *and* $\boldsymbol{\mu} = (\mu_u = -1.9, \mu_v = -0.35)$ *and for a latent space dimension of* $M = 3$*. Note POD-LSPG's inability to model the shock behavior, leading to a diffusive and inaccurate solution that does not accurately model the advection-driven behavior of the problem. The best-case interpolated dLSPG and GD-LSPG solutions model the shock behavior with much higher accuracy, while the worst-case GD-LSPG solution is more accurate than the worst-case interpolated dLSPG solution. Note that an animated version of this figure is found in the supplementary material at journals.cambridge.org in the provided mp4 files.*

worst-case GD-LSPG solutions at the same latent space dimension. This indicates that the interpolated CNN-based autoencoder is prone to failure in generalizing for parameter sets not seen during training.

Finally, Figure 15 presents the local errors in the pressure field at parameter sets $\boldsymbol{\mu} = (\mu_u = -1.3, \mu_v = -0.65)$ and $\boldsymbol{\mu} = (\mu_u = -1.9, \mu_v = -0.35)$ at time $t = 0.3$ for POD-LSPG, and the best- and worst-case interpolated dLSPG, and GD-LSPG. Here, it is evident that the POD-LSPG solution introduces considerable error to the pressure field throughout the domain. Alternatively, for best-case solutions, both interpolated dLSPG and GD-LSPG introduce relatively small errors isolated

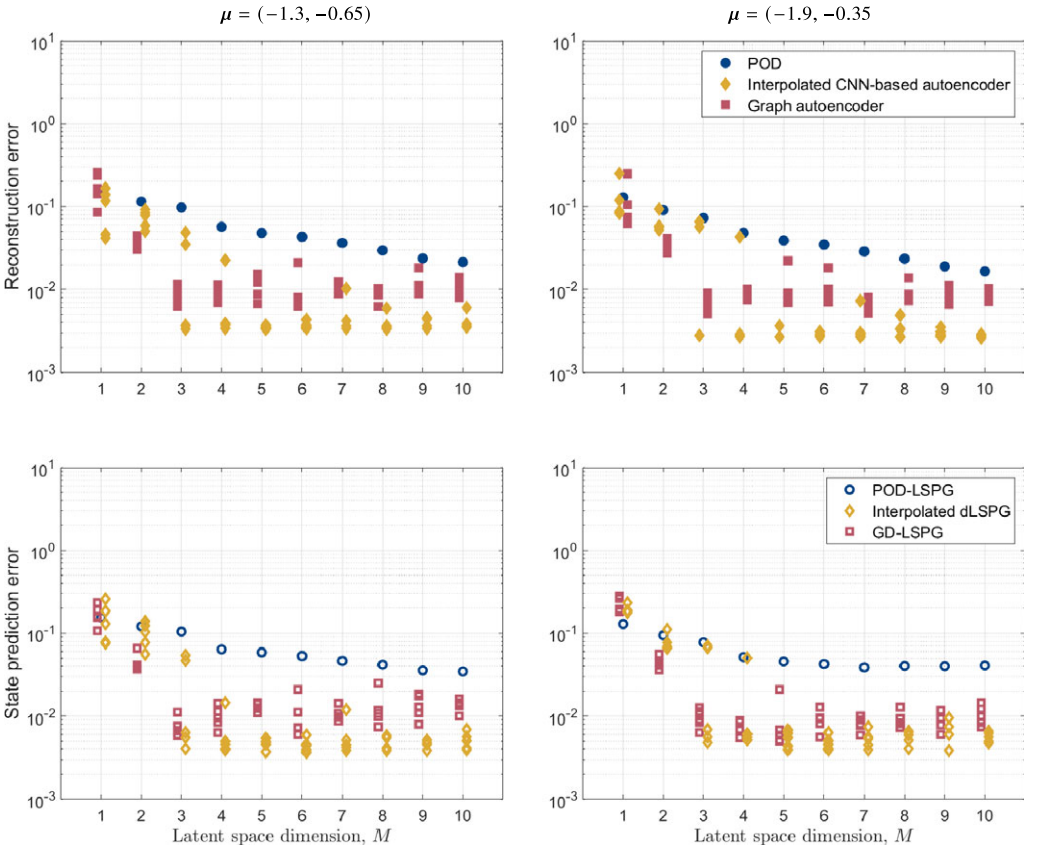

Figure 14. *Reconstruction and state prediction errors for two choices of test parameter sets (each represented by a single column) for the Riemann problem setup. We repeat the training for the interpolated CNN-based autoencoder and the graph autoencoder five times, which correspond to the five points at each latent space dimension. We only obtain the POD modes once, which corresponds to only one point at each latent space dimension for POD reconstruction and POD-LSPG. The top row includes plots of POD, interpolated CNN-based autoencoder, and graph autoencoder reconstruction errors evaluated from Eq. (5.1) to Eq. (5.2) with respect to the latent space dimension, M. Note that the graph autoencoder performs similar level of reconstruction accuracy as the interpolated CNN-based autoencoder. The bottom row demonstrates POD-LSPG, interpolated dLSPG, and GD-LSPG state prediction errors evaluated from Eq. (5.3) with respect to the latent space dimension, M. For small latent space dimensions, GD-LSPG (and often interpolated dLSPG) outperforms POD-LSPG in terms of accuracy. Additionally, the interpolated CNN-based autoencoder used in the dLSPG solution fails to generalize well in this application for several latent space dimensions. Note: one of the interpolated dLSPG solutions for $M = 1$ failed to converge.*

around the shock and rarefaction waves, indicating that the primary source of error for these methods is a phase lag. However, the worst-case GD-LSPG solution is considerably more accurate than the worst-case interpolated dLSPG solution, establishing a better generalization capabilities of GD-LSPG in this setting.

5.2.2. Bow shock generated by flow past a cylinder problem

In our second setting, we model a bow shock generated by flow past a cylinder using an unstructured finite volume mesh for a parameterized version of a test case found in Nishikawa and Kitamura (2008) to

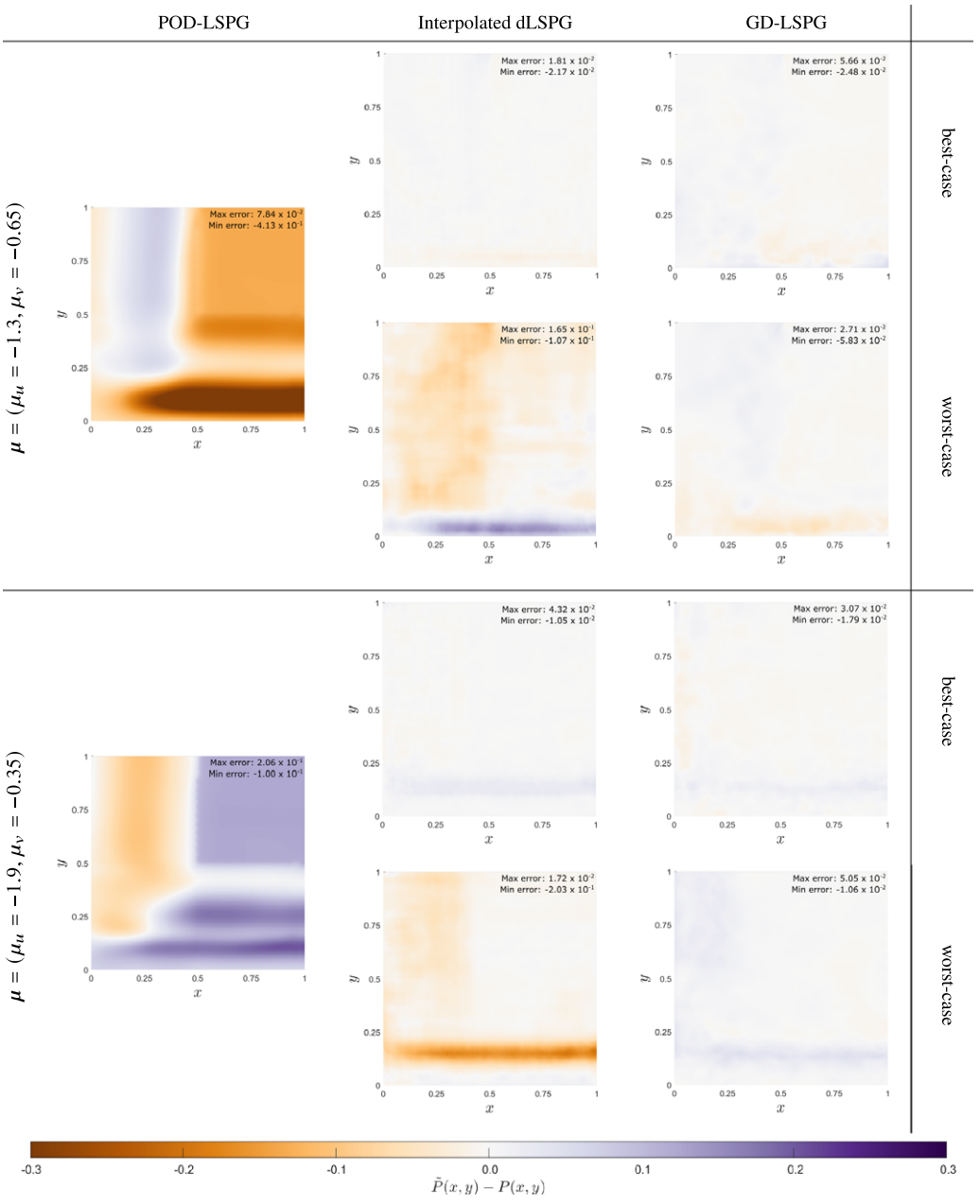

Figure 15. *ROM prediction errors associated with the best- and worst-case solutions for the Riemann problem setup. The left, middle and right columns represent the errors for POD-LSPG, interpolated dLSPG and GD-LSPG, respectively, at latent space dimension $M = 3$. The results are depicted for the parameter sets $\boldsymbol{\mu} = (\mu_u = -1.3, \mu_v = -0.65)$ and $\boldsymbol{\mu} = (\mu_u = -1.9, \mu_v = -0.35)$ at time $t = 0.3$. We define local error to be $\tilde{P}(x,y) - P(x,y)$ at a given time step. The POD-LSPG solution fails to accurately model the advection-driven nature of the solution and shows errors as high as 25% in some parts of the domain. The best-case interpolated dLSPG and GD-LSPG solutions model the shock wave much more accurately, with only small errors localized near the shock, highlighting the slight phase misalignment between FOM and ROM shock locations as the primary source of error. The worst-case interpolated dLSPG solution has considerably higher errors than the worst-case GD-LSPG solution. Note that an animated version of this figure is found in the supplementary material at journals.cambridge.org in the provided mp4 files.*

