A twin-decoder structure for incompressible laminar flow reconstruction with uncertainty estimation around 2D obstacles

J Chen, J Viquerat, Frederic Heymes, Elie Hachem

A twin-decoder structure for incompressible laminar flow reconstruction with uncertainty estimation around 2D obstacles

A Preprint

J. Chen
MINES Paristech, CEMEF
PSL - Research University

J. Viquerat*
MINES Paristech, CEMEF
PSL - Research University
jonathan.viquerat@mines-paristech.fr

F. Heymes
MINES Alès
Institute for Risk Science

E. Hachem
MINES Paristech, CEMEF
PSL - Research University

October 1, 2021

Abstract

Over the past few years, deep learning methods have proved to be of great interest for the computational fluid dynamics community, especially when used as surrogate models, either for flow reconstruction, turbulence modeling, or for the prediction of aerodynamic coefficients. Overall, exceptional levels of accuracy have been obtained, but the robustness and reliability of the proposed approaches remain to be explored, particularly outside the confidence region defined by the training dataset. In this contribution, we present an autoencoder architecture with twin decoder for incompressible laminar flow reconstruction with uncertainty estimation around 2D obstacles. The proposed architecture is trained over a dataset composed of numerically-computed laminar flows around 12,000 random shapes, and naturally enforces a quasi-linear relation between a geometric reconstruction branch and the flow prediction decoder. Based on this feature, two uncertainty estimation processes are proposed, allowing either a binary decision (accept or reject prediction), or proposing a confidence interval along with the flow quantities prediction (u, v, p). Results over dataset samples as well as unseen shapes show a strong positive correlation of this reconstruction score to the mean-squared error of the flow prediction. Such approaches offer the possibility to warn the user of trained models when the provided input shows too large deviation from the training data, making the produced surrogate model conservative for fast and reliable flow prediction.

***K*eywords** Neural networks; Autoencoders; Anomaly detection; Computational fluid dynamics; Surrogate model

1 Introduction

During the last few years, the computational fluid dynamics (CFD) community has largely benefited from the fast-paced development of the machine learning (ML) field, and more specifically from that of

*Corresponding author

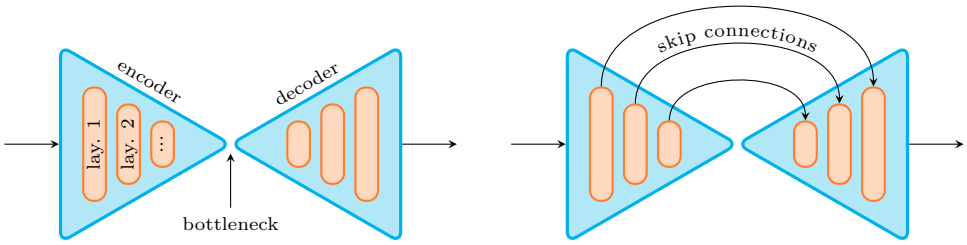
the neural networks (NN) domain. In many cases, a part of the usual numerical resolution process is replaced with a trained NN, in order to reduce its computational cost. Examples for these applications are the prediction of closure terms in RANS [1, 2] or LES [3] computations. In other situations, a supervised network is trained to directly predict a flow profile: in [4], an autoencoder is used to obtain steady state flow predictions around elementary and real-life shapes; in [5], a fusion convolutional neural network (CNN) is trained to predict velocity snapshots around a cylinder in weakly turbulent flows, using the time history of pressure around the cylinder as an input; in [6], a neural network is trained to predict unsteady flow around a circular cylinder, by minimizing a physical loss function composed of regression error and conservation laws. CNNs were also directly applied to predict lift and drag coefficients of 2D airfoils [7] or arbitrary shapes [8].

Still, in most of the proposed works, the question of the reliability of the predictions produced by the trained models is left out. Indeed, the topological complexity of the input space can make it hard to determine whether or not a given element, provided by an external user, lies within the boundaries of the dataset used during training. While very few approaches were proposed to tackle such issues in the context of NN-assisted CFD, several outlier detection techniques have been proposed in other domains. Among them, unsupervised methods have attracted much attention, as they do not require labeled data. In particular, several autoencoder (AE) based techniques were developed for medical and industrial applications: in [9], a fully connected AE with three hidden layers is applied to breast cancer detection; in [10], a convolutional AE (CAE) is used to detect cracking and spalling defects on concrete structures; in [11], CAE is used to detect miss-printed logo images on mobile phones, and in [12], the authors compared several variants of AE on anomaly segmentation in brain magnetic resonance images.

Autoencoder are feedforward neural networks that are widely applied to dimension reduction and feature extraction of high-dimensional data [13]. As shown in the sketch of figure 1a, AEs are composed of a contractive path, named *encoder*, whose role is to compress input data to a space of reduced dimension called *latent space*. The latent space representation of the input variables is obtained at the *bottleneck* of the structure, which is followed by a *decoder* branch, mirror of the encoder one, responsible for the reconstruction of the input. Autoencoders can be trained both in unsupervised or supervised way, depending on the application scenario. In the case of unsupervised learning, AEs are usually exploited to infer the latent space structure of a given dataset. In the CFD community, this functionality makes AE potential candidate tools for model reduction. In [14], AEs are used in conjunction with convolutional layers to learn low-dimensional features of fluid systems. In [15] and [16], the authors combine recurrent neural networks with CAE to learn the dynamics of the extracted low dimensional features. Adversely, in the case of supervised learning, AEs are exploited to perform various full-field flow prediction tasks [4, 5, 6]. Among the multiple variations of AE structures, the special case of U-net [17] must be mentioned. As sketched in figure 1b, U-nets structures contain *skip connections* from the encoder to the decoder, the role of which is to concatenate low-level features from the contractive path to the expansion path (here, concatenation means stacking tensors together along the channel axis). By allowing the mixing of low-level features with the high-level latent-space representation, U-nets usually achieve excellent performance levels on segmentation [17] and regression tasks: in [18], the authors exploit U-nets to infer the velocity and pressure fields of turbulent flow around airfoils computed in a Reynolds-averaged Navier-Stokes framework; in [19], a U-net-like architecture is used to reconstruct turbulent flows from extremely coarse flow field images with remarkable accuracy; in [20], a recurrent U-net architecture is trained to predict the instationary velocity and pressure fields in porous membranes.

In the present paper, we introduce an autoencoder architecture with a twin-decoder as a possible tool for outlier detection in the context of fluid flow predictions. New contributions of this work include:

- A novel twin-decoder architecture displaying a strong correlation between the input reconstruction and the flow prediction error levels by taking advantage of proper skip connections between the two decoder branches. We find that this correlation is almost linear, at the expense of a slightly lower flow prediction accuracy than that of a U-net with similar structure;
- Two uncertainty estimation procedures taking advantage of the latter property: (i) a qualitative procedure based on a user-provided error threshold level, providing binary decisions regarding the prediction (accept or reject), and (ii) a quantitative procedure, providing the



(a) Autoencoder architecture (b) U-net architecture

Figure 1: **Sketch of autoencoder architectures.** Standard autoencoders (left) are composed of an encoder and a decoder paths, and can be exploited either for end-to-end regression tasks (in a supervised way, with labels), or for the inference of latent space representations (in an unsupervised way, without labels). U-net autoencoders are a specific class of AE, in which skip connections are added from the encoder branch to the decoder one in order to mix high-level features from the latent space with low-level one from the contractive path. They usually present a superior level of performance on regression tasks.

user with a error level interval on top of the flow prediction. Results over the considered dataset as well as unseen shapes proved these methods to be efficient to detect the applicability limits of the trained model. Both methods rely on simple concepts, and can be easily applied to other end-to-end prediction tasks.

The paper is organized as follows: the problem settings and dataset construction are presented in section 2. Insights about the proposed twin-decoder architecture and its training procedure are given in section 3. The concepts of both the qualitativee and quantitativ trust-level methods are then described. In section 4, the performance of the method is first explored through a hyper-parameter calibration, then the best architecture is selected based on a cost-to-accuracy ratio. Finally, the correlation levels of the trained model are presented, and the two trust-level methods are put into practice. Finally, future perspectives are given. The base code used in this paper is available at https://github.com/jviquerat/twin_autoencoder (see section A for additional details).

2 Dataset construction

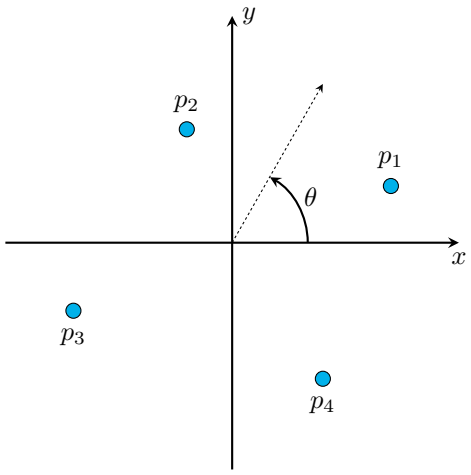
This section provides insights on the random shape dataset generation used to train networks in the next sections. This dataset was initially used in [8] (section 3.5), thus only the main lines are sketched here. For more details, the reader is referred to [8]. First, we describe the steps to generate arbitrary shapes by means of connected Bezier curves. Then, the solving of the Navier-Stokes equations with an immersed method is presented. Finally, details about the dataset are given.

