Journal of Computational Physics 535 (2025) 114027

Contents lists available at ScienceDirect

Journal of Computational Physics

journal homepage: www.elsevier.com/locate/jcp

Learning the solution operator of two-dimensional incompressible Navier-Stokes equations using physics-aware convolutional neural networks ☆

Viktor Grimm [a,*], Alexander Heinlein [c,*], Axel Klawonn [a,b,*]

[a] Department of Mathematics and Computer Science, University of Cologne, Weyertal 86-90, 50931 Köln, Germany
[b] Center for Data and Simulation Science, University of Cologne, 50923 Köln, Germany
[c] Delft Institute of Applied Mathematics, Delft University of Technology, Mekelweg 4, 2628 CD, Delft, Netherlands

ARTICLE INFO

Keywords:
Convolutional neural networks
Computational fluid dynamics
Machine learning
Scientific machine learning

ABSTRACT

In recent years, the concept of introducing physics to machine learning has become widely popular. Most physics-inclusive ML-techniques however are still limited to a single geometry or a set of parametrizable geometries. Thus, there remains the need to train a new model for a new geometry, even if it is only slightly modified. With this work we introduce a technique with which it is possible to learn approximate solutions to the steady-state Navier–Stokes equations in varying geometries without the need of parametrization. This technique is based on a combination of a U-Net-like CNN and well established discretization methods from the field of the finite difference method. The results of our physics-aware CNN are compared to a state-of-the-art data-based approach. Additionally, it is also shown how our approach performs when combined with the data-based approach.

1. Introduction

Fluid behavior is important in various fields such as civil, mechanical, and biomedical engineering, aerospace, meteorology, and geosciences. The governing equations for fluid behavior are typically the Navier-Stokes equations, which are solved using discretization approaches like finite difference, finite volume, or finite element methods. However, such computational fluid dynamics (CFD) simulations can be computationally intensive, especially for turbulent flow and complex geometries, and changing the geometry requires recomputing the entire simulation. Hence, there is a need for a quick surrogate model for CFD simulations. Such surrogate models encompass a variety of approaches, including linear reduced order models [15,37], such as reduced basis [50] and proper orthogonal decomposition [55] models, as well as neural network-based models [14], like convolutional neural networks (CNNs) [10,13,27,39,45,31,63] and neural operators [35,44].

In present work, we focus on using neural networks as an approximation for CFD simulations. Instead of relying on a large dataset, we leverage the known governing equations of fluids to construct a physics-aware loss function and train our model to satisfy these equations discretely. This approach has recently become increasingly popular and was applied to dense neural networks (DNN) to solve

☆ This work is based on the doctoral dissertation of the first author at the Faculty of Mathematics and Natural Science of the University of Cologne.

* Corresponding authors.
E-mail addresses: viktor.grimm@uni-koeln.de (V. Grimm), a.heinlein@tudelft.nl (A. Heinlein), axel.klawonn@uni-koeln.de (A. Klawonn).

https://doi.org/10.1016/j.jcp.2025.114027
Received 3 August 2023; Received in revised form 29 September 2024; Accepted 18 April 2025

Available online 23 April 2025

partial differential equations (PDEs) with little training data [52] or without training data [65] as well as inverse problems with limited training data [25,52]. More recently, this idea was also applied to convolutional neural networks (CNN) by using physics-aware loss functions to solve PDEs [2,11,16,57,61,71], upscale and denoise solutions [17,30], generally improve the predictive quality of a model [60,70], or learn PDEs from data [42,43]. For a comprehensive overview on scientific machine learning (SciML), we refer to [5,69].

However, previous physics-informed machine learning approaches have imposed geometric constraints, such as rectangularity or parametrizability, or have been limited to specific geometries or even to a single geometry. In practice, these conditions are usually not met. Furthermore, the exact geometry is often unknown or at least not known in sufficient detail. This is, for example, the case with medical imaging procedures. Therefore, we aim to develop a CNN that is capable of learning the flow field and pressure under physics constraints without labeled data for, with some restrictions, arbitrary geometries. We explicitly do not use methods such as coordinate transformations, which would limit us to specific geometries. This approach aims to advance existing methods for more realistic applications, and to the authors' knowledge, it is the first attempt to use a single CNN for multiple irregular geometries without labeled data.

The rest of this paper is organized as follows. We first define our stationary boundary value problem in section 2. Then, we introduce CNNs as surrogate models in section 3 and directly afterwards extend this framework to physics-aware label-free learning of PDE solutions in section 4 and its application to the Navier-Stokes equations in section 5. Next, in section 6, we explain the architecture of our surrogate model. The creation of the training data used here is described in section 7. We present results for the data-based and physics-aware approach in section 9. Finally, we draw conclusions in section 10.

In the remainder of this section, we will discuss how our approach is positioned in the areas of *physics-informed machine learning* and *operator learning*.

1.1. Physics–informed machine learning

The method described in this paper can be categorized as physics-informed Machine Learning (ML) [47]. This term is used to describe ML methods for which prior knowledge, often in terms of known physical laws, are explicitly incorporated into the model. Since we use the governing equations to train our physics-aware CNN, our method can also be referred to as a physics-informed NN (PINN). However, it is important to note that the discretization in our approach differs from that in classical PINNs, as introduced in [51,52] and pioneered in earlier work [7,36]. In particular, in classical PINNs, the NN model itself is employed to discretize the differential equations; the input of a classical PINN are the spatio-temporal coordinates, and the output approximates of the solution at that time and location. Then, the derivatives required to evaluate the residual of the differential equations are computed exactly via back-propagation [59], that is, using automatic differentiation. Other approaches related to PINNs that are based on the variational formulation are the Deep Ritz method [8] and variational PINNs [32].

In our approach, however, the model input is an image of the geometry and the output are images of the solution field. Then, we employ a finite difference discretization of the differential equation using the centers of pixel images as the grid nodes. This can be seen as a different way of discretizing in physics-informed machine learning compared with the aforementioned approaches. As discussed in the introduction, physics-informed discretizations using CNNs on images have also been employed in [2,11,16,57,61,71]. In contrast to other works, our approach focuses on making predictions for variations in the geometry of the computational domain. Note that while we focus on finite difference discretizations on pixel images for a proof-of-concept, our approach could also be extended to finite element or finite volume discretizations.

For an overview of physics-informed machine learning, we refer to the literature; see, for instance, [4,29,3,47,6].

1.2. Operator learning

In [35], the authors introduce the concept of neural operators as the use of neural networks for approximating maps between infinite-dimensional function spaces. Neural operators are intended to be discretization-invariant in the sense that they can be applied to different discretizations of the input function, can be evaluated at any point in the output domain, and converge to a continuous operator when refining the discretization. Certain architectures have been proven to be universal approximators for operators: they can approximate any continuous operator to arbitrary precision, given sufficient model capacity. Popular approaches are Fourier neural operators (FNOs) [41] and deep operator networks (DeepONets) [44], many variants of which have been introduced in recent years. Furthermore, in [54], convolutional neural operators (CNOs) have been introduced, which satisfy structure-preserving continuous-discrete equivalence and hence enable learning operators without discretization-dependent aliasing errors. The CNO architecture is an extension of the U-Net [58], which is also the basis for the architecture employed in this paper.

In this work, as briefly mentioned before, we aim at learning approximating the solution operator, mapping a parametrization of the geometry to the solution of the stationary Navier–Stokes equations. Therefore, our approach is an example of operator learning. Strictly speaking our model does not necessarily satisfy the discretization-invariance required in [35]. However, since our model is fully convolutional, it could be applied to a different discretization than it was trained on; we leave these investigations to future research.

Note that we could use, for instance, FNO or CNO architectures since they are also based on structured-grid input data. The classical DeepONet or extensions to graph neural networks [21], such as the DeepGraphONet introduced in [66], naturally employ other data structures and are therefore not directly applicable or comparable without data interpolation. However, in principle, physics-informed DeepONet models could [22] also be combined with the approach introduced here, when using different input and output data structures.

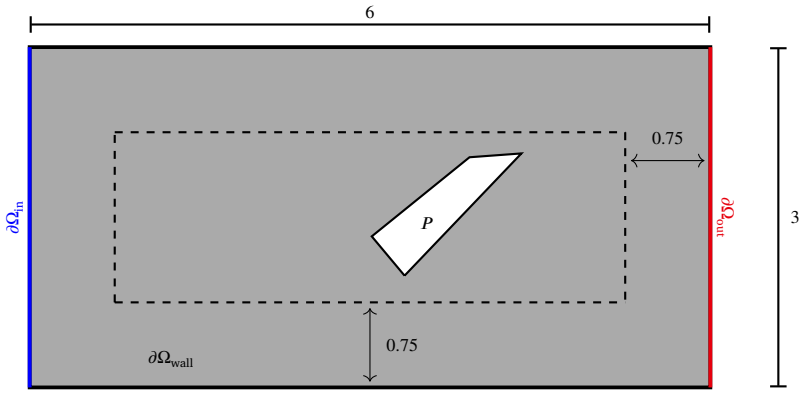


Fig. 1. Example of a channel geometry Ω with a star-shaped obstacle. The obstacle is confined within a box (dashed line) with a distance of 0.75 to the boundary of Ω.

2. Model problem

We consider the stationary Navier–Stokes equations describing incompressible Newtonian fluids with constant density

$$\begin{aligned}(\vec{u}\cdot\nabla)\vec{u} - \nu\Delta\vec{u} + \nabla p = 0 \quad &\text{in } \Omega,\\ \nabla\cdot\vec{u} = 0 \quad &\text{in } \Omega,\end{aligned} \tag{1}$$

where $\vec{u}$ is the velocity field, p the pressure, and ν the kinematic viscosity.

In our experiments, we consider rectangular channels $\Omega = (0,6)\times(0,3)$ from which we have cut a star-shaped obstacle P; cf. Fig. 1 for an example. This design is inspired by [27,10]. We apply the following boundary conditions: On the inlet $\partial\Omega_{in} := 0\times(0,3)$, we prescribe a constant inflow velocity $u = (3,0)^\top$, and at the outlet $\partial\Omega_{out} := 6\times(0,3)$, we fix the pressure to $p = 0$. The lower and upper parts of the boundary $0\times[0,6]$ and $3\times[0,6]$, respectively, correspond to walls, and hence we enforce no-slip conditions $u = (0,0)^\top$. Finally, we choose $\nu = 5\cdot 10^{-2}$. The Reynolds number based on the channel height is $Re = (u_m\cdot d)/\nu = 180$, where $u_m = 3$ is the velocity averaged over the cross-section of the channel. Depending on the shape and position of the obstacle, this setup leads to strongly varying flow patterns, making the channel problem a challenging benchmark for a CFD surrogate model.

3. Surrogate models based on CNNs

Let us first discuss the approach to construct surrogate models via CNNs from [27,10], which is the basis for this work. CNNs [38] are artificial neural networks (ANNs) that employ linear transformations based on discrete convolutions within the network layers making them well suited for structured temporal or spatial data, where neighboring coefficients correspond to neighboring points in space or time, respectively.

CNNs are therefore also suitable for the approximating the solutions of partial differential equations on a structured tensor product grid: even though interaction is typically global, it is strongest for neighboring nodes. If the data structure is based on unstructured grids, graph convolutional networks (GCNs) [34] can be employed as an alternative. Here, we only consider tensor product-structured data and therefore restrict ourselves to classical CNNs.

In the approach from [27,10], a CNN that maps from the geometry of the computational domain to solution field(s) of the corresponding boundary value problem is trained; here, we specifically aim at predicting the velocity and pressure fields satisfying the Navier–Stokes equations eq. (1). In order to be able to employ standard CNNs, the geometry and solution fields are therefore interpolated to a tensor product grid. In two dimensions, the resulting data has a simple matrix structure; see Fig. 2 for an illustration of this process and Fig. 9 for an exemplary pair of input and output data. As can be seen, due to the matrix structure of the input and output data, they can be directly identified as pixel images. This also allows us to use a large variety of techniques from the application of CNN models to image data.

Let us now give a formal introduction of the approach. Therefore, let $I_g \in \mathbb{R}^{w\times h}$ be the pixel image matrix representing some computational domain Ω_g; g indicates a generic index for a specific geometry. Moreover, let $u_g^i \in \mathbb{R}^{w\times h}$ be a matrix representation of the ith component of the solution field of the boundary value problem solved on the computational domain Ω_g. Here, w and h correspond to the width and height of the pixel images, as well as the number of interpolation nodes in the x and y directions. By assembling the tensors u_g^i, $i = 1,\dots,d$, we obtain a third-order tensor $u_g \in \mathbb{R}^{d\times w\times h}$, assuming that all solution components are defined on the same pixel grid. For a scalar-valued problem d is one, and for a vector-valued problem, d is larger than one. For instance, for the Navier–Stokes equations in two dimensions, we have two velocity components and one pressure component, such that d is three. In analogy to pixel images, each component of the solution field is regarded as one channel of the output image.

