TECHNISCHE UNIVERSITÄT WIEN

Diplomarbeit
Data-Driven Reduced Models for Numerical Simulations

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Diplom-Ingenieurs (Dipl.-Ing. oder DI), eingereicht an der TU Wien, Fakultät für Maschinenwesen und Betriebswissenschaften, von

Mark RIEGLER

Mat.Nr.: 01613683

unter der Leitung von
Univ.Prof. Dr.-Ing. Stefanie Elgeti
Jaewook Lee, M.Sc.

Institut für Leichtbau und Struktur-Biomechanik, E317

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass die vorliegende Arbeit nach den anerkannten Grundsätzen für wissenschaftliche Abhandlungen von mir selbstständig erstellt wurde.
Alle verwendeten Hilfsmittel, insbesondere die zugrunde gelegte Literatur, sind in dieser Arbeit genannt und aufgelistet. Die aus den Quellen wörtlich entnommenen Stellen, sind als solche kenntlich gemacht.
Das Thema dieser Arbeit wurde von mir bisher weder im In- noch Ausland einer Beurteilerin/einem Beurteiler zur Begutachtung in irgendeiner Form als Prüfungsarbeit vorgelegt. Diese Arbeit stimmt mit der von den Begutachterinnen/Begutachtern beurteilten Arbeit überein.

Wien, Juni, 2023

Unterschrift

Contents

List of Symbols

Notation	Description
Greek symbols	
α	Incline of domain for SLDCPE case
λ	Regularization constant
ν	Fluid kinematic viscosity
ϕ	Activation function
φ	Arbitrary feature
$\boldsymbol{\varphi}$	Solution vector of variable φ
ϱ	Fluid density
$\boldsymbol{\theta}$	Neural network parameters
Latin symbols	
$\boldsymbol{b}$	Bias vector of hidden layer
d_1	Geometry parameter of LDC case
d_2	Geometry parameter of LDC case
d_{max}	Largest Euclidean distance to origin
$\boldsymbol{f}$	Body force acting on fluid
i	Parameter configuration index
$\boldsymbol{z}$	Latent vector
J	Objective function
L	Number of neural network layers
$\boldsymbol{h}$	Output of neural network's hidden layer
N_{train}	Number of parameter configurations
$N_{samples}$	Number of training samples of parameter configuration
N_t	Number of timesteps
$\boldsymbol{o}$	Network output vector for parameter configuration
p	Fluid pressure
N_{nodes}	Number of nodes per snapshot for parameter configuration
r	Cylinder radius for Karman vortex street case
t	Time
u	Fluid velocity component in x-direction
u_{in}	Inlet velocity for Karman vortex street case
u_{top}	Amplitude of periodic excitation at top boundary for SLDCPE case
v	Fluid velocity component in y-direction
$\boldsymbol{u}$	Vector of fluid velocity
$\boldsymbol{W}$	Weight matrix of hidden layer

Notation	Description
$\boldsymbol{x}$	Input vector to neural network
x_{ref}	x-coordinate in the reference domain
$\tilde{\boldsymbol{y}}$	Output vector of neural network
y_{ref}	y-coordinate in the reference domain

Acronyms

Notation	Description
CFD	Computational Fluid Dynamics
DL	Deep Learning
FEM	Finite Element Method
LDC	Lid driven cavity flow
nw	Network width
PDE	Partial differential equation
POD	Proper orthogonal decomposition
ReLU	Rectified linear unit
ROM	Reduced order model
SDF	Signed distance function
SLDC	Skewed lid driven cavity flow
SLDCPE	Skewed lid driven cavity flow with periodic excitation
t-SNE	T-distributed Stochastic Neighbour Embedding

List of Figures

List of Tables

1. Introduction

In modern engineering, simulations are an indispensable tool for product design. Many engineering problems are concerned with mathematical models where a set of parameters may characterize its behavior. These parameters include, but are not limited to, geometric features, boundary conditions or material properties [19]. One application for simulations is shape optimization. The goal is to find a shape which is optimal with regards to a certain quantity of interest, e.g. optimizing the shape of an airfoil for drag reduction or attaining a target pressure distribution [44]. The optimality of a shape is evaluated by its corresponding value of an objective function. Typically, the shape is parametrized so that different shapes are obtained by tuning a set of geometry parameters. In the course of the optimization process, forward simulations are run with a set of parameters to yield a corresponding objective function value. The aim is to find a set of parameters whose corresponding objective function value is optimal [16].

In the course of the optimization process the solution to a large number of parameters is sought after. Because performing high-fidelity simulations for this would be prohibitively expensive, other methods have been introduced. Therefore, one needs an efficient and accurate way to produce these solutions. One such method is called reduced modeling, where the goal is to build a simplified model of the high-fidelity system, which can be evaluated magnitudes faster than the original simulation [4, 18]. This reduced model has to be built, which involves an initial one-time computational investment. For this reduced model, there is a trade off between accuracy and evaluation speed [19].

Within reduced order modeling, one can distinguish between model-based and data-based methods. The former uses the underlying model for building a reduced model while the latter builds a reduced model solely from data, for example from simulations of the high-fidelity model [3, 5]. This thesis only discusses the data-driven methods. Readers interested in model-based methods are referred to [28, 18].

The field of machine learning and big data has had success in modeling many spatiotemporal systems and many different methods have emerged for this use case [13, 11, 24]. In particular, deep learning (DL) methods have been developed to alleviate the problems arising from classical methods like the Finite Element Method (FEM). One application is the forward problem, where the solution to a PDE for a set of parameters is sought after. In classical methods, performing a simulation from scratch is needed to yield the solution for this set of parameters. Neural networks have been employed to build and efficiently evaluate such a parameter-to-solution mapping [37].

One of the most widely used reduced order modeling techniques in Computational Fluid Dynamics (CFD) is the Proper Orthogonal Decomposition (POD) [33]. POD as a linear model reduction technique belongs to the projection-based methods, where a set of appropriate basis vectors span an intrinsic solution subspace, which approximates

the full order model's most important features. Thus, linearity for this subspace is assumed. In contrast, deep learning has the capability for nonlinear model reduction where the solution is embedded in a nonlinear manifold rather than a linear subspace [42]. The advantages of nonlinear reduction techniques over its linear counterpart lie in the decreased number of parameters needed and thus a lower storage cost for a given desired accuracy [8, 6, 26, 22].

This work aims to develop a deep learning architecture which can be used as a reduced model for fluid flows. In particular, methods for learning the velocity field are investigated. The goal is to study the effectiveness of these methods and the influence of data preprocessing and network size. In Section 2, the fundamentals of fluid dynamics and the reduced models with neural networks are presented. Section 3 shows the implementation details of the neural network and the data preprocessing. In Section 4, the results of the experiments are discussed. Finally, Section 5 discusses the findings of the experiments and possible improvements of the network.

2. Fundamentals

2.1. Incompressible flow

The problems considered in this thesis comprise incompressible fluid dynamics. This means that the density and the viscosity of the fluid are considered constant. In that case, a fluid's behaviour may be modeled by the following set of partial differential equations (PDEs):

$$\nabla \cdot \boldsymbol{u} = 0 \tag{1a}$$

$$\frac{\partial \boldsymbol{u}}{\partial t} + \nabla \cdot (\boldsymbol{u}\boldsymbol{u}) = -\frac{\nabla p}{\varrho} + \nu \nabla^2 \boldsymbol{u} + \boldsymbol{f} \tag{1b}$$

with $\boldsymbol{u}$ being the fluid velocity, t the time, ϱ the fluid density, p the presssure, ν the kinematic viscosity, and $\boldsymbol{f}$ the body force acting on the fluid [21].

In practice, these equations are solved numerically. Therefore, one relies on discretization methods in time and space. In our simulations, we used the Space-Time Finite Element Method as the discretization method. The interested reader is referred to [38, 39].

2.2. Reduced model using neural networks

Model reduction with neural networks is typically done in two phases: in the first phase, the so-called offline phase, a large number of numerical simulations are performed over a set of parameter samples. The aim is to create a dataset with the results of these simulations. Subsequently, a network is trained to predict the results of the performed flow simulations. In the second phase, the online phase, the trained network can then be used to predict the results of a new flow simulation with, e.g., given input values which were not in the training data. This prediction by evaluating the network is typically magnitudes faster than performing a simulation from scratch and should yield similar results [43].

An artificial neural network is a chain of functions applied to an input vector $\boldsymbol{x}$. One of the most prominent examples of a neural network is a fully-connected feedforward network, where an input vector $\boldsymbol{x}$ is passed through a chain of functions $f^{(l)}$ with $l = 1, \ldots, L$ one after the other to yield an output $\tilde{\boldsymbol{y}}$:

$$\tilde{\boldsymbol{y}} = \left(f^{(L)} \circ f^{(L-1)} \circ \cdots \circ f^{(1)}\right)(\boldsymbol{x}), \tag{2}$$

where L as the number of "layers" is called the network's depth[1]. In particular, the functions $f^{(i)}$ for $i = 1, \dots, L$ consist of a linear mapping and a subsequent application of a nonlinear function. Let $\boldsymbol{h}^0 = \boldsymbol{x}$ denote the input of the neural network. In a fully-connected feedforward network, for $k > 0$, $\boldsymbol{h}^k$ is the output of the k-th hidden layer, which is computed:

$$\boldsymbol{h}^k = \phi\left(\boldsymbol{b}^k + \boldsymbol{W}^k \boldsymbol{h}^{k-1}\right), \tag{3}$$

where $\boldsymbol{h}^{k-1} \in \mathbb{R}^{d_{k-1}}$ and $\boldsymbol{h}^k \in \mathbb{R}^{d_k}$ are called hidden layers, ϕ is called the activation function, a nonlinear function which is applied element-wise, $\boldsymbol{b}^k \in \mathbb{R}^{d_k}$ the bias vector and $\boldsymbol{W}^k \in \mathbb{R}^{d_k \times d_{k-1}}$ the weight matrix. The dimension of a hidden layer $\boldsymbol{h}^k$ is called the width of the layer and the values of $\boldsymbol{b}^k$ and $\boldsymbol{W}^k$ the network parameters $\boldsymbol{\theta}$. Thus, the output of the network $\tilde{\boldsymbol{y}} = \boldsymbol{h}^L$ can be written as the output of the L-th hidden layer [14, 42].

Neural networks are used to approximate a function which maps a set of input vectors $\boldsymbol{x}$ to corresponding output vectors $\boldsymbol{y}$. The goal is to find optimal network parameters $\boldsymbol{\theta}$ such that the function is most closely approximated. This is done my minimizing an objective function $J(\boldsymbol{\theta})$ which penalizes any deviation from the network's output to the ground truth output. [14].