2.1 Random shape generation

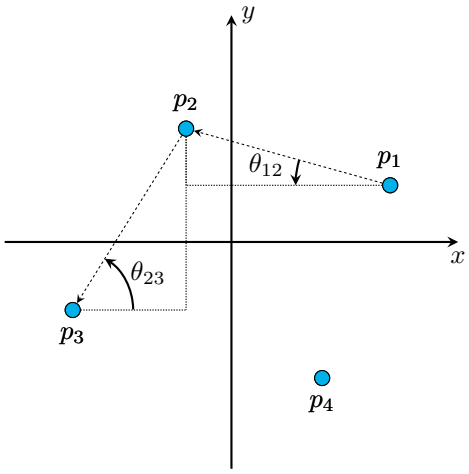
In the first step, n_s random points are drawn in $[0,1]^2$, and translated so their center of mass is positioned in $(0,0)$. An ascending trigonometric angle sort is then performed (see figure 2a), and the angles between consecutive random points are then computed. An average angle is then computed around each point (see figure 2b) using:

$$\theta_i^* = \alpha\theta_{i-1,i} + (1-\alpha)\theta_{i,i+1},$$

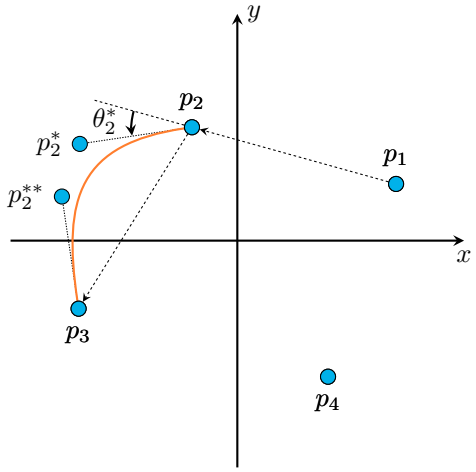
with $\alpha \in [0,1]$. The averaging parameter α allows to alter the sharpness of the curve locally, maximum smoothness being obtained for $\alpha = 0.5$. Then, each pair of points is joined using a cubic Bézier curve, defined by four points: the first and last points, p_i and p_{i+1}, are part of the curve, while the second and third ones, p_i^* and p_i^{**}, are control points that define the tangent of the curve at p_i and p_{i+1}. The tangents at p_i and p_{i+1} are respectively controlled by θ_i^* and θ_{i+1}^* (see figure 2c). A final sampling



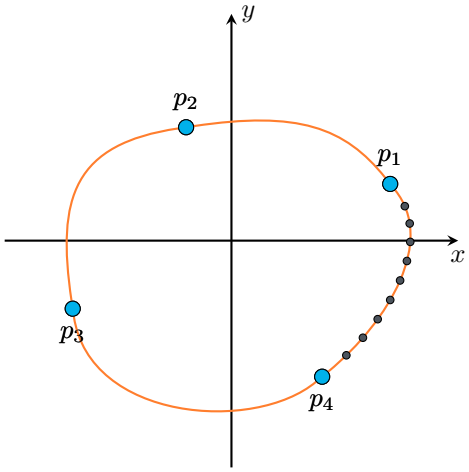
(a) Draw n_s random points, translate them around $(0, 0)$ and sort them by ascending trigonometric angle



(b) Compute angles between random points, and compute an average angle around each point θ_i^*



(c) Compute control points coordinates from averaged angles and generate cubic Bézier curve



(d) Sample all Bézier lines and export for mesh immersion

Figure 2: **Random shape generation with cubic Bézier curves**.

of the successive Bézier curves leads to a boundary description of the shape (figure 2d). Using this method, a wide variety of shapes can be attained, as shown in figure 3.

2.2 Numerical resolution of the Navier-Stokes equations

The flow motion of incompressible newtonian fluids is described by the Navier-Stokes (NS) equations:

$$\begin{cases} \rho\left(\partial_t \mathbf{v} + \mathbf{v}\cdot\nabla\mathbf{v}\right) - \nabla\cdot\left(2\eta\boldsymbol{\epsilon}(\mathbf{v}) - p\mathbf{I}\right) = \mathbf{f}, \\ \nabla\cdot\mathbf{v} = 0, \end{cases} \tag{1}$$

where $t \in [0, T]$ is the time, $\mathbf{v}(x, t)$ the velocity, $p(x, t)$ the pressure, ρ the fluid density, η the dynamic viscosity and $\mathbf{I}$ the identity tensor. In order to efficiently construct the dataset, an immersed volume method is used for resolution instead of the usual body-fitted method, avoiding a systematic re-meshing

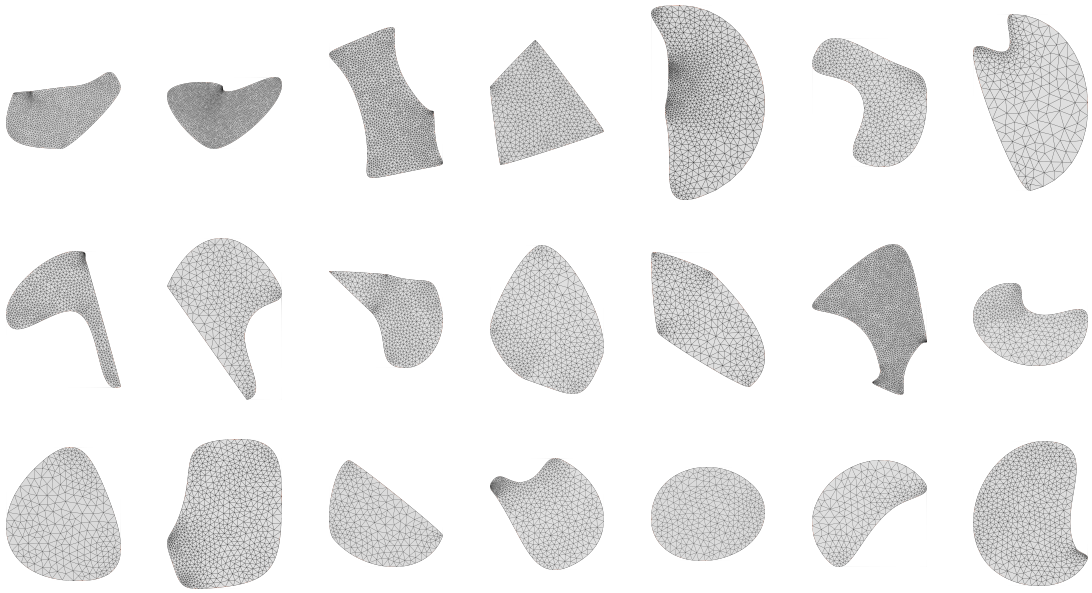


Figure 3: **Shape examples drawn from the dataset.** A wide variety of shape is obtained using a restrained number of points ($n_s \in [4, 6]$), as well as a local curvature r and averaging parameter α.

of the whole domain for each shape. This method rely on a unified fluid-solid Eulerian formulation based on level-set description of the geometry [21], and leads to the following set of modified equations:

$$\begin{cases} \rho^*(\partial_t \mathbf{v} + \mathbf{v} \cdot \nabla \mathbf{v}) - \nabla \cdot (2\eta \boldsymbol{\epsilon}(\mathbf{v}) + \boldsymbol{\tau} - p\mathbf{I}) = \mathbf{f}, \\ \nabla \cdot \mathbf{v} = 0, \end{cases} \tag{2}$$

where we have introduced the following mixed quantities:

$$\begin{aligned} \boldsymbol{\tau} &= H(\alpha)\boldsymbol{\tau}_\mathrm{s}, \\ \rho^* &= H(\alpha)\rho_\mathrm{s} + (1 - H(\alpha))\rho_\mathrm{f}, \end{aligned}$$

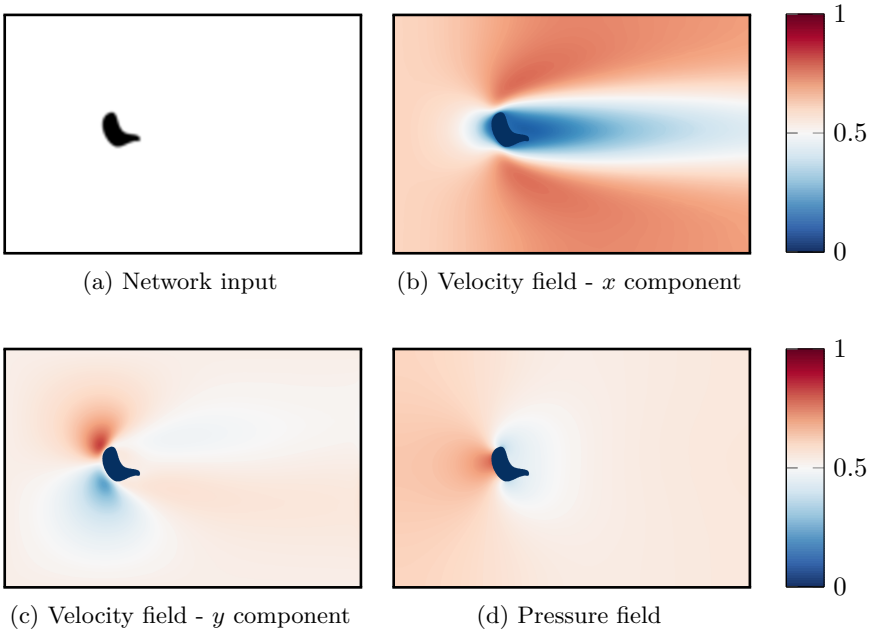
where the subscripts f and s respectively refer to the fluid and the solid, and $H(\alpha)$ is the Heaviside function:

$$H(\alpha) = \begin{cases} 1 \text{ if } \alpha > 0, \\ 0 \text{ if } \alpha < 0. \end{cases} \tag{3}$$

The reader is referred to [22] for additional details about formulation (2). Eventually, the modified equations (2) are cast into a stabilized finite element formulation, and solved using a variational multi-scale (VMS) solver [23, 24, 25, 26, 27].