Our goal is to train a CNN that approximates the solution operator

$$\begin{aligned}\mathcal{U} : \mathbb{R}^{w\times h} &\to \mathbb{R}^{d\times w\times h}\\ I_g &\to u_g,\end{aligned}$$

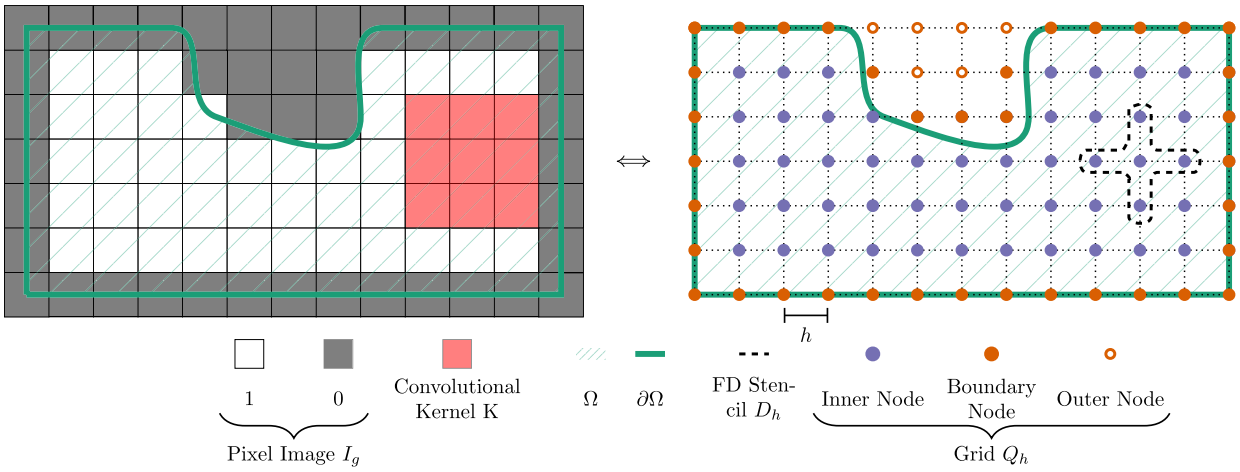


Fig. 2. A pixel image I_g (left) and the corresponding FD-grid Ω_h (right) of a geometry Ω, whose border $\partial\Omega$ is drawn as a thick solid green line in the figure. Applying a five-point finite difference stencil D_h^k to the FD-grid is equivalent to applying a convolutional filter K with fixed weights to the pixel image I_g. Note that the values associated with the pixels and their grid node counterparts are not depicted. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

that is, the operator that maps a pixel representation of the geometry of the computational domain to a pixel representation of the solution of the corresponding boundary value problem. Hence, our approach can be seen as an example of operator learning; cf. the related DeepONet [44] and Fourier neural operator [41] approaches, which employ different network architectures.

Let us denote the CNN model by f_{NN}^{Ψ}, where Ψ are the trainable network parameters. In [27,10], a CNN f_{NN} has then been trained to approximate the solution operator $\mathcal{U}$ in a purely data-driven way. In particular, high-fidelity simulation data has been employed as the reference data u_g, and the model has been trained to minimize the mean squared error (MSE) between the model output and u_g. This corresponds to the minimization problem:

$$\arg\min_{\Psi} \frac{1}{|T|} \sum_{g \in T} \left\| f_{NN}^{\Psi}(I_g) - u_g \right\|^2, \tag{2}$$

where T is a set of geometries used as training data. Note training the model f_{NN}^{Ψ} with this loss function requires the availability of reference data u_g; this means that a large number of measurement or high fidelity simulation data has to be available before the model training.

It remains to discuss how to construct I_g and u_g for a specific geometry g. The approach is not restricted to a specific image representation of the geometry I_g, and in [10,27], a binary or signed distance function (SDF)-based image of the geometry has been employed. It can be observed that the SDF input yields slightly better results. However, it comes at a computational cost, and the computation of the exact SDF input image requires precise knowledge about the boundary of the geometry; in practical applications, for instance, when the geometry is only known from medical image data, the SDF function can only be computed approximately based on the available image data. A binary input image can be generated more easily by checking if the center or most of the volume of each pixel lies within the computational domain Ω_g. Here, we only consider binary input images.

The output pixel images u_g can be constructed from a reference solution $\hat{u}$ by an interpolation operator, for instance, pointwise interpolation in the center points of the pixels or by averaging over the pixels (Clément-type interpolation). Here, $\hat{u}$ could be high-fidelity simulation or measurement data. We discuss the data processing for this paper in more detail in section 7.

Next, we introduce the main novelty of this paper, that is, our approach to replace the data-based loss function eq. (2) by a physics-aware loss function that requires only the knowledge of the PDE.

4. A physics-aware surrogate model based on CNNs

In this section, we extend the approach from Section 3 by incorporating knowledge of the mathematical PDE model during network training. While the loss function in eq. (2) relies on reference data, our novel approach only requires a mathematical formula for the PDE residual, although a combination of both loss functions is possible. By considering the network output as a discrete finite difference solution on the same grid, we can approximate the PDE residual using finite difference stencils. This method can be extended to other discretization approaches on a structured grid, such as finite element or finite volume methods, but we focus on finite difference discretization for simplicity.

In this section, we extend the approach in section 3 by including knowledge of the mathematical PDE model into network training. Whereas the loss function in eq. (2) relies on reference data, our novel approach only requires a mathematical formula for the residual of the PDE; a combination of both loss functions is also possible. By considering the network output as a discrete finite difference solution on the pixel image grid, we can approximate the PDE residual using finite difference stencils. This method can be extended to

other discretization approaches on a structured grid, such as finite element or finite volume methods, but we focus on finite difference discretization for simplicity and leave other discretization approaches to future work.

Like the data-driven approach described in section 3, our new approach yields a surrogate model capable of predicting solutions for a range of geometries of the computational domain. It can be trained without using reference data or in combination with reference data. To introduce the approach, we first explain how to apply it to a stationary diffusion equation; see also [24] for preliminary results for the stationary diffusion equation. Then, we discuss specifically how to apply the approach to the two-dimensional Navier–Stokes equations and the handling of boundary conditions.

4.1. Finite differences and discrete convolutions

In order to derive the implementation of finite difference stencils based on the discrete convolution respectively cross-correlation operation, which is generally used in convolutional neural networks, we first consider a simple stationary diffusion problem: find the function u such that

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f \quad \text{in } \Omega = (0,1)^2, \tag{3}$$

where we assume that f is a continuous function on Ω. Later, in section 5, we discuss the application to the Navier–Stokes equations, which are our focus in this work. Note that, in this whole subsection, we neglect the treatment of boundary conditions; we discuss this directly in the context of the application of our approach to the Navier–Stokes equations in section 4.3.

Now, let us introduce a uniform grid $\Omega_h = \left\{ \boldsymbol{x}_{ij} \mid 1 \le i, j \le n+1 \right\}$, with $\boldsymbol{x}_{ij} = ((i-1)\,h, (j-1)\,h)$ and $h = \frac{1}{n}$, and let $U^h = \left(U^h_{ij}\right)_{ij} \in \mathbb{R}^{(n+1)\times(n+1)}$ with $U^h_{ij} \approx u\left(\boldsymbol{x}_{ij}\right)$ the matrix representation of a discretization of u. Then, $u^h = \left(u^h_i\right)_i \in \mathbb{R}^{(n+1)\cdot(n+1)}$ with

$$u^h_{(j-1)\cdot(n+1)+i} = U^h_{ij} \tag{4}$$

is the corresponding vector representation in lexicographical order. Then, discretizing eq. (3) using central differences involves the approximation

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{U^h_{i+1,j} - 2U^h_{i,j} + U^h_{i-1,j}}{h^2} \quad \text{and} \quad \frac{\partial^2 u}{\partial y^2} \approx \frac{U^h_{i,j+1} - 2U^h_{i,j} + U^h_{i,j-1}}{h^2}, \tag{5}$$

which could also be rewritten in terms of the entries of u^h using eq. (4). This leads to a linear system of equations

$$Au^h = f^h. \tag{6}$$

For the right-hand side vector, let $F^h = \left(F^h_{ij}\right)_{ij} \in \mathbb{R}^{(n+1)\times(n+1)}$ be the matrix representation with $F^h_{ij} = f\left(\boldsymbol{x}_{ij}\right)$ and $f^h = \left(f^h_i\right)_i \in \mathbb{R}^{(n+1)\cdot(n+1)}$ be the corresponding vector representation, where

$$f^h_{(j-1)\cdot(n+1)+i} = F^h_{ij}. \tag{7}$$

We can observe that

$$Au^h = f^h \quad \Leftrightarrow \quad U^h * K = F^h, \tag{8}$$

where $*$ is the cross-correlation operation and

$$K = \frac{1}{h^2}\begin{pmatrix} K_{-1,-1} & K_{-1,0} & K_{-1,1} \\ K_{0,-1} & K_{0,0} & K_{0,1} \\ K_{1,-1} & K_{1,0} & K_{1,1} \end{pmatrix} = \frac{1}{h^2}\begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}, \tag{9}$$

is the kernel which corresponds to the finite difference stencil of the central difference scheme eq. (5). The cross-correlation is the linear transformation implemented in the convolutional layers in CNNs of current state-of-the-art machine learning libraries; cf. [20, section 9.1].

In general form, the cross-correlation is given by

$$(I * K)_{ij} = \sum_m \sum_n I_{i+m,j+n} K_{m,n}, \tag{10}$$

where I is some matrix and K is, again, a kernel matrix, such as the one given in eq. (9). As by convention, we omit the range of the sums, and regard each matrix coefficient as zero which is outside the range of indices. Note that flipping the kernel in eq. (10) yields the discrete convolution

$$(I \tilde{*} K)_{ij} = \sum_m \sum_n I_{i-m,j-n} K_{m,n};$$

see, for instance, [20, Section 9.1]. Since, in a CNN, the entries of the kernel are generally trainable, the cross-correlation and the discrete convolution are equivalent in that sense.

By numbering the rows and columns of K from -1 to 1, as done in eq. (9), we can easily show the equality of the left hand sides in eq. (8):

$$\begin{aligned}\left(Au^h\right)_{(j-1)(n+1)+i} &\overset{eq.(5)}{=} \frac{1}{h^2}\left(U^h_{i-1,j} + U^h_{i,j+1} - 4U^h_{i,j} + U^h_{i,j-1} + U^h_{i+1,j}\right)\\ &= \sum_{m=-1}^{1}\sum_{n=-1}^{1} U_{i+m,j+n}K_{m,n} \overset{eq.(10)}{=} \left(U^h * K\right)_{ij}\end{aligned}$$

The equality of the right hand sides follows from eq. (7), showing that a finite difference discretization can be implemented using cross-correlation in a CNN. Next, we will use this analogy to derive a physics-aware loss function in the context of CNNs.

4.2. Derivation of a physics-aware loss function

Let us consider a generic system of PDEs given in implicit form on the same computational domain $\Omega = [0,1]^2$:

$$F\left(x,\ u(x),\ \frac{\partial u}{\partial x_1}(x),\ \frac{\partial u}{\partial x_2}(x),\ \dots\right) = 0, \quad x \in \Omega. \tag{11}$$

Here, F is a nonlinear function which may depend on partial derivatives of u of any order. Therefore, eq. (11) is a generalization of the diffusion equation eq. (3).

Analogously to section 4.1, we can discretize eq. (11) by approximating the derivatives using finite differences on a structured $n+1 \times n+1$ grid. Appropriate finite differences schemes for various PDEs can be found in the literature; see, for instance, [40,62,64]. In section 5, we discuss the specific case of the Navier–Stokes equations, which are the main application discussed in this work. As mentioned before, other discretization schemes on structured grids, such as finite element or finite volume discretizations can also be used.

Let $U^h \in \mathbb{R}^{d\times(n+1)\times(n+1)}$ be the tensor representation of the discrete solution with d components. Then, the discrete problem corresponding to eq. (11) can be written as

$$F^h\left(X^h,\ U^h,\ U^h * D^x,\ U^h * D^y,\ \dots\right) = 0, \tag{12}$$

where $X^h = \left(\mathbf{x}^h_{ij}\right)_{ij} \in \mathbb{R}^{2\times(n+1)\times(n+1)}$ is the tensor containing all the grid nodes, and D^x and D^y are kernel matrices corresponding to the finite difference discretization of the partial derivatives

$$\frac{\partial u}{\partial x_1} \quad \text{and} \quad \frac{\partial u}{\partial x_2},$$

respectively. As shown in section 4.1 for the example of a standard five-point stencil, any finite difference discretization of a partial derivative can be written as the cross-correlation with the corresponding finite difference stencil. Higher derivatives can therefore be treated analogously.

The generally nonlinear system of equations eq. (12) can be reformulated as a least-squares problem for the discrete residual

$$\arg\min_{U^h} \left\| F^h\left(X^h,\ U^h,\ U^h * D^x,\ U^h * D^y,\ \dots\right)\right\|_2^2. \tag{13}$$

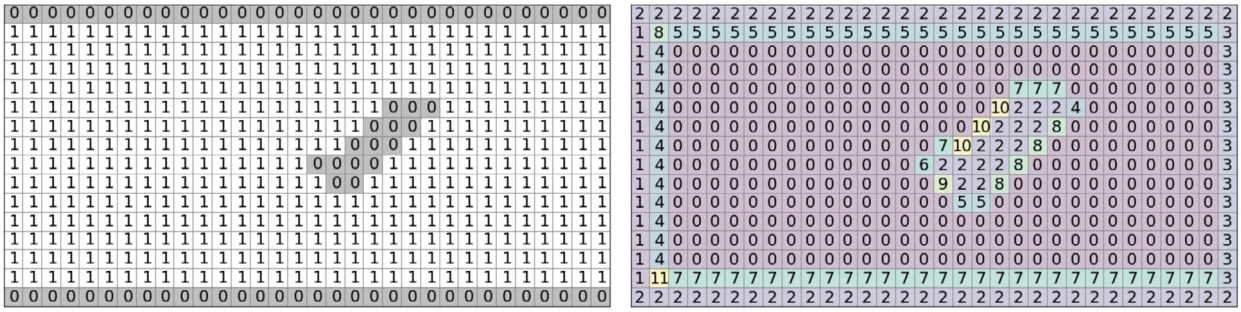
Both problems are equivalent when the same boundary conditions are imposed. While solving eq. (12) benefits classical numerical solvers, the minimization problem eq. (13) is better suited for a neural network approach. The solution tensor U^h on the grid Ω_h can be replaced by the CNN output $f^{\Psi}_{N!N}$, resulting in

$$\arg\min_{\Psi} \left\| F^h\left(X^h,\ f^{\Psi}_{NN},\ f^{\Psi}_{NN} * D^x,\ f^{\Psi}_{NN} * D^y,\ \dots\right)\right\|_2^2, \tag{14}$$

where Ψ represents the network parameters. The cross-correlation can be easily implemented since it is a standard operation in state-of-the-art deep learning libraries.