Using a method called backpropagation, the gradient of the objective function with respect to the network parameters (weights and biases) can be efficiently calculated. It makes use of the chain rule of differentiation by computing the gradient starting from the last layer and successively moving backwards to the first layer [42]. The interested reader is referred to [17].

The optimization of the network's parameters with regards to the objective function $J(\boldsymbol{\theta})$ is called training and usually gradient-based optimization algorithms are applied where backpropagation is used to calculate this gradient. In practice, a method called Adam is often used [27, 37, 26]. The interested reader is referred to [23].

In practice, the ReLU-function is a common choice for the activation function [11, 24, 27]. The function is $ReLU(x) = \max\{0, x\}$. Its strengths lie in its simplicity, effectiveness and computational speed, which stems from the fact that there is no need to compute exponentials or divisions. [14, 31]

One of the most appealing properties of deep neural networks is that they are universal approximators: they can approximate any finite-dimensional function up to arbitrary accuracy [20, 34, 36].

Autodecoder Here, we describe the autodecoder network proposed in [27].

[1] Although there is no unified definition of a network's depth [42], for simplicity's sake, we define depth as the number of layers.

The autodecoder is a fully connected feedforward neural network used for generative modeling. In the paper, a neural network is trained to learn signed distance functions[2] (SDFs) of a family of similar 3D-shapes (e.g. cars). The desired outcome is a model which can embed common properties of a set of shapes in a low dimensional latent space, where a specific shape is encoded as a latent vector $\boldsymbol{z}$ in this low dimensional space. Specifically, a function f_θ should be learned which takes a latent code $\boldsymbol{z}_i$ corresponding to some shape i and in addition a queried 3D location $\boldsymbol{x}$ and approximates the shape's SDF:

$$f_\theta(\boldsymbol{z}_i, \boldsymbol{x}) \approx SDF^i(\boldsymbol{x}) \tag{4}$$

The network is trained by trying to minimize a cost function consisting of a sum of reconstruction errors and a regularization term for the latent codes. This yields a single model which is capable of representing multiple shapes in a low-dimensional subspace. Thus, this property can be used for reduced modeling. Furthermore, the network is resolution independent, because any arbitrary query location $\boldsymbol{x}$ can be used as an input to the network.

Because the latent codes $\boldsymbol{z}_i$ and the network parameters $\boldsymbol{\theta}$ are arguments of the cost function, the network parameters and the latent vector can be optimized during training by using backpropagation. Thus, this network does not require a dedicated network to embed the high-dimensional data into a low-dimensional latent space.

[2]The signed distance function is used for implicitly describing a watertight surface. For a given spatial point, the signed distance function yields the distance to the nearest surface. If the point lies inside the surface, it yields a negative value and if it lies outside, it has a positive value.

3. Implementation

3.1. Data generation

To generate data, we performed two-dimensional stationary and instationary fluid flow simulations governed by the incompressible Navier-Stokes equations (Equation 1). We used the space-time method as the time-discretization method. After the simulations were run, the data was taken by extracting the values from the lower time level.

The simulations were performed using XNS, an in-house CFD code developed by the Chair for Computational Analysis of Technical Systems of the RWTH Aachen University.

The meshes were generated using an in-house mesh generator. For each geometry parametrization , we meshed from scratch in order to keep the reduced model independent of the triangulation.

3.1.1. Skewed lid driven cavity (SLDC)

This fluid flow is based on the single lid driven cavity (LDC) flow. It consists of a fluid in a rectangular container where the fluid at the top wall moves with constant velocity in positive x-direction. The velocity at the other boundaries are zero [25].

The skewed lid driven cavity flow (SLDC) [7, 9] is a variant of the standard LDC flow where the geometry is slanted to form a parallelogram.

For the generation of the dataset, we run 1323 simulations, each of which is characterized by two geometry and one flow parameters. The chosen parameter values are displayed in Table 3. The density of 1 kg/m^3 and the constant velocity magnitude of 1 m/s at the top wall is the same for all simulations. The boundary conditions are displayed in Figure 3. We start with zero velocity and pressure everywhere but at the top boundary. Each simulation was run for 500 timesteps with a timestep size of 0.1 s to achieve a quasi-stationary flow.

For the flow parametrization we change the viscosity ν[3]. For the domain, we take a $[0, 1] \times [0, 1]$ square domain and change its width and inclination with two geometry parameters d_1 and d_2 (see Figure 1). The number of nodes is between 4704 and 7966 and the number of elements between 4892 and 8224.

Case with periodic boundary conditions In this case, we apply a periodic velocity boundary condition on the top wall. This results in a more dynamic behavior of the

[3]The viscosities were not equidistantly sampled. If we define the Reynolds number as $Re = \frac{u \cdot L}{\nu}$ with $u = 1$ as the velocity at the top boundary and $L = 1 - d_2$ as the length of the top boundary, the sampled Reynolds numbers are $[200, 400, 800, 1000, 1200, 1500, 2000]$.

fluid. We denote this case as skewed lid driven cavity flow with periodic excitation (SLDCPE). We simulated for 10 s which corresponds to 2 full cycles of this excitation.

The dataset consists of the results of 637 simulations. We keep the fluid properties density and viscosity the same for all configurations: $\varrho = 1\ kg/m^3$ and $\nu = 2.5{\cdot}10^{-3}\ Ns/m^2$. The boundary conditions are the same as the previous case with the sole exception of the velocity at the top boundary, where we prescribe a dynamic boundary condition of

$$u(t) = u_{top} \cdot \left(1 - \cos\left(\frac{2\pi t}{5}\right)\right). \tag{5}$$

The amplitude u_{top} of the periodic excitation is parametrized with values shown in Table 4.

For the geometry parametrization (see Figure 2) we again start with a $[0,1] \times [0,1]$-square domain and change the incline angle α whose values are shown in Table 4. For the mesh, the number of nodes ranges from 8074 to 8312 and the number of elements from 7420 to 7664.

Parameter	Unit	Sample values
ν	Ns/m^2	0.0003 to 0.005
d_1	m	-1.0 to 1.0 in increments of 0.1
d_2	m	0.0 to 0.4 in increments of 0.05

Table 3: Parameter values for stationary SLDC simulations.

Parameter	Unit	Sample values
α	$^\circ$	-45 to 45 in increments of 1
u_{top}	m/s	$\{0.5, 0.7, 1.0, 1.2, 1.5, 2.0, 3.0\}$

Table 4: Parameter values for SLDCPE simulations.

3.1.2. Karman vortex street

The Karman vortex street is a case where fluid flows past a cylinder inducing periodic vortex shedding downstream of the cylinder [2].

We performed 726 simulations for 50 seconds with a timestep size of 0.05 s. The end time is chosen so that all parameter configurations exhibit periodic vortex shedding. For all configurations of this case, we kept a constant dynamic viscosity of $0.02\ Ns/m^2$. The domain and the boundary conditions for our simulations are displayed in Figure 4. The prescribed boundary conditions are an inlet velocity u_{in} and zero velocity at the cylinder. The initial conditions are zero velocity and pressure everywhere but the inelt velocity at the left boundary.

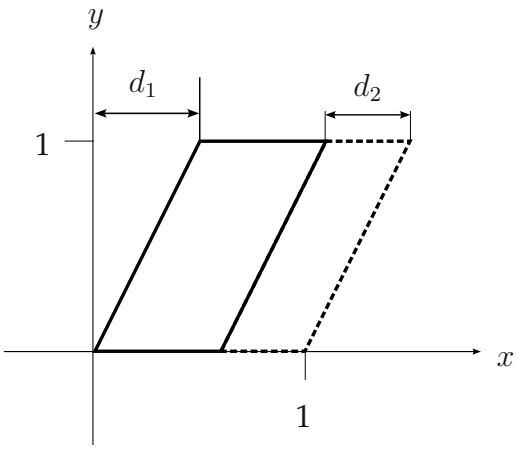


Figure 1: Geometry parametrization of the skewed lid driven cavity. The domain is obtained by taking a $[0, 1] \times [0, 1]$-square domain, shifting the top wall by d_1 in positive x-direction and the right wall by d_2 in negative x-direction.

For flow parametrization, we changed the inlet velocity u_{in} and the fluid density ϱ. For geometry parametrization we changed the cylinder radius r. The sampled parameter values are displayed in Table 5.

The number of nodes ranges from 12960 to 13526 and the number of elements from 12737 to 13305. We have a rectangular box (see Figure 4) around the cylinder where the triangulation of the mesh is finer than outside of it. For the data extraction we only used the values in that box, consisting of 5030 to 5364 nodes, and ignored the other ones.

Parameter	Unit	Sample values
u_{in}	m/s	1.0 to 2.0 in increments of 0.1
ϱ	kg/m^3	2.0 to 4.0 in increments of 0.2
r	m	0.25 to 0.5 in increments of 0.05

Table 5: Parameter values for von-Karman vortex street simulations.

3.2. Data preprocessing

Min-max-normalization This method is used to scale the values of each feature in the dataset to a range of [-1,1]. For an arbitrary feature φ, firstly, we search for the maximum φ_{max} and minimum value φ_{min} in the whole dataset. Finally, a data entry φ_i is scaled as follows:

$$\hat{\varphi}_i = \frac{\varphi_i - \varphi_{min}}{\varphi_{max} - \varphi_{min}} \cdot 2 - 1, \tag{6}$$

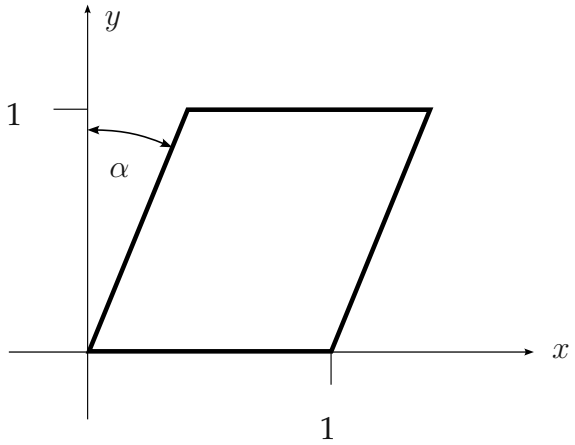


Figure 2: Geometry parametrization of the skewed lid driven cavity with periodic excitation

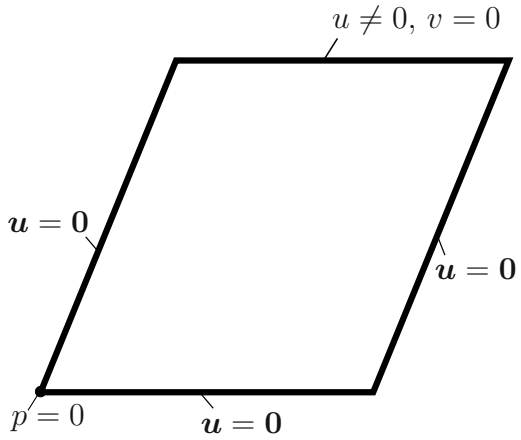


Figure 3: Skewed lid driven boundary conditions

where $\hat{\varphi}_i$ is the normalized value of φ_i.