2.3 Dataset

The dataset is composed of 12.000 shapes, along with their steady-state velocity and pressure fields at $Re = 10$ (see figure 4). All the labels were computed using CimLib-CFD [22], following the methods exposed in section 2.2. The input fields are resized to 2D 100×150 arrays before being provided to the network. As is customary in neural networks training, a channel-wise normalization is applied, mapping all the pixels' value into $[0, 1]$. In following sections we use a color scale to visualize the velocity and pressure fields, but each of them is always a 2D array. For additional details about the distribution of the elements in the dataset, the reader is referred to [8] (section 3.5).



(a) Network input (b) Velocity field - x component

(c) Velocity field - y component (d) Pressure field

Figure 4: **Network input, velocity field and pressure field for a dataset element.** The shape is shown in its computational domain (upper left), along with the computed velocity field (top right, lower left) and pressure field (lower right).

3 Network architecture and training

3.1 Twin-decoder architecture

The general autoencoder architecture with twin-decoder proposed in this contribution is shown in figure 5. Its input consists in a boolean 1-channel 2D tensor containing the computing domain and the obstacle. Its outputs are (i) a 1-channel 2D tensor containing the reconstructed input, and (ii) a 3-channels tensor containing the predicted velocity components and pressure fields. The encoder branch consists in stacked convolution-convolution-max-pooling blocks using 3×3 kernel size, stride size equal to 1, and zero-padding. The convolutional layers exploit rectified linear unit (ReLU) as activation functions. After every pooling operation, the number of kernels used for convolutional layers is doubled, until reaching the bottleneck.

The decoder is composed of two branches, hereafter denoted by "shape decoder" and "flow decoder". Both decoder branches are composed of deconvolution-convolution-convolution blocks, and share similar structures. The deconvolution layers use 2×2 kernel size, stride size equal to 2, zero padding and ReLU activation. Symmetrically to the encoder branch, the number of kernels is halved after each block in the decoder branches. Finally, a convolutional layer with a 1×1 kernel is applied to set the final number of channels (1 for the shape decoder, and 3 for the flow decoder). The output of our network hence contains 1-channel tensor representing reconstructed input, and a 3-channel tensor representing the velocity and pressure.

The key ingredient of the proposed architecture lies in the skip connections that link the shape decoder and the flow decoder. The output of each shape decoder block is concatenated (along the channel axis, to preserve shape) to the output of the deconvolution layer of each flow decoder block. This idea is similar to that of U-net, except that low-level features do not originate from the encoder branch, but from the reconstruction of the input. Forcing such dependence between the two decoder branches is expected to induce a strong correlation between their performance levels. In essence, by enforcing a relation between the shape reconstruction error and the flow prediction error, the proposed method allows to reject possible outliers based on the reconstruction error. As the latter can be computed for

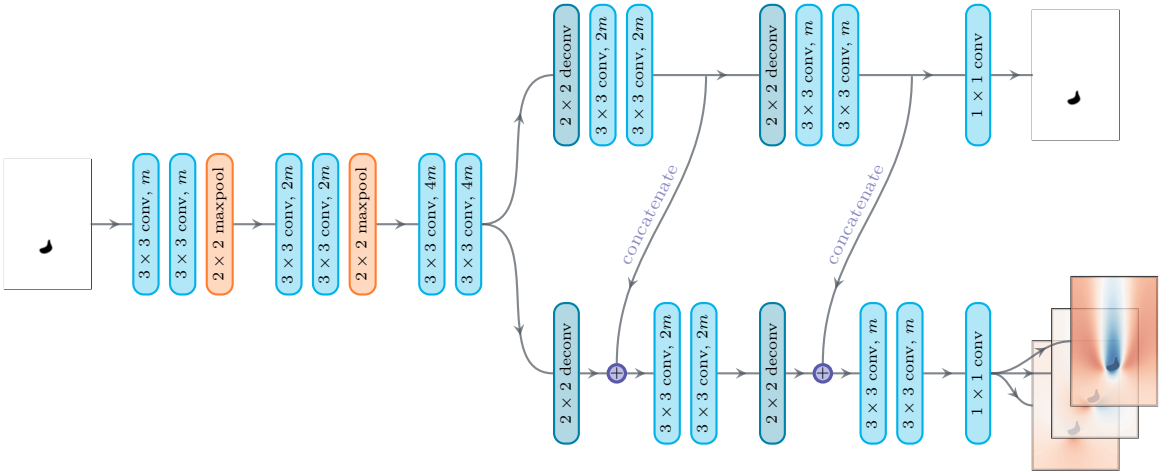


Figure 5: **Proposed twin-decoder architecture.** The encoder is based on a pattern made of two convolutional layers followed by a max-pooling layer. At each occurence of the pattern, the image size is divided by two, while the number of filters, noted m, is doubled. In both decoder paths, a transposed convolution step is first applied to the input, while the number of filters is halved. The output of this layer in the flow decoder is then concatenated with its mirror counterpart in the shape decoder. Finally, two convolution layers are applied. At the end of the last layer, a 1×1 convolution is applied on each decoder to obtain a final 3D tensor with 4 channels. Every channel has the same dimension as the input.

an arbitrary input, the end-user can be warned whether the network prediction can be trusted or not. More details on the acceptance/rejection procedure are provided in section 3.3.

3.2 Training procedure

The loss function used to train the twin-decoder architecture is a weighted sum of the shape and the flow decoder losses. Both decoders use the regular mean squared error (MSE) as loss function:

$$L = \frac{1}{hwn_c} \sum_{d,i,j} (y_{d,i,j} - \hat{y}_{d,i,j})^2, \tag{4}$$

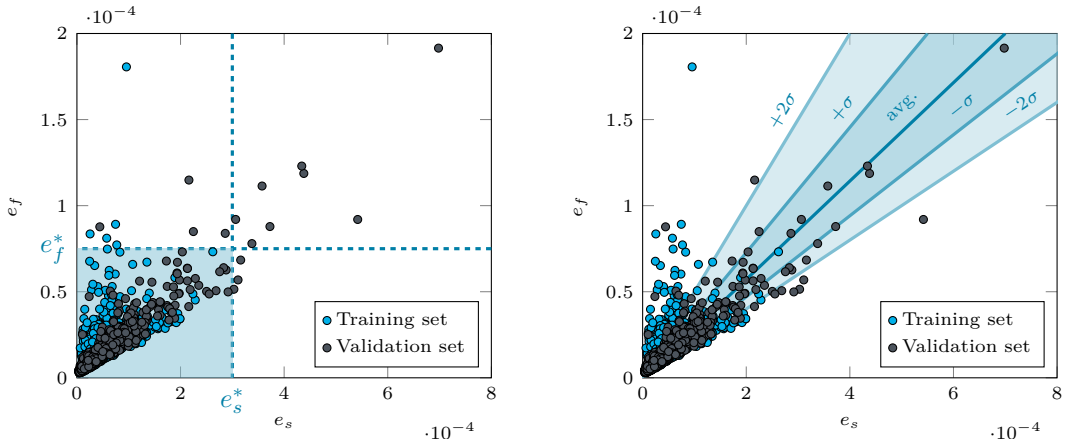
where h, w and n_c represent respectively channel height, width and the number of channels. Expression (4) represents the average squared error over all the pixels of a 3D tensor. The final loss function used for training is:

$$L_{\text{twin}} = L_{\text{flow}} + \beta L_{\text{shape}}, \tag{5}$$

where β is a weighting parameter that remains to be tuned (see section 4). The network is trained with the Adam optimizer using an initial learning rate of 1×10^{-3}, which is reduced to 1×10^{-4} after 600 epochs. To prevent overfitting, the validation loss is monitored, and early stopping is used to determine the end of the training. The network parameters are initialized using a truncated Guassian distribution, following [17]. Training is performed using a Tesla V100 GPU, using mini-batches of size 128 to limit the required computational resources. As different models are evaluated and caompared, their training times are given in section 4.

3.3 Trust level based on shape reconstruction

Given a trained twin-decoder neural network, it is feasible to evaluate the trust level of flow prediction by input reconstruction. As the error levels of the twin decoders are strongly correlated, a qualitative



(a) Qualitative method

(b) Quantitative method

Figure 6: **Description of the qualitative and quantitative methods** on the scatter plots of e_f versus e_s on training and validation sets for a reference twin-decoder architecture. (Left) Qualitative method: given a threshold e_f^*, the corresponding optimal e_s^* is found by solving problem (6). Then, the end-user rejects predictions that produce shape reconstruction errors superior to e_s^*. Only the predictions falling within the bottom left quarter (in orange) are accepted. (Right) Quantitative method: e_f is modeled as an affine function of e_s with an uncertainty interval, as shown in relation (7). Based on e_s, the method provides an estimated e_f level, along with a confidence interval for the prediction.

and a quantitative trust-level methods are proposed, both based on the reconstruction error. Their general concepts are proposed in the following sub-sections, while their applications on trained networks are presented in section 4.4. In the following, the MSE for flow and shape reconstructions are respectively denoted e_f and e_s.