This approach is related to physics-informed neural networks (PINNs) that classically use dense feedforward neural networks for discretization. In classical PINNs, the residual of the partial differential equation is also minimized in a least-squares sense. Therefore, the differential operator is evaluated by automatic differentiation of the network function using the back propagation algorithm [59]. Our CNN-based approach differs as it employs a classical discretization, for instance, based on finite difference stencils. In our approach, the neural network predicts the coefficients of the discrete solution; cf. section 4.1.

In eq. (14), the CNN input is intentionally omitted. In fact, if the solution of eqs. (12) and (13) is unique due to appropriate boundary conditions and finite difference discretization, the solution does not depend on any input parameters. Hence, it is sufficient to train the neural network for a constant output; this could be simply realized via a bias vector in the output layer. The approximation of the solution of a single boundary value problem using this approach is similar to the classical PINNs approach but using a finite difference discretization; we refer to the discussion in section 1.1. This would require training the neural network from scratch for every geometry and is therefore not as efficient as training a single neural network for approximating the operator mapping the geometry to the corresponding solution; cf. sections 1.2 and 4.3. For solving a single boundary value problem, classical numerical solvers are currently more efficient. In line with this, the discussion in [24] shows that, for a simple stationary diffusion problem,



(a) Geometry image (b) Boundary image

Fig. 3. Low-resolution pixel image inputs that are used by our model. The geometry image Fig. 3a is passed as input to the CNN and the boundary image Fig. 3b is used for the construction of the physics-aware loss. The geometry represented here was previously described in Fig. 1.

solving eq. (14) using an SGD-based optimizer cannot compete with solving the discrete system in eq. (12) using classical numerical solvers, such as gradient descent or the conjugate gradient method.

However, the physics-aware loss function eq. (14) becomes relevant once the CNN model serves as a surrogate model for multiple configurations parameterized by the input of the CNN model; cf. section 3 for the data-based approach. Analogously, we consider image representations describing the geometry of the computational domain.

4.3. Geometry-dependency and boundary conditions

To extend the physics-aware loss function for variations in the geometry of the computational domain, we introduce a discretization of eq. (11) that is compatible with the pixel image representation employed in the CNN. For this purpose, we consider a rectangle Q encompassing Ω and use an equidistant grid Q_h with grid step size h to discretize it. The set of grid nodes is denoted as $X = \{x_{i,j}\}$, with w grid nodes in the x direction and h grid nodes in the y direction; cf. Fig. 2.

Now, let I_g be an input image describing the geometry of the computational domain Ω; even though we can generally employ any geometry representation, it is important that the boundary pixels are uniquely determined because we explicitly use them in our approach; see, for instance, Fig. 3a. Plugging the CNN model $f_{NN}(Ig)$, as described in section 3, into the physics-aware minimization problem eq. (14), we then obtain

$$\underset{\Psi}{\operatorname{argmin}} \left\| F^h\left(X^h,\ C_{I_g}\left(f_{NN}^{\Psi}\left(I_g\right)\right),\ C_{I_g}\left(f_{NN}^{\Psi}\left(I_g\right)\right) * D^x,\ C_{I_g}\left(f_{NN}^{\Psi}\left(I_g\right)\right) * D^y,\ \dots\right)\right\|_2^2. \tag{15}$$

Here, we only enforce the physics-awareness for those pixels which are inside the computational domain, as indicated by the input image representation I_g, and the operator C_{I_g} corresponds to enforcing the boundary conditions. In the following we will abbreviate the notation of F^h and write $\left\| F^h\left(X^h, f_{NN}^{\Psi}\left(I_g\right), \dots\right)\right\|$ for ease of readability.

In order for eq. (15) to be well-defined, we have to prescribe boundary conditions at the boundary pixels. There are at least two ways of enforcing boundary conditions in the context of physics-based neural network models. In particular, we can either add a loss term associated with the boundary conditions or explicitly encode the boundary conditions in the network function. In the literature, the former is also denoted as *soft enforcement* of boundary conditions, whereas the latter is denoted as *hard enforcement* of boundary conditions; cf. [65] for a more detailed discussion. It has been observed in the literature that soft enforcement of boundary conditions can be problematic in different ways: it can make the training less robust and also may lead to cases where the training does not converge to the solution; see for example [65]. Therefore, we focus on hard enforcement of boundary conditions. For instance, in case of Dirichlet boundary conditions, can be easily done by explicitly writing the correct values in the output image of the neural network before applying the loss function; at the same time, and as in classical discretization methods, we do not enforce the physical loss in those pixels. The explicit enforcement of the boundary conditions is indicated by the operator C_{I_g} in eq. (15). If different boundary conditions are prescribed on different parts of the boundary, we encode this by specific values in the input image; cf. Fig. 3. See also section 5.2 for a specific discussion of our implementation of boundary conditions for the Navier–Stokes equations.

Now, we extend the training of the surrogate model to multiple geometries. Therefore, analogously to the data-driven case eq. (2), we optimize the loss function over a training data set of geometries T, resulting in the following loss function:

$$\underset{f_{NN}}{\operatorname{argmin}} \frac{1}{|T|} \sum_{g \in T} \left\| F^h\left(X^h, f_{NN}^{\Psi}\left(I_g\right), \dots\right)\right\|_2^2. \tag{16}$$

The main difference to the data-based loss function eq. (2) from section 3 is that no reference flow data f_g but only the mathematical model of the PDE is necessary for the training. Of course, both loss functions can also be combined into a hybrid loss function

$$\underset{\Psi}{\operatorname{argmin}} \frac{1}{|T|} \sum_{g \in T} \omega_{\text{PDE},g} \left\| F^h\left(X^h, f_{NN}^{\Psi}\left(I_g\right), \dots\right)\right\|_2^2 + \omega_{\text{data},g} \left\| f_{NN}^{\Psi}\left(I_g\right) - u_g\right\|^2, \tag{17}$$

where the weights $\omega_{\text{PDE},g}$ and $\omega_{\text{data},g}$ balance the two loss terms. Different variants of the hybrid loss function are possible, for instance,

$$\begin{aligned} \omega_{\text{data},g} &= \alpha \quad \wedge \quad \omega_{\text{PDE},g} = 0, \quad \text{if reference data is available,} \\ \omega_{\text{data},g} &= 0 \quad \wedge \quad \omega_{\text{PDE},g} = \beta, \quad \text{otherwise,} \end{aligned}$$

or

$$\begin{aligned} \omega_{\text{data},g} &= \alpha \quad \wedge \quad \omega_{\text{PDE},g} = \beta, \quad \text{if reference data is available,} \\ \omega_{\text{data},g} &= 0 \quad \wedge \quad \omega_{\text{PDE},g} = \beta, \quad \text{otherwise.} \end{aligned}$$

Here, $\alpha, \beta > 0$ are some weight parameters. The choice of weights α and β in the loss function is key to balancing the contributions of the data-based loss and the physics-aware loss. Choosing appropriate values for α and β can significantly influence the performance of the model and its ability to generalize. The optimal values for these weights depend on the specific problem and dataset. As a general guideline, however, if α is too large relative to β, the model may fit the data well but fail to respect the physical constraints, potentially leading to unrealistic solutions. Conversely, if β is chosen to be large, the model may adhere to strictly to the physics and fail to capture the nuances in the data. A carefully tuned balance allows the model to capitalize on the strengths of both the data and the underlying physics. Other strategies for choosing the weights are, of course, also possible. For a theoretical discussion based on the neural tangent kernel on how to balance PDE and data loss terms for classical PINNs, see [68].

Next, we discuss the details of our model for the specific problem considered here, that is, the Navier–Stokes equations eq. (1).

5. Application to the Navier–Stokes equations

In this work, we are concerned with the application of our approach to the Navier–Stokes equations. Therefore, we discuss, in this section, the derivation of the physics-aware loss function and the treatment of the boundary conditions.

5.1. Physics-aware loss function

We have already introduced the Navier–Stokes equations in eq. (1) of section 2. If we expand it in terms of the individual components, we obtain

$$u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + \frac{\partial p}{\partial x} - \nu\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) = 0 \tag{18}$$

$$u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + \frac{\partial p}{\partial y} - \nu\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) = 0 \tag{19}$$

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0, \tag{20}$$

where u and v are the x- and y-component of the flow field. Here, eq. (20) is the continuity equation and eq. (18) and eq. (19) are the components of the momentum equation.

We discretize eqs. (18) to (20) using the central-difference stencils

$$\left(D^x u_h\right)_{i+j} := \frac{u_h^{i+1,j} - u_h^{i-1,j}}{2h}, \quad \left(D^{xx} u_h\right)_{i+j} := \frac{u_h^{i+1,j} - 2u_h^{i,j} + u_h^{i-1,j}}{h^2},$$

$$\left(D^y u_h\right)_{i+j} := \frac{u_h^{i,j+1} - u_h^{i,j-1}}{2h}, \quad \left(D^{yy} u_h\right)_{i+j} := \frac{u_h^{i,j+1} - 2u_h^{i,j} + u_h^{i,j-1}}{h^2}.$$

The resulting discretized Navier–Stokes equations read

$$u_h \odot D^x u_h + v_h \odot D^y u_h + D^x p_h - \nu\left(D^{xx} u_h + D^{yy} u_h\right) = 0 \tag{21}$$

$$u_h \odot D^x v_h + v_h \odot D^y v_h + D^y p_h - \nu\left(D^{xx} v_h + D^{yy} v_h\right) = 0 \tag{22}$$

$$D^x u_h + D^y v_h = 0, \tag{23}$$

where $\odot$ is the Hadamard product, that is, element-wise product. Following the discussion in section 4.1, this can equivalently be written using the cross-correlation operation $*$ as follows:

$$U_h \odot U_h * D^x + V_h \odot U_h * D^y + P_h * D^x - \nu\left(U_h * D^{xx} + U_h * D^{yy}\right) = 0 \tag{24}$$

$$U_h \odot V_h * D^x + V_h \odot V_h * D^y + P_h * D^y - \nu\left(V_h * D^{xx} + V_h * D^{yy}\right) = 0 \tag{25}$$

$$U_h * D^x + V_h * D^y = 0, \tag{26}$$

where, for simplicity, we overload the notation for the discrete differential operators D^x, D^{xx}, D^y, and D^{yy} with the corresponding stencil matrices, and U_h, V_h, and P_h are the matrix representations of the solution fields corresponding to u_h, v_h, and p_h, respectively. Furthermore, for simplicity, we omit the treatment of the boundary conditions for now and refer to section 5.2 for a detailed discussion.

Equations (24) to (26) correspond to the discrete nonlinear system of equations

$$N(U_h, V_h) + G(P_h) = 0 \quad \text{in } \Omega, \tag{27}$$

$$D(U_h, V_h) = 0 \quad \text{in } \Omega, \tag{28}$$

where each of the operators N, G, and D can be implemented using the cross-correlation and the Hadamard product as the building blocks; cf. eqs. (24) to (26). The operator N is nonlinear, whereas G and D are both linear operators.

Now, we apply the physics-aware surrogate modeling approach described in section 4.3 to predict the solution of eqs. (27) and (28) for varying geometries. The surrogate model takes the form of

$$f_{NN}^{\Psi} : \mathbb{R}^{w \times h} \to \mathbb{R}^{3 \times w \times h}$$
$$I_g \to \begin{pmatrix} U_{NN}^{\Psi}(I_g) \\ V_{NN}^{\Psi}(I_g) \\ P_{NN}^{\Psi}(I_g) \end{pmatrix},$$

where I_g is, again, the pixel image representation of a geometry g, and $U_{NN}^{\Psi}(I_g)$, $V_{NN}^{\Psi}(I_g)$, and $P_{NN}^{\Psi}(I_g)$ correspond to the CNN predictions for the matrices resp. images U_h, V_h, and P_h, respectively.

Combining our physics-aware approach for multiple geometries as described in section 4.3 with this CNN model and the discrete residual of the Navier–Stokes equations eqs. (27) and (28), we obtain the loss function

$$\frac{1}{|T|} \sum_{g \in T} \left(\omega_M \| N(U_{NN}^{\Psi}(I_g), V_{NN}^{\Psi}(I_g)) + G(P_{NN}^{\Psi}(I_g)) \|_2^2 + \omega_D \| D(U_{NN}^{\Psi}(I_g), V_{NN}^{\Psi}(I_g)) \|_2^2 \right). \tag{29}$$

Here, ω_M and ω_D are weights for the two loss terms, and T is, again, the set of all training geometries g.

To complete our discussion of the application of the physics–aware approach to the Navier–Stokes equations, we discuss the specific treatment of the boundary conditions in the next section.

5.2. Treatment of boundary conditions

As discussed in section 4.3, we enforce Dirichlet boundary conditions explicitly by hard-coding the values of the pixels in the output image. In particular, for our boundary value problems, as introduced in section 2, we consider the following boundary conditions: For inlet boundary condition, we set

$$U_{1,j} = 3, V_{1,j} = 0, \quad \forall j = 1, \dots, h-1, \tag{30}$$

where h (height) is the number of pixels in y direction. Moreover, the no-slip boundary conditions

$$U_{i,1} = U_{i,h} = V_{i,1} = V_{i,h} = 0, \quad \forall i = 1, \dots, w, \tag{31}$$

are enforced at the lower and upper walls as well as the zero pressure boundary condition

$$P_{w,j} = 0, \quad \forall j = 1, \dots, h-1, \tag{32}$$

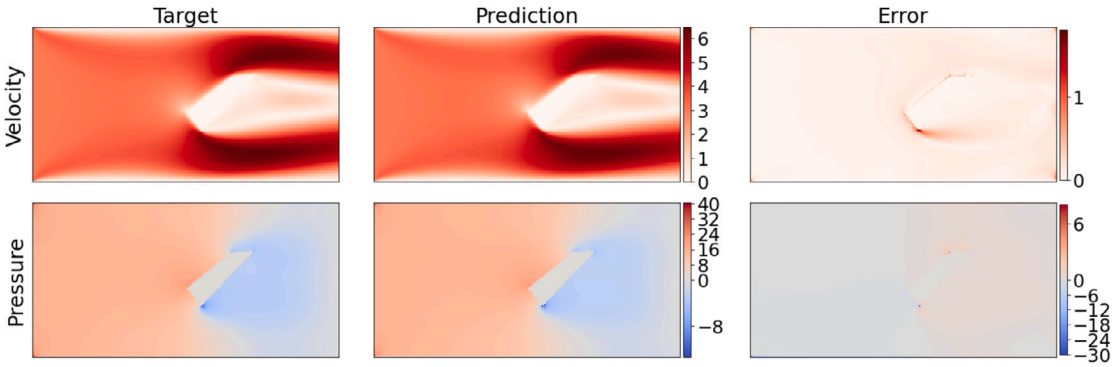
at the outlet. Here, w (width) is the number of pixels in x direction.