Min-max normalization allows gradient-based optimization methods a better convergence rate [1].

Here, we apply min-max-normalization to all input- and output variables.

Unitsphere normalization This method is used in [27]. Here, we try to preserve the proportions of the data to each other the same while still scaling to a range of $[-1, 1]$. Suppose the data has m features and n samples. Firstly, mean-centering is applied to to each of the features. For an arbitrary dataset entry φ_i (for $i = 1, \ldots, n$), we denote the mean-centered value as φ_i^*. Then, each of the n samples is treated as a coordinate in $\mathbb{R}^m$. In those n samples, we search for the highest Euclidean distance to the origin d_{max}. Finally, scaling is applied such that all of those coordinates lie within the unitsphere in $\mathbb{R}^m$ which yields the final value:

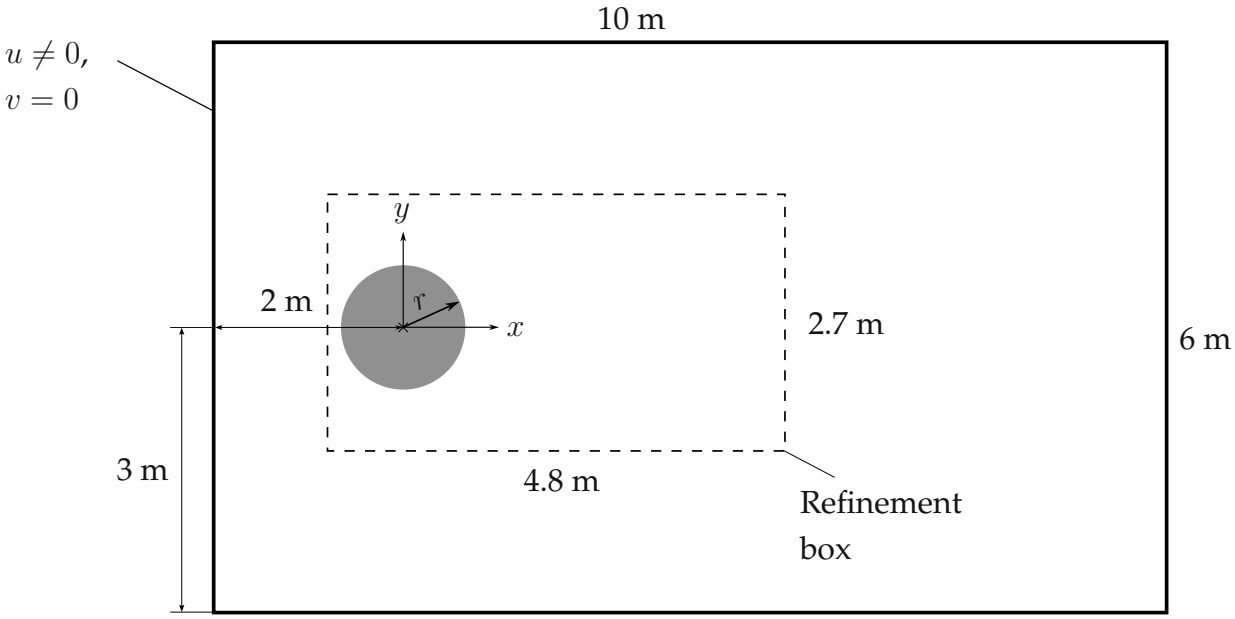


Figure 4: Karman vortex street setup. All but the radius of the cylinder (grey) are to scale. The triangulation inside the refinement box is finer than outside of it. This box is a rectangular domain spanning $[-0.8, 4.0] \times [-1.3, 1.4]$.

$$\hat{\varphi}_i = \frac{\varphi_i^*}{d_{max}}. \tag{7}$$

For our experiments, we apply min-max-normalization on the input variables x_{ref}, y_{ref}, and t. For the output variables u and v, we apply unitsphere-normalization on a configuration per configuration basis such that each parameter configuration has its own scaling.

3.3. Network architecture and training

We closely followed the network setup of [27] to predict two-dimensional flow fields. An overview of the adapted network is shown in Figure 5. The input is a latent code and input variables which are the x- and y-coordinates in a reference domain and a given time t. For a stationary flow, we only pass the latent code and the reference coordinates into the network. Since we are only interested in predicting the flow velocities, the output is a 2-dimensional vector consisting of u and v: the fluid velocities in x- and y-direction respectively. In particular, we want to learn the following mapping:

$$f_\theta : [\boldsymbol{z}_i, x_{ref}, y_{ref}, t] \mapsto \boldsymbol{o}_i, \tag{8}$$

where $\boldsymbol{z}_i$ is a latent vector representing a fluid flow which is determined by a parameter configuration i and $\boldsymbol{o}_i$ the output of the network. For our implementation, it is important to note that the latent code size must be smaller than the network width

because the latent code and the input variables are injected into the fourth layer and the number of values must not exceed the network width. In all our experiments the number of hidden layers is 8. All the layers consist of a fully-connected layer and a ReLU-activation function. However, we replace the ReLU-activation function at the last layer with a tanh-activation function so that negative values are possible for the output.

The loss function consists of the prediction error and a regularization term for the latent codes:

$$\mathcal{L} = \sum_{i=0}^{N_{train}} \frac{1}{N_{samples,i}} \left(\sum_{j=0}^{N_{samples,i}} \| f_\theta \left(\boldsymbol{z}_i, x_{ref}, y_{ref}, t \right) - \boldsymbol{o}_{orig,j} \|_{L_1} + \lambda \| \boldsymbol{z}_i \|_{L_2} \right), \tag{9}$$

where N_{train} is the number of flow field configurations, $N_{samples,i}$ the number of training samples of configuration i, $\boldsymbol{o}_{orig}$ the vector of original values and λ a regularization constant.

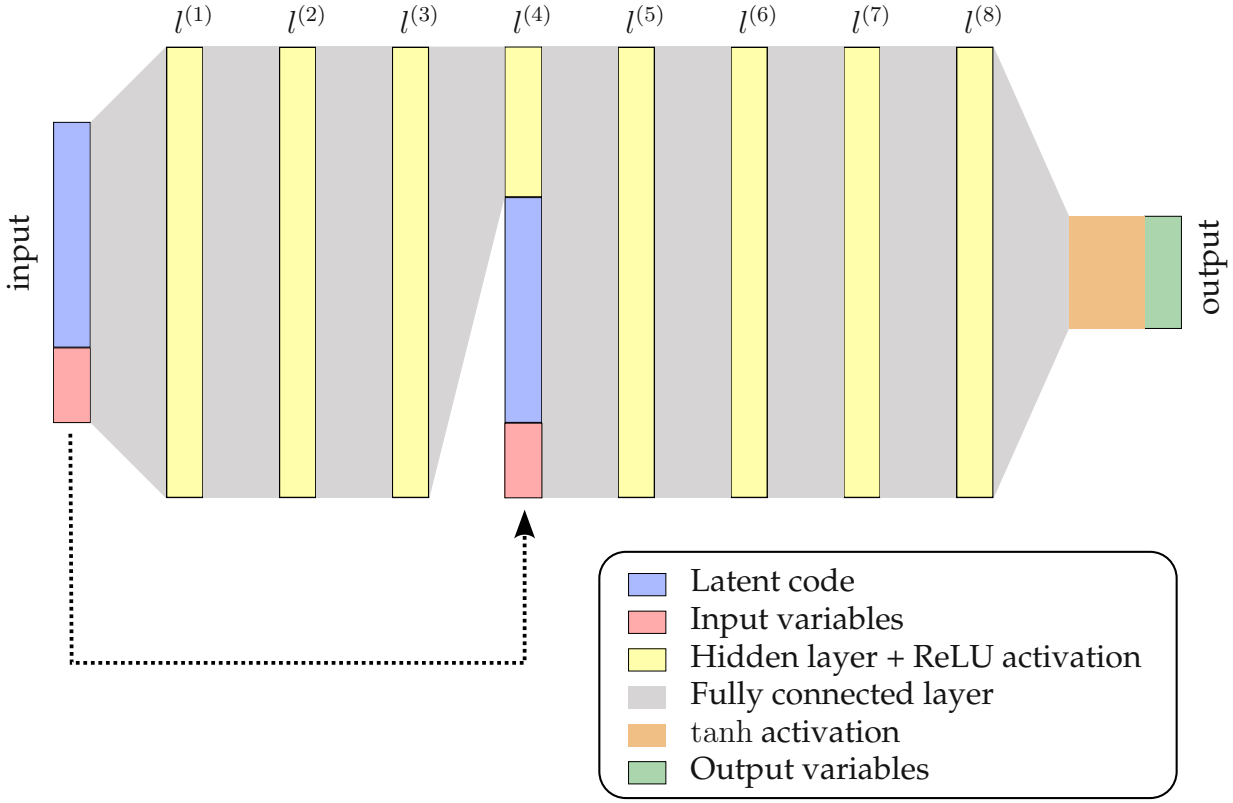


Figure 5: Visualization of autodecoder network architecture. The latent code and the input variables are injected into layer $l^{(4)}$ and concatenated with the output of layer $l^{(3)}$. Note that the output of layer $l^{(3)}$ has less values than that of most other layers so that the network has the same width in every layer.

Similarly to [27], we apply dropout [35] with a dropout probability of 0.2 and weight normalization [32] at all hidden layers. For the optimization of network parameters

and the latent vector, we use the Adam optimizer [23] with a stepwise-decreasing learning rate similar to [27]: after every 250 epochs, the learning rate was halved. We have separate initial values of the learning rate: 0.0005 for optimizing the network parameters and 0.001 for the optimization of the latent code. Every training was run for 2000 epochs. The latent code regularization parameter λ in Equation 9 for all trainings is 0.0001.

For detailed information about the network architecture and training, the reader is referred to [27].

3.4. Data inference

In order to obtain field values, a concatenated vector of a latent code $\boldsymbol{z}$, coordinates x_{ref}, y_{ref} of a point in the reference domain and a given time t, which needs to be normalized, has to be passed through the trained network. Because the output is normalized, the values have to be scaled back by reversing the preprocessing step. Note that the output of the network is in the range $[-1, 1]$. Therefore, the predicted velocities are bounded by the minimum and maximum value found in the training dataset.