3.3.1 Qualitative method

In this case, the end-user provides an acceptable MSE level e_f^* on the flow prediction. The method consists in selecting the associated shape reconstruction error e_s^* that minimizes the probability of taking wrong decisions when supposing that the two error levels are linearly correlated. The threshold shape reconstruction error e_s^* is the solution to the following minimization problem:

$$e_s^* = \frac{1}{N} \min_e \left[\text{card}\left(e_s < e \text{ and } e_f > e_f^*\right) + \text{card}\left(e_s > e \text{ and } e_f < e_f^*\right) \right], \tag{6}$$

where N is the number of MSE scatter points (e_s, e_f) taken into account. The numerator sums the amount of wrong decisions as the error of both decoders are assumed to be positively correlated. Such a formulation prevents the end-user from making mistakes when accepting or rejecting predictions. An illustration of the method is shown in figure 6a.

3.3.2 Quantitative method

With the quantitative method, e_f and an associated confidence interval δ_f are directly estimated from e_s using the following relation:

$$e_f = a e_s + b + \epsilon, \tag{7}$$

where ϵ follows a normal law of the form:

$$\epsilon \sim \mathcal{N}(0, (c e_s)^2). \tag{8}$$

The assumption is supported by the linear pattern of error scatter plots. In essence, indexing the standard deviation on e_s in (8) represents the growing uncertainty on flow prediction as shape reconstruction turns worse. Under this formulation, the likelihood of e_f on N scatter points is:

$$\prod_{i=1}^{N} p(e_f^i | e_s^i; a, b, c) = \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi(ce_s^i)^2}} \exp\left(-\frac{(e_f^i - ae_s^i - b)^2}{2(ce_s^i)^2}\right) \tag{9}$$

The optimal parameters a^*, b^* and c^* are obtained by minimizing the negative log likelihood:

$$(a^*, b^*, c^*) = \min_{a,b,c} \frac{1}{2} \sum_i \left(\frac{ae_s^i e_f^i - b}{ce_s^i}\right)^2 + \sum_i \log(ce_s^i) + \frac{N}{2}\log(2\pi), \tag{10}$$

which is achieved using a gradient descent algorithm. An illustration of the method is shown in figure 6b. The minimum is determined by the scatter points of the training and validation error, then evaluated on the test set. To the difference of the qualitative method, in which the predictions are either plainly accepted or rejected, such formulation allows one to estimate e_f with a confidence interval δ_f, without a need of pre-selecting a threshold accuracy, thus providing an additional flexibility. As an example, given a shape reconstruction error e_s, the associated flow prediction accuracy level e_f would fall into $[(a-2c)e_s + b, (a+2c)e_s + b]$ with 95% probability.

4 Results

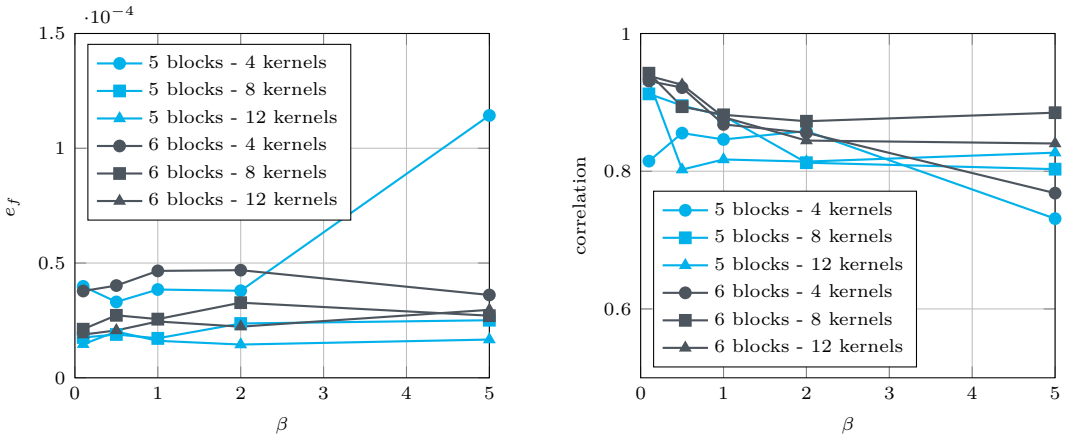
In this section, the performance of the twin-decoder architecture is explored. First, a minimal hyper-parameter calibration is presented, that highlights the impact of the loss weighting parameter β, the number of convolutional blocks used in the decoder structure, and the number of kernels used in the convolutional layers. The resulting architecture is evaluated on the test set, showing good performance on unseen shapes. Then, the qualitative and quantitative methods respectively presented in sections 3.3.1 and 3.3.2 are put into practice. Finally, a comparison with two other AE-based architectures is proposed, revealing the contribution of skip connections between shape and flow decoders.

4.1 Hyper-parameter calibration

In this section, the impact of three different parameters is explored: (i) the weighting parameter β, (ii) the number of convolutional blocks, and (iii) the number of convolutional kernels. In total, 30 configurations were tested, the network being evaluated not only on its flow field prediction performance, but also on the correlation coefficient between e_f and e_s on the validation set. As the different explorations are detailed below, the corresponding results can be found in figure 7.

Convolution blocks The number of convolution blocks in the encoder (or equivalently the number of deconvolution blocks in the decoders) proved to be determinant regarding the performance of the flow decoder. Based on previous experiments (not shown here), only architectures with 5 or 6 blocks were considered (less blocks showing low performance, while more blocks being too costly to train). Architectures with 5 blocks proved to outperform deeper ones in terms of flow prediction, probably due to a too high compression rate in the latent space when using 6 blocks. Adversely, architectures with 6 blocks presented a slightly better correlation coefficient between the two decoder errors.

Convolution kernels The flow prediction accuracy was highly dependent on the number of convolution kernels used in each blocks. Since the architecture is symmetric and scalable, we use the number of kernels in the first convolutional layer to represent this hyper-parameter. Results show a clear advantage of using 8 kernels over 4, while increasing from 8 to 12 is not as beneficial. Hence, it is not necessary to use too many kernels, as a CNN complexity scales with the square of the kernel number. Similar conclusions hold for the correlation coefficient.



(a) Flow prediction MSE

(b) Correlation coefficient between e_s and e_f

Figure 7: **Hyper-parameter calibration.** The performance is evaluated on the validation set. To compare the accuracy of flow prediction between different architectures, the MSE is averaged over the entire dataset.

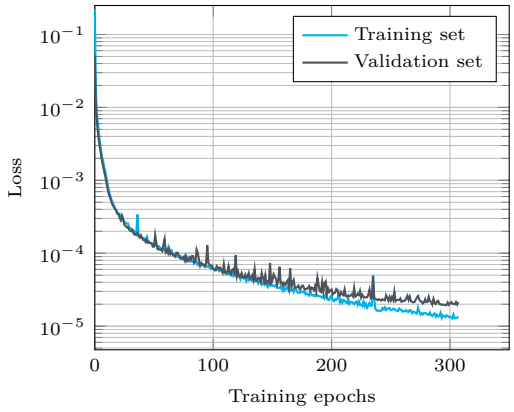


Figure 8: **Training history of the model.** The training is ceased when the validation loss does not decrease for 10 successive epochs. A slight overfitting can be observed on the trained model.

Weighting parameter The β parameter in the loss function (5) proved to have a crucial impact on correlation of the two decoder errors. Five different values were considered (from 0.1 to 5), with larger values giving more weight to the shape decoder during the training process. We found that small β values were very beneficial to the correlation level, while alleviating the differences caused by different number of convolutional blocks. With $\beta = 0.1$, a 5-block architecture with 12 kernels in the first convolutional layer obtained a correlation level of 93%, which represents a strong linear relation. The impact of β on the flow prediction accuracy is not as clear according to the obtained results.

4.2 Prediction cost and accuracy

In order to evaluate the computational cost of the training, we provide the best performance attained by each combination with respect to the number of learnable parameters in table 1. Each point represents the smallest e_f value obtained by tuning β. By using the 5-block architecture with 12 kernels, e_f values as low as 1.4×10^{-5} can be reached, requiring 1.4 million parameters, making it the architecture choice with the best cost-accuracy ratio. Learning time on a Tesla V100 GPU is approximately 0.7 hours. The training curves for the training and validation subsets are presented in figure 8.

Table 1: **Flow prediction performance obtained for architectures of various complexities**. A good ratio must be found between the final achievable accuracy and the total number of learnable parameters, as training time rises dramatically with the network complexity. Here, best performance is obtained using a 5-block architecture and 12 initial kernels, with a total of 1.4 million parameters.

Architecture	Best e_f	Nb. of parameters
5 blocks - 4 kernels	3.3063×10^{-5}	157,208
5 blocks - 8 kernels	1.7216×10^{-5}	627,500
5 blocks - 12 kernels	$\mathbf{1.4491 \times 10^{-05}}$	**1,410,880**
6 blocks - 4 kernels	3.6089×10^{-5}	592,024
6 blocks - 8 kernels	2.1178×10^{-5}	2,365,484
6 blocks - 12 kernels	1.8826×10^{-5}	5,320,384

On the test set, the proposed model reaches a flow reconstruction accuracy $e_f = 1.38 \times 10^{-5}$, thus showing good generalization capabilities on unseen data. In figure 9, a prediction example from the test set is shown, along with the associated exact solution and a pixel-wise error map. As can be seen, the velocity and pressure fields are well recovered by the network, the error being concentrated in the vicinity of the obstacle, *i.e.* in the area of large pressure and velocity gradients. As the obstacle itself only represents a small portion of the predicted field, a second metric is proposed: for each element of the test set, the mean pixel-wise relative error is computed on a smaller area surrounding the obstacle. The corresponding error distributions are plotted in figure 10. Overall, the average relative error is 3.92% for horizontal velocity, 3.57% for vertical velocity and 3.55% for pressure, with very few elements exceeding 6% of relative error, showing again decent generalization capabilities.