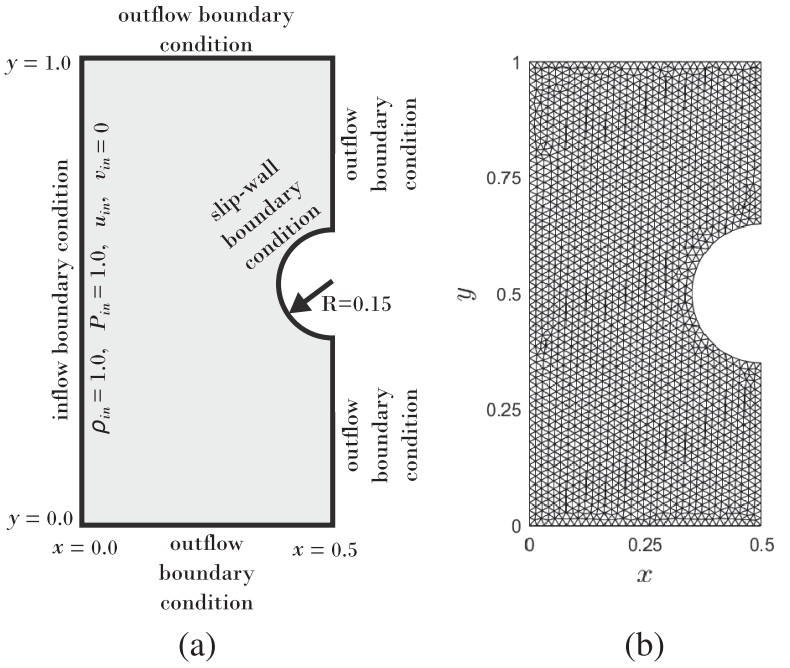

Figure 16. *(a) Setup for the parametric Euler equations to be solved by a Riemann solver with an unstructured finite volume mesh. A rectangular domain with the leading edge of a cylinder is modeled with the noted boundary conditions. Depending on the freestream Mach number, μ_{in}, which will be varied as this problem's parameter, a shock can form along the leading edge of the cylinder and propagate through the domain at different speeds. (b) Mesh used to solve 2D Euler equations for the bow shock generated by flow past a cylinder problem. In this case, the physical domain's geometry is more complex than in the Riemann problem setup, therefore benefiting from the unstructured mesh.*

demonstrate the flexibility of GD-LSPG. As seen in Figure 16a, the model consists of a rectangular domain and the leading edge of a cylinder, where, for the selected range of parameter values, it is expected that a shock will form on the leading edge and propagate through the domain. Because of the domain's geometric irregularity, it is natural to employ an unstructured mesh, which is presented in Figure 16b. Once again, we solve Eq. (5.6 and 5.7) on the domain that has been spatially discretized into 4148 finite volume cells using Gmsh (Geuzaine and Remacle, 2009). However, this time we model an inflow boundary condition on the left boundary of the domain, outflow boundary condition on the right side of the domain and the top and bottom boundaries, and a slip-wall boundary condition on the surface of the cylinder. We parameterize the model by the freestream Mach number, μ_{in}, that is,

$$\rho_{\text{in}} = 1.0, \quad P_{\text{in}} = 1.0, \quad u_{\text{in}} = \mu_{\text{in}} \sqrt{\frac{\gamma P_{\text{in}}}{\rho_{\text{in}}}}, \quad v_{\text{in}} = 0, \tag{5.14}$$

where ρ_{in}, P_{in}, u_{in}, and v_{in} denote the freestream density, freestream pressure, and freestream velocities in the x and y directions, respectively, and $\mu_{in} = 1.0, 1.050, \ldots, 1.250$, which results in solutions to six different parameters. The solutions exhibit a shock that develops along the leading edge of the cylinder at varying rates. Additionally, the shock propagates at different speeds through the domain and incurs varying pressure, density, and velocity differences across the shock depending on the freestream Mach number. We take $\Delta t = 0.001$ and $T_f = 1.0$, therefore collecting 1001 snapshots for each parameter including the initial conditions.

In this numerical experiment, we present POD-LSPG and GD-LSPG solutions. To generate the POD-LSPG solution, we set the tolerance κ in Eq. (4.5) to be 10^{-4}, whereas, the tolerance is set to 10^{-3} for GD-LSPG. The step size $\beta^{(j)}$ for both models is set to 1.0 at all time steps. Figure 17 demonstrates the pressure and density fields at two different time steps and at the test parameter $\mu_{\text{in}} = 1.125$ for the ground truth solution, POD-LSPG, and GD-LSPG, both using a latent space dimension $M = 2$. Additionally, Figure 17 provides the local errors between both ROM solutions versus the ground truth solution for the pressure field. As before, note that POD-LSPG smooths out the shock and fails to accurately model the moving discontinuity. Alternatively, GD-LSPG models the moving shock behavior much more faithfully without generating spurious high-pressure regions. It is evident that the majority of the errors for GD-LSPG are isolated around the shock.

We further assess the interpretability of the GD-LSPG solution by presenting the Jacobian of the decoder for the graph autoencoder alongside the POD modes used in the POD-LSPG solution for the latent space dimension $M = 2$ and the test parameter $\mu_{\text{in}} = 1.125$ in Figure 18. Specifically, we plot the component of the Jacobian of the decoder and the POD modes that correspond to the energy state variable (i.e., ρE in Eq. [5.7]) over the entire physical domain. While the POD modes are time invariant, the Jacobian of the decoder depends on the latent state vector and is shown at two distinct time instances. At $t = 0.25$, mode 1 primarily captures information about the moving shock, while mode 2 captures information about the moving shock and the high-pressure region behind it. Additionally, at $t = 0.75$, mode 1 continues to represent the shock, and additionally captures part of the high-pressure region, whereas mode 2 still captures both features. On the other hand, the POD modes remain highly diffusive and time invariant, limiting their ability to represent the moving shock as well as the graph autoencoder.

Finally, Figure 19 reports the POD and graph autoencoder reconstruction and state prediction errors. For this numerical example, we consider latent space dimensions 1 to 10, where the dimension of the FOM is 4148 * 4 = 16592. The graph autoencoder presents significantly lower reconstruction errors compared to POD for the latent space dimensions of choice. In addition, the GD-LSPG state prediction errors remain considerably lower than those of POD-LSPG. We note that the main reason that GD-LSPG state prediction errors for this example are not as low as those achieved in the 1D Burgers' model and 2D Riemann problem setup is the more significant phase lag error as demonstrated in Figure 17. Furthermore, we emphasize that identifying an appropriate error metric for advection-dominated problems is an important area of future work.

5.3. Performance with noisy training data

In this section, we evaluate the ability of our graph autoencoder to approximate the underlying physics when trained on noisy data. The numerical experiment is conducted on the flow past a cylinder problem, following the same training strategy outlined in Appendix A, with the only difference being the addition of Gaussian noise to the training data. Specifically, Gaussian noise is added to each state variable $(\rho, \rho u, \rho v, \rho E)$, proportional to its range within the training data,

$$(\overline{\rho})_i^n = (\rho)_i^n + \epsilon \sigma_{\text{noise}} \left((\rho)^{max} - (\rho)^{min} \right), \tag{5.15}$$

$$(\overline{\rho u})_i^n = (\rho u)_i^n + \epsilon \sigma_{\text{noise}} \left((\rho u)^{max} - (\rho u)^{min} \right), \tag{5.16}$$

$$(\overline{\rho v})_i^n = (\rho v)_i^n + \epsilon \sigma_{\text{noise}} \left((\rho v)^{max} - (\rho v)^{min} \right), \tag{5.17}$$

$$(\overline{\rho E})_i^n = (\rho E)_i^n + \epsilon \sigma_{\text{noise}} \left((\rho E)^{max} - (\rho E)^{min} \right), \tag{5.18}$$

where $(\rho)_i^n$ and $(\overline{\rho})_i^n$ denote the clean and noisy density values, respectively, for the i^{th} cell and the n^{th} training snapshot, and $(\rho)^{max}$ and $(\rho)^{min}$ represent the maximum and minimum density values across the entire training set. The same notation applies to the momentum components, $(\overline{\rho u})_i^n$, $(\overline{\rho v})_i^n$, and the energy term, $(\overline{\rho E})_i^n$). In Eqs. (5.15–5.18), $\sigma_{\text{noise}} \in \mathbb{R}_+$ denotes the noise level, and $\epsilon \sim \mathcal{N}(0,1)$ represents the added noise drawn from a standard normal distribution. In this study, we vary the noise level as