Coordinate mapping from reference to physical domain For all three fluid flow cases, the mappings from the reference to the physical domain can be described by a known function which is determined by a set of geometry parameters. The mappings are shown in Table 6.

Case	$x_{ref} \mapsto x_{phys}$	$y_{ref} \mapsto y_{phys}$
SLDC	$\frac{x_{ref}+1}{2} \cdot (1 - d_2) + \frac{y_{ref}+1}{2} \cdot d_1$	$\frac{y_{ref}+1}{2}$
SLDCPE	$\frac{x_{ref}+1}{2} + \frac{y_{ref}+1}{2} \cdot \tan(\alpha)$	$\frac{y_{ref}+1}{2}$
Karman	$-0.8 + 4.8 \cdot \frac{x_{ref}+1}{2}$	$-1.3 + 2.7 \cdot \frac{y_{ref}+1}{2}$

Table 6: Coordinate mapping from reference to physical domain for all three cases.

In order to predict the flow field for a set of parameters which are not in the training dataset, a suitable latent code has to be generated. This can be done by latent space interpolation: the values of the resulting latent code are obtained by performing interpolation of the latent codes obtained from the training where their corresponding parameter values are used as the variables. If the parameter space is bigger than 1, then multivariate interpolation has to be applied. Here, we apply linear multivariate interpolation. This is done by triangulating the parameter samples in parameter space and performing interpolation within the respective triangulation element.

4. Results

4.1. Test case evaluation

Here, the reduced models are applied to the three test cases described in subsection 3.1.

To quantify the accuracy of the reduced models, we adopted the metric from [10], which is defined as:

$$\varepsilon_{rel}\left(\varphi\right) = \frac{1}{N_{train}} \sum_{i=1}^{N_{train}} \left(\frac{\sqrt{\sum_{k=1}^{N_t} \left\| \boldsymbol{\varphi}_k(\boldsymbol{\mu}_i) - \tilde{\boldsymbol{\varphi}}_k(\boldsymbol{\mu}_i) \right\|_2^2}}{\sqrt{\sum_{k=1}^{N_t} \left\| \boldsymbol{\varphi}_k(\boldsymbol{\mu}_i) \right\|_2^2}} \right). \tag{10}$$

where N_{train} denotes the number of parameter configurations which were simulated, N_t the number of timesteps, $\boldsymbol{\varphi}_k$ is the solution vector of variable φ at timestep t_k for $k = 1, \ldots, N_t$, $\boldsymbol{\mu}_i$ the parameter configuration vector for case i and $\tilde{\boldsymbol{\varphi}}$ is the predicted solution vector from the autoencoder.

For the stationary SLDC case, we introduce the following error metric:

$$\varepsilon_{RMSE,mean}(\varphi) = \frac{1}{N_{train}} \sum_{i=1}^{N_{train}} \left(\sqrt{\frac{\sum_{j}^{N_{nodes,i}} \left\| \boldsymbol{\varphi}^j(\boldsymbol{\mu}_i) - \tilde{\boldsymbol{\varphi}}^j(\boldsymbol{\mu}_i) \right\|_2^2}{N_{nodes,i}}} \right), \tag{11}$$

where N_{train} is the number of parameter configurations, $N_{nodes,i}$ is the number of nodes for parameter configuration i, φ a fluid flow variable or a vector of those parameters (e.g. $[u, v]$, where the vectors u and v are stacked horizontally), $\boldsymbol{\varphi}^j(\boldsymbol{\mu}_i)$ the value of variable(s) φ at node j for configuration i, μ_i the parameter configuration vector for configuration i and $\tilde{\boldsymbol{\varphi}}_i^j$ the value obtained from the reduced model.

These measures are used to determine the mean accuracy of a reduced model for a whole dataset of simulation results. Both are based on a relative error. We chose these metrics because different dimensions of the solution vectors and also zero values in the solution vectors are allowed.

Here, the effects of the normalization method, the network width and the size of the latent code are investigated. For each of the three cases, we trained networks with the same training setups but with different values of the aforementioned network parameters. For the stationary SLDC case, we trained smaller networks than the ones of the transient cases, because the stationary case has less training samples simply due to the

fact that time is an additional input dimension for instationary cases. Additionally, the latent code sizes and the network widths are powers of 2.

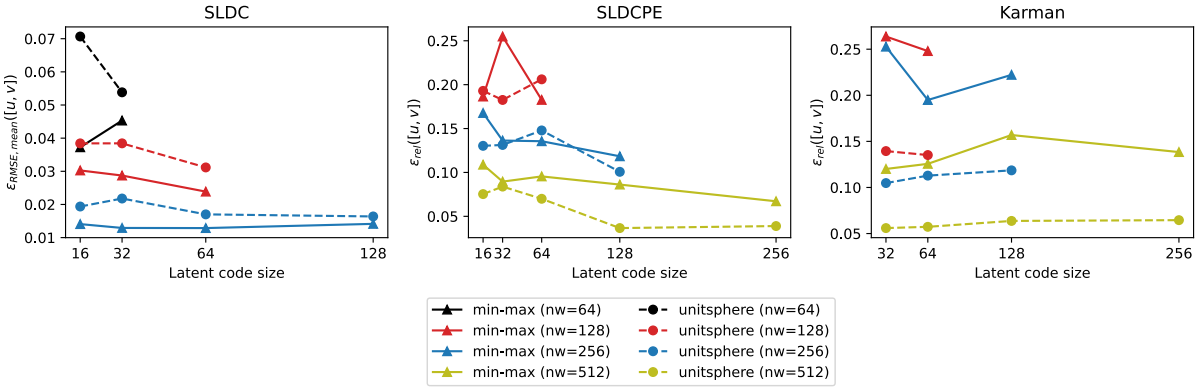


Figure 6: Relative errors of cases. nw denotes the network width. min-max and unitsphere denote the different normalization methods.

The relative errors (Equation 11, Equation 10) of the absolute velocity for all trained networks and cases are displayed in Figure 6.

The effects of the latent code size on the accuracy depend on the fluid flow case. For the cases SLDC and SLDCPE, increasing the latent code size leads to a smaller error for the most part. In the Karman vortex street case however, the increased latent code size attributes to a larger error, especially for networks with greater width. This may be due to the latent injection, where the layer at which the latent code and the input is injected has less incoming full connections than the other layers (see Figure 5). When the latent code is bigger, then the number of incoming full connections is less. The error becoming bigger with greater latent code sizes thus indicates that having more full connections between layers has a greater ability to capture this type of flow.

The type of data normalization also shows different results. While the min-max-normalization in the SLDC case generally performs better, the unitsphere-normalization is more accurate in the transient cases. Especially in the Karman case, the effects on accuracy are more pronounced: the errors of the min-max-normalized data is approximately double the ones of the unitsphere-normalized data for almost all training configurations.

In all cases, an increase in network width leads to a smaller error. This can be attributed to a wider network having more parameters and hence being able to model more complex behaviour.

4.1.1. Common occurring errors

One of the factors most contributing to the error is the failure to correctly predict flow structures like the vortex shedding in the Karman vortex street case or the vortex in the

SLDCPE case. To demonstrate this, we are concentrating on a representative example: the Karman vortex street case with $u_{in} = 1.6$, $\varrho = 1.6$, $r = 0.45$.

Effects of network width For this, we compare the network widths of 256 and 512 for unitsphere-normalization and a latent code size of 32.

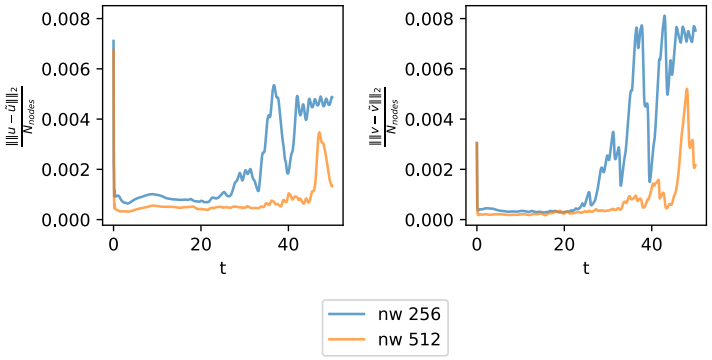


Figure 7: Karman vortex case: error evolution of network widths 256 and 512 (latent code size 32, unitsphere-normalization, case with $u = 1.6$, $\varrho = 2.2$, $r = 0.45$)

Figure 7 displays how the errors of the velocities evolve with time. One noticeable large error occurs at the first timestep, where the initial values are mainly zero. The initial snapshot and the corresponding field predictions and errors are displayed in Figure 8.

The plot also demonstrates that the network with width 256 exhibits larger errors than the one with the larger network width. Both networks show a large increase in error at some time step, where the network with the lower width exhibits this spike at an earlier timestep.

The effects of choosing different network widths are displayed in Figure 9 and Figure 10: both networks capture the motion of vortex shedding while the network with width 256 exhibits larger errors, which mainly stem from the inaccuracies of the predicted vortex shedding. This can be seen in Figure 9 in the absolute error. In Figure 10, the network with width 256 fails to capture the vortex shedding motion and instead outputs a stationary flow. On the other hand, the network with width 512 can still capture the motion of the vortex shedding.

The effects of the latent size and the data normalization type for this parameter configuration are discussed in subsection A.2.

SLDCPE case In subsection A.1, the effects of the network width on the predictions' accuracy of the vortex in the SLDCPE case is presented.