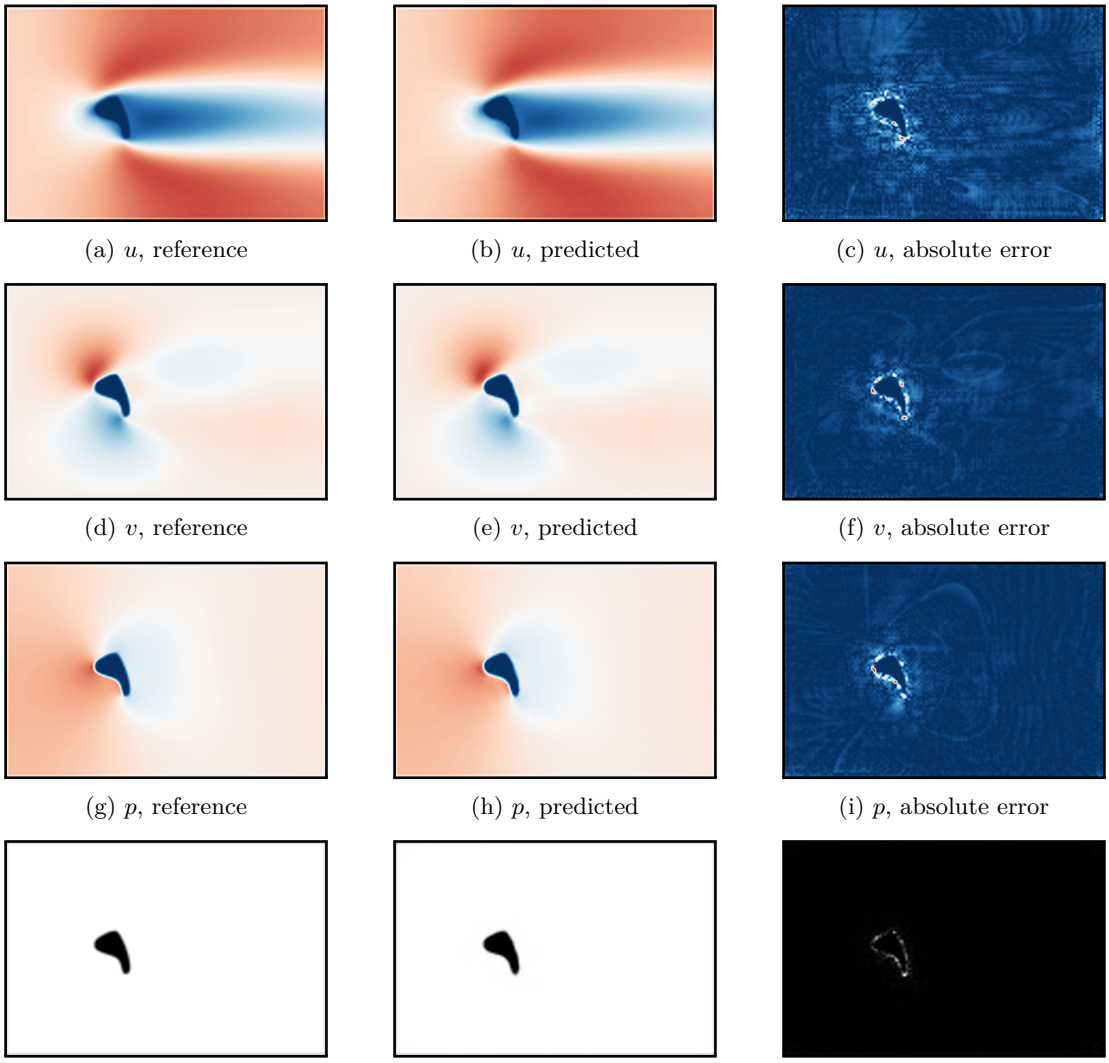
4.3 Correlation levels

In this section, the correlation levels obtained between the shape reconstruction and the flow prediction errors are commented. To further show the interest of the autoencoder architecture with twin-decoder (twin-AE), two close network architectures are considered, namely the dual autoencoder (dual-AE), and the U-net dual autoencoder (U-dual-AE). The dual-AE architecture is simply obtained by removing the skip connections of the twin-AE, while the U-dual-AE exploits skip connections coming from the encoder path instead of the reconstruction path. To ensure a fair comparison, the same configuration is used for all three architectures. A concatenation with constant tensor is applied to the flow decoder of the dual-AE, so its number of parameters is equal to that of the U-dual-AE and the twin-AE.

The scatter plots obtained with the twin-AE on the training, the validation and the test sets are respectively shown in figures 11a, 11b and 11c. Associated correlation levels are 0.772, 0.931 and 0.954, meaning that strong linear relations are observed on the validation and test set. Regarding the training set, the weaker correlation is interpreted as a consequence of the slight overfitting observed during training (see figure 8). In comparison, the obtained correlation level of the dual AE on the test set is 0.830, which is significantly weaker than that of the twin AE, thus proving the interest of the skip connections between the two decoder branches (see figure 11d). The twin-AE also has slightly lower relative errors than its dual-AE counterpart (3.92%, 3.57% and 3.55% respectively for u, v, p, against 4.30%, 4.05% and 4.00%). Finally, the U-dual-AE architecture exhibits almost no correlation between e_s and e_f, with a computed correlation level of 0.385 (see figure 11e). Adversely, the relative error levels of the U-dual-AE are significantly lower (3.01%, 1.94% and 1.94%), which is in line with results from the literature [28].

4.4 Trust level based on input reconstruction

This section presents the application of the trust level methods detailed in section 3.3. Again, the underlying concept is to take advantage of the strong correlation between shape and flow reconstruction



(a) u, reference (b) u, predicted (c) u, absolute error

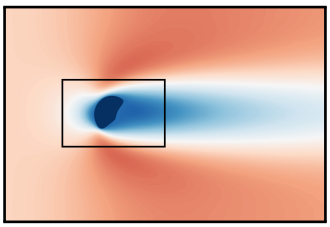
(d) v, reference (e) v, predicted (f) v, absolute error

(g) p, reference (h) p, predicted (i) p, absolute error

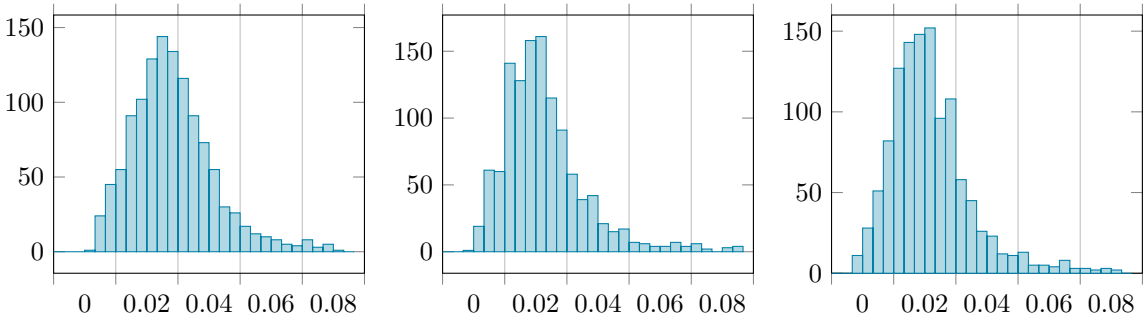
(j) Shape, reference (k) Shape, predicted (l) Shape, absolute error

Figure 9: **Flow and shape predictions around an obstacle from the test set.** On this instance, the flow prediction error is $e_f = 1.57 \times 10^{-5}$, with most of the error concentrated on the boundary of the shape, *i.e.* in the area of large pressure and velocity gradients. For the u, v and p predictions, the pixels' value range is still $[0, 1]$. An RGB colormap is used for better visualization

.



(a) Area taken into account to compute relative error histograms.



(b) Relative error for u. (c) Relative error for v. (d) Relative error for p.

Figure 10: **Relative error for flow predictions** over test set. The black rectangle around the obstacle indicates the area on which the u, v and p relative errors are computed. The histograms indicate the error levels obtained when comparing predictions to labels on the 1200 elements of the test set.

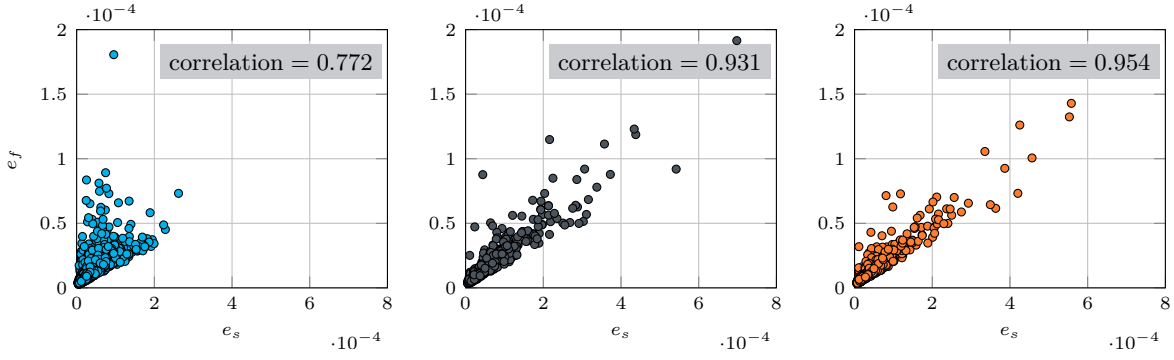
error levels (see section 4.3) to propose an uncertainty estimation (either qualitative or quantitative) along with the flow prediction.