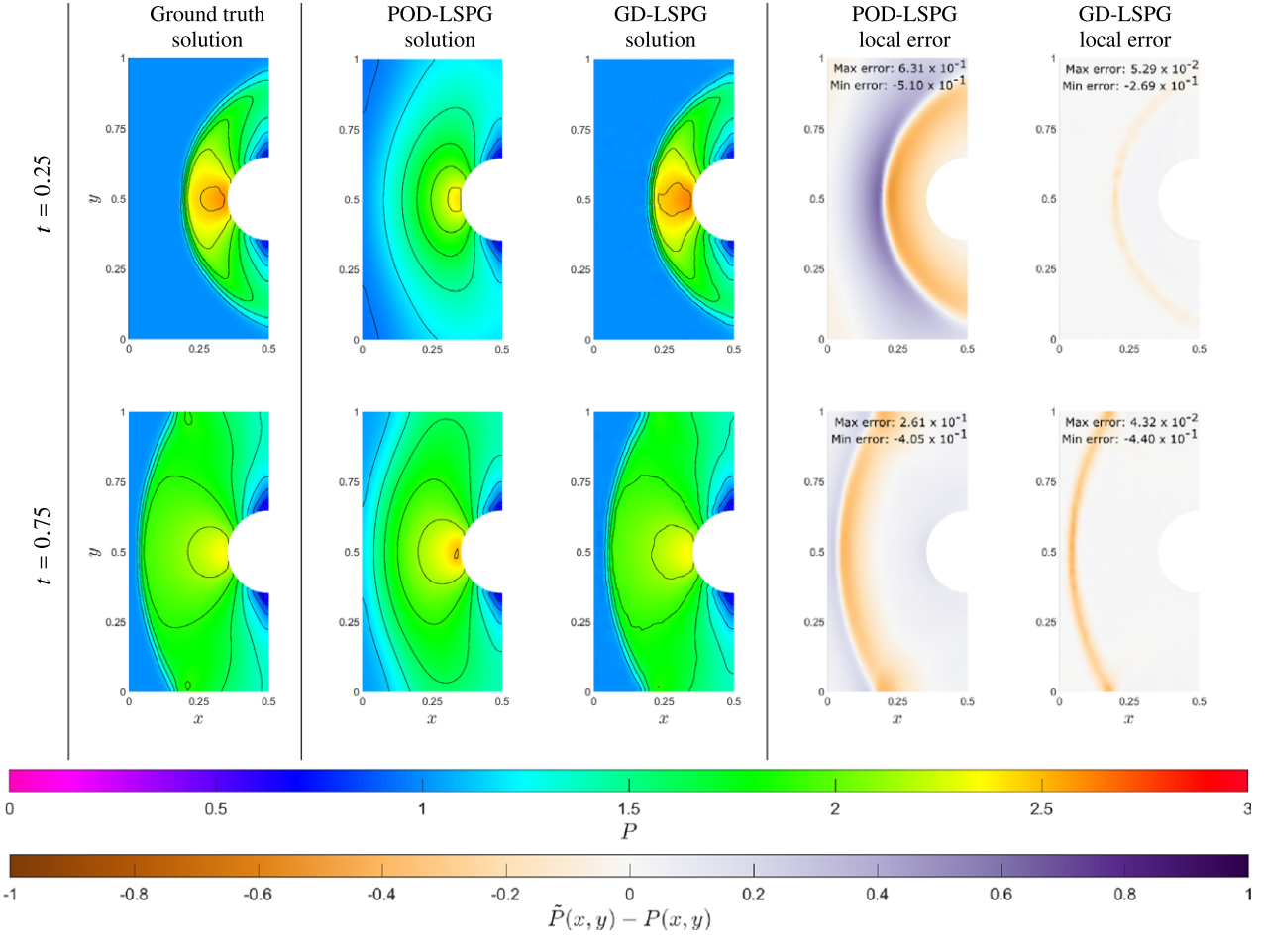

Figure 17. *Pressure field (denoted by color) and density field (denoted by contours) are demonstrated for the ground truth solutions (first column) as well as the POD-LSPG (second column) and GD-LSPG solutions (third column). Additionally, the local errors, $\tilde{P}(x,y) - P(x,y)$, between the corresponding ROMs and the ground truth solution for the bow shock generated by flow past a cylinder are presented in columns 4 and 5. The results are provided at two time steps, namely $t = 0.25$ (top row) and $t = 0.75$ (bottom row), for test parameter $\mu_{\text{in}} = 1.125$. The ROMs are constructed using a latent space dimension of $M = 2$. Like before, POD-LSPG struggles to accurately model the advection-driven shock wave occurring in this model. Alternatively, GD-LSPG models the shock behavior with higher accuracy where errors are mainly isolated around the shock. The top color bar refers to the pressure field solutions while the bottom color bar refers to the local error plots.*

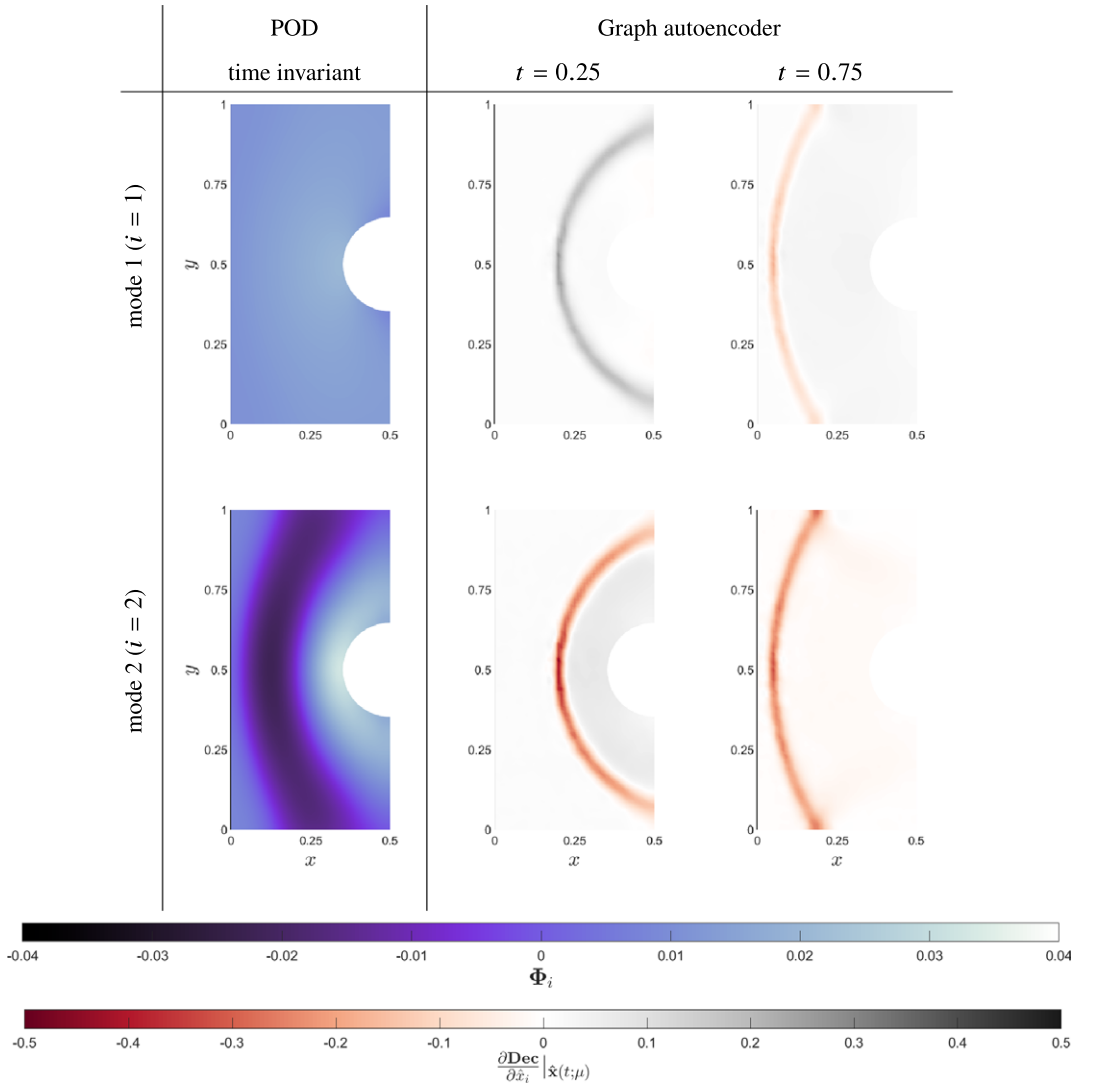

Figure 18. *POD modes, Φ_i, (left) and Jacobian of the decoder for the graph autoencoder of the i^{th} latent state variable, $\frac{\partial \mathbf{Dec}}{\partial \hat{x}_i}\Big|_{\hat{\mathbf{x}}(t;\boldsymbol{\mu})}$ (right) for the energy state variable (ρE in Eq. [5.7]) for the POD-LSPG and GD-LSPG solutions, respectively. The results are shown for the latent space dimension $M = 2$ and for the test parameter $\mu_{\text{in}} = 1.125$. Note that the POD modes are time invariant and therefore remain fixed for all time steps, while the Jacobian of the decoder for the graph autoencoder presents time-varying mode shapes (here shown for $t = 0.25$ and $t = 0.75$) . The Jacobian of the decoder of the graph autoencoder reveals that the graph autoencoder's latent variables primarily contain information about the bow shock that forms on the leading edge of the cylinder, offering interpretability into the GD-LSPG solution. In contrast, the POD modes used in POD-LSPG are time invariant, independent of the latent state vector, and highly diffusive. Note that the top colorbar (black/blue/white) is used to present the POD modes, while the bottom color bar (red/white/black) is used to present the Jacobian of the decoder for the graph autoencoder.*

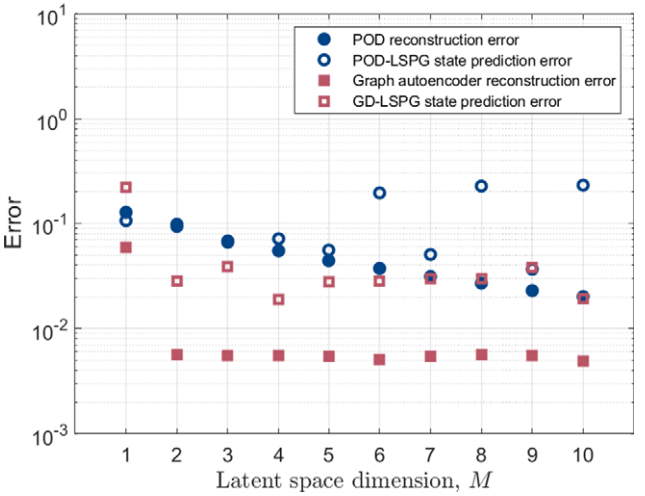

Figure 19. *POD reconstruction error from Eq. (5.2), graph autoencoder reconstruction error from Eq. (5.1), and state prediction errors from Eq. (5.3) plotted with respect to the latent space dimension M for 2D Euler equations for the bow shock generated by flow past a cylinder at* $\mu_{\text{in}} = 1.125$. *For latent space dimensions 2–10, GD-LSPG performs as well as or considerably better than POD-LSPG in terms of accuracy.*

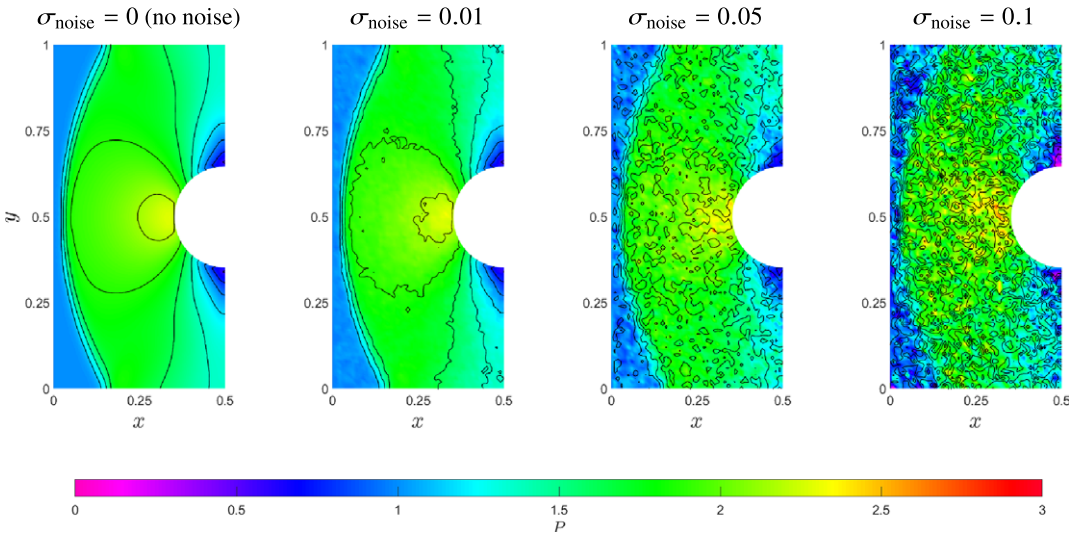

Figure 20. *Pressure field (denoted by color) and density field (denoted by contours) are demonstrated for four different noise levels,* $\sigma_{\text{noise}} = 0$, 0.01, 0.05, *and* 0.1 *for the training parameter* $\mu_{\text{in}} = 1.1$ *at* $t = 0.75$. *Notice that for* $\sigma_{\text{noise}} = 0.1$, *the shock features are still present, but are significantly blurred in comparison to* $\sigma_{\text{noise}} = 0$.