Neumann pressure boundary conditions can be implemented by introducing ghost-nodes outside our computational domain. However, in this work we do not consider Neumann boundary conditions and instead refer to [16]. Note, though, that the use of ghost-nodes may not be feasible in the case of irregular obstacle boundaries. Here, interpolation to a pixel image may introduce corner points on the boundary where the normal vector is not well defined.

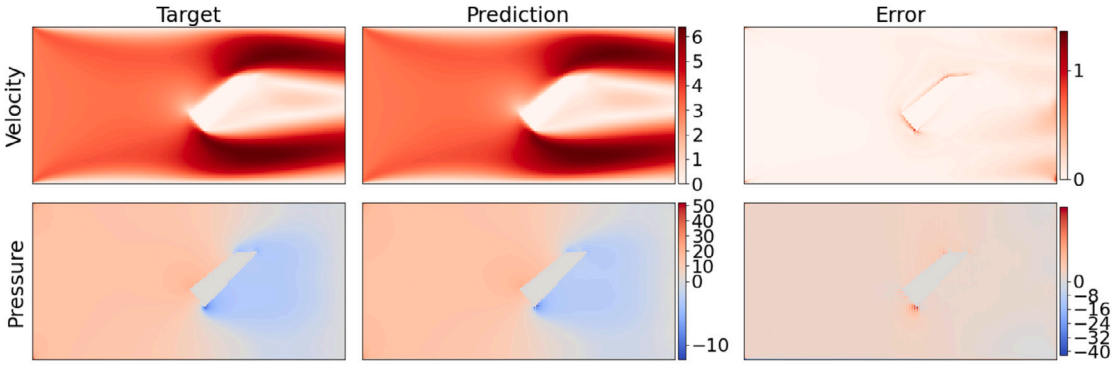
Further, to avoid the usage of pressure values in pixels where the pressure is not defined, we employ one-sided differences in pixels adjacent to corresponding boundaries. We encode the different stencils in an additional input image; cf. Fig. 3b. The numbering scheme used for the grid nodes is as follows: 0 represents internal nodes, 1 corresponds to inflow boundary nodes, 2 represents no-slip boundary nodes, and 3 denotes outflow boundary nodes. Nodes with numbers 4 and above require one-sided approximations for the pressure gradient. It is important to note that some pixels correspond to nodes outside the original domain Ω due to obstacles. These nodes are marked as 0 in the geometry image and 2 in the boundary image. Velocity and residual values are set to 0 in these nodes. This is necessary as the governing equations are not defined in those nodes.

5.3. Example on a single geometry

In order to verify that the physics-aware loss enables us to learn a solution of the Navier–Stokes equations, we consider a single fixed geometry. This is not useful in practice since we could more efficiently directly discretize the Navier–Stokes equations using finite differences and solve the discrete system using suitable numerical solvers; see [24] for a comparison for a simple Laplace problem.



(a) Velocity and pressure for the physics-aware approach (Prediction) compared to the OpenFOAM simulation on the *locally refined* mesh (Target).



(b) Velocity and pressure for the physics-aware approach (Prediction) compared to the OpenFOAM simulation on the *Cartesian rasterized* mesh (Target).

Fig. 4. Results for a single geometry as shown in Fig. 1 and Fig. 7. CNN model with Swish activation function and $5 \cdot 10^{-5}$ as the learning rate for the Adam optimizer.

We compare the model prediction against FVM simulations with OpenFOAM on two different meshes: a locally refined mesh (Fig. 6a) and using the same pixel grid as the CNN model (removing the pixels inside the obstacle). The results are plotted in Fig. 4. Compared with the simulation on a locally refined mesh, we obtain low relative L_2 errors (defined in eq. (33)) of 2.6% for u and 2.8% for p. As can be seen in Fig. 4a, the velocity error is particularly high near the obstacle, presumably due to non-resolved boundary layers. The comparison against the simulation on the rasterized mesh in Fig. 4b shows a visual improvement of these errors, and the relative L_2 error for the velocity reduces to 2.2%. This suggests that part of the error is due to insufficient mesh resolution. An error of 0 cannot be obtained since the CNN model is based on a finite difference discretization whereas the reference data is computed using FVM simulations for both types of meshes.

In total, we conclude from the results for a single geometry that a CNN model with physics-aware loss may learn a good approximation of the solution of the Navier–Stokes equations. Later, in section 9, we will investigate the performance of the CNN-based surrogate model trained on a data set consisting of multiple geometries, introducing another level of complexity to the model.

6. Architecture of the convolutional neural network

In this section, we describe the network architecture of our CNN-based surrogate models, utilizing the same architecture type for both the data-based and physics-aware models described in sections 3 and 4, respectively. We employ a fully convolutional neural network that only performs convolutions, up- or downsampling. For a comprehensive understanding of CNNs, we refer to [20, Chapt. 9] and the references therein.

Our CNN architecture draws inspiration from the U-Net architecture [58]. It consists of an encoder, transforming input image(s) into a lower dimensional representation in the bottleneck, and a decoder, transforming the bottleneck output into velocity and pressure output images. The U-Net architecture's symmetric encoder and decoder paths are connected via skip connections. The performance of data-based surrogate models with U-Net architecture is generally superior compared to bottleneck CNNs without skip connections, as discussed in [10].

The encoder of our network consists of blocks comprising a 3×3 convolutional layer, followed by a 2×2 convolutional layer with a stride of 2, both followed by an activation. The first convolution extracts input features, while the second convolution reduces spatial dimensions. We avoid max pooling for downsizing to prevent high-frequency artifacts as discussed in [28]. The decoder mirrors the encoder and includes upsampling layers with nearest-neighbor interpolation followed by a convolutional layer with an activation. This upsampling technique helps avoiding checkerboard artifacts mentioned in [48] that can occur with deconvolutional and downward convolutional layers. Matching encoder and decoder blocks are connected by skip connections following the U-Net architecture. The encoder block output is concatenated with the upsampling layer output, doubling the number of filters. Refer to Fig. 5 for an

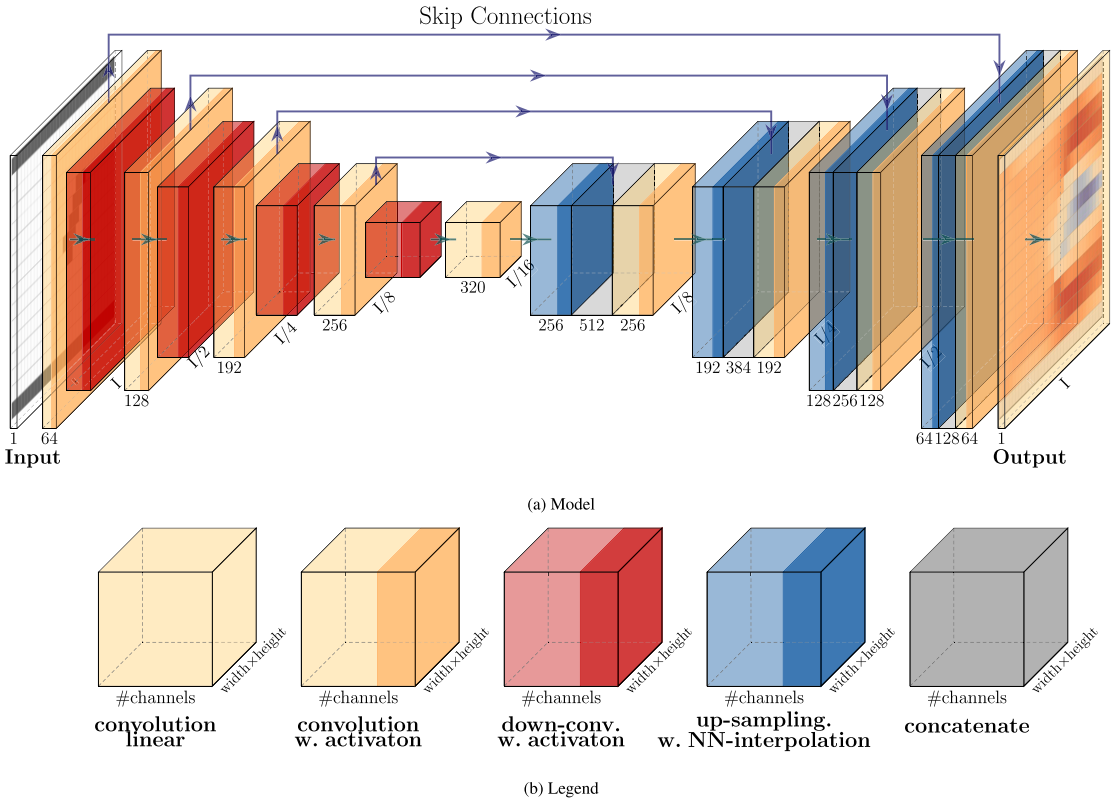


Fig. 5. Exemplary model architecture with four levels and one decoder path. Note, that our models use separate decoder paths for each scalar output field, U, V, and P.

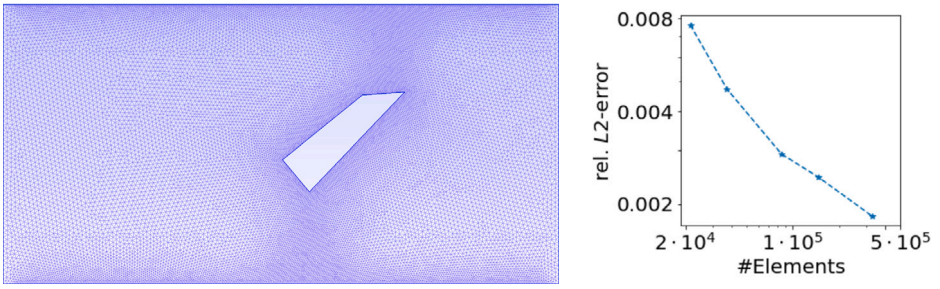


Fig. 6. (a) Example of a locally refined mesh used for simulations. (b) Mesh Convergence plot with relative errors compared against on a reference simulation for a fine mesh with ≈ 1 300 000 elements. The depicted mesh corresponds to the second node in the convergence plot.

illustration of this architecture with one decoder path and four levels. In our experiments in section 9, unless stated otherwise, we employ an 8-level model. Additionally, it should be noted that our models use separate decoder paths for each scalar output field, U, V, and P, for a total of three separate decoder paths. The total number of trainable parameters for this model are just under 40 million.

We refer to section 8 for additional comments on the choice of hyper parameters, including the model architecture.

7. Generation of (training) data

Training our surrogate models for the challenging task of predicting solutions across a wide range of geometries necessitates a large dataset. Note that previous publications have employed datasets of up to 100 000 geometries; see [27,9]. For the fully data-driven model, reference data is required along with input images of the geometries. In contrast, the physics-aware model solely relies on the input images, as the physics loss replaces the need for reference data. Thus, generating training data involves creating random obstacle

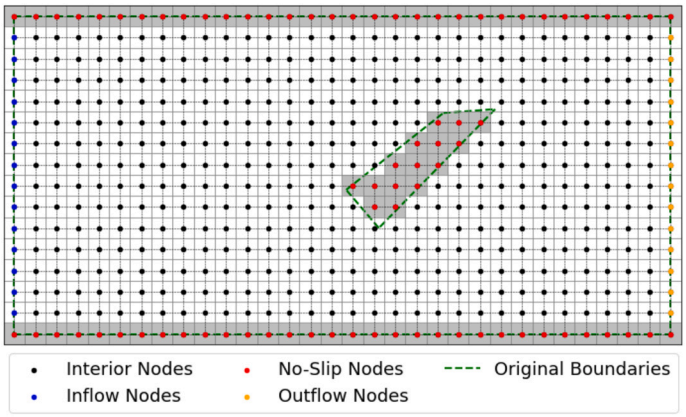


Fig. 7. Exemplary representations of the interpolation process, here with a lower resolution of 32×16. The green dotted line shows the border of the computational domain Ω_P. Note that the original boundary is plotted behind the outer nodes.