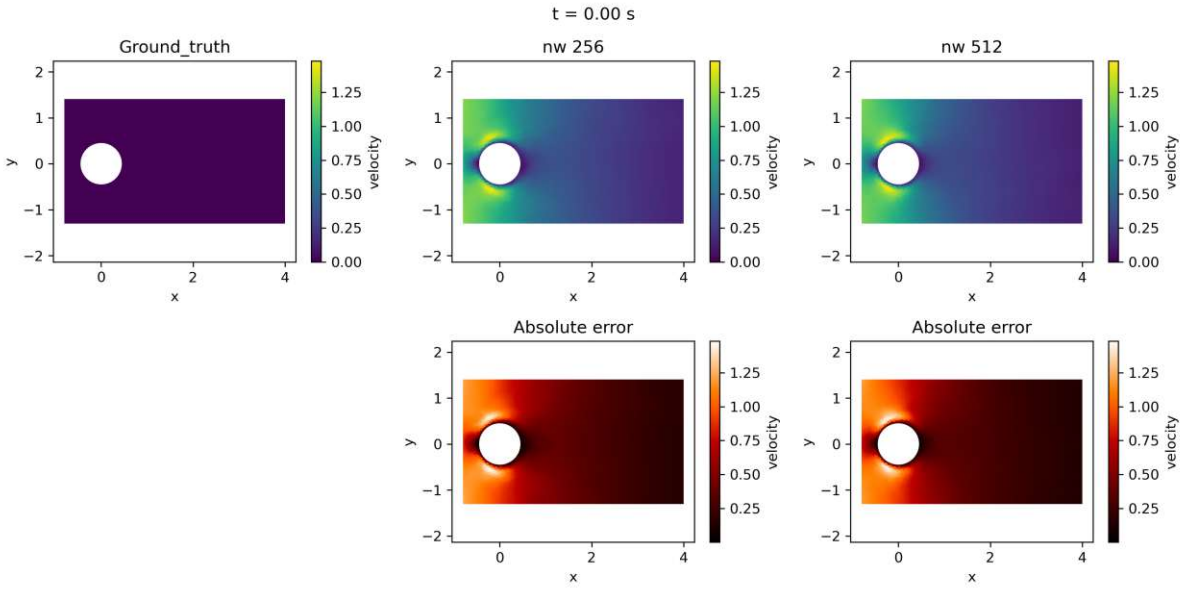


Figure 8: Karman vortex case: velocity prediction and absolute error of network widths 256 and 512, first timestep (latent code size 32, unitsphere-normalization, case with $u = 1.6$, $\varrho = 2.2$, $r = 0.45$)

4.2. Latent space exploration

4.2.1. Data interpolation

To visualize the latent space, which for all our experiments has more than 3 dimensions, we are applying t-distributed Stochastic Neighbor Embedding (t-SNE) [40], a nonlinear unsupervised reduction method used for visualization of high-dimensional data. It attempts to embed high-dimensional data into a lower dimension while preserving the global structure of the data. Interested readers may be referred to [40, 41, 12]. Here, the method is applied to the latent code space to display patterns of the data on a 2D-embedded space. Values of a parameter are marked with different colors to better display the separability.

SLDCPE case Figure 11 displays the application of t-SNE to the 64-dimensional latent vectors obtained for the training of the network with width 512 where the colorization groups the parameters for the flow field together. For both parameters, the separation between the values is clearly visible.

Karman vortex street For the Karman vortex street, we are comparing the network with the lowest error (latent code size of 32 (see 12a) and network width 512) with the network with the same width but with latent code size 256 (see 12b). The latter shows a clear grouping of the parameters' values and a clear structure, especially for

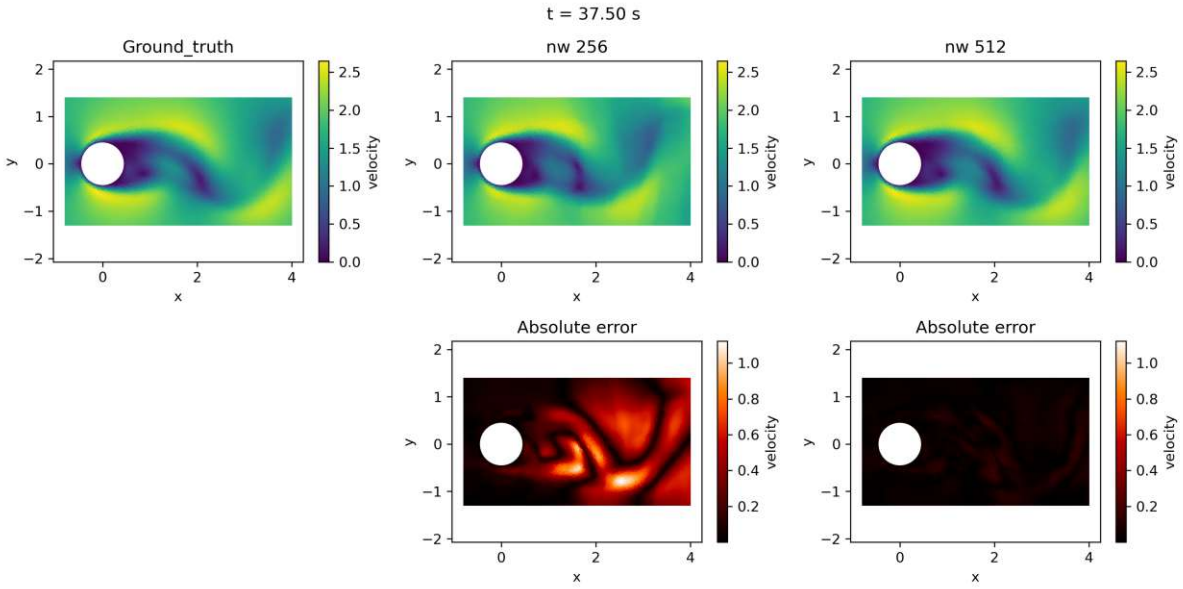


Figure 9: Karman vortex case: velocity prediction and absolute error of network widths 256 and 512, timestep 750 (latent code size 32, unitsphere-normalization, case with $u = 1.6$, $\varrho = 2.2$, $r = 0.45$)

the velocity and the radius, is noticeable. The network with width 32 on the other hand still shows some separability of the clusters but especially for the density, the clusters cannot be easily distinguished.

4.2.2. Solution inference on unseen parameter configurations

SLDCPE case Inference is tested on a square domain ($\alpha = 0°$) and the time step $t = 9.0\ s$ for the following top boundary velocities: $u_{top} \in \{0.6, 0.8, 1.3, 1.6, 2.2, 2.6\}$. These velocity values are not in the training data set and hence interpolation has to be applied. The time step is chosen so that the vortex in the flow is fully developed. The network width is 512 and the latent code size is 256.

Figure 13 shows the inferred field values of the min-max-normalization method. We omitted displaying the inferred velocity field of the unitsphere-normalized data, because there is little to no visible difference.

For the parameter samples used in Figure 13, we compare the effects of the data normalization on the accuracy of the velocity at the top boundary. The comparison of relative errors is displayed in Figure 14. The ground truth is computed using Equation 5 where the values at the left and right boundary are omitted because they are zero.

For both data normalization methods, the velocity profile is captured. Regarding the relative error, the unitsphere-normalized data exhibited approximately the same relative errors across all six cases. The min-max-normalized data on the other hand shows

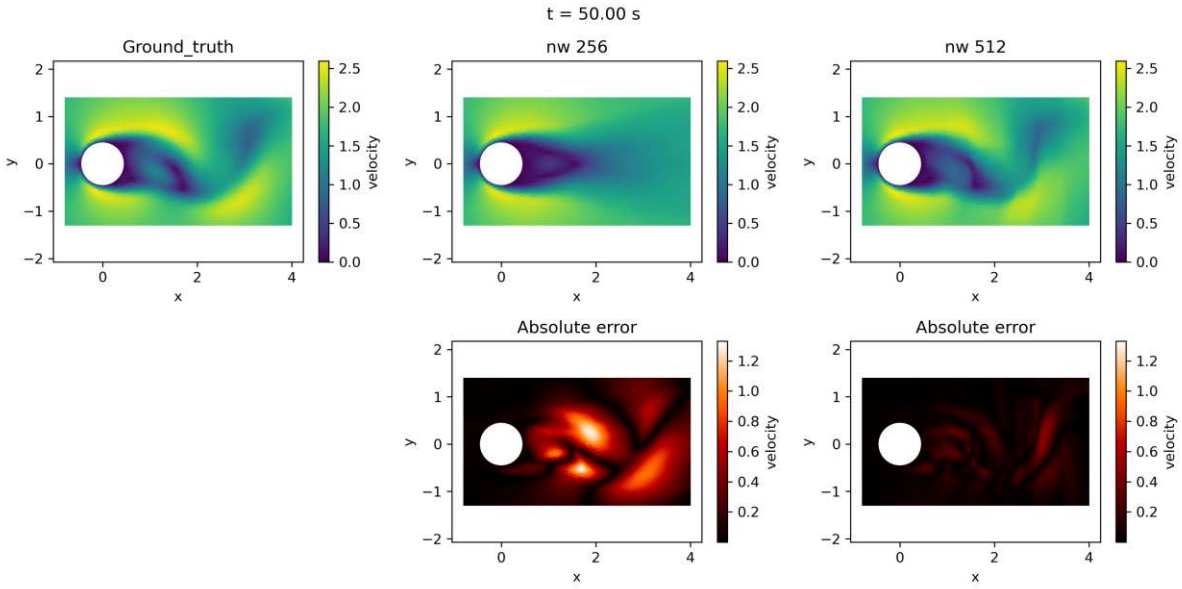


Figure 10: Karman vortex case: velocity prediction and absolute error of network widths 256 and 512, last timestep (latent code size 32, unitsphere-normalization, case with $u = 1.6$, $\varrho = 2$, $r = 0.45$)

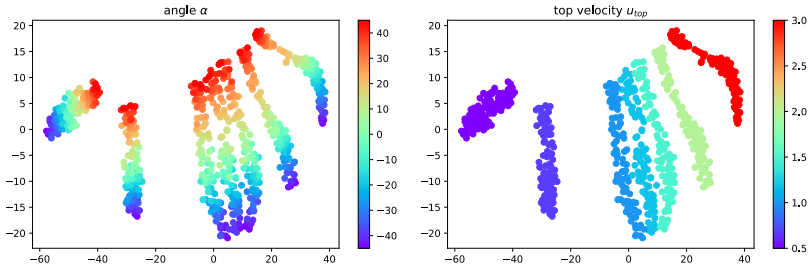
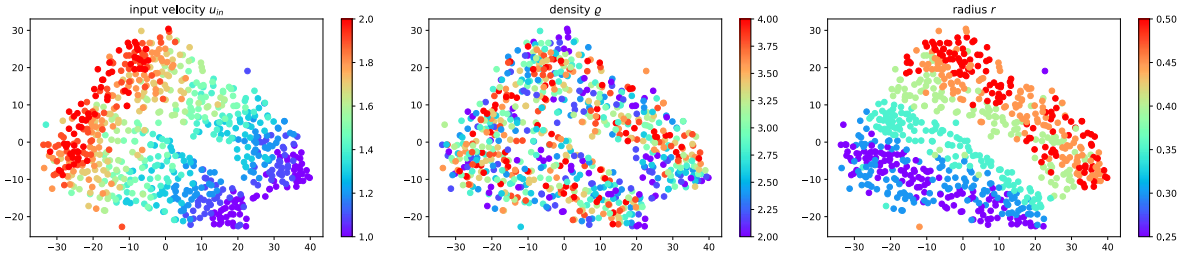


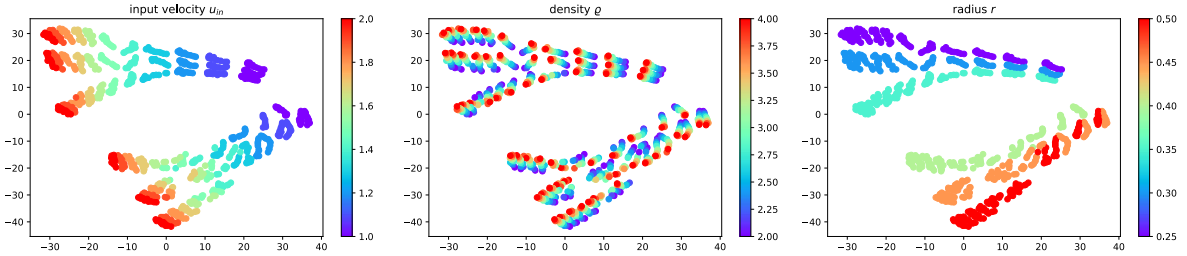
Figure 11: SLDCPE case: t-SNE plot of latent codes of network with width 512 and latent code size 64 trained on min-max-normalized data. The case's parameters are color-coded.

different error behaviour for the cases. While the relative errors of $u_{top} = 2.2$ and especially 0.6 are noticeably higher than the other ones, the other cases, especially for $u_{top} \in \{1.3, 1.6\}$ are noticeably lower than the rest.