$\sigma_{\text{noise}} = 0.0$, 0.01, 0.05, and 0.1 to assess the sensitivity of the graph autoencoder to different noise levels. Figure 20 presents examples of noisy training solutions for $\mu_{\text{in}} = 1.1$ at time $t = 0.75$.

Once trained over all studied noise levels, the graph autoencoder is evaluated on its ability to generate the clean (i.e., noise-free), ground truth solutions for a test parameter not seen during training. Figure 21 presents the ground truth solution and GD-LSPG solutions for latent space dimensions $M = 2$, and 10, and

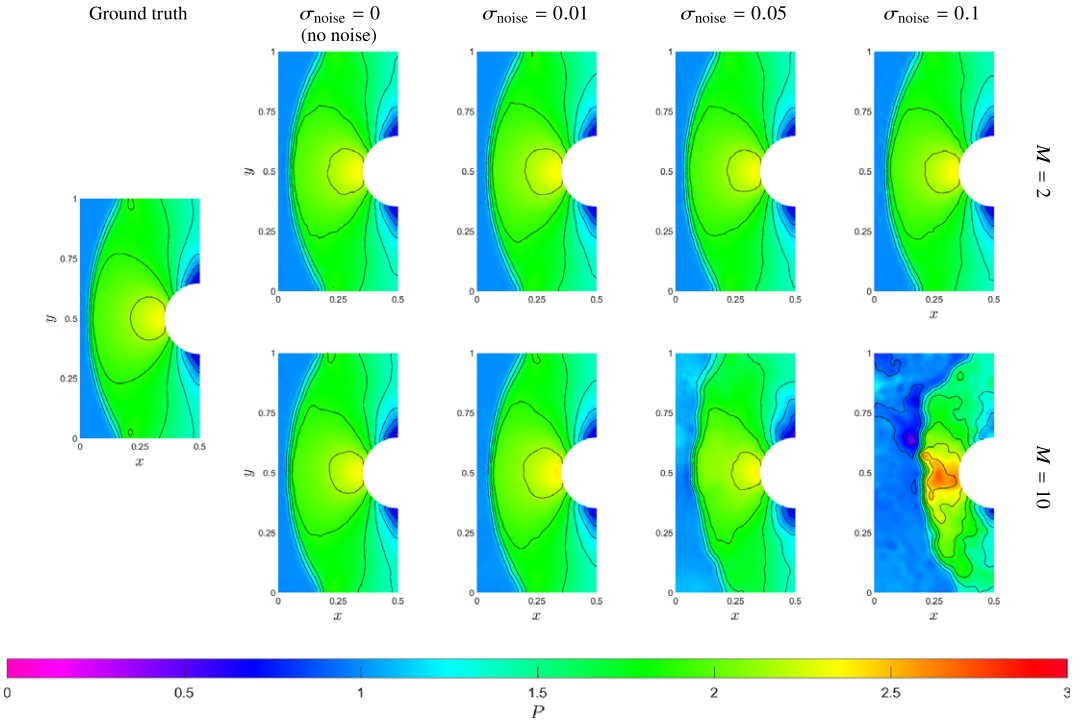

Figure 21. *Pressure field (denoted by color) and density field (denoted by contours) are demonstrated for the ground truth and for GD-LSPG for latent space dimensions $M = 2$, and 10 and noise levels $\sigma_{noise} = 0$, 0.01, 0.05, and 0.1 at time $t = 0.75$ for test parameter $\mu_{in} = 1.125$. For $M = 2$, we find all solutions to be very accurate predictions of the ground truth solution. However, for $M = 10$, we find that, as more noise is introduced, the GD-LSPG solution becomes more inaccurate due to noise. Still, the shock features appear to be present.*

for test parameter $\mu_{in} = 1.125$ at time $t = 0.75$. For $M = 2$, the GD-LSPG solution remains consistently accurate across all studied noise levels. In contrast, for $M = 10$, the GD-LSPG solution maintains accuracy for $\sigma_{noise} = 0$, and 0.01, but accuracy degrades at noise levels $\sigma_{noise} = 0.05$, and 0.1. While the shock features appear to still be present in the solutions, there appears to be a considerable amount of artifacts introduced into the solution, especially for noise level $\sigma_{noise} = 0.1$.

Finally, the reconstruction and state prediction errors are reported in Figure 22 for latent space dimensions $M = 1$–10, and noise levels $\sigma_{noise} = 0.0$, 0.01, 0.05, and 0.1. As expected, introducing higher noise levels increased both the graph autoencoder reconstruction errors and the GD-LSPG state predictions errors. Additionally, we observe that the GD-LSPG state prediction errors for the noisy solutions increase with larger latent space dimensions. We speculate that the graph autoencoder with smaller latent space dimensions are constrained to capture only the signal itself, whereas those with larger latent space dimensions may overfit to noise. It should also be noted that several GD-LSPG solutions failed to converge due to non-physical solutions (e.g., negative pressure), primarily at higher noise levels, specifically at noise level $\sigma_{noise} = 0.05$ for $M = 1$, and at noise level $\sigma_{noise} = 0.1$ for $M = 5,6,9$).

5.4. Wall clock training times

Lastly, in addition to online computational costs, we report the times to generate the hierarchies of graphs as well as the times to train each model used in this study, which were both carried out using an NVIDIA L40S GPU. Wall-clock times to generate the hierarchy of graphs for each example are reported in Table 1 and wall-clock times to train each model are reported in Table 2. For the second numerical experiment

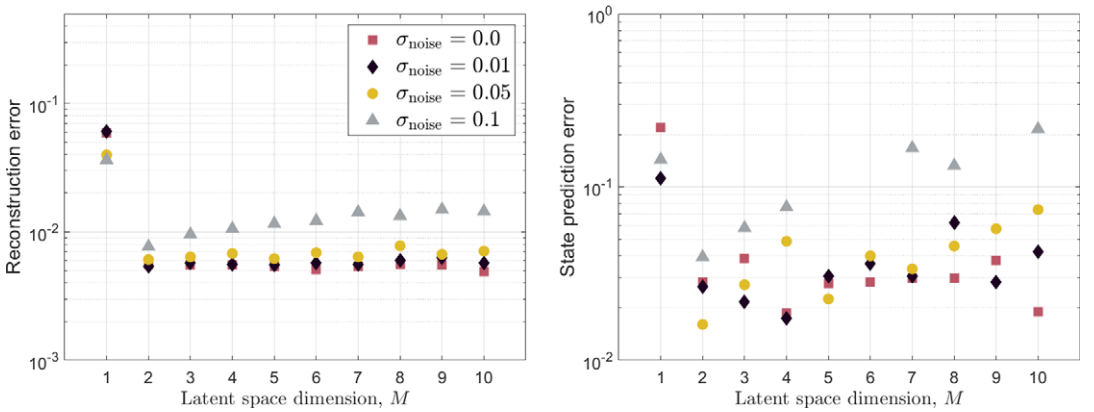

Figure 22. *Graph autoencoder reconstruction errors (left) evaluated from Eq. (5.1) and GD-LSPG state prediction errors (right) evaluated from Eq. (5.3) for the bow shock generated by flow past a cylinder problem when trained using noisy training data with noise levels* $\sigma_{noise} = 0$, 0.01, 0.05, *and* 0.1 *for test parameter* $\mu = 1.125$. *Higher noise levels in the training data lead to increased reconstructions errors, as well as higher state prediction errors. Note that GD-LSPG failed to converge due to non-physical solutions (i.e., negative pressure) for latent space dimension* $M = 1$ *at noise level* $\sigma_{noise} = 0.05$ *and for latent space dimensions* $M = 5$, *6, and 9 at* $\sigma_{noise} = 0.1$.

Table 1. *Average wall-clock times (in seconds) for generating the hierarchy of graphs for each example, where* $|\mathcal{V}^i|$ *and* $|\mathcal{V}^{i+1}|$ *denote the number of nodes in the* i^{th} *and* $(i+1)^{th}$ *layers in the hierarchy of graphs, respectively. The time reported for each clustering is the average of ten repeated runs of the clustering algorithm*

	Burgers' model			Riemann problem			Bow shock		
Hierarchy level, i	$\lvert\mathcal{V}^i\rvert$	$\lvert\mathcal{V}^{i+1}\rvert$	Wall-clock time (s)	$\lvert\mathcal{V}^i\rvert$	$\lvert\mathcal{V}^{i+1}\rvert$	Wall-clock time (s)	$\lvert\mathcal{V}^i\rvert$	$\lvert\mathcal{V}^{i+1}\rvert$	Wall-clock time (s)
0	316	64	15.391	4328	512	676.098	4148	512	702.034
1	64	16	0.029	512	64	1.116	512	64	1.095
2	16	4	0.037	64	8	0.008	64	8	0.008
3	4	2	0.002	8	2	0.003	8	2	0.002
Total			15.425			677.224			703.140

(Riemann problem), where we repeat the training five times, we simply report the average time for training the autoencoders for each latent space dimension. Immediately, it is worth noting that the graph autoencoder in its current form is more expensive to train than the traditional CNN-based autoencoders and considerably more expensive to train than the POD.

The wall-clocks times reported in Table 1 are computed by repeating the hierarchical spectral clustering algorithm ten times for each model and averaging the results for each level. For the Burgers' model, we employ the K-means++ initialization scheme (Arthur and Vassilvitskii, 2007), as it resulted in more intuitive cluster assignments. Generating the graph hierarchy is not yet computationally efficient enough to be used on-the-fly. More efficient clustering schemes will be necessary before GD-LSPG can be effectively applied to models without a fixed mesh, such as those using adaptive meshes, Lagrangian frameworks, or arbitrary Lagrangian–Eulerian methods.