Qualitative method A threshold mean squared error tolerance for the flow reconstruction is provided by the user, and is here chosen to be $e_f^* = 5 \times 10^{-5}$. The corresponding threshold shape reconstruction error e_s^* is obtained by solving the minimization problem (6). As the dataset size is relatively limited, the problem is solved by an exhaustive search, the results of which are shown in figure 12a. One observes that choosing $e_s^* = 1.9 \times 10^{-4}$ minimizes the risks of accepting bad predictions and rejecting good predictions. When testing the procedure on the elements of the validation and testing sets, it is observed that the mistake rate is close to 1% in both cases. In figure 12b, the false classification rate on the three subsets is plotted as a function of e_f^*, showing that stricter e_f^* choices inevitably lead to worse performances with this method. The area of accepted predictions is plotted in figure 14a, along with a representation of the elements of the test set.

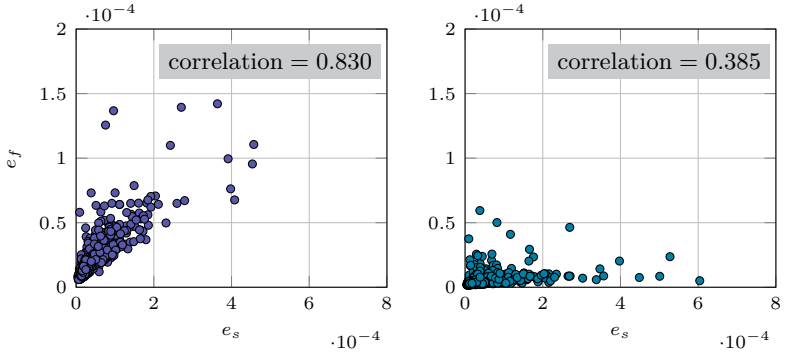
Quantitative method A gradient descent algorithm is used to minimize the negative log-likelihood problem (10) over the training set, in order to obtain the optimal parameter set (a^*, b^*, c^*). With an initial value $(a_0, b_0, c_0) = (0.1, 0, 0.1)$, the algorithm converges after 6 iterations (see figure 13). The optimal parameters retained are $(a^*, b^*, c^*) = (0.257157, 2.248\,20 \times 10^{-6}, 0.105841)$, which minimize the negative log likelihood over the validation set. Hence, e_f can be estimated from e_s as:

$$e_f = 0.257157\, e_s + 2.248\,20 \times 10^{-6} + \mathcal{N}(0, (0.105841\, e_s)^2). \tag{11}$$

As shown in figure 14b, the regression line and its 1σ confidence interval matches well with the distribution of the test set, with only a handful of (e_f, e_s) couples falling outside of the range. The fact that the confidence interval widens with larger values of e_s translates the increasing scarcity of samples in the test set when e_s rises. For the majority of predictions, though, it provides a good grasp

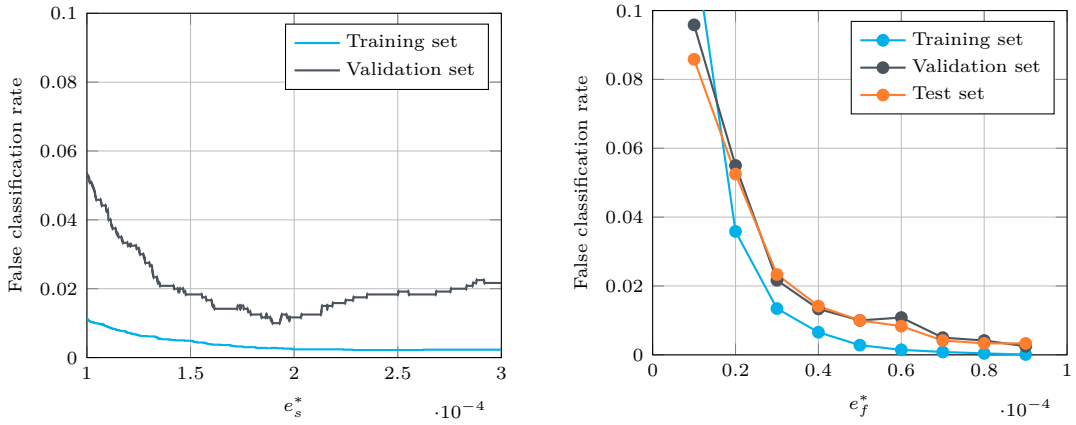


(a) Twin AE, training set (b) Twin AE, validation set (c) Twin AE, test set



(d) Dual AE, test set (e) U-dual AE, test set

Figure 11: **Comparison of scatter plots of e_f versus e_s for different sets and architectures**. Top row: the correlation levels obtained with the twin AE on the training, validation and test sets are respectively 0.772, 0.931 and 0.954. Bottom row: dual AE and U-dual AE architectures show weaker correlation levels on the test set, with respective levels of 0.830 and 0.385.



(a) False classification rate as a function of e_s^* for $e_f^* = 5 \times 10^{-5}$.

(b) False classification rate achieved on different subsets for different e_f^* values.

Figure 12: **Trust level identification based on the e_s indicator.** (Left) The optimal threshold is obtained by minimizing the mistaken classification rate on the training and the validation sets. (Right) The mistake rate rises significantly on all three subsets when decreasing the threshold e_f^* value. For the optimal e_s^* value, the mistake rate on the validation and test sets is approximately 1%.

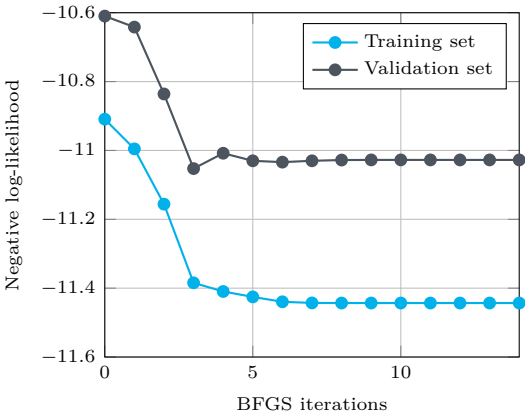
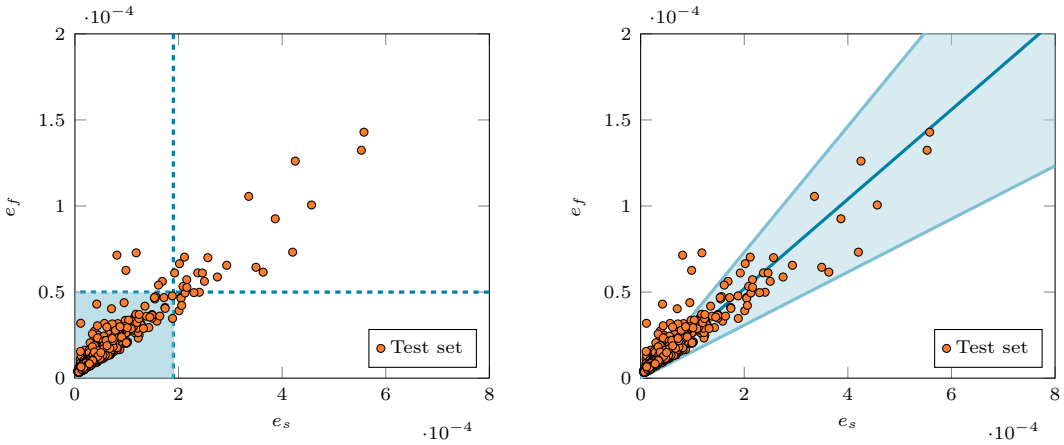


Figure 13: **Minimization of the negative log-likelihood problem (10) using the BFGS algorithm.**



(a) Qualitative method: accepted prediction area when choosing $e_f^* = 5 \times 10^{-5}$ and selecting e_s^* following (6).

(b) Quantitative method: regression line with its 1σ confidence interval obtained by solving problem (10).

Figure 14: **Representations of the qualitative and quantitative methods** along with the test set scatter plot.

of the flow prediction quality. The 1σ (68% probability) and 2σ (95% probability) confidence intervals for sampled e_s values are provided in table 2.

4.5 Flow prediction on outliers

In this section, the capabilities of the qualitative and quantitative methods to detect invalid inputs and outliers are evaluated. To do so, multiple shapes are generated that do not fit within the dataset, including polygons with sharp edges (see figure 15), shapes included in the dataset but misplaced in the input domain (*i.e.* moved away from the position used in the dataset), and enlarged shapes from the dataset. In total, 120 outliers are tested. In table 3, one can see that the relative errors on the different classes of outliers are systematically larger than those obtained on the dataset shapes. While the prediction errors on polygons are only slightly higher that those from the test set, field prediction errors for misplaced shapes are systematically superior to 10%. Enlarged shapes present extremely high reconstruction and prediction errors, higher than 100%. An example of prediction on enlarged Bezier shape is shown in figure 17, illustrating the interest of incorporating uncertainty estimation and outlier detection processes in neural network architectures.