Karman vortex street To test the inference capabilities, we test for a density $\varrho = 2.3$, which is not in the training parameter samples, a radius $r = 0.45$, which is taken from the training parameter samples, and input velocities u_{in} of $[1.5, 1.52, 1.54, 1.56, 1.58, 1.6]$, where all but the first and last velocity value are not in the training parameter samples. The corresponding latent codes are obtained by performing interpolation. Again, we choose the network with the lowest relative error: network width 512 and latent code size 32.



(a) Latent code size 32



(b) Latent code size 256

Figure 12: Karman vortex case: t-SNE plots of latent codes of network with width 512 and different latent code sizes trained on min-max-normalized data. The latent codes are color-coded for different values of the parameters.

Figure 15 shows the errors of the inferred velocities in x-direction where the inlet velocity u_{in} is changed. For an error comparison, we compare the error with the results shown in Figure 7 because they share the same radius r, a similar density ϱ and the input velocity of $u_{in} = 1.6$ appear in both. When comparing the errors of the velocity u, the ones from the inferred parameter configurations are approximately one magnitude higher than the one from the training dataset.

Figure 16 shows the inferred velocities at the last timestep. In all cases, the vortex shedding behavior is reproduced.

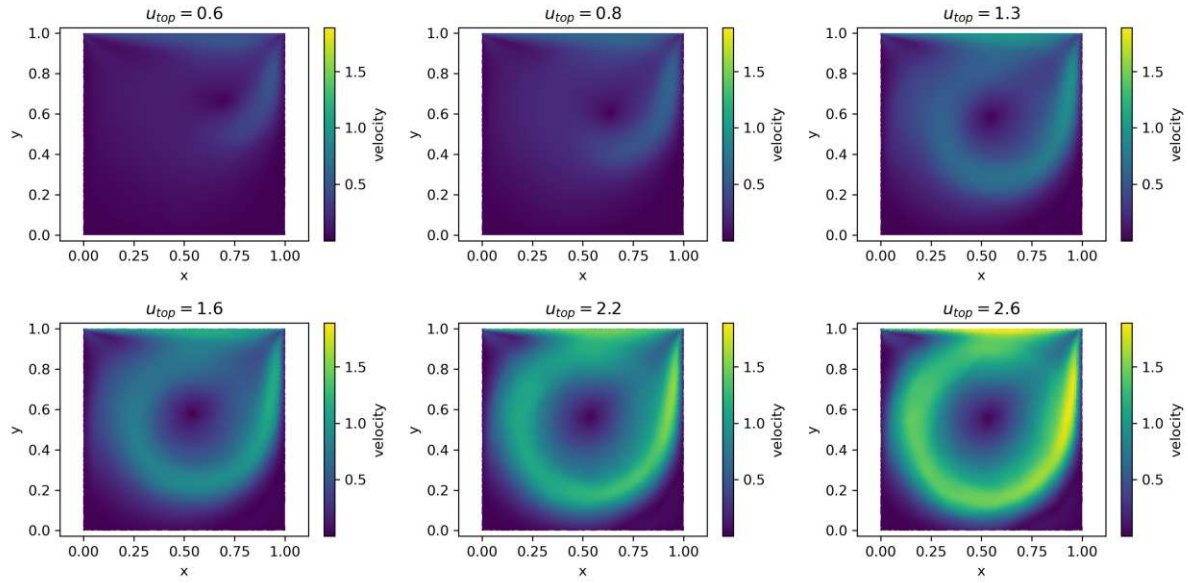


Figure 13: SLDCPE case: inferred velocity contour plots of min-max-normalization (latent code size 256, network width 512, cases with $\alpha = 0°$ and $t = 9.0\ s$)

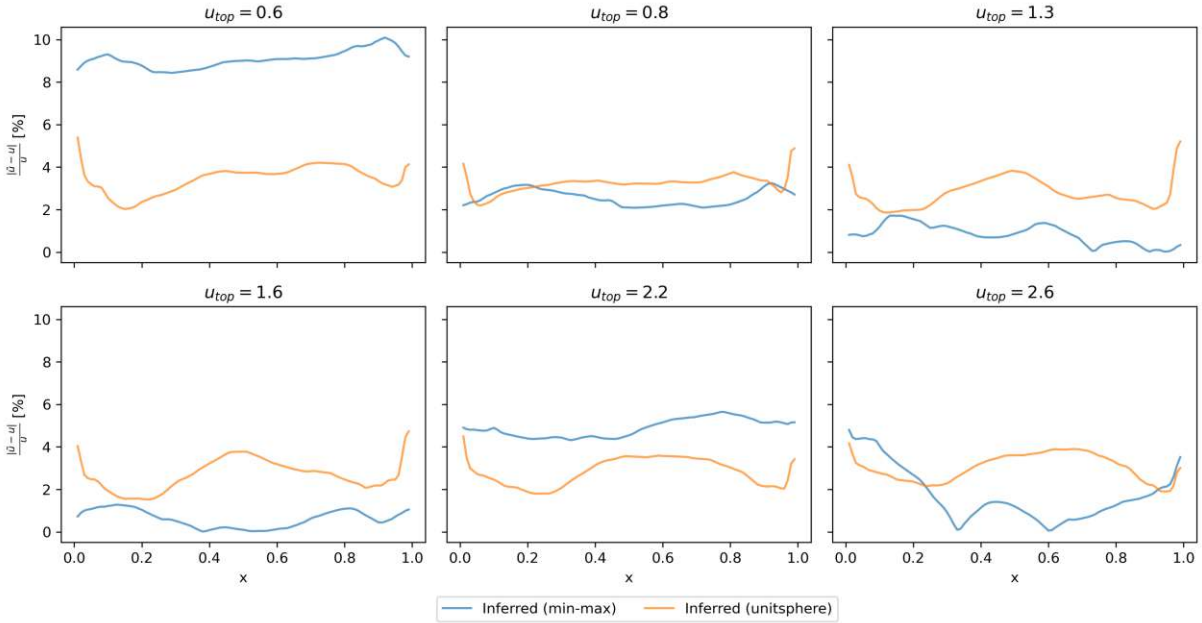


Figure 14: SLDCPE case: relative errors of inferred velocities at the top boundary of both data normalization methods (network with latent code size 256, width 512, cases with $\alpha = 0°$ and $t = 9.0s$)

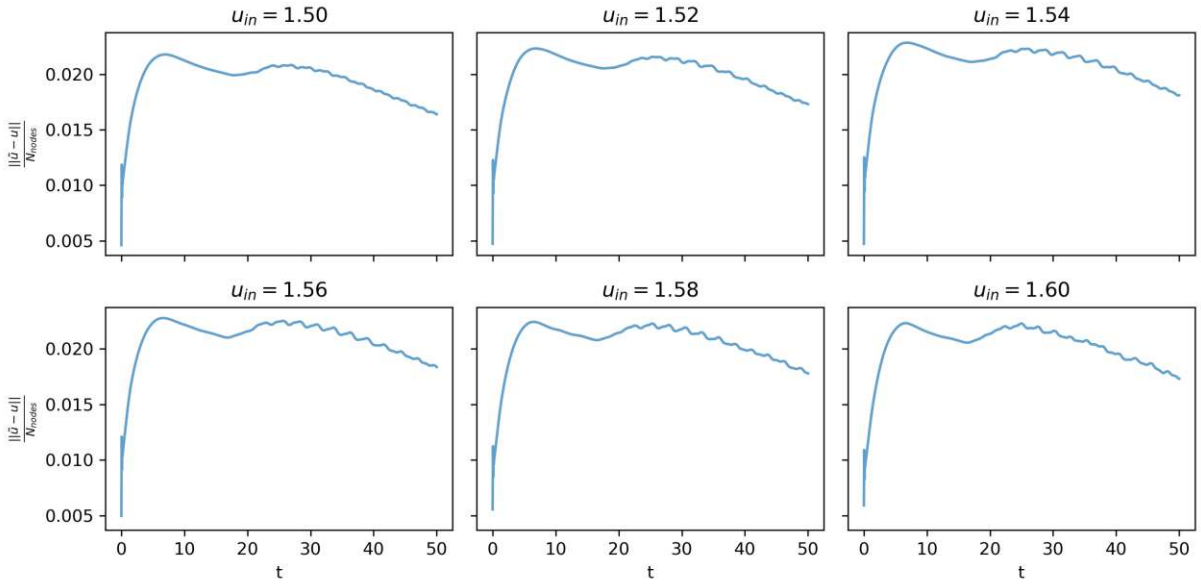


Figure 15: Karman vortex street case: errors of inferred velocity (latent code size 32, network width 512, min-max-normalization, cases with $\varrho = 2.3$, $r = 0.45$)