Another noteworthy observation is that, although the solution states in the Riemann problem setup and the bow shock generated by flow past a cylinder problem have considerably larger dimensions than the

Table 2. *Wall-clock training times (in hours) for POD, CNN-based autoencoder, and graph autoencoder for all numerical examples. Note that the POD is only obtained once for all latent space dimensions where the POD basis is determined by truncating the left singular basis to the user-specified latent space dimension,* M. *Alternatively, each latent space dimension has a unique CNN-based autoencoder and graph autoencoder that must be trained individually. For the repeated training of the autoencoders for the Riemann problem, we report the average time to train each model*

Latent space dimension, M	Burgers' model			Riemann problem			Bow shock	
	POD	CNN-based autoencoder	Graph autoencoder	POD	Interpolated CNN-based autoencoder	Graph autoencoder	POD	Graph autoencoder
1	1.39×10^{-3}	1.73	4.70	0.03	2.01	5.08	0.02	4.48
2		1.30	5.48		1.98	5.01		4.52
3		1.16	5.40		1.90	5.00		4.38
4		1.30	4.64		1.91	5.06		4.41
5		1.30	4.68		1.94	5.00		4.40
6		1.13	4.77		2.23	4.98		3.83
7		1.12	4.77		2.21	5.03		3.80
8		1.17	4.71		2.20	5.88		4.67
9		1.30	4.65		1.86	4.95		3.91
10		1.28	4.70		1.86	5.02		4.64

Burgers' model, the cost to train the autoencoders does not appear considerably larger. There are likely several factors contributing to this. First, the larger models are likely easier to parallelize on the NVIDIA L40S GPU. Furthermore, the number of batches provided at each epoch during training is a major driving component of the training time. As seen in Appendix A, for all autoencoders, a batch size of 20 was chosen due to random access memory limitations. However, due to the varying number of training solutions, the number of batches per epoch varies considerably depending on the problem. To elaborate, the autoencoders for the 1D Burgers' model have 36080 solution states used for training, meaning each epoch will have 1804 batches. On the other hand, the autoencoders for the 2D Riemann problem setup have 7000 solution states used for training, meaning each epoch will have 350 batches. Finally, the autoencoders for the 2D bow shock generated by flow past a cylinder problem have 5500 solution states used for training, meaning each epoch will have 275 batches.

6. Conclusions and future work

In this paper, we present GD-LSPG, a PMOR method that leverages a graph autoencoder architecture to perform model reduction directly on unstructured meshes. The graph autoencoder is constructed by first generating a hierarchy of reduced graphs to emulate the compressive capabilities of CNNs. Next, message passing operations are trained for each layer in the hierarchy of reduced graphs to emulate the filtering capabilities of CNNs. In an online stage, the graph autoencoder is leveraged to perform a nonlinear manifold LSPG projection to obtain solutions to parameter sets not seen during training. We investigate the interpretability of such solutions by analyzing the Jacobian of the decoder.

In this work, we assess the performance of GD-LSPG with three different numerical experiments. First, to benchmark GD-LSPG, we apply it to a 1D Burgers' model commonly used in the literature. Specifically, this benchmarking case deploys a structured mesh and we can therefore compare the results directly to both POD-LSPG (Carlberg et al., 2011, 2017) and dLSPG (Lee and Carlberg, 2020, 2021). We find that GD-LSPG has accuracy that is comparable to that of dLSPG and that both methods provide

considerable improvement in accuracy over POD-LSPG as they capture the moving shock behavior found in this solution more accurately in comparison to POD-LSPG. By plotting the Jacobian of the decoder, we find that the latent space of the graph autoencoder primarily captures information about the moving shock front. In the second and third numerical experiments, we apply GD-LSPG to a 2D Euler equations model that leverages an unstructured mesh in two different settings. The first setting involves a square domain that solves a Riemann problem setup. Because the domain is regular, we can interpolate the unstructured mesh onto a structured grid that can be provided to a CNN-based autoencoder and applied in dLSPG. While we find that this method can produce results that outperform GD-LSPG, it is also prone to failing to generalize well for parameter sets not seen during training. Alternatively, GD-LSPG consistently generalizes well for parameter sets not seen during training while also considerably outperforming POD-LSPG in terms of accuracy due to its ability to model the moving shock and rarefaction waves present in this problem. The second setting models a bow shock generated by flow past a cylinder with an unstructured mesh. In this setting, the modeled geometry is more naturally represented by an unstructured mesh, and therefore demonstrates the flexibility of GD-LSPG. We compare GD-LSPG to POD-LSPG and, once again, find that GD-LSPG often outperforms POD-LSPG in terms of accuracy as GD-LSPG better preserves the sharp features of the bow shock that appear on the cylinder. Alternatively, POD-LSPG smooths out the shock and introduces spurious high-pressure regions in the solution. As with the 1D Burgers' model, the Jacobian of the decoder for the graph autoencoder indicates that the latent state variables contain information about the moving bow shock. Finally, we also investigate the ability of the graph autoencoder to filter noise by training it on data generated by perturbing the ground truth solutions with Gaussian noise at various noise levels for the bow shock generated by flow past a cylinder problem. This setting naturally aligns with our graph autoencoder framework, as the noisy model is constrained to the same underlying governing equations as the clean data for which the residual can be computed. At smaller latent space dimensions, the graph autoencoder accurately captures the shock behavior. However, accuracy decreases as noise level and latent space dimension increase.

As is present across an extensive amount of literature employing end-to-end training of autoencoders (Lee and Carlberg, 2020, 2021; Fries et al., 2022; Barwey et al., 2023), our graph autoencoder is limited to models with a dimension of only a few thousand. Additionally, we emphasize that the hierarchical spectral clustering scheme is limited by the scalability of the eigen decomposition of the graph Laplacian and $K-$means clustering operation which we leave as areas of future work. One potential future direction to achieve cost savings is to introduce a latent space dynamics model similar to operator inference (Peherstorfer and Willcox, 2016), latent space dynamics identification (Fries et al., 2022), or sparse identification of nonlinear dynamics (Brunton et al., 2016). An added benefit of these methods is that, unlike the LSPG schemes presented in Section 4, they do not often require explicit knowledge of the residual. This feature makes them potentially suitable for use with experimental data or computational models where the residual is unavailable or difficult to compute. Another possible future direction is to develop a hyper-reduction scheme to achieve cost savings. In GD-LSPG's current formulation, the high-dimensional residual must be computed and projected onto the low-dimensional latent space, forcing the operation count complexity to scale on the dimension of the FOM. A hyper-reduction scheme would alleviate this limitation by sampling and generating a sparse representation of the high-dimensional residual, thereby eliminating an operation count complexity that scales on the dimension of the FOM. To date, hyper-reduction has been achieved for dLSPG using shallow decoders (Kim et al., 2022). However, GD-LSPG uses a deep decoder. As a result, we leave this as an open area of investigation.

Abbreviations

CNN	Convolutional neural network
dLSPG	Deep least-squares Petrov–Galerkin
ELU	Exponential linear unit
FEM	Finite element method
FOM	Full-order model
FVM	Finite volume method

GD-LSPG	Geometric deep least-squares Petrov–Galerkin
GNN	Graph neural network
MLP	Multilayer perceptron
MPP	Message passing and pooling
ODE	Ordinary differential equation
PDE	Partial differential equation
PMOR	Projection-based model-order reduction
POD	Proper orthogonal decomposition
POD-LSPG	Proper orthogonal decomposition-based least-squares Petrov–Galerkin
ROM	Reduced-order model
R-RHLL	Rotated Roe, Harten, Lax, and van Leer
UMP	Unpooling and message passing

Supplementary material. The supplementary material for this article can be found at http://doi.org/10.1017/dce.2025.10030.

Data availability statement. The data that support the findings of this study are openly available at https://doi.org/10.5281/zenodo.17088969 (Magargal et al., 2025a) and https://github.com/liam-magargal/GeometricDeepLSPG (Magargal et al., 2025b).

Acknowledgments. Portions of this research were conducted on Lehigh University's Research Computing infrastructure, partially supported by NSF Award 2019035.

Author contribution. Conceptualization: L.K.M.; P.K.; S.N.R. Methodology: L.K.M.; P.K.; S.N.R. Formal analysis: L.K.M; P.K., S.N.R. Data curation: L.K.M.; P.K. Funding acquisition: P.K.; J.W.J. Project administration: P.K.; J.W.J; J.G.M. Writing – original draft: L.K.M.; P.K. Writing – review & editing: L.K.M.; P.K.; S.N.R.; J.W.J.; J.G.M. All authors approved the final submitted draft.

Funding statement. L. K. Magargal is supported by the Department of Defense (DoD) through the National Defense Science & Engineering Graduate (NDSEG) Fellowship Program. P. Khodabakhshi acknowledges the support of the National Science Foundation under award 2450804. L. K. Magargal and J. W. Jaworski acknowledge the financial support of the Department of Energy under grant DE-EE0008964. S. N. Rodriguez and J. G. Michopoulos acknowledge the support of the Office of Naval Research through U.S. Naval Research Laboratory core funding. J. W. Jaworski acknowledges the partial support of the National Science Foundation under CAREER award 1846852/2402397 and the Office of Naval Research under award N00014-24-2111.

Competing interests. The authors declare none.

Ethical standard. The research meets all ethical guidelines, including adherence to the legal requirements of the study country.

References

Ahmed SE and San O (2020) Breaking the Kolmogorov barrier in model reduction of fluid flows. *Fluids 5*(1), 26.

Amsallem D, **Zahr MJ and Farhat C** (2012) Nonlinear model order reduction based on local reduced-order bases. *International Journal for Numerical Methods in Engineering 92*(10), 891–916.

Antoulas A (2005) *Approximation of Large-Scale Dynamical Systems*. Philadelphia, PA: Society for Industrial and Applied Mathematics.

Arthur D and Vassilvitskii S (2007) *K-means++: The advantages of careful seeding*. New Orleans, Louisiana: Society for Industrial and Applied Mathematics, pp. 1027–1035.

Baker N, **Alexander F**, **Bremer T**, **Hagberg A**, **Kevrekidis Y**, **Najm H**, **Parashar M**, **Patra A**, **Sethian J**, **Wild S**, *et al.* (2019) *Tech. Rep. Workshop Report on Basic Research Needs for Scientific Machine Learning: Core Technologies for Artificial Intelligence*. Washington, D.C. (United States): USDOE Office of Science (SC).

Bank D, **Koenigstein N and Giryes R** (2023) *Autoencoders. Machine learning for data science handbook: data mining and knowledge discovery handbook*, 353–374.

Barnett J and Farhat C (2022) Quadratic approximation manifold for mitigating the Kolmogorov barrier in nonlinear projectionbased model order reduction. *Journal of Computational Physics 464*, 111348.

Barnett J, **Farhat C and Maday Y** (2023) Neural-network-augmented projection-based model order reduction for mitigating the Kolmogorov barrier to reducibility. *Journal of Computational Physics 492*, 112420.