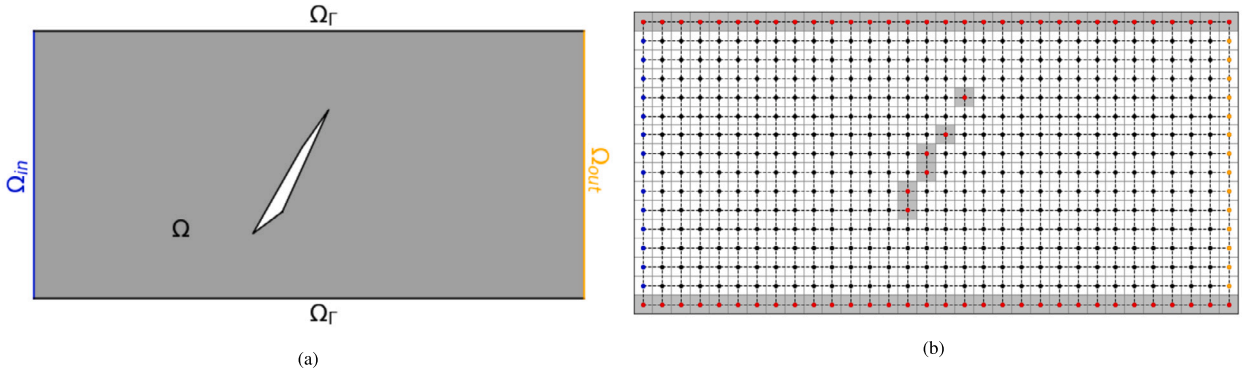


Fig. 8. Exemplary representations of gross distortion caused by too acute angles. (a) The original geometry and (b) the reduced pixel image, here with a lower resolution of 32×16. The green dotted line shows the borders of the computational domain Ω_P. Note that there the original boundary is plotted behind the outer nodes.

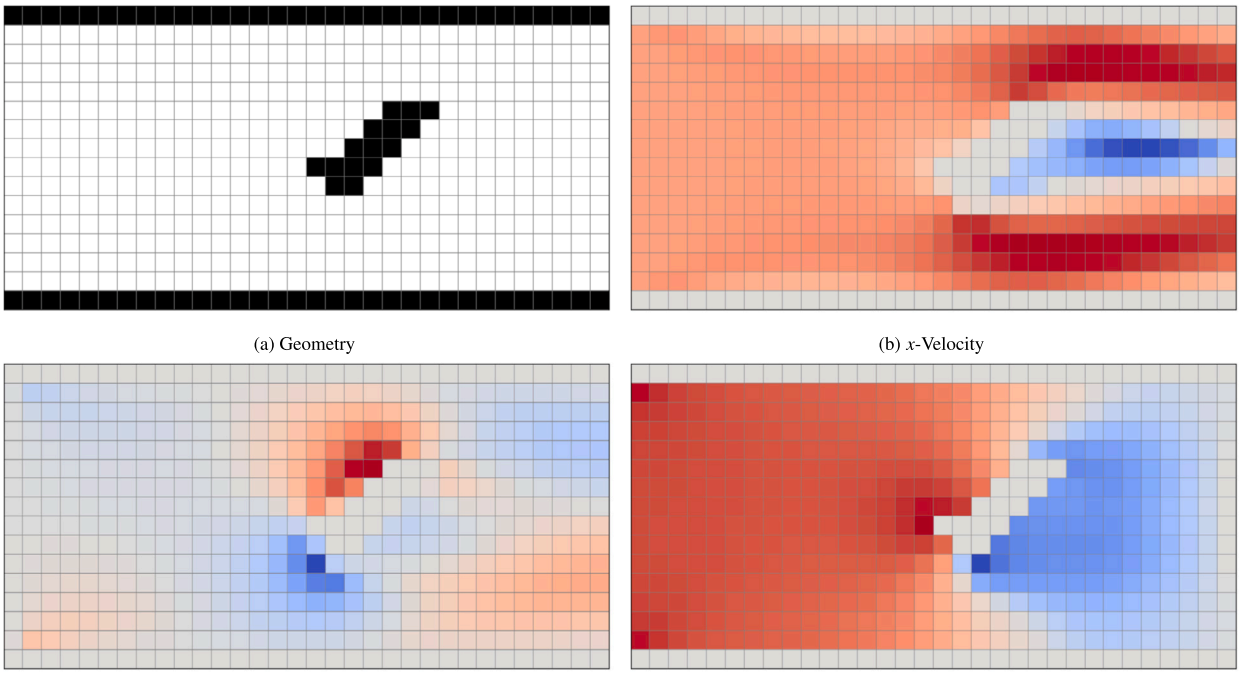
geometries and their pixel image representations. To validate our model, we generate simulation meshes and conduct corresponding CFD simulations as reference data.

7.1. Input data

The input data is needed for the training of the data-driven and physics-aware model as well as of hybrid variants.

Geometry generation. In line with section 2, our focus lies on two-dimensional rectangular channel geometries featuring star-shaped obstacles that do not touch the boundary (inlet, outlet, upper and lower walls). The computational domain is given by $\Omega = [0, 6] \times [0, 3]$ from which we have removed a star-shaped obstacle P defined by its corners. The obstacle's corners are randomly positioned around a central point, inspired by the approach outlined in [9]. We only consider obstacles with a maximum width of 50% of the channel's height and a minimum distance of 0.75 from any boundary. To prevent significant distortion of the geometry in the image representation, we impose a minimum angle of 10° at each vertex of the obstacle. This prevents excessively acute angles, as shown in Fig. 8, which could result in a disconnected obstacle representation.

Geometry image representation. As discussed before, CNNs rely on input data with a tensor-product structure. Therefore, we interpolate the geometry to a binary 256×128 pixel image. In Fig. 7, an exemplary geometry is shown in its pixel image representation, where white pixels (encoded as 1) correspond to fluid cells, while gray pixels (encoded as 0) represent walls and the obstacle. The pixel value is determined based on whether the center of the pixel is within the fluid domain, or not. In previous studies, signed distance function (SDF) representations were also used to describe the geometry, generally leading to slightly better results; cf. [9,10,27]. For simplicity, we restrict ourselves to binary input images, as they are close to practically relevant cases; for instance, binary images can be directly generated from imaging techniques, such as magnetic resonance imaging (MRI).



(a) Geometry (b) x-Velocity

(c) y-Velocity (d) Pressure

Fig. 9. Exemplary representations of the pixel images, here with a lower resolution of 32×16.

7.2. Reference output data

Reference output data is required in order to compute the data loss eq. (2), which is required for the data-driven and hybrid modeling approaches. In case of the fully physics-aware approach, the reference data is only used for validation.

Mesh generation. For each obstacle chosen as discussed in section 7.1, we generate a computational mesh using Gmsh [18]; we refer to its documentation[1] for details on the mesh generation. In particular, we use the Frontal Delaunay algorithm [56] to generate an unstructured triangular mesh for each case. Then, we refine the mesh near all walls, that is, near the upper and lower wall as well as near the obstacle, to resolve the boundary layers of the flow. In order to find a suitable level of refinement for the simulations, we performed a mesh convergence study on a representative geometry; cf. Fig. 6b. Note that we created a new mesh for each level of refinement, rather than refining an existing mesh. Fig. 6a shows an mesh with refinement near the boundaries with $\approx 40\,000$ elements, whereas we used meshes with $\approx 160\,000 - 200\,000$ to generate our simulations; we do not display such a mesh for the sake of clarity.

CFD simulations. The CFD simulations for the generation of the reference data have been performed using OpenFOAM v8 [67], a software based on the finite volume method (FVM). We utilized the simpleFoam solver, which solves the stationary incompressible Navier–Stokes equations using the semi-implicit method for pressure linked equations (SIMPLE) algorithm [49,12]. We refer to the OpenFOAM documentation[2] for more details on the simpleFoam solver. The configuration is based on the pitzDaily example, adapted to laminar flow and with stricter convergence criteria.

Interpolation of the simulation data. To compare the surrogate model's output with the reference simulation data, we interpolate the simulation data onto the same pixel grid. This involves evaluating the FVM solution at the centroids of the pixels, resulting in pixel images U_h, V_h, and P_h representing the velocity in the x and y directions and the pressure, respectively.

It is important to note that values outside the computational domain Ω_p are explicitly set to 0 in both the simulation and the model prediction. This is clearly visible in our plots, such as in Fig. 9. To achieve this, we mask the output images based on the geometry representation in the input image.

[1] https://gmsh.info/doc/texinfo/gmsh.html.
[2] https://doc.cfd.direct/openfoam/user-guide-v8/.

8. Some comments on hyperparameter choices

Our surrogate models depend on numerous hyperparameters, and their specific choices can significantly affect performance. These hyperparameters encompass model architecture, such as the number of channels per convolutional layer, the depth of the U-net architecture, and the activation function. They also include optimizer parameters like the learning rate, learning rate schedule, and batch size. Additionally, there are hyperparameters related to the loss function, such as the weights assigned to individual loss terms and discretization parameters for the physics-aware loss, including the employed FD stencils. Furthermore, there are hyperparameters in a broader sense, for instance, the resolution of input and output images.

Considering the large number of hyperparameters, an exhaustive investigation of their impact is impractical. Therefore, instead of conducting a comprehensive grid-search, we have fixed some hyperparameters while varying individual ones. To maintain brevity, we provide qualitative discussion of the outcomes rather than presenting extensive results for this process.

Network architecture. Hyperparameters related to the model architecture determine the number of parameters Ψ and hence the model's capacity to approximate the solution operator; cf. section 3. Due to the high complexity of the solution operator, we expect that the model requires a large number of parameters, whereas a too large number of parameters may lead to overfitting. We individually optimize hyperparameters for the network architecture with regards to the validation errors; note that the errors are computed with respect to the reference simulation data. For the configurations described in section 7.1 and tested in section 9, we have determined that a depth of 8 levels and 64 channels in the first layer yield a good compromise: lower values result in reduced approximation properties of the model, while higher values lead to an increased computational effort and decreased generalization properties (in terms of the validation error).

Activation function and learning rate. As the activation function we either use the rectified linear unit (ReLU) [19] or the Swish function [53]. Whereas for the single geometry case discussed in section 5.3 the use of the Swish activation function was beneficial, the ReLU function generally led to better results for training a surrogate model for multiple geometries; cf. section 9. In combination with ReLU we always achieved the best results with a learning rate of $1e-4$. With swish the optimal learning rate depended on the considered geometry and ranged from $1e-5$ to $1e-4$.

Optimizer and batch size. The best results for our surrogate model were obtained by using the stochastic gradient descent optimizer with adaptive moment estimation (Adam) [33] and a batch size of 1. Other optimizers were unable to reliably find suitable minima and greater batch sizes led to greatly increased errors.

Image resolution. Our fully CNN model architecture, can be applied to any resolution with a power of 2 number of pixels in both the x and y directions. The number of pixels in each direction does not have a systematic dependence. However, if the resolution is too low, the geometry representation may be inaccurate, and the FD discretization error could be high.

Conversely, higher image resolutions may lead to increased computational effort and reduced accuracy due to limited model capacity. We have observed that models for higher resolutions require increased depth to achieve meaningful predictions. In addition, training larger and deeper models presents a more difficult optimization problem. We have found that models for pixel images with a width of 512 pixels and a height of 256 pixels and larger do not converge to suitable minima as reliably as models for smaller pixel images. Based on these considerations, we have chosen an image resolution of 256 pixels in width and 128 pixels in height, which has also been used in previous studies; cf. [27,9].

9. Computational results

In this section, we present numerical results for our surrogate modeling approach. We investigate the fully data-driven approach (section 9.1), the fully physics-aware approach (section 9.2), and the hybrid approach (section 9.3), which combines both.

We evaluate the models using on the relative L_2-errors

$$\frac{\|U_{NN} - U_h\|_2}{\|U_h\|_2} \tag{33}$$

as the performance measure. Here, U_{NN} is the prediction of our model, and U_h is the reference solution; unless otherwise stated, the reference solution corresponds to the result of an OpenFOAM simulation on a locally refined mesh evaluated at the midpoints of the pixels; cf. the discussion in section 7. We then compute the L_2-error on the pixel grid employed by the surrogate model with the functions being constant on each pixel; as a result the relative L_2-norm is equivalent to the relative l_2-norm.

The dataset we use consists of $\approx 5\,000$ geometries with randomly generated obstacles with 3, 4, 5, 6, or 12 edges, with $\approx 1\,000$ geometries for each number of edges. Fig. 10 shows four example geometries from this dataset as they are used as input for our model. All computations were performed on NVIDIA V100-GPUs with CUDA 10.1 using Python 3.6 and tensorflow-gpu 2.7 [1].

9.1. Data-based approach

First, we analyze the fully data-driven approach as discussed in section 3; cf. [9]. In addition to the velocity field, which was the focus of [9], we also learn the pressure field here; we also did some changes in the network architecture to improve upon the results in [9]. Later, in sections 9.2 and 9.3, we will use the results for the fully data-driven model as a baseline for comparison.

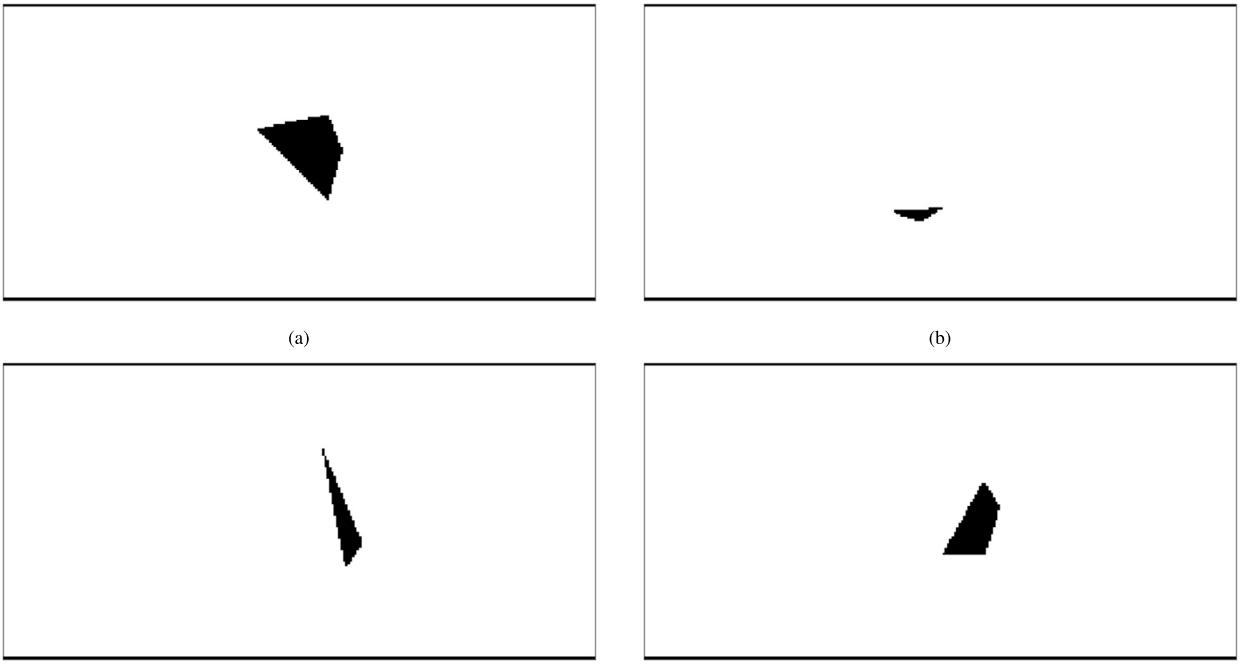


Fig. 10. Four exemplary images of geometries as they are used as input for our model.