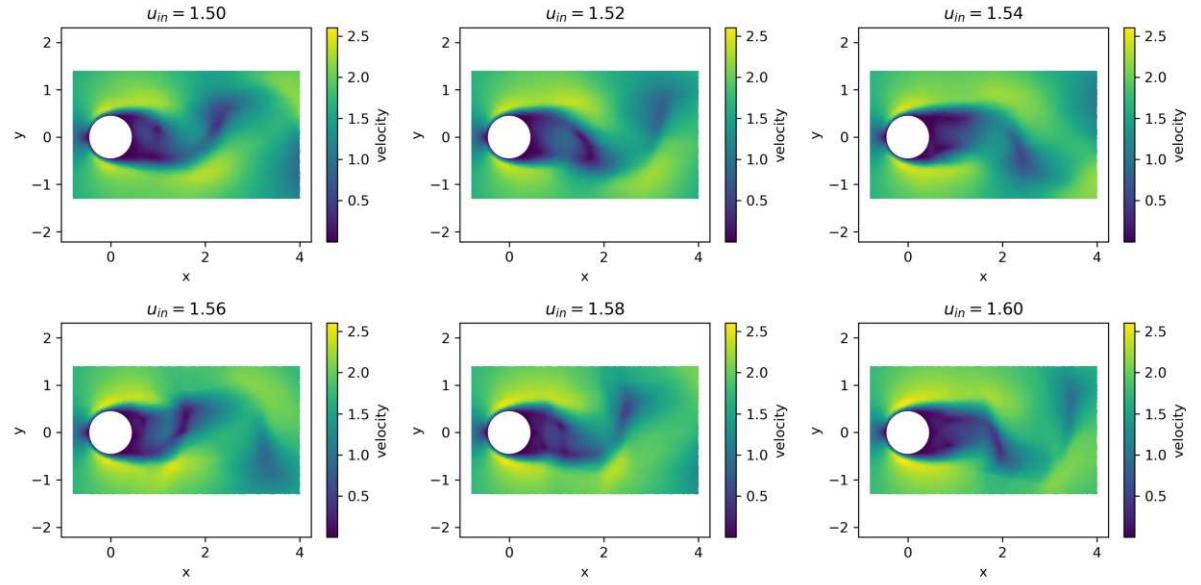


Figure 16: Karman vortex street case: inferred velocity contour plots of min-max-normalization at last timestep (latent code size 32, network width 512, cases with $\varrho = 2.3$, $r = 0.45$)

5. Conclusion

In this work, the application of feedforward neural networks for building reduced models of fluid flows was studied. A particular focus was put on the effects of flow and geometry parametrization on the accuracy of velocity prediction. The results of the numerical experiments show that the type of data normalization and the network width have the largest impact on the the prediction errors of the autodecoder. The type of data normalization shows different results: while the min-max-normalization performs better in the stationary LDC-case, the unitsphere-normalization displayed a lower error on the SLDCPE-case and the Karman vortex street-case. Especially in the Karman vortex street-case, unitsphere-normalization performs considerably better. Despite the lower error, it poses a challenge in practice: for unseen parameter configuration, not only does the interpolation have to be performed for the latent codes, it also has to be performed for the scaling values, which are different for each parameter configuration run. This may lead to scaling errors.

Increasing the latent code size also showed different results. While in the LDC- and the SLDCPE-case increasing the latent code size mainly decreases the error, the Karman vortex-case showed a contrary behaviour, where it mainly led to greater errors. In all cases, the data normalization method and the network width had a greater impact on the accuracy than the latent size.

In all cases, the autodecoder was able to represent the main flow features (e.g. vortex shedding), albeit with varying accuracy. The main errors in the simulations come from inaccuracies or even failures of predicting the geometry of the main flow structures (e.g. the vortex in the SLDCPE-case and the vortex shedding in the Karman vortex street), especially the outline of those vortices, contributed a substantial amount to the error measure.

Furthermore, we presented a way of the autoencoder network to yield solutions to unseen parameters. This feature can then be used for applications of reduced models, e.g. shape optimization.

Since the output dimension of the network can arbitrarily be expanded, it is also possible to predict the pressure. Especially in the absence of pressure singularities, this network should be applicable. Furthermore, one can expand the output dimensions to additionally predict the geometry, which might change in time. This could be used to learn the flow field in fluid-structure interactions (see [15] for example) where the fluid domain moves with time. Another possible extension is changing the input dimensions, where parameters like the input velocity for e.g. the Karman-vortex-street are an additional input.

Finally, one has the option to include physical laws by modifying the objective function such that any deviation from the physical law is penalized, similar to [29, 30].

A. Appendix

A.1. SLDCPE case: effects of network width

In the following we present results for the case of $\alpha = -30°$ and $u_{top} = 3$. For this we look at the effects of the network widths using the min-max-normalization and a latent code size of 64.

Figure 17 shows the L_2-error evolution of the velocities. The network with a lower width exhibits higher errors due less network parameters being available and thus the network loses some ability to predict more complex interactions.

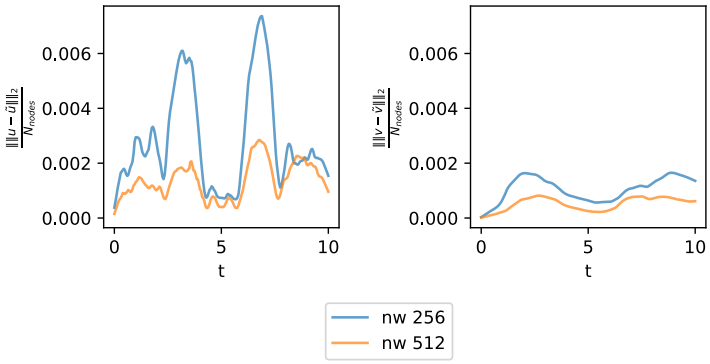


Figure 17: SLDCPE case: L_2 velocity error evolution of network widths 256 and 512 (latent code size 64, min-max-normalization, case with $\alpha = -30°$ and $u_{top} = 3$)

Figure 18 shows the flow field at the first peak. The network with width of 256 has a lower velocity at the top than the one with a width of 512. Furthermore, there is also a visual distinction between the networks' outputs and the ground truth: the vortex in the predictions for the wider network is captured whereas the network with the lower width mainly exhibits errors in the outline of the swirl. However, both display an error in the right upper corner where the velocity is discontinuous.

A.2. Karman case: effects of normalization method and latent code size

Here the case with $u_{in} = 1.6, \varrho = 2.2, r = 0.45$ is investigated.

Figure 19 and Figure 20 display the velocity error evolution over time. In the former, we compare latent code sizes 32 and 128 for a min-max-normalized dataset with network width 512 and in the latter we compare the normalization methods for a network with width 512 and latent code size 32.

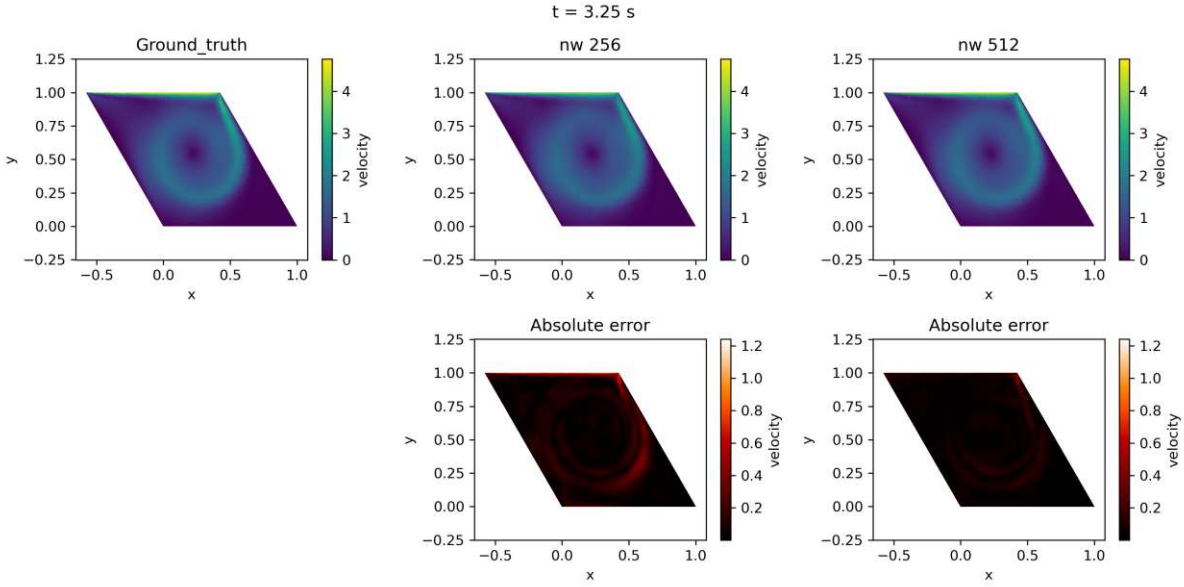


Figure 18: SLDCPE case: velocity prediction and absolute error of network widths 256 and 512 (min-max-normalization, latent code size 64, case with $\alpha = -30°$ and $u_{top} = 3$)

In both figures, the networks cannot sufficiently predict the shedding behaviour and show a big error increase. Both figures support the observations in Figure 6 in that the unitsphere-normalization and the smaller latent code sizes lead to smaller prediction errors.

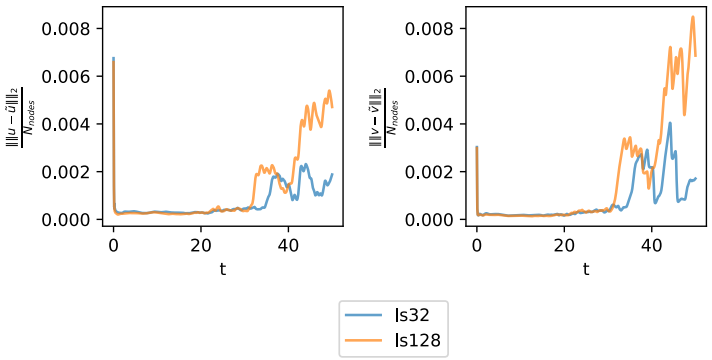


Figure 19: Karman vortex case: error evolution of latent codes sizes 32 and 128 (network width 512, min-max-normalization, case with $u_{in} = 1.6$, $\varrho = 2.2$, $r = 0.45$)

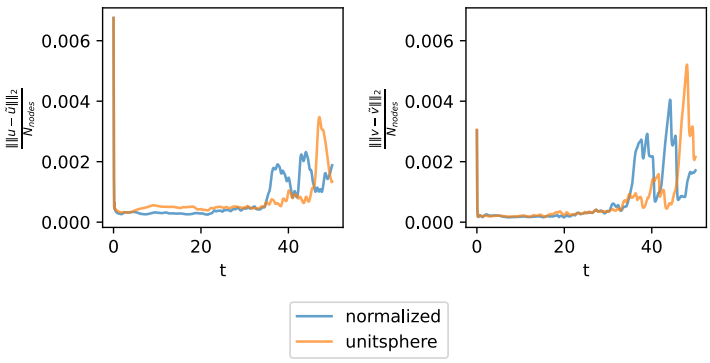


Figure 20: Karman vortex case: error evolution of min-max- and unitsphere-normalization, denoted as *normalized* and *unitsphere* respectively in the legend (latent code size 32, network width 512, case with $u_{in} = 1.6$, $\varrho = 2.2$, $r = 0.45$)

References

[1] Charu C. Aggarwal. *Linear Algebra and Optimization for Machine Learning*. Springer Nature Switzerland AG, 1 edition, 2020. doi: https://doi.org/10.1007/978-3-030-40344-7.