Barrault M, **Maday Y**, **Nguyen NC and Patera AT** (2004) An 'empirical interpolation' method: Application to efficient reduced basis discretization of partial differential equations. *Comptes Rendus Mathematique 339*(9), 667–672.

Barwey S, **Shankar V**, **Viswanathan V and Maulik R** (2023) Multiscale graph neural network autoencoders for interpretable scientific machine learning. *Journal of Computational Physics 495*, 112537.

Battaglia PW, **Hamrick JB**, **Bapst V**, **Sanchez-Gonzalez A**, **Zambaldi V**, **Malinowski M**, **Tacchetti A**, **Raposo D**, **Santoro A**, **Faulkner R,** *et al* (2018) *Relational inductive biases, deep learning, and graph networks. Preprint, arXiv:1806.01261*.

Baur U, **Beattie C**, **Benner P and Gugercin S** (2011) Interpolatory projection methods for parameterized model reduction. *SIAM Journal on Scientific Computing 33*(5), 2489–2518.

Benner P, **Goyal P**, **Kramer B**, **Peherstorfer B and Willcox K** (2020) Operator inference for non-intrusive model reduction of systems with non-polynomial nonlinear terms. *Computer Methods in Applied Mechanics and Engineering 372*, 113433.

Benner P, **Gugercin S and Willcox K** (2015) A survey of projection-based model reduction methods for parametric dynamical systems. *SIAM Review 57*(4), 483–531.

Benner P, **Ohlberger M**, **Cohen A and Willcox K** (2017) *Model Reduction and Approximation*. Philadelphia, PA: Society for Industrial and Applied Mathematics.

Bern M and Plassmann P (2000) Chapter 6 - mesh generation. In Sack J and Urrutia J (eds.), *Handbook of Computational Geometry*. Amsterdam: North-Holland, pp. 291–332.

Bongini P, **Bianchini M and Scarselli F** (2021) Molecular generative graph neural networks for drug discovery. *Neurocomputing 450*, 242–252.

Brunton SL, **Proctor JL and Kutz JN** (2016) Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences 113*(15), 3932–3937.

Bui-Thanh T, **Willcox K and Ghattas O** (2008) Model reduction for large-scale systems with high-dimensional parametric input space. *SIAM Journal on Scientific Computing 30*(6), 3270–3288.

Carlberg K, **Barone M and Antil H** (2017) Galerkin v. least-squares Petrov-Galerkin projection in nonlinear model reduction. *Journal of Computational Physics 330*, 693–734.

Carlberg K, **Bou-Mosleh C and Farhat C** (2011) Efficient non-linear model reduction via a least-squares Petrov–Galerkin projection and compressive tensor approximations. *International Journal for Numerical Methods in Engineering 86*(2), 155–181.

Carlberg K, **Farhat C**, **Cortial J and Amsallem D** (2013) The GNAT method for nonlinear model reduction: Effective implementation and application to computational fluid dynamics and turbulent flows. *Journal of Computational Physics 242*, 623–647.

Chan J, **Demkowicz L**, **Moser R and Roberts N** (2010) A class of discontinuous Petrov–Galerkin methods. Part V: Solution of 1D burgers and Navier–Stokes equations. *ICES Report 29*, 13.

Chaturantabut S and Sorensen DC (2010) Nonlinear model reduction via discrete empirical interpolation. *SIAM Journal on Scientific Computing 32*(5), 2737–2764.

Chen PY, **Xiang J**, **Cho DH**, **Chang Y**, **Pershing G**, **Maia HT**, **Chiaramonte MM**, **Carlberg K and Grinspun E** (2022) *CROM: Continuous reduced-order modeling of PDEs using implicit neural representations. Preprint, arXiv:2206.02607*.

Clevert DA, **Unterthiner T and Hochreiter S** (2016) Fast and accurate deep network learning by exponential linear units (ELUs). In *International Conference on Learning Representations (ICLR)*.

Dihlmann M, **Drohmann M and Haasdonk B** (2011) Model reduction of parametrized evolution problems using the reduced basis method with adaptive time partitioning. In *Proceedings of the International Conference on Adaptive Modeling and Simulation*.

Doshi-Velez F and Kim B (2017) *Towards a rigorous science of interpretable machine learning. Preprint, arXiv:1702.08608*.

Drmac Z and Gugercin S (2016) A new selection operator for the discrete empirical interpolation method—Improved a priori error bound and extensions. *SIAM Journal on Scientific Computing 38*(2), A631–A648.

Drohmann M, **Haasdonk B and Ohlberger M** (2011) Adaptive reduced basis methods for nonlinear convection–diffusion equations. In *Finite Volumes for Complex Applications VI Problems & Perspectives: FVCA 6, International Symposium, Prague*. Berlin, Heidelberg: Springer, pp. 369–377.

Dutta S, **Rivera-Casillas P**, **Styles B and Farthing MW** (2022) Reduced order modeling using advection-aware autoencoders. *Mathematical and Computational Applications 27*(3), 34.

Eivazi H, **Le Clainche S**, **Hoyas S and Vinuesa R** (2022) Towards extraction of orthogonal and parsimonious non-linear modes from turbulent flows. *Expert Systems with Applications 202*, 117038.

Farhat C, **Chapman T and Avery P** (2015) Structure-preserving, stability, and accuracy properties of the energy-conserving sampling and weighting method for the hyper reduction of nonlinear finite element dynamic models. *International Journal for Numerical Methods in Engineering 102*(5), 1077–1110.

Fey M and Lenssen JE (2019) *Fast graph representation learning with PyTorch Geometric. Preprint, arXiv:1903.02428*.

Franco N, **Manzoni A and Zunino P** (2023) A deep learning approach to reduced order modelling of parameter dependent partial differential equations. *Mathematics of Computation 92*(340), 483–524.

Fresca S, **Dede L and Manzoni A** (2021) A comprehensive deep learning-based approach to reduced order modeling of nonlinear time-dependent parametrized PDEs. *Journal of Scientific Computing 87*(2), 1–36.

Fresca S and Manzoni A (2022) POD-DL-ROM: Enhancing deep learning-based reduced order models for nonlinear parametrized PDEs by proper orthogonal decomposition. *Computer Methods in Applied Mechanics and Engineering 388*, 114181.

Fries WD, **He X and Choi Y** (2022) Lasdi: Parametric latent space dynamics identification. *Computer Methods in Applied Mechanics and Engineering 399*, 115436.

Fukami K, **Nakamura T and Fukagata K** (2020) Convolutional neural network based hierarchical autoencoder for nonlinear mode decomposition of fluid field data. *Physics of Fluids 32*(9), 095110.

Gao H and Ji S (2019) Graph U-nets. In *International Conference on Machine Learning, Proceedings of Machine Learning Research 97*, 2083–2092.

Geelen R and Willcox K (2022) Localized non-intrusive reduced-order modelling in the operator inference framework. *Philosophical Transactions of the Royal Society A 380*(2229), 20210206.

Geelen R, **Wright S and Willcox K** (2023) Operator inference for non-intrusive model reduction with quadratic manifolds. *Computer Methods in Applied Mechanics and Engineering 403*, 115717.

Geuzaine C and Remacle JF (2009) Gmsh: A 3-D finite element mesh generator with built-in pre-and post-processing facilities. *International Journal for Numerical Methods in Engineering 79*(11), 1309–1331.

Glorot X and Bengio Y (2010) Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics. Proceedings of Machine Learning Research 9*, 249–256.

Gruber A, **Gunzburger M**, **Ju L and Wang Z** (2022) A comparison of neural network architectures for data-driven reduced-order modeling. *Computer Methods in Applied Mechanics and Engineering 393*, 114764.

Hamilton WL (2020) *Graph Representation Learning*. San Rafael, CA: Morgan & Claypool Publishers.

Hamilton WL, **Ying Z and Leskovec J** (2017) Inductive representation learning on large graphs. *Advances in Neural Information Processing Systems 30*.

Hartman D and Mestha LK (2017) A deep learning framework for model reduction of dynamical systems. In *2017 IEEE Conference on Control Technology and Applications (CCTA)*, IEEE, 1917–1922.

Hasegawa K, **Fukami K**, **Murata T and Fukagata K** (2020) Machine-learning-based reduced-order modeling for unsteady flows around bluff bodies of various shapes. *Theoretical and Computational Fluid Dynamics 34*(4), 367–383.

He X, **Choi Y**, **Fries WD**, **Belof JL and Chen JS** (2023) gLaSDI: Parametric physics-informed greedy latent space dynamics identification. *Journal of Computational Physics 489*, 112267.

He H and Zou R (2021) *functorch: JAX-like composable function transforms for PyTorch.* https://github.com/pytorch/functorch.

Kang YE, **Yang S and Yee K** (2022) Physics-aware reduced-order modeling of transonic flow via β-variational autoencoder. *Physics of Fluids 34*(7), 076103.

Kashima K (2016) Nonlinear model reduction by deep autoencoder of noise response data. In *2016 IEEE 55th Conference on Decision and Control (CDC)*. IEEE, pp. 5750–5755.

Khodabakhshi P and Willcox K (2022) Non-intrusive data-driven model reduction for differential–algebraic equations derived from lifting transformations. *Computer Methods in Applied Mechanics and Engineering 389*, 114296.

Kim B, **Azevedo VC**, **Thuerey N**, **Kim T**, **Gross M and Solenthaler B** (2019) Deep fluids: A generative network for parameterized fluid simulations. *Computer Graphics Forum 38*(2), 59–70.

Kim Y, **Choi Y**, **Widemann D and Zohdi T** (2022) A fast and accurate physics-informed neural network reduced order model with shallow masked autoencoder. *Journal of Computational Physics 451*, 110841.

Kingma DP and Ba J (2014) *Adam: A method for stochastic optimization. Preprint, arXiv:1412.6980*.

Kurganov A and Tadmor E (2002) Solution of two-dimensional Riemann problems for gas dynamics without Riemann problem solvers. *Numerical Methods for Partial Differential Equations: An International Journal 18*(5), 584–608.

Lam SK, **Pitrou A and Seibert S** (2015) Numba: A LLVM-based python JIT compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, pp. 1–6.

Lax PD and Liu XD (1998) Solution of two-dimensional Riemann problems of gas dynamics by positive schemes. *SIAM Journal on Scientific Computing 19*(2), 319–340.

Lee K and Carlberg K (2020) Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders. *Journal of Computational Physics 404*, 108973.

Lee K and Carlberg K (2021) Deep conservation: A latent-dynamics model for exact satisfaction of physical conservation laws. *Proceedings of the AAAI Conference on Artificial Intelligence 35*(1), 277–285.