Table 1
Performance of the data-based approach on multiple geometries from the channel dataset compared to OpenFOAM simulations on *locally refined* meshes. The divergence and momentum residuals are averaged over all configurations and pixels.

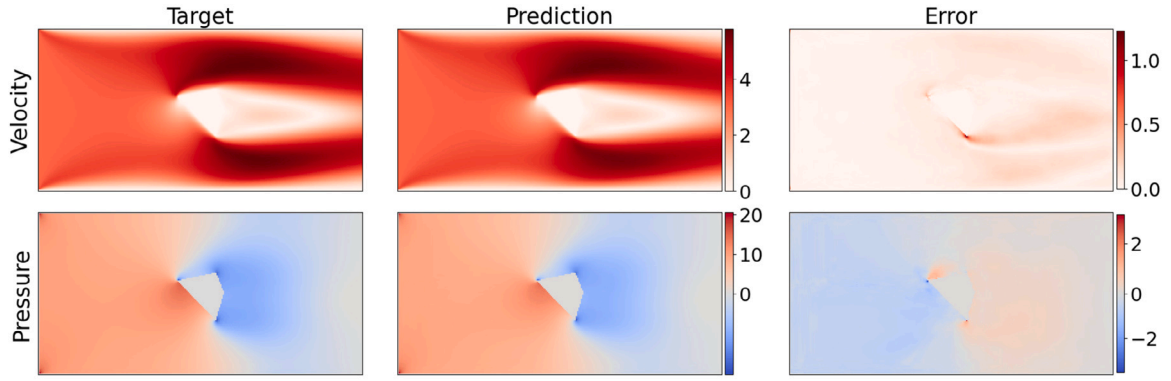
training data	error	$\frac{\|u_{NN}-u\|_2}{\|u\|_2}$	$\frac{\|p_{NN}-p\|_2}{\|p\|_2}$	divergence residual	momentum residual	# epochs
10%	training	2.07%	10.98%	$1.1 \cdot 10^{-1}$	$1.4 \cdot 10^{0}$	500
	test	4.48%	15.20%	$1.6 \cdot 10^{-1}$	$1.7 \cdot 10^{0}$	
25%	training	1.93%	8.45%	$9.1 \cdot 10^{-2}$	$1.2 \cdot 10^{0}$	500
	test	3.49%	10.70%	$1.2 \cdot 10^{-1}$	$1.4 \cdot 10^{0}$	
50%	training	1.48%	8.75%	$9.0 \cdot 10^{-2}$	$1.1 \cdot 10^{0}$	500
	test	2.70%	10.09%	$1.1 \cdot 10^{-1}$	$1.2 \cdot 10^{0}$	
75%	training	1.43%	7.30%	$1.0 \cdot 10^{-1}$	$1.5 \cdot 10^{0}$	500
	test	2.52%	8.67%	$1.2 \cdot 10^{-1}$	$1.5 \cdot 10^{0}$	

In order to investigate the performance of the data-driven approach, we trained several models with increasing percentages of training data on the channel dataset (5 000 configurations); the remaining data is used as test data. The training and test performance, in terms of the relative L_2-errors and averaged residual norms, for this approach are summarized in Table 1. We observe that the data-driven model is able to learn the velocity and pressure fields very well, where the errors on the velocity are generally lower. Moreover, the training accuracy is always a bit lower compared to the test accuracy, which indicates some overfitting. We can observe that using a larger share of training data improves the performance of the model and slightly reduces the overfitting.

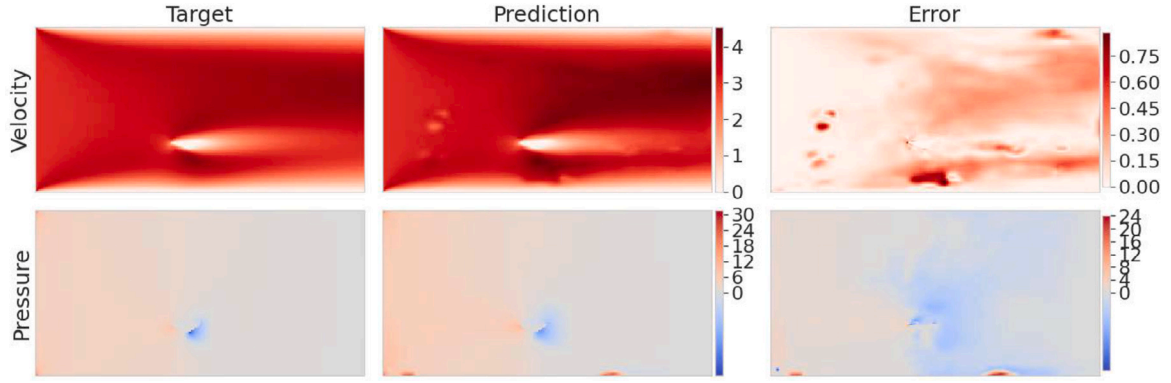
For the best model, with 75% training data, we also present plots of the velocity and pressure fields in Figs. 11a and 11b, comparing the reference and prediction data. Fig. 11a shows a typical example a quantitatively and qualitatively good prediction, whereas the prediction in Fig. 11b exhibits some clearly unphysical artifacts in the pressure and velocity fields despite a feasible average error. We conjecture that this is due to the pure data loss, which does not include any physical knowledge; as we will discuss in section 9.2.1, the physics-aware loss improves this model behavior.

9.2. Physics-aware approach

In this subsection, we will analyze the proposed physics-aware approach in detail. The results on the whole channel dataset (5 000 configurations) for varying percentages of training data are summarized in Table 2. We observe that the physics-aware surrogate model can be extended from a single geometry (section 5.3) to multiple geometries, as discussed theoretically in section 4. In particular,



(a) A good prediction for the geometry depicted in fig. 10a. The relative L_2-error in u is 2.5% and 6.4% in p.



(b) A bad prediction for the geometry depicted in fig. 10b. The relative L_2-error in u is 5.7% and 28.6% in p.

Fig. 11. Prediction of velocity and pressure for the data-based approach (Prediction) compared to the OpenFOAM simulation on the *locally refined* mesh (Target). This model was trained on 3750 geometries. Both geometries are test geometries.

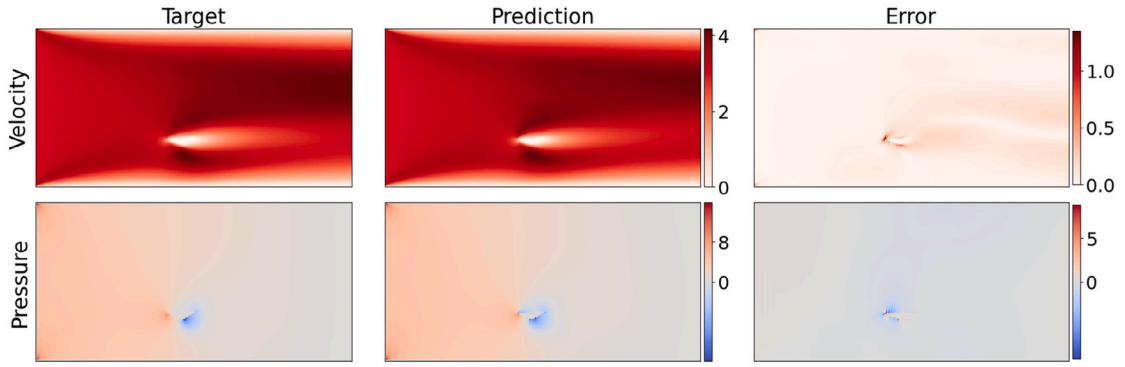
Table 2
Performance of the physics-aware approach on multiple geometries from the channel dataset compared to OpenFOAM simulations on *locally refined* meshes. The divergence and momentum residuals are averaged over all configurations and pixels.

training data	error	$\frac{\|u_{NN}-u\|_2}{\|u\|_2}$	$\frac{\|p_{NN}-p\|_2}{\|p\|_2}$	divergence residual	momentum residual	# epochs
10%	training	4.34%	9.75%	$2.8 \cdot 10^{-02}$	$7.4 \cdot 10^{-02}$	2 500
	test	5.70%	12.81%	$5.7 \cdot 10^{-02}$	$2.0 \cdot 10^{-01}$	
25%	training	4.17%	9.61%	$2.5 \cdot 10^{-02}$	$6.1 \cdot 10^{-02}$	2 500
	test	4.82%	10.73%	$4.4 \cdot 10^{-02}$	$1.3 \cdot 10^{-01}$	
50%	training	4.16%	9.47%	$2.4 \cdot 10^{-02}$	$5.7 \cdot 10^{-02}$	2 500
	test	4.37%	9.68%	$3.7 \cdot 10^{-02}$	$1.0 \cdot 10^{-01}$	
75%	training	3.82%	8.71%	$1.8 \cdot 10^{-02}$	$4.0 \cdot 10^{-02}$	2 500
	test	3.91%	8.65%	$2.8 \cdot 10^{-02}$	$8.0 \cdot 10^{-02}$	

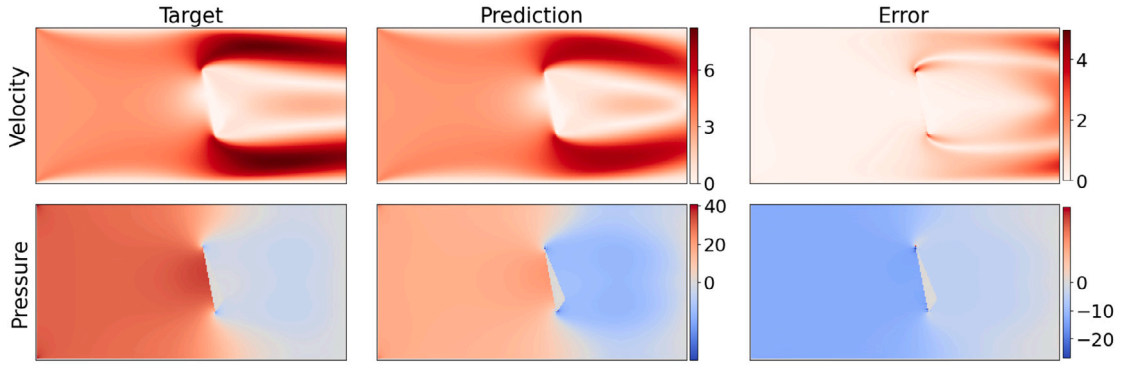
we obtain predictions with low errors on the velocity and pressure fields, and the performance improves slightly when increasing the share of training data. Interestingly, the overfitting effect is rather small, even when using only 10% of the data for training the model.

In Fig. 12, we showcase different predictions from this model. We observe smooth solutions in all cases, without any unphysical artifacts visible. However, when inspecting the error plots, we observe that the error is highest in the vicinity of the obstacle, indicating that the uniform pixel grid cannot fully resolve the boundary layers of the flow. Hence, we observe some error compared with the reference data, which has been computed on a locally refined mesh; cf. Fig. 6a.

In the following, we will discuss the results in more detail: in section 9.2.1, we compare the results with the results for the data-based approach in section 9.1; in section 9.2.2, we discuss the correlation of high maximum velocities in the flow field and high prediction errors; and in section 9.2.3, we discuss the influence of the pixel grid on the prediction performance.



(a) A good prediction for the geometry depicted in fig. 10b. The relative L_2-error in u is 2.3% and 6.0% in p.



(b) A bad prediction for the geometry depicted in fig. 10c. The relative L_2-error in u is 21.0% and 45.8% in p.

Fig. 12. Velocity and pressure for the physics-aware approach (Prediction) compared to the OpenFOAM simulation on *locally refined* meshes (Target). The model was trained on 3 750 geometries. All shown geometries are test geometries.

9.2.1. Comparison to the data-based approach

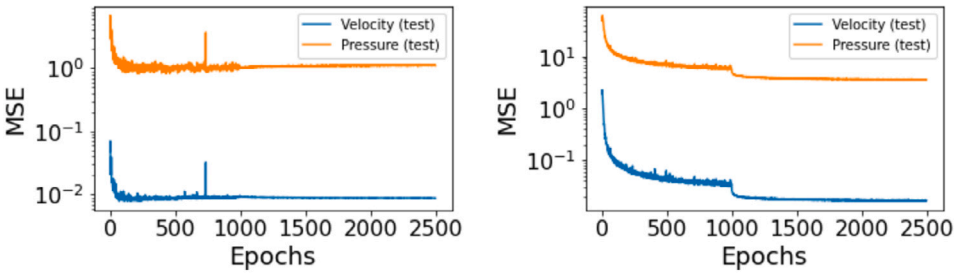
Comparing the results in Tables 1 and 2 for the data-based and the physics-aware approach, respectively, we observe that the data-based model generally has a better performance based on the relative errors compared with the reference data. On the other hand, the residuals of the divergence and momentum equations on the pixel grid are lower for the physics-aware model. This can be easily explained by the fact that the data-based model is trained against the reference data, whereas the physics-aware model is trained to minimize the residuals.