[2] E Amalia, M A Moelyadi, and M Ihsan. Effects of turbulence model and numerical time steps on von karman flow behavior and drag accuracy of circular cylinder. *Journal of Physics: Conference Series*, 1005(1):012012, apr 2018. doi: 10.1088/1742-6596/1005/1/012012. URL https://dx.doi.org/10.1088/1742-6596/1005/1/012012.

[3] Peter Benner, Stefano Grivet-Talocia, Alfio Quarteroni, Rozza Gianluigi, Wil Schilders, and Luis Miguel Silveira. *Model Order Reduction: Volume 1 System- and Data-Driven Methods and Algorithms*. De Gruyter, Berlin, Boston, 2021. ISBN 9783110498967. doi: doi:10.1515/9783110498967. URL https://doi.org/10.1515/9783110498967.

[4] Peter Benner, Stefano Grivet-Talocia, Alfio Quarteroni, Rozza Gianluigi, Wil Schilders, and Luis Miguel Silveira. *Model Order Reduction: Volume 2 Snapshot-Based Methods and Algorithms*. De Gruyter, Berlin, Boston, 2021. ISBN 9783110671490. doi: doi:10.1515/9783110671490. URL https://doi.org/10.1515/9783110671490.

[5] Peter Benner, Stefano Grivet-Talocia, Alfio Quarteroni, Rozza Gianluigi, Wil Schilders, and Luis Miguel Silveira. *Model Order Reduction: Volume 3 Aplications*. De Gruyter, Berlin, Boston, 2021. ISBN 9783110499001. doi: doi:10.1515/9783110499001. URL https://doi.org/10.1515/9783110671490.

[6] Andrea Bonito, Albert Cohen, Ronald DeVore, Diane Guignard, Peter Jantsch, and Guergana Petrova. Nonlinear methods for model reduction. *arXiv preprint arXiv:2005.02565*, 2020.

[7] I. Demirdžić, Ž. Lilek, and M. Perić. Fluid flow and heat transfer test problems for non-orthogonal grids: Bench-mark solutions. *International Journal for Numerical Methods in Fluids*, 15(3):329–354, 1992. doi: https://doi.org/10.1002/fld.1650150306. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/fld.1650150306.

[8] Ronald A. DeVore. Nonlinear approximation. *Acta Numerica*, 7:51–150, 1998. doi: 10.1017/S0962492900002816.

[9] E. Erturk and B. Dursun. Numerical solutions of 2-d steady incompressible flow in a driven skewed cavity. *ZAMM - Journal of Applied Mathematics and Mechanics / Zeitschrift für Angewandte Mathematik und Mechanik*, 87(5):377–392, 2007. doi:

https://doi.org/10.1002/zamm.200610322. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/zamm.200610322.

[10] Stefania Fresca and Andrea Manzoni. 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, jan 2022. doi: 10.1016/j.cma.2021.114181. URL https://doi.org/10.1016%2Fj.cma.2021.114181.

[11] Kai Fukami, Kazuto Hasegawa, Taichi Nakamura, Masaki Morimoto, and Koji Fukagata. Model order reduction with neural networks: Application to laminar and turbulent flows. *SN Computer Science*, 2, 2021. ISSN 2661-8907. doi: https://doi.org/10.1007/s42979-021-00867-3.

[12] Benyamin Ghojogh, Mark Crowley, Fakhri Karray, and Ghods Ali. *Elements of Dimensionality Reduction and Manifold Learning*. Springer Nature Switzerland AG, 2023. ISBN 978-3-031-10602-6. doi: https://doi.org/10.1007/978-3-031-10602-6.

[13] Francisco J. Gonzalez and Maciej Balajewicz. Deep convolutional recurrent autoencoders for learning low-dimensional feature dynamics of fluid systems. *arXiv preprint arXiv:1808.01346*, 2018. doi: https://doi.org/10.48550/arXiv.1808.01346.

[14] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[15] Renkun Han, Yixing Wang, Weiqi Qian, Wenzheng Wang, Miao Zhang, and Gang Chen. Deep neural network based reduced-order model for fluid-structure interaction system. *Physics of Fluids*, 34(7), 07 2022. ISSN 1070-6631. doi: 10.1063/5.0096432. URL https://doi.org/10.1063/5.0096432.

[16] J. Haslinger and R. A. E. Mäkinen. *Introduction to Shape Optimization*. Society for Industrial and Applied Mathematics, 2003. doi: 10.1137/1.9780898718690. URL https://epubs.siam.org/doi/abs/10.1137/1.9780898718690.

[17] Hecht-Nielsen. Theory of the backpropagation neural network. In *International 1989 Joint Conference on Neural Networks*, pages 593–605 vol.1, 1989. doi: 10.1109/IJCNN.1989.118638.

[18] Jan S. Hesthaven, Gianluigi Rozza, and Benjamin Stamm. *Certified Reduced Basis Methods for Parametrized Partial Differential Equations*. Springer Cham, 2016. ISBN 978-3-319-22470-1. doi: https://doi.org/10.1007/978-3-319-22470-1.

[19] J.S. Hesthaven and S. Ubbiali. Non-intrusive reduced order modeling of nonlinear problems using neural networks. *Journal of Computational Physics*, 363:55–78, 2018. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2018.02.037. URL https://www.sciencedirect.com/science/article/pii/S0021999118301190.

[20] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989. ISSN 0893-6080. doi: https://doi.org/10.1016/0893-6080(89)90020-8. URL https://www.sciencedirect.com/science/article/pii/0893608089900208.

[21] Takeo Kajishima and Kunihiko Taira. *Computational Fluid Dynamics: Incompressible Turbulent Flows*. Springer Cham, 2016. ISBN 978-3-319-45304-0. doi: https://doi.org/10.1007/978-3-319-45304-0.

[22] Youngkyu Kim, Youngsoo Choi, David Widemann, and Tarek Zohdi. A fast and accurate physics-informed neural network reduced order model with shallow masked autoencoder. *Journal of Computational Physics*, 451:110841, 2022. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2021.110841. URL https://www.sciencedirect.com/science/article/pii/S0021999121007361.

[23] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980*, 2017.

[24] Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces. *arXiv preprint arXiv: 2108.08481*, 2023.

[25] Hendrik C. Kuhlmann and Francesco Romanò. *The Lid-Driven Cavity*, pages 233–309. Springer International Publishing, Cham, 2019. ISBN 978-3-319-91494-7. doi: 10.1007/978-3-319-91494-7_8. URL https://doi.org/10.1007/978-3-319-91494-7_8.

[26] Kookjin Lee and Kevin T. Carlberg. Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders. *Journal of Computational Physics*, 404:108973, 2020. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2019.108973. URL https://www.sciencedirect.com/science/article/pii/S0021999119306783.

[27] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[28] Alfio Quarteroni, Andrea Manzoni, and Federico Negri. *Reduced Basis Methods for Partial Differential Equations*. Springer International Publishing Switzerland, 2015. ISBN 978-3-319-15431-2. doi: https://doi.org/10.1007/978-3-319-15431-2.

[29] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations. *arXiv preprint arXiv:1711.10561*, 2017.

[30] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

[31] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

[32] Tim Salimans and Diederik P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pages 901–909, Red Hook, NY, USA, 2016. Curran Associates Inc. ISBN 9781510838819.

[33] Peng Shengfang, Zhang Junjie, and Zhang Chunyu. Efficient aerodynamic shape optimization through reduced order cfd modeling. *Optimization and Engineering*, 21:1599–1611, 2020. ISSN 1573-2924. doi: 10.1007/s11081-020-09489-9. URL https://doi.org/10.1007/s11081-020-09489-9.

[34] Sho Sonoda and Noboru Murata. Neural network with unbounded activation functions is universal approximator. *Applied and Computational Harmonic Analysis*, 43(2):233–268, 2017. ISSN 1063-5203. doi: https://doi.org/10.1016/j.acha.2015.12.005. URL https://www.sciencedirect.com/science/article/pii/S1063520315001748.

[35] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

[36] Shaoqiang Tang and Yang Yang. Why neural networks apply to scientific computing? *Theoretical and Applied Mechanics Letters*, 11(3):100242, 2021. ISSN 2095-0349. doi: https://doi.org/10.1016/j.taml.2021.100242. URL https://www.sciencedirect.com/science/article/pii/S2095034921000490.

[37] Derick Nganyu Tanyu, Jianfeng Ning, Tom Freudenberg, Nick Heilenkötter, Andreas Rademacher, Uwe Iben, and Peter Maass. Deep learning methods for partial differential equations and related parameter identification problems. *arXiv preprint arXiv:2212.03130*, 2023.

[38] Tayfun Tezduyar, Marek Behr, and James Liou. A new strategy for finite element computations involving moving boundaries and interfaces—the deforming-spatial-domain/space-time procedure: I. the concept and the preliminary numerical tests. *Computer Methods in Applied Mechanics and Engineering*, 94:339–351, 02 1992. doi: 10.1016/0045-7825(92)90059-S.

[39] Tayfun Tezduyar, Marek Behr, Sanjay Mittal, and James Liou. A new strategy for finite element computations involving moving boundaries and interfaces—the deforming-spatial-domain/space-time procedure: Ii. computation of free-surface flows, two-liquid flows, and flows with drifting cylinders. *Computer Methods in Applied Mechanics and Engineering*, 94:353–371, 02 1992. doi: 10.1016/0045-7825(92)90060-W.

[40] Laurens van der Maaten. Learning a parametric embedding by preserving local structure. In David van Dyk and Max Welling, editors, *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*, volume 5 of *Proceedings of Machine Learning Research*, pages 384–391, Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA, 16–18 Apr 2009. PMLR. URL https://proceedings.mlr.press/v5/maaten09a.html.

[41] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[42] Loc Vu-Quoc and Alexander Humer. Deep learning applied to computational mechanics: A comprehensive review, state of the art, and the classics. *arXiv preprint arXiv:2212.08989*, 2022. doi: https://doi.org/10.48550/arXiv.2212.03130.

[43] Genki Yagawa and Atsuya Oishi. *Computational Mechanics with Deep Learning*. Springer Nature Switzerland AG, 2023. ISBN 978-3-031-11847-0. doi: https://doi.org/10.1007/978-3-031-11847-0.

[44] Weigang Yao, Simao Marques, Trevor Robinson, Cecil Armstrong, and Liang Sun. A reduced-order model for gradient-based aerodynamic shape optimisation. *Aerospace Science and Technology*, 106:106120, 2020. ISSN 1270-9638. doi: https://doi.org/10.1016/j.ast.2020.106120. URL https://www.sciencedirect.com/science/article/pii/S1270963820308026.