LeVeque RJ (2002) *Finite Volume Methods for Hyperbolic Problems. Cambridge Texts in Applied Mathematics*. Cambridge, UK: Cambridge University Press.

Lieu T, **Farhat C and Lesoinne M** (2006) Reduced-order fluid/structure modeling of a complete aircraft configuration. *Computer Methods in Applied Mechanics and Engineering 195*(41), 5730–5742.

Liska R and Wendroff B (2003) Comparison of several difference schemes on 1D and 2D test problems for the Euler equations. *SIAM Journal on Scientific Computing 25*(3), 995–1017.

MacQueen J (1967) Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability.* Berkeley, CA: University of California Press, *1*, 281–297.

Madenci E and Oterkus E (2013) *Peridynamic Theory and its Applications*. New York: Springer.

Magargal LK, **Khodabakhshi P**, **Rodriguez SN**, **Jaworski JW and Michopoulos JG** (2025a) Replication Data for: Projection based model-order reduction via graph autoencoders suited for unstructured meshes. https://doi.org/10.5281/zenodo.17088969, zenodo, V2.

Magargal LK, **Khodabakhshi P**, **Rodriguez SN**, **Jaworski JW and Michopoulos JG** (2025b) *Replication Data for: Projection based model-order reduction via graph autoencoders suited for unstructured meshes.* https://github.com/liam-magargal/GeometricDeepLSPG.

Maulik R, **Lusch B and Balaprakash P** (2021) Reduced-order modeling of advection-dominated systems with recurrent neural networks and convolutional autoencoders. *Physics of Fluids 33*(3), 037106.

Mazumder S (2015) *Numerical Methods for Partial Differential Equations: Finite Difference and Finite Volume Methods*. Cambridge, MA: Academic Press.

McQuarrie SA, **Huang C and Willcox K** (2021) Data-driven reduced-order models via regularised operator inference for a single-injector combustion process. *Journal of the Royal Society of New Zealand 51*(2), 194–211.

Monaghan JJ (1992) Smoothed particle hydrodynamics. *Annual Review of Astronomy and Astrophysics 30*, 543–574.

Moore B (1981) Principal component analysis in linear systems: Controllability, observability, and model reduction. *IEEE Transactions on Automatic Control 26*(1), 17–32.

Nair NJ and Balajewicz M (2019) Transported snapshot model order reduction approach for parametric, steady-state fluid flows containing parameter-dependent shocks. *International Journal for Numerical Methods in Engineering 117*(12), 1234–1262.

Newman ME, **Watts DJ and Strogatz SH** (2002) Random graph models of social networks. *Proceedings of the National Academy of Sciences 99*, 2566–2572.

Nishikawa H and Kitamura K (2008) Very simple, carbuncle-free, boundary-layer-resolving, rotated-hybrid Riemann solvers. *Journal of Computational Physics 227*(4), 2560–2581.

Nocedal J and Wright SJ (1999) *Numerical Optimization*. New York, NY: Springer.

Ohlberger M and Rave S (2013) Nonlinear reduced basis approximation of parameterized evolution equations via the method of freezing. *Comptes Rendus Mathematique 351*(23-24), 901–906.

Paardekooper SJ (2017) Multidimensional upwind hydrodynamics on unstructured meshes using graphics processing units–I. Two-dimensional uniform meshes. *Monthly Notices of the Royal Astronomical Society 469*(4), 4306–4340.

Pan S, **Brunton SL and Kutz JN** (2023) Neural implicit flow: A mesh-agnostic dimensionality reduction paradigm of spatiotemporal data. *Journal of Machine Learning Research 24*(41), 1–60.

Paszke A, **Gross S**, **Massa F**, **Lerer A**, **Bradbury J**, **Chanan G**, **Killeen T**, **Lin Z**, **Gimelshein N**, **Antiga L**, *et al.* (2019) Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems 32*.

Peherstorfer B (2020) Model reduction for transport-dominated problems via online adaptive bases and adaptive sampling. *SIAM Journal on Scientific Computing 42*(5), A2803–A2836.

Peherstorfer B (2022) Breaking the Kolmogorov barrier with nonlinear model reduction. *English (US). Notices of the American Mathematical Society 69*(5), 725–733.

Peherstorfer B and Willcox K (2016) Data-driven operator inference for nonintrusive projection-based model reduction. *Computer Methods in Applied Mechanics and Engineering 306*, 196–215.

Pichi F, **Moya B and Hesthaven JS** (2024) A graph convolutional autoencoder approach to model order reduction for parametrized PDEs. *Journal of Computational Physics 501*, 112762.

Qian E, **Kramer B**, **Peherstorfer B and Willcox K** (2020) Lift & learn: Physics-informed machine learning for large-scale nonlinear dynamical systems. *Physica D: Nonlinear Phenomena 406*, 132401.

Reddy J (2005) *An Introduction to the Finite Element Method*. New York, NY: McGraw-Hill Education.

Regazzoni F, **Dedè L and Quarteroni A** (2019) Machine learning for fast and reliable solution of time-dependent differential equations. *Journal of Computational Physics 397*, 108852.

Ren YX (2003) A robust shock-capturing scheme based on rotated Riemann solvers. *Computers & Fluids 32*(10), 1379–1403.

Rennie B and Dobson A (1969) On Stirling numbers of the second kind. *Journal of Combinatorial Theory 7*(2), 116–121.

Rewieński MJ (2003) *A Trajectory Piecewise-Linear Approach to Model Order Reduction of Nonlinear Dynamical Systems*. PhD thesis, Massachusetts Institute of Technology.

Rezaian E and Duraisamy K (2023) Data-driven balanced truncation for predictive model order reduction of Aeroacoustic response. *AIAA Journal 61*(10), 4524–4545.

Rodriguez SN, **Iliopoulos AP**, **Carlberg K**, **Brunton SL**, **Steuben JC and Michopoulos JG** (2022) Projection-tree reduced order modeling for fast N-body computations. *Journal of Computational Physics 459*, 111141.

Roe P (1981) Approximate Riemann solvers, parameter vectors, and difference schemes. *Journal of Computational Physics 43*(2), 357–372.

Rusch TK, **Bronstein MM and Mishra S** (2023) *A survey on oversmoothing in graph neural networks. arXiv Preprint, arXiv: 2303.10993*.

Schulz-Rinne CW, **Collins JP and Glaz HM** (1993) Numerical solution of the Riemann problem for two-dimensional gas dynamics. *SIAM Journal on Scientific Computing 14*(6), 1394–1414.

Simonyan K, **Vedaldi A and Zisserman A** (2013) *Deep inside convolutional networks: Visualising image classification models and saliency maps. arXiv Preprint, arXiv:1312.6034*.

Sirovich L (1987) Turbulence and the dynamics of coherent structures part I: Coherent structures. *Quarterly of Applied Mathematics 45*(3), 561–571.

Taira K, **Brunton SL**, **Dawson STM**, **Rowley CW**, **Colonius T**, **McKeon BJ**, **Schmidt OT**, **Gordeyev S**, **Theofilis V and Ukeiley LS** (2017) Modal analysis of fluid flows: An overview. *AIAA Journal 55*(12), 4013–4041.

Von Luxburg U (2007) A tutorial on spectral clustering. *Statistics and Computing 17*, 395–416.

Wei YC and Cheng CK (1989) Towards efficient hierarchical designs by ratio cut partitioning. In *1989 IEEE International Conference on Computer-Aided Design. Digest of Technical Papers*, IEEE, 298–301.

Welper G (2017) Interpolation of functions with parameter dependent jumps by transformed snapshots. *SIAM Journal on Scientific Computing 39*(4), A1225–A1250.

Wentland CR, **Duraisamy K and Huang C** (2023) Scalable projection-based reduced-order models for large multiscale fluid systems. *AIAA Journal 61*(10), 4499–4523.

Wiewel S, **Becher M and Thuerey N** (2019) Latent space physics: Towards learning the temporal evolution of fluid flow. *Computer Graphics Forum 38*(2), 71–82.

Zhou J, **Cui G**, **Hu S**, **Zhang Z**, **Yang C**, **Liu Z**, **Wang L**, **Li C and Sun M** (2020) Graph neural networks: A review of methods and applications. *AI Open 1*, 57–81.

Appendix A. Architecture details and training

All models are trained on an NVIDIA L40S GPU. We initialize all weights and biases using Xavier initialization (Glorot and Bengio, 2010). No attempt was made to mitigate oversmoothing (or overdiffusion) for any of the models, which can be a concern for extremely deep GNNs. The reader is directed to Rusch et al (2023) for an in-depth study on quantifying and mitigating oversmoothing for GNNs. The details of the graph and CNN-based autoencoders for the test problems in Section 5 are provided in Tables A1 and A2, respectively. In all test problems, the loss is evaluated using Eq. (3.19). Stochastic gradient descent is performed using the Adam optimizer (Kingma and Ba, 2014) to update the weights and biases at each epoch. The learning rate for all problems is chosen to be 10^{-4}, and the activation functions are taken to be ELU (Clevert et al., 2016).

To train the autoencoders in the 1D Burgers' model, we first perform a training/validation split, where 4000 solution states are stored for validation, and the remaining 36080 solution states are used for training. We train the models for 1000 epochs where, at each epoch, the training set is passed through the autoencoder in batches of 20. In training the *graph autoencoder*, we noticed that the architecture struggles to model the solution state near the boundaries, especially when the shock approaches the boundary. We believe that this issue is related to the nonlocality of our graph autoencoder, as the boundary nodes do not receive adequate information from the space outside of the domain. This behavior is often found across the field of nonlocal modeling, including peridynamics (Madenci and Oterkus, 2013) and smoothed-particle hydrodynamics (Monaghan, 1992). We leave this as an open area for future investigation, but for now, we present a simple procedure for including padding nodes in a graph autoencoder. This procedure appends 30 nodes to the left side of the domain and sets their features to be the value of the left boundary condition, that is, μ_1. Along the right boundary, we find that solving Burgers' model with the finite volume solver for 30 finite volume cells to the right of the right boundary and prescribing the computed velocity values to the features of the padding nodes is appropriate. Our decoder reconstructs the solution for the nodes in the physical domain as well as those in the padding zones but only computes the loss with respect to the nodes in the physical domain of the problem. During the hierarchical spectral clustering algorithm, radius r^i for layer i is chosen such that Eq. (3.1) gives roughly seven edges for each node, that is, $r^i = \left(x_{\text{right}} - x_{\text{left}}\right)\left(\frac{7}{2|\mathcal{V}^i|}\right)$, where $x_{\text{right}} \in \mathbb{R}$ and $x_{\text{left}} \in \mathbb{R}$ are the positions of the rightmost and leftmost padding nodes in $\mathcal{V}^0$, respectively. To train the *CNN-based autoencoder*, a kernel size of 25 is chosen at each layer, where half-padding is used. In the decoder, the transposed convolution layers are given an output padding of 1.