Table 2: **Estimating e_f for given e_s values.**

e_s	e_f (1σ interval)	e_f (2σ interval)
1×10^{-5}	$[3.76 \times 10^{-6}, 5.88 \times 10^{-6}]$	$[2.70 \times 10^{-6}, 6.94 \times 10^{-6}]$
2×10^{-5}	$[5.27 \times 10^{-6}, 9.51 \times 10^{-6}]$	$[3.16 \times 10^{-6}, 1.16 \times 10^{-5}]$
4×10^{-5}	$[8.30 \times 10^{-6}, 1.68 \times 10^{-5}]$	$[4.07 \times 10^{-6}, 2.10 \times 10^{-5}]$
8×10^{-5}	$[1.44 \times 10^{-5}, 3.13 \times 10^{-5}]$	$[5.89 \times 10^{-6}, 3.98 \times 10^{-5}]$
1.6×10^{-4}	$[2.65 \times 10^{-5}, 6.03 \times 10^{-5}]$	$[9.52 \times 10^{-6}, 7.73 \times 10^{-5}]$
3.2×10^{-4}	$[5.07 \times 10^{-5}, 1.18 \times 10^{-4}]$	$[1.68 \times 10^{-5}, 1.52 \times 10^{-4}]$

Table 3: **Model performance on test set and polygons**. The comparison is based on the average pixel-level relative error in the 30×45 rectangular zone around the obstacles.

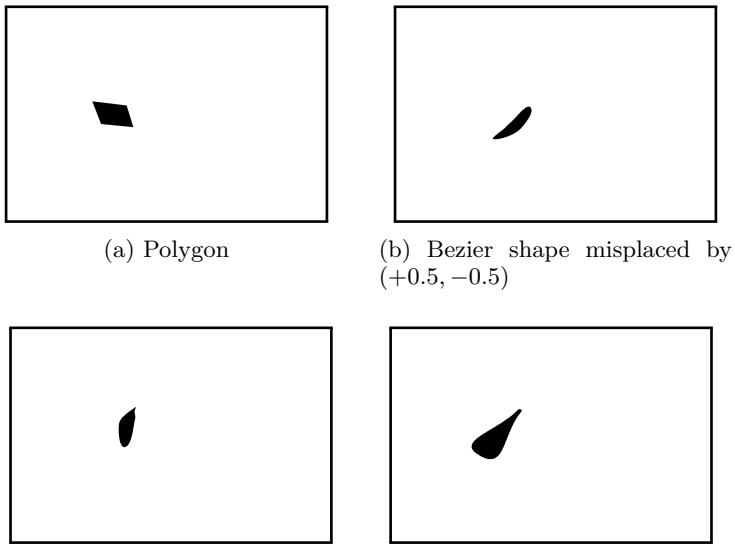
	Shape rel. error	**u rel. error**	**v rel. error**	**p rel. error**
Bezier Test Set	**3.43%**	**3.92%**	**3.57%**	**3.55%**
Polygons	4.16%	6.11%	4.74%	4.70%
Lower right	16.4%	13.5%	17.1%	16.2%
Upper right	16.9%	11.4%	18.2%	14.3%
Enlarged	214%	244%	153%	146%

In figure 16b, the (e_s, e_f) scatter plot of the test set is shown, along with the position of the 120 outliers, both for the qualitative and the quantitative methods. As can be seen, most polygons are located in the accepted region of the qualitative method, indicating that the higher average relative error shown in table 3 is caused by a few polygon outliers with large relative errors. When inspecting the polygons set, it was found out that those with largest e_f errors were presenting edges features that were considerably sharper than the others. Almost all the misplaced and enlarged shapes fall into the rejected area of the qualitative method, while also matching well the $\pm 2\sigma$ interval of the quantitative method (27 enlarged shapes out of 30 are not shown due to their out-of-range MSE.). Still, a handful of outliers presenting low e_s with large e_f are noticed, indicating that the proposed methods are still missing a small amount of outliers. Overall, the qualitative method efficiently detects most of the outliers, with only 10 out of 120 bad predictions not detected (*i.e.* 91.6% of outliers detected). The $\pm 1\sigma$ interval of the quantitative method covers 85 out of 120 (70.8%) outliers, while $\pm 2\sigma$ interval covers 106 out of 120 (88.3%) outlier predictions. The results of figure 16b also indicate that the uncertainty levels of misplaced and enlarged inputs are partly under-estimated.

Overall, the qualitative method efficiently diminishes the risk of mis-use of the trained model, by catching more than 90% of the bad inputs. Still, it remains a limited, binary method, and by construction approximately discards 1% of the dataset points. Conversely, the quantitative method also provides adequate confidence range for almost all the elements from the test set, and for nearly 90% of the outliers. However, the provided error intervals for inputs leading to very large e_f errors are underestimated. Finally, similarly to the qualitative one, the quantitative method cannot account for a handful of points presenting large e_f values in conjunction with small e_s values.

5 Conclusion

In the present contribution, a twin-AE architecture for 2D incompressible laminar flow prediction with embedded uncertainty estimation was presented. The underlying motivation was to propose a method to naturally incorporate outlier detection and uncertainty estimation in the training procedure, in order to provide a decisional tool to the potential end-user. The embedded uncertainty estimation relies on the coupling of the autoencoder for flow prediction with a second autoencoder for input



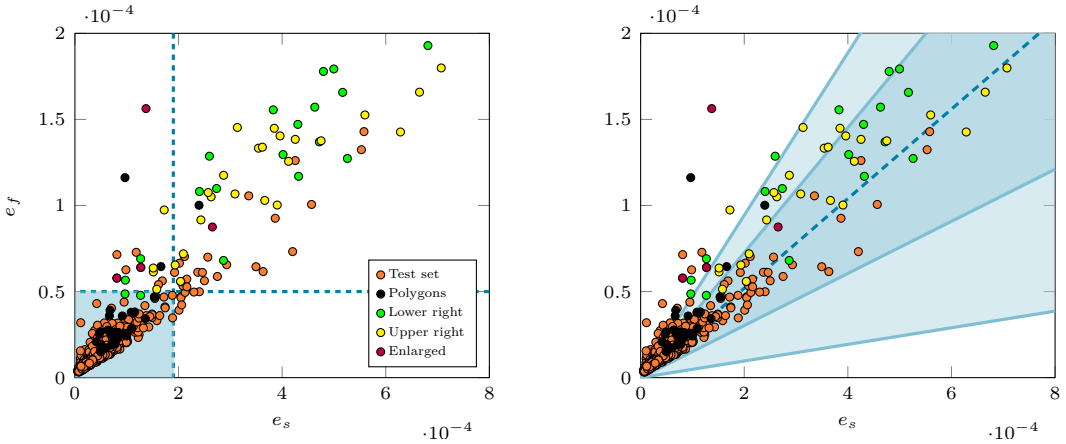
(a) Polygon

(b) Bezier shape misplaced by $(+0.5, -0.5)$

(c) Bezier shape misplaced by $(+0.5, +0.5)$

(d) Enlarged Bezier shape

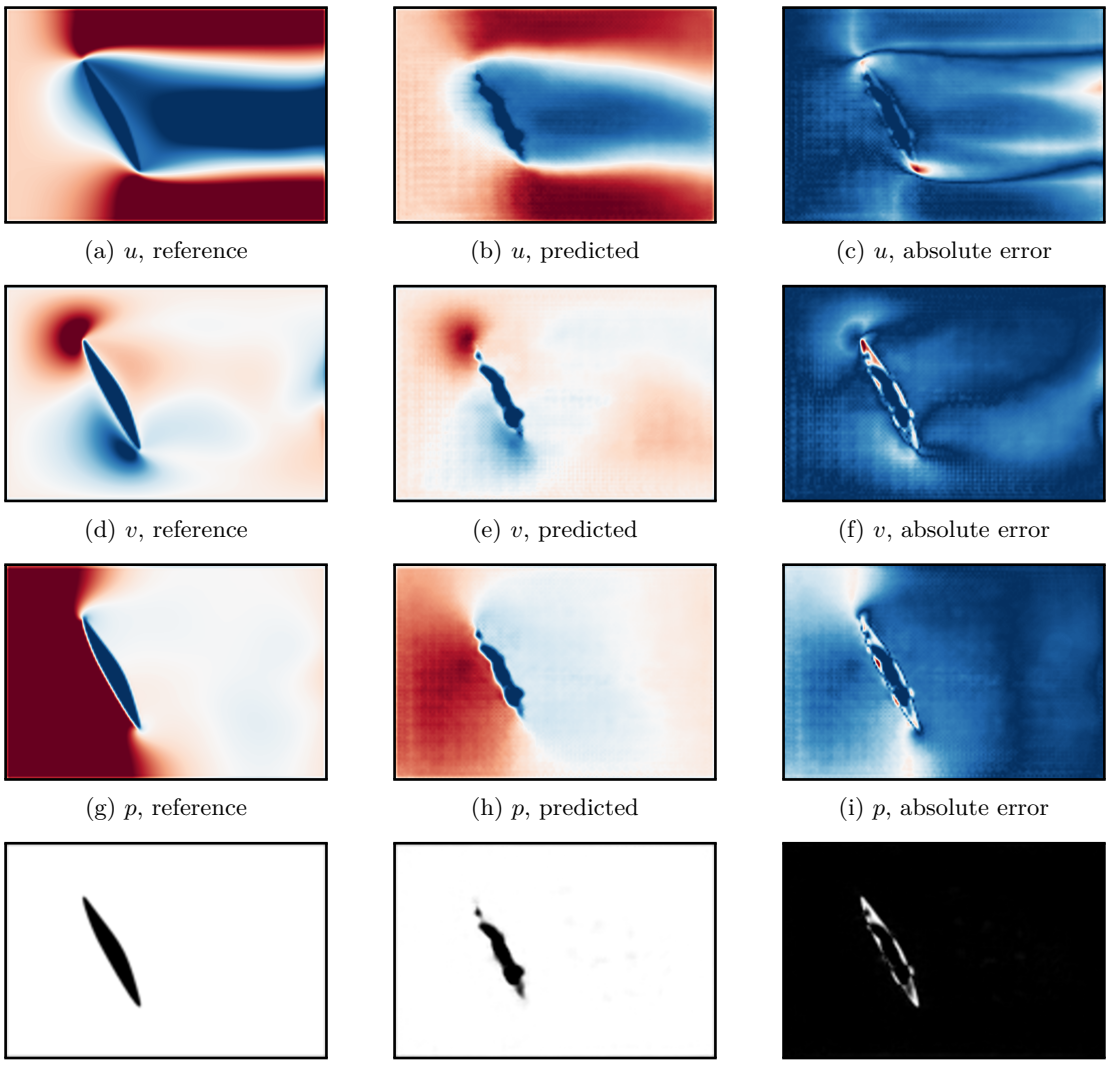
Figure 15: **Outlier examples for model evaluation.** Polygons are generated with fewer sampling points and sharper edges than the dataset shapes. Misplaced and enlarged shapes have the same curve characteristics as the original data set, but are ill-positioned, or significantly larger than those of the dataset.