At first sight, this may seem contradictory since we would expect that, for the same boundary value problem, a lower residual might also results in a lower error. However, as discussed in section 7.2, the reference data is generated based on solving the Navier–Stokes equations with FVM on a locally refined mesh, whereas we evaluate the residuals for the physics-aware model on a uniform pixel grid. This means that the physics-aware model can never reach relative velocity and pressure errors of zero, with our current setting. Likewise, the data-based model will not minimize the residuals on the uniform pixel grid.

Interestingly, the physics-aware model is less prone to overfitting than the data-based model. In particular, despite slightly worse prediction errors, the gap between the training and test errors is clearly lower for the physics-aware model, indicating better generalization capabilities.

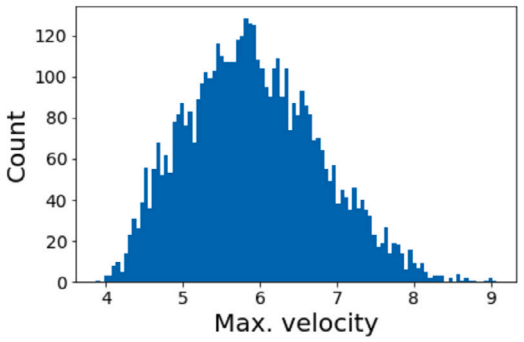
Tables 1 and 2 also indicate that we performed a significantly larger number of epochs to train the physics-aware than the data-based model on the same data set; in particular, we ran the training for 2 500 instead of 500 epochs. In order to illustrate this, we present plots of the evolution of the mean squared errors for the velocity and pressure over the training process for the physics-aware model with 75% training data and a new data-based model that we also trained for 2 500 epochs on 75% training data in Fig. 13. Note that this data-based model is not the same model for which we have presented results in this section so far. The test errors of the data-based model reach their minimum very quickly, see Fig. 13a. It can be clearly seen that with a training of more than 500 epochs, the test errors do not decrease further. On the contrary, they even increase slightly. Thus, longer training of the data-based model would only lead to stronger overfitting. In contrast, the test errors for the physics-aware model decrease more slowly and reach their lowest value only in the further course of the training, see Fig. 13b.

Finally, we briefly discuss those cases where the data-based or the physics-aware model performs badly. As mentioned in section 9.1, the data-based approach occasionally makes predictions with unphysical artifacts in the flow and pressure fields. In particular, we show present in Fig. 11b one example from the test data set where this is apparent. The corresponding prediction of the physics-aware model for the same sample is shown in Fig. 12a. We do not observe the same artifacts. In alignment with the lower overfitting of the physics-aware model, we conclude that the physics-aware model indeed learns better the actual flow behavior based on the

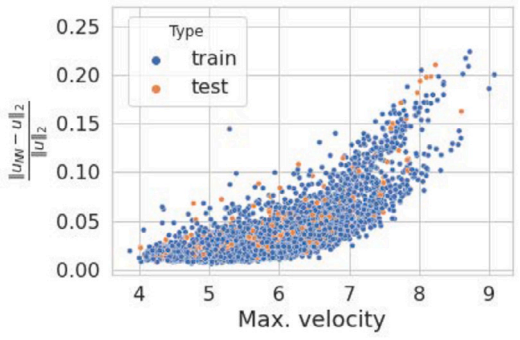


(a) Data-based approach. (b) Physics-aware approach

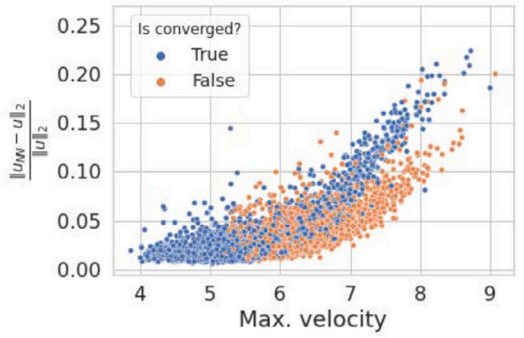
Fig. 13. Test loss curves for the velocity and pressure over the trained epochs.



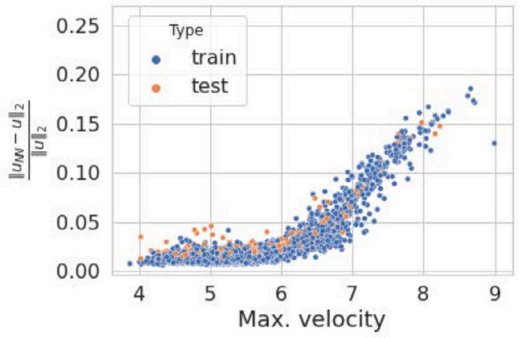
(a) Histogram of the maximum velocity for the channel dataset



(b) Relative L_2-error for u for the physics-aware approach compared to OpenFOAM simulations on *locally refined* meshes. The color indicates training and



(c) Relative L_2-error for u for the physics-aware approach compared to OpenFOAM simulations on *locally refined* meshes. The color indicates convergence of OpenFOAM simulations on *rasterized* meshes



(d) Relative L_2-error for u for the physics-aware approach compared to OpenFOAM simulations on *rasterized* meshes. The color indicates training and test data

Fig. 14. Results investigating the correlation of the error with the maximum velocity appearing in the flow field.

residuals of the Navier–Stokes equations. On the other hand, we often see larger errors in the vicinity of the obstacle for the physics-aware approach. This might be attributed due to the uniform pixel grid, which is not specifically refined for resolving the boundary layers in the physics-aware approach. In particular, it seems that the error originates at the obstacle and propagates downstream.

9.2.2. Correlation of errors and velocities

In further analyzing the prediction errors, we observed a systematic correlation between the relative error in the velocity and the maximum velocity appearing in the flow field. In particular, depending on the size and position of the obstacle, the maximum velocity can vary significantly; see, e.g., the examples in Figs. 11 and 12. We observe that geometries with a maximum velocity above 6 exhibit higher average errors compared to those below 6: 5.8% for u and 12.7% for p versus 2.5% for u and 5.7% for p.

In Fig. 14a, we observe that the maximum velocity ranges roughly from 4 to 9, following almost a normal distribution; hence, the lower and higher maximum velocities do not appear as often as maximum velocities of 6. Despite fewer cases with lower maximum velocities, we observe that the prediction error generally increases with an increasing maximum velocity; cf. Fig. 14b. Moreover, there seems to be no relation to whether the configuration is in the training or test set.

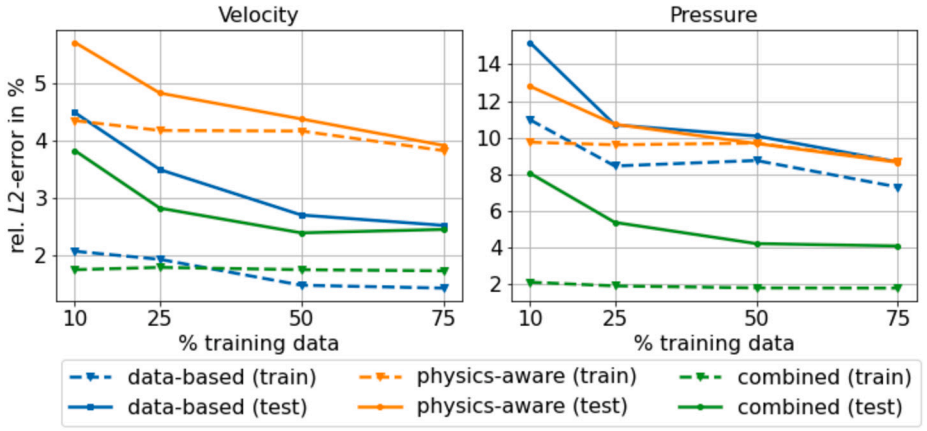


Fig. 15. Performance of the hybrid approach for abundant simulation results.

Besides arguing based on the distribution of maximum velocities in the data set, it is not surprising that higher maximum velocities lead to higher errors since this might correspond to higher Reynolds numbers and more complex flow patterns. Moreover, our physics-aware loss, as defined in eq. (29), incorporates second-order central stencils for all terms, including the convective terms. However, central stencil approximations for convective terms can be problematic when the cell Reynolds number exceeds 2; see, for instance, [46, Sec. 2.3]. For our uniform pixel grid, this occurs when $|u| > 4.25\frac{m}{s}$ in our case. Therefore, it may be necessary to consider alternative approximations for the convective terms. However, this is beyond the scope of this article.

9.2.3. *Influence of the pixel grid*

As mentioned before, there seems to be an effect from an insufficient resolution of the boundary layers around the obstacle. In order to investigate potential effects of the resolution of the pixel grid, we rerun all configurations in our data set on the pixel grid; due to their structure, we also denoted these as *rasterized meshes* in section 5.3.

First of all, we observe that a significant number of OpenFOAM simulations on the rasterized meshes did not converge; cf. Fig. 14c. Furthermore, we also see a correlation maximum velocity and convergence in this case. For geometries with obstacles narrower than 1 m and flow fields with maximum velocities below $6\ \frac{m}{s}$ almost all simulations converged. Conversely, for larger obstacles and faster flow fields only a part converged. This is in alignment with our observation on higher errors for higher maximum velocity cases.

Finally, we evaluate the physics-aware model only on those cases where the simulations on the rasterized meshes successfully converged. Fig. 14d displays the errors of the predictions of the physics-aware model against the rasterized simulations. Comparing Figs. 14b and 14d, we can observe a much better match when using the rasterized simulations as the reference. For geometries with maximum velocities below $6\ \frac{\mathrm{m}}{\mathrm{s}}$, the average L_2-error in u decreases from 2.2% to 1.5%. Similarly, for geometries with maximum velocities above $6\ \frac{\mathrm{m}}{\mathrm{s}}$, the average L_2 error in u decreases from 6.7% to 5.4%. This shows that the pixel grid has an influence on the prediction performance. Further investigations of this aspect are out of the scope of this paper but will be subject of future work.

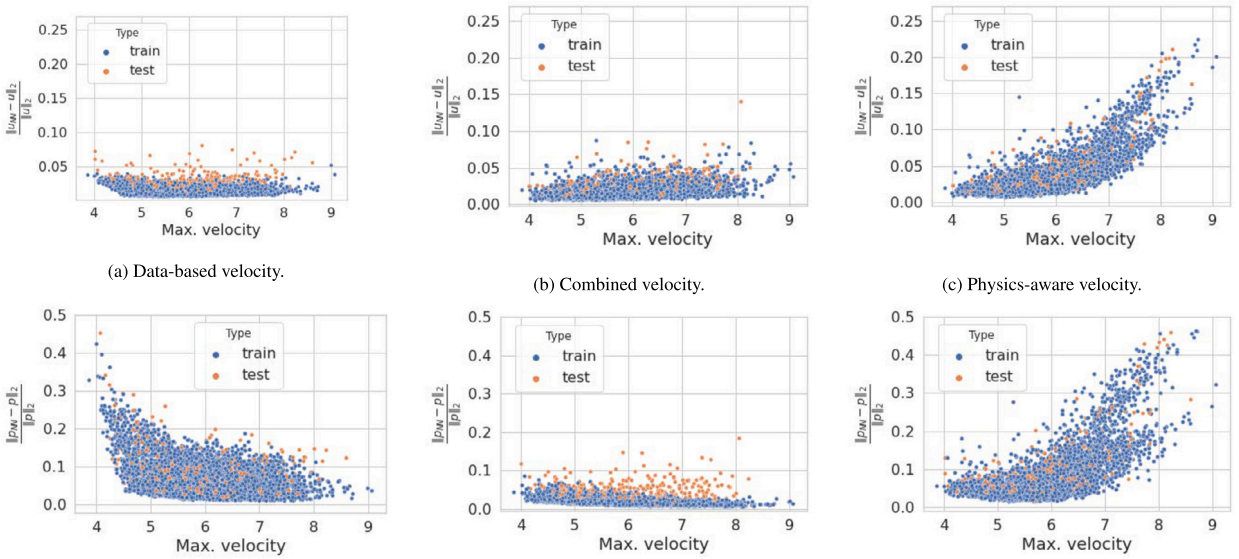
9.3. *Hybrid approach*

As discussed in section 9.2.1, both approaches have their advantages due to the different loss functions considered. The main advantages of the physics-aware approach are its generalization properties as well as the fact that no reference data is required. The main disadvantage in the CNN approach is that a uniform grid is used, which, in our setting, is not fine enough to fully resolve boundary effects and high velocities. The data-based approach, on the other hand, is able to better capture these. This is presumably because the reference data in the data loss encodes effects which cannot be fully resolved by the pixel grid. However, the data-based model is more prone to overfitting and unphysical flow artifacts.

In order to combine some of the strength of both approaches, we propose a hybrid approach, which employs a weighted sum of the data-based loss function eq. (2) and the physics-aware loss function eq. (29). This could, for example, be relevant:

- if a sufficient number of data samples is available and the generalization properties or physical consistency of the model should be enhanced or
- if only an insufficient number of data samples to cover the range of geometries is available; this could specifically be the case if measurement data is used or the simulations are prohibitively expensive. In this case, the missing data can be replaced by using the physics-aware loss.

Fig. 15 compares the overall performance of the data-based, physics-aware, and hybrid approaches. Here, the hybrid approach uses both the data-based loss and the physics-aware loss, both with an equal weight of 1. It can be observed that, for all ratios of training data, the hybrid model outperforms the data-based and physics-aware models in terms of the relative errors in the velocity and pressure; in particular, the prediction performance on the pressure improves significantly, by roughly 50%, compared with the



(a) Data-based velocity.

(b) Combined velocity.

(c) Physics-aware velocity.

(d) Data-based pressure.

(e) Combined pressure.

(f) Physics-aware pressure.