In the 2D Euler equations for the Riemann problem setup, we first perform a training/validation split, where 525 solution states are stored for validation, and the remaining 7000 solution states are used for training. We train the model for 5000 epochs, where, at each epoch, the training set is passed through the autoencoder in batches of 20. To compute the hierarchy of reduced graphs in the *graph autoencoder*, at each layer, Eq. (3.1) uses a radius that aims for nine edges for each node. We found that padding along the boundaries was unnecessary for this problem and therefore did not include it. In the *CNN-based autoencoder*, a kernel size of 5×5 is chosen at each layer, where half-padding is used. Stride is taken as 2 for all layers. In the decoder, the transposed convolution layers are given an output padding of 1.

To train the *graph autoencoders* used in the 2D Euler equations for the bow shock generated by flow past a cylinder problem, we first perform a training/validation split, where 506 solution states are stored for validation, and the remaining 5500 solution states are used for training. We train the model for 5000 epochs where, at each epoch, the training set is passed through the autoencoder in batches of 20. To compute the hierarchy of reduced graphs, at each layer, Eq. ((3.1) uses a radius that aims for nine edges for each node, that is, $r^i = \sqrt{\frac{9}{\pi|\mathcal{V}^i|}}$. We found that padding along the boundaries was unnecessary for these models, and therefore did not include it.

We found stacking message passing operations to be beneficial to the accuracy of the graph autoencoders. In the MPP layers, we perform message passing operations multiple times before pooling. In the UMP layers, multiple message passing operations are performed after unpooling. The number of message passing operations is represented in Table A1 under the "# of MP operations" column. Lastly, the final message passing operation of the decoder does not have an activation function associated with it, as the output of the last UMP layer should fall in the range $[0, 1]$ because of the rescaling operations.

We additionally report the total number of tunable parameters for each layer of the autoencoders. A single SageCONV message passing operation consists of $2N_F^{i-1}N_F^i$ tunable parameters. When we stack multiple message passing operations in the same MPP layer, we ultimately have $2N_F^{i-1}N_F^i + 2N^{MPP}N_F^iN_F^i$ tunable parameters in the i^{th} layer, where $N^{MPP} \in \mathbb{N}$ denotes the number of additional message passing operations in an MPP layer. Likewise, when we stack multiple message passing operations in the same UMP layer, we will have $2N_F^{i-1}N_F^i + 2N^{UMP}N_F^{i-1}N_F^{i-1}$ tunable parameters in the i^{th} layer, where $N^{UMP} \in \mathbb{N}$ denotes the number of additional message passing operations in a UMP layer. For both CNN layers and transposed CNN layers, the total number of parameters in the i^{th} layer is $N_F^{i-1}N_F^i\iota + N_F^i$, where $\iota \in \mathbb{N}$ denotes the kernel size. Note that the CNN-based autoencoders use biases in the MLP/fully connected layers, whereas the graph autoencoders do not.

A.1. Details of the graph autoencoders

Table A1. *Outputs of each layer of the encoder and decoder of the graph autoencoder for all test problems, where* i *represents the layer number,* $|\mathcal{V}^i|$ *denotes the number of nodes in the output graph,* N_F^i *denotes the number of features for each node in the output graph, and* M *denotes the dimension of the latent space. The number of nodes in the output graph of the preprocessing and postprocessing layers in the 1D Burgers' model includes the 30 padding nodes on both sides of the domain. Parentheses in "# of MP operations" column denote the number of features in the output of intermediate message passing operations*

		i	Layer type	$\lvert\mathcal{V}^i\rvert$	N_F^i	# of MP operations	Vector length	# of tunable parameters
1D Burgers' model	Encoder	0	Preprocessing	316	1	N/A	N/A	N/A
		1	MPP	64	8	2 (8)	N/A	144
		2	MPP	16	16	2 (16)	N/A	768
		3	MPP	4	32	2 (32)	N/A	3072
		4	MPP	2	64	2 (64)	N/A	12288
		5	MLP_{enc}	N/A	N/A	N/A	M	$128M$
	Decoder	0	MLP_{dec}	2	64	N/A	N/A	$128M$
		1	UMP	4	32	2 (64)	N/A	12288
		2	UMP	16	16	2 (32)	N/A	3072
		3	UMP	64	8	2 (16)	N/A	768
		4	UMP	316	1	2 (8)	N/A	144
		5	Postprocessing	N/A	N/A	N/A	316	N/A
2D Riemann problem	Encoder	0	Preprocessing	4328	4	N/A	N/A	N/A
		1	MPP	512	16	2 (16)	N/A	640
		2	MPP	64	64	2 (64)	N/A	10240
		3	MPP	8	128	2 (128)	N/A	49152
		4	MPP	2	256	2 (256)	N/A	196608
		5	MLP_{enc}	N/A	N/A	N/A	M	$512M$
	Decoder	0	MLP_{dec}	2	256	N/A	N/A	$512M$
		1	UMP	8	128	2 (256)	N/A	196608
		2	UMP	64	64	2 (128)	N/A	49152
		3	UMP	512	16	2 (64)	N/A	10240
		4	UMP	4328	4	2 (16)	N/A	640
		5	Postprocessing	N/A	N/A	N/A	4328 × 4	N/A
Bow shock problem	Encoder	0	Preprocessing	4148	4	N/A	N/A	N/A
		1	MPP	512	16	2 (16)	N/A	640
		2	MPP	64	64	2 (64)	N/A	10240
		3	MPP	8	128	2 (128)	N/A	49152
		4	MPP	2	256	2 (256)	N/A	196608
		5	MLP_{enc}	N/A	N/A	N/A	M	$512M$
	Decoder	0	MLP_{dec}	2	256	N/A	N/A	$512M$
		1	UMP	8	128	2 (256)	N/A	196608
		2	UMP	64	64	2 (128)	N/A	49152
		3	UMP	512	16	2 (64)	N/A	10240
		4	UMP	4148	4	2 (16)	N/A	640
		5	Postprocessing	N/A	N/A	N/A	4148 × 4	N/A

A.2. Details of the CNN-based autoencoders

Table A2. *Outputs of each layer of the encoder and decoder of the CNN-based autoencoder for test problems 1 and 2, where* i *denotes the layer number, and* M *denotes the dimension of the latent space. Note: the MLP_{dec} in the CNN-based autoencoder has a bias term and is followed by an ELU activation function (unlike the graph autoencoder). Empirically, we found that the inclusion of the bias term and activation function slightly improves accuracy for the CNN-based autoencoder, but not the graph autoencoder. Note that the CNN-based autoencoder for the 2D Riemann problem has 4096 grid points in the interpolated mesh versus the 4328 cells in the original mesh*

		i	Layer type	Grid points	Channels	Stride	Vector length	# of tunable parameters
1D Burgers' model	Encoder	0	Preprocessing	256	1	N/A	N/A	N/A
		1	1D convolution	128	8	2	N/A	208
		2	1D convolution	32	16	4	N/A	3216
		3	1D convolution	8	32	4	N/A	12832
		4	1D convolution	2	64	4	N/A	51264
		5	MLP_{enc}	N/A	N/A	N/A	M	$129M$
	Decoder	0	MLP_{dec}	2	64	N/A	N/A	$128(M+1)$
		1	1D transposed convolution	8	32	4	N/A	51232
		2	1D transposed convolution	32	16	4	N/A	12816
		3	1D transposed convolution	128	8	4	N/A	3208
		4	1D transposed convolution	256	1	2	N/A	201
		5	Postprocessing	N/A	N/A	N/A	256	N/A
2D Riemann problem	Encoder	0	Preprocessing	4096	4	N/A	N/A	N/A
		1	2D convolution	1024	8	2	N/A	208
		2	2D convolution	256	16	2	N/A	3216
		3	2D convolution	64	32	2	N/A	12832
		4	2D convolution	16	64	2	N/A	51264
		5	MLP_{enc}	N/A	N/A	N/A	M	$129M$
	Decoder	0	MLP_{dec}	16	64	N/A	N/A	$128(M+1)$
		1	2D transposed convolution	64	32	2	N/A	51232
		2	2D transposed convolution	256	16	2	N/A	12816
		3	2D transposed convolution	1024	8	2	N/A	3208
		4	2D transposed convolution	4096	4	2	N/A	204
		5	Postprocessing	N/A	N/A	N/A	4096×4	N/A

Appendix B. Proper orthogonal decomposition

To compute the set of orthonormal POD basis vectors, we use the method of snapshots (Sirovich, 1987) in which a snapshot matrix of the time history of the FOM solutions is generated, $\mathbf{X}^{POD} = [\mathbf{x}^1, \ldots, \mathbf{x}^{N_{\text{train}}}] \in \mathbb{R}^{N \times N_{\text{train}}}$, where $N_{\text{train}} \in \mathbb{N}$ is the number of training snapshots, $\mathbf{x}^i \in \mathbb{R}^N$ is the solution state vector of snapshot i with N being the dimension of the state vector. Next, singular value decomposition is performed on the snapshot matrix, $\mathbf{X}^{POD}$:

$$\mathbf{X}^{POD} = \mathbf{V}\boldsymbol{\Sigma}\mathbf{U}^T \tag{B.1}$$

where $\mathbf{V} = [\mathbf{v}_1, \ldots, \mathbf{v}_N] \in \mathbb{R}^{N \times N}$ is a matrix of N orthonormal vectors which represent the POD modes in the order of decreasing singular values, $\boldsymbol{\Sigma} = \text{diag}(\sigma_1, \ldots, \sigma_N) \in \mathbb{R}^{N \times N}$ is the diagonal matrix of singular values ordered as $\sigma_1 \geq \ldots \geq \sigma_N$, and $\mathbf{U} = [\mathbf{u}_1, \ldots, \mathbf{u}_{N_{\text{train}}}] \in \mathbb{R}^{N \times N_{\text{train}}}$ provides information about the time dynamics. The POD basis is created by truncating the first M left singular vectors of the snapshot matrix, that is, $\boldsymbol{\Phi} = [\mathbf{v}_1, \ldots, \mathbf{v}_M] \in \mathbb{R}^{N \times M}$, which is made up of M orthonormal vectors that describe the dominant mode shapes of the system. The POD basis vectors are orthonormal and optimal in the L^2 sense, making them a common choice in the context of PMOR (Taira et al., 2017).

Cite this article: Magargal L, Khodabakhshi P, Rodriguez S, Jaworski J and Michopoulos J (2025). Projection-based model-order reduction via graph autoencoders suited for unstructured meshes. *Data-Centric Engineering*, 6, e52. doi:10.1017/dce.2025.10030