(a) Qualitative method: using the selection criterion $(e_f^*, e_s^*) = (5\times10^{-5}, 1.9\times10^{-4})$ from previous analysis, 110 bad predictions out of 120 outliers are detected.

(b) Quantitative method: using the regression line from equation 11, 110 bad predictions out of 120 outliers fall into the 2σ confidence interval.

Figure 16: **Prediction and reconstruction error on the outliers.** The qualitative and quantitative methods are illustrated in (a) and (b).



(a) u, reference (b) u, predicted (c) u, absolute error

(d) v, reference (e) v, predicted (f) v, absolute error

(g) p, reference (h) p, predicted (i) p, absolute error

(j) Shape, reference (k) Shape, predicted (l) Shape, absolute error

Figure 17: **Flow and shape predictions around an enlarged Bezier shape.** As this input shape is an outlier, the shape reconstruction is poor, and is associated with large prediction errors ($e_f = 1.316 \times 10^{-2}$). For the u, v and p predictions, the color scales are the same as for figure 4.

reconstruction, using well-chosen skip connections. Doing so naturally enforces a quasi-linear relation between the flow prediction error and the input reconstruction error. Building on this particular trait, simple yet effective qualitative and quantitative techniques were proposed to detect outliers and provide uncertainty prediction on any input provided to the trained network.

The proposed architecture was trained on a dataset of 12000 laminar flows around random 2D shapes, generated using Bezier curves. After hyper-parameter calibration, the correlation coefficient between the reconstruction error and flow prediction error reached 0.95 on the test set. The two methods were then tested on true outliers presenting different flaws (polygonal shapes that did not belong to the dataset, shapes from the dataset misplaced in the input domain, shapes significantly larger than those of the dataset), and proved efficient to either reject shapes with high flow prediction errors, or provide adequate uncertainty range. Still, a handful of outliers remained undetected, or their associated uncertainty range was under-estimated. Possible improvements could be brought to these methods, either by improving the network architecture to improve the flow error/reconstruction error correlation level, or by gaining more control on the dataset generation, in order to avoid the inclusion of possible outliers.

These results underline the potential of the proposed approach. Indeed, the implementation of such methods in prediction tasks can significantly lower the risk of the end-user taking decisions based on network predictions using inadequate inputs. Efforts shall be pursued for more accurate input reconstruction. From the experiments on U-Dual-AE, low-level features from its encoder bring great benefits to flow prediction. If the shape decoder of twin-AE provides equivalently beneficial features, we expect an improved flow prediction while keeping the strong correlation between its twin decoders.

Acknowledgements

This work is supported by the Carnot M.I.N.E.S. Institute through the M.I.N.D.S. project.

A Open source code

The code of this project is available on the following github repository: https://github.com/jviquerat/twin_autoencoder[2].

References

[1] J. Ling, A. Kurzawski, and J. Templeton. Reynolds averaged turbulence modelling using deep neural networks with embedded invariance. Journal of Fluid Mechanics, 807:155–166, 2016.

[2] B. D. Tracey, K. Duraisamy, and J. J. Alonso. A machine learning strategy to assist turbulence model development. In 53rd AIAA Aerospace Sciences Meeting, pages 1–23, 2015.

[3] A. D. Beck, D. G. Flad, and C.-D. Munz. Deep neural networks for data-driven turbulence models. arXiv, 2018.

[4] X. Guo, W. Li, and F. Iorio. Convolutional neural networks for steady flow approximation. In 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 481–490, 2016.

[5] X. Jin, P. Cheng, W. L. Chen, and H. Li. Prediction model of velocity field around circular cylinder over various reynolds numbers by fusion convolutional neural networks based on pressure on the cylinder. Physics of Fluids, 30(4), 2018.

[6] S. Lee and D. You. Data-driven prediction of unsteady flow over a circular cylinder using deep learning. Journal of Fluid Mechanics, 879:217–254, 2019.

[7] Y. Zhang, W. J. Sung, and D. N. Mavris. Application of convolutional neural network to predict airfoil lift coefficient. In 2018 AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, 2018.

[2]The code will be released upon publication of the present manuscript

[8] J. Viquerat and E. Hachem. A supervised neural network for drag prediction of arbitrary 2d shapes in laminar flows at low reynolds number. Computers & Fluids, 210:104645, 2020.

[9] S. Hawkins, H. He, G. Williams, and R. Baxter. Outlier detection using replicator neural networks. In Data Warehousing and Knowledge Discovery, pages 170–180. Springer Berlin Heidelberg, 2002.

[10] J.K. Chow, Z. Su, J. Wu, P.S. Tan, X. Mao, and Y.H. Wang. Anomaly detection of defects on concrete structures with the convolutional autoencoder. Advanced Engineering Informatics, 45:101105, 2020.

[11] M. Ke, C. Lin, and Q. Huang. Anomaly detection of logo images in the mobile phone using convolutional autoencoder. In 4th International Conference on Systems and Informatics (ICSAI), pages 1163–1168, 2017.

[12] C. Baur, B. Wiestler, S. Albarqouni, and N. Navab. Deep autoencoding models for unsupervised anomaly segmentation in brain mr images. In Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries, page 161–16. Springer, 2018.

[13] G. E. Hinton and R. Salakhutdinov. Reducing the dimensionality of data with neural networks. Science, 313:504 – 507, 2006.

[14] K. Lee and K. T. Carlberg. Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders. Journal of Computational Physics, 404:108973, 2020.

[15] S. R. Bukka, A. R. Magee, and R. K. Jaiman. Deep convolutional recurrent autoencoders for flow field prediction, 2020.

[16] F. J. Gonzalez and M. Balajewicz. Deep convolutional recurrent autoencoders for learning low-dimensional feature dynamics of fluid systems, 2018.

[17] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015, pages 234–241, 2015.

[18] N. Thuerey, K. Weißenow, L. Prantl, and X. Hu. Deep learning methods for reynolds-averaged navier–stokes simulations of airfoil flows. AIAA Journal, 58(1):25–36, 2020.

[19] K. Fukami, K. Fukagata, and K. Taira. Super-resolution reconstruction of turbulent flows with machine learning. arXiv preprints, 2018.

[20] Serveh Kamrava, Pejman Tahmasebi, and Muhammad Sahimi. Physics- and image-based prediction of fluid flow and transport in complex porous membranes and materials by deep learning. Journal of Membrane Science, 622:119050, 2021.

[21] J. Bruchon, H. Digonnet, and T. Coupez. Using a signed distance function for the simulation of metal forming processes: Formulation of the contact condition and mesh adaptation. International Journal for Numerical Methods in Engineering, 78(8):980–1008, 2009.

[22] E. Hachem, S. Feghali, R. Codina, and T. Coupez. Immersed stress method for fluid structure interaction using anisotropic mesh adaptation. International Journal for Numerical Methods in Engineering, 94:805–825, 2013.

[23] T.E. Tezduyar, S. Mittal, S.E. Ray, and R. Shih. Incompressible flow computations with stabilized bilinear and linear equal-order-interpolation velocity-pressure elements. Computer Methods in Applied Mechanics and Engineering, 95(2):221–242, 1992.

[24] Y. Bazilevs, V.M. Calo, J.A. Cottrell, T.J.R. Hughes, A. Reali, and G. Scovazzi. Variational multiscale residual-based turbulence modeling for large eddy simulation of incompressible flows. Computer Methods in Applied Mechanics and Engineering, 197(1):173–201, 2007.

[25] K. Takizawa, T. E. Tezduyar, and Y. Otoguro. Stabilization and discontinuity-capturing parameters for space–time flow computations with finite element and isogeometric discretizations. Computational Mechanics, 62(5):1169–1186, 2018.

[26] Y. Otoguro, K. Takizawa, T. E. Tezduyar, K. Nagaoka, R. Avsar, and Y. Zhang. Space–time vms flow analysis of a turbocharger turbine with isogeometric discretization: computations with time-dependent and steady-inflow representations of the intake/exhaust cycle. Computational Mechanics, 64(5):1403–1419, 2019.

[27] Y. Otoguro, K. Takizawa, and T. E. Tezduyar. Element length calculation in b-spline meshes for complex geometries. Computational Mechanics, 65:1085–1103, 2020.

[28] J. Chen, J. Viquerat, and E. Hachem. U-net architectures for fast prediction of incompressible laminar flows, 2019.