Fig. 16. Comparison of the relative L_2-error distribution for u and p with regards to the maximum occurring velocity for the data-based ((a) and (d)), combined ((a) and (e)) and physics-aware ((c) and (f)) approaches compared to OpenFOAM simulations on *locally refined* meshes. All models were trained on 3 750 geometries.

other approaches. However, unfortunately, the gap between training and test performance overfitting is on a similar level as for the data-based model.

The performance of the data-based, the physics-aware, and the hybrid approaches with respect to the maximum velocity is shown in Fig. 16. Whereas the data-based and the physics-aware models show a correlation between the prediction error and the maximum velocity, the hybrid model seems to be rather robust; interestingly, for the data-based approach, we observe a slight deterioration of the performance in the pressure prediction for lower maximum velocities.

The results indicate that, if high-fidelity reference data is available, a combination of the data-based and physics-aware loss functions yields the best results.

9.4. Weighting of loss terms

There are some elements whose modification may improve the prediction quality of our approach. This includes, for example, varying the weight of the loss terms, see eq. (29).

The initial prediction of the convolutional neural network (CNN) does not fulfill the divergence-free equation due to the random initialization of its weights. During the training process, the minimization of the sum of squared residuals eq. (29) is pursued, where equal weights ($\omega_{\mathrm{M}} = \omega_{\mathrm{D}} = 1$) may cause the learned prediction to satisfy the momentum equation more than the mass equation. While a valid solution should satisfy both the mass equation and the momentum equation, the mass equation can be seen as primarily serving as a constraint, limiting the space of valid solutions. Furthermore, in our approach, we employ a variant of the Navier–Stokes equations, specifically the momentum equation, where the assumption $\nabla \cdot \vec{u} = 0$ is explicitly employed to simplify the derivation, as discussed in [23]. Therefore, although our primary interest lies in solving the momentum equations, it may be beneficial to confine the search space to velocity fields that comply with the divergence-free condition. Due to architectural constraints preventing easy modification of our CNNs to guarantee divergence-free predictions, we endeavor to achieve a similar outcome by augmenting the weight ω_{D} of the mass residual loss term in the loss function.

Shown in Fig. 17 is the distribution of relative errors L_2 in velocity and pressure over the maximum occurring velocity for three models for whose training we varied the weight ω_{M} of the mass residual in the physics-aware loss from 1 over 10 to 100. The averaged relative L_2 errors are 4.8%, 3.7%, and 4.5% in the velocity and 10.4%, 7.9%, and 7.9% in the pressure, for values of 1, 10, and 100 of ω_{M}, respectively. Note that we used sixth-order finite difference stencils for all models here, as opposed to second-order stencils in section 9.2. In addition, the training and test data sets are not identical to those used in the previous sections. Therefore, the error values reported in this section are not necessarily directly comparable to the previous ones.

With an increase in the weight of the mass residual, we see a reduction in the error in the velocity as well as in the pressure, especially at higher velocities. This effect is very clear for $\omega_{\mathrm{M}} = 10$. In the averaged errors, this model improves by about 1% in velocity and about 2.5% in pressure compared to the model trained with equal weights, i.e. $\omega_{\mathrm{M}} = 1$. However, for the model trained with $\omega_{\mathrm{M}} = 100$ we see larger errors in the velocity overall. Here, even for low velocities, the errors in the pressure become larger. An even further increase of the weight ω_{M} led to a deterioration of the predictive capabilities, because while the predictions of the model increasingly satisfied the divergence-free constraint, the momentum residual grew.

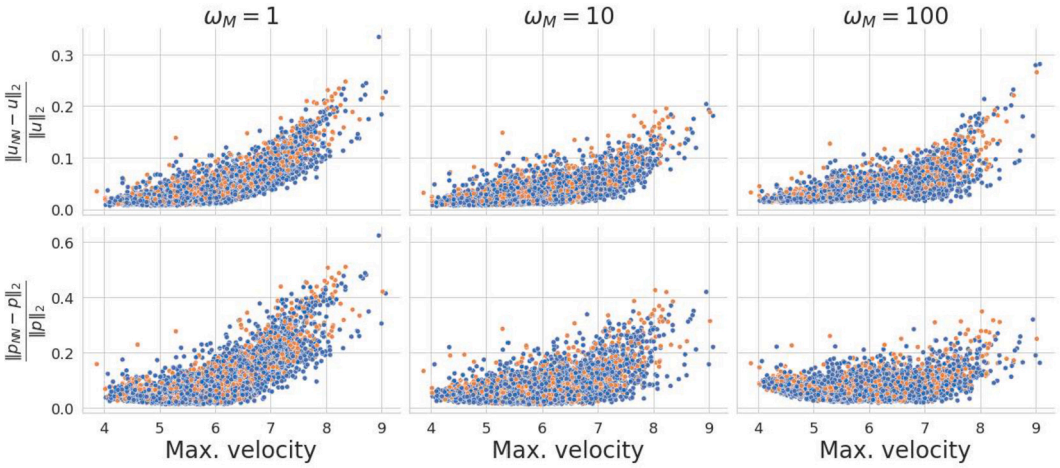


Fig. 17. Comparison of the relative L_2-error distribution for u and p with regards to the maximum occurring velocity for different approaches compared to OpenFOAM simulations on *locally refined* meshes. All models were trained on 3 500 geometries.

We have thus demonstrated that increasing the weight of the mass residual in the physics-aware loss can significantly improve the predictive capabilities of the model.

9.5. Out-of-distribution data

So far, our predictions have been limited to geometries that were present in the training or test datasets. These datasets exclusively pertain to the model problem, as illustrated in Fig. 1. Notably, these geometries encompass obstacles in the form of star-shaped polygons with up to 12 vertices. In this section, we will showcase predictions obtained using the physics-aware convolutional neural network for geometries that possess alternative types of obstacles. In doing so, we assess how well the model can generalize to previously unseen geometries, extrapolate beyond the training data, and effectively handle new and unique shapes.

The first geometry we will test the model on is a circle with radius 0.4 that we place in the middle of the channel. This type of obstacle is a highly distinct and different type of obstacle compared to the star-shaped polygons present in the training dataset. Circles have a continuous curved boundary, which contrasts with the sharp edges of the star-shaped polygons, making them significantly novel geometries for the model. We show the prediction of our model for this geometry in Fig. 18a. Smooth predictions are observed. High errors occur only near the obstacle. This prediction shows that our model can handle the curvature of a circle very well without having seen a single curved obstacle during training.

The second geometry we test our model on is a composition of an oval and a flower with 5 petals. The oval has a horizontal radius of 0.45 and a vertical radius of 0.25. The flower has a maximum radius of 0.4. The 5-petaled flower also differs from a circle because the curvature is not uniform throughout, but is interrupted by sharp bends where the petals meet. We show the prediction of our model for the second geometry in Fig. 18b. Again, we see smooth predictions with high errors occurring only near the obstacle.

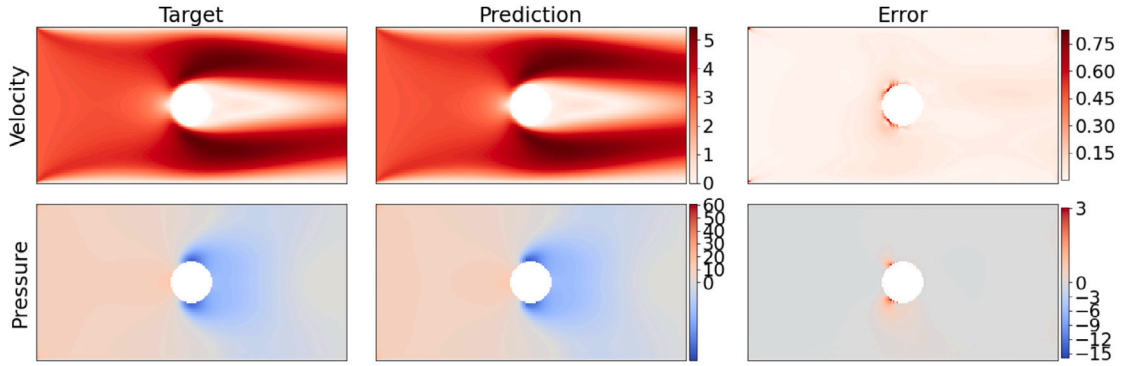
These two predictions exemplify that our model is capable of making reasonable and accurate predictions for geometries with significantly different obstacles.

9.6. Computation time

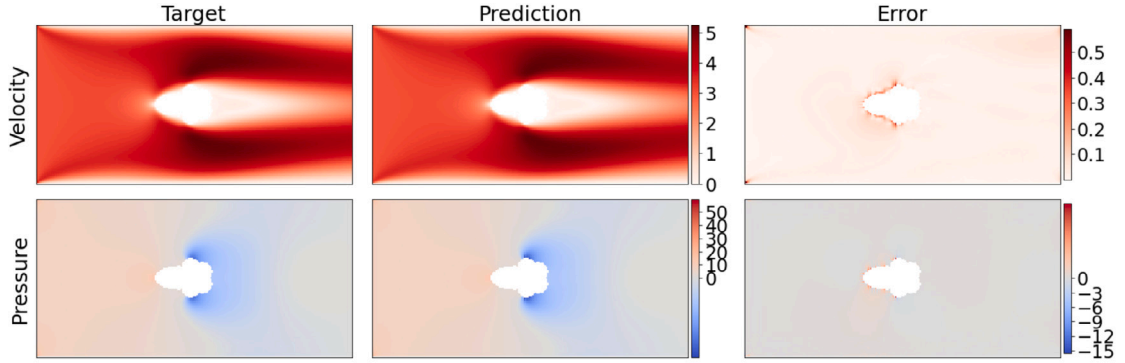
An important aspect of a surrogate model is the speed with which it can be evaluated. Therefore, in this section we want to compare the time needed to evaluate the surrogate model with the time needed for a reference CFD simulation. A CFD simulation described in section 7.2 takes between 10 and 60 minutes, depending on the geometry, and in individual difficult cases the simulation may take longer than 60 minutes. In comparison, we need only roughly 6 milliseconds (ms) to evaluate our surrogate models on a geometry. Thus, the evaluation of the surrogate model is between 100 000 and 600 000 times faster than a CFD simulation. This does not include the time required to mesh the geometry, which can be very time-consuming depending on the geometry and mesh fineness, and to set up the CFD simulation.

The time required for generating the dataset and train the surrogate model is very high. However, we exclude this time from our discussion because we attribute it to an offline phase, which only has to be carried out once. Then, once our model has been trained, we can apply it to many different configurations without retraining the model, and the savings in computational time will depend on the number of configurations to be tested.

The generation of our dataset is the most time-consuming part. While training the model discussed in section 9.2, which has been trained on about 3 750 geometries, takes roughly 5 days, generating the training data took more than 30 days of serial computations. The data generation includes, for each geometry, mesh generation, carrying out the corresponding CFD simulation, and interpolating the data onto the pixel grid. However, the computations can be sped up significantly because each geometry is independent and the computations can be easily parallelized.



(a) A circular obstacle. The relative L_2-error in u is 1.2% and 4.9% in p.



(b) A composite obstacle. The relative L_2-error in u is 0.7% and 4.8% in p.

Fig. 18. Velocity and pressure for the physics-aware approach (Prediction) on out-of-distribution geometries compared to the OpenFOAM simulation on *locally refined* meshes (Target). The model was trained on 3 750 geometries.

In each training step (forward and backward propagation), the physics-aware loss has to be computed once; note that our implementation has not been optimized, and we expect that lower computing times can be achieved. On average, one training step for one geometry takes about 50 ms when using only the physics-aware loss and 45 ms when using only the data-driven loss. This indicates an increase in the computing time of only roughly 10% when using the physics-aware loss. When using both the physics and the data loss, we require roughly 55 ms for one geometry.

10. Conclusion

We have introduced a novel physics-aware approach to train convolutional neural networks as surrogate models that relies exclusively on the physics modeling fluid behavior in multiple irregular geometries. Our approach does not rely on reference data and only requires the geometry image and boundary conditions. However, we have demonstrated that incorporating the physics-aware loss in the training process improves upon the data-based approach when reference data is available. This approach serves as an excellent surrogate model, with the evaluation being in the order of $O\left(10^5\right)$ times faster than a conventional CFD simulation. However, this speedup comes at the cost of having to create a large data set and a time-consuming process of training the model.

Our physics-aware approach performs well for low velocity geometries and demonstrates strong generalization capabilities. In contrast, the data-based approach struggles to generalize for the same low velocity geometries. Despite using a coarser resolution and finite differences, which may not be ideal for Navier–Stokes, our models achieve excellent predictions close to the reference solution for most geometries. However, for cases where our physics-aware models did not match the reference solution, even higher resolution finite volume methods failed to obtain a converged solution. Consequently, accurate predictions cannot be expected in such cases.

In future work, an extension to three-dimensional geometries is envisioned; see [26] for preliminary results.

CRediT authorship contribution statement

Viktor Grimm: Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Validation, Visualization, Writing – original draft, Writing – review & editing. **Alexander Heinlein:** Conceptualization, Supervision, Writing – original draft, Writing – review & editing, Investigation. **Axel Klawonn:** Conceptualization, Project administration, Supervision, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work was performed as part of the Helmholtz School for Data Science in Life, Earth and Energy (HDS-LEE) and received funding from the Helmholtz Association of German Research Centers. We gratefully acknowledge the use of the computational facilities of the Center for Data and Simulation Science (CDS) at the University of Cologne and of the Department of Mathematics and Computer Science of the Technische Universität Bergakademie Freiberg operated by the University Computing Center (URZ) and funded under grant application No. 100376434 to the State Ministry for Higher Education, Research and the Arts of the Federal State of Saxony (SMWK) on Artificial Intelligence and Robotics for GeoEnvironmental Modeling and Monitoring.

Data availability

The data that has been used can be directly produced using OpenFoam.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: large-scale machine learning on heterogeneous systems, https://www.tensorflow.org/, 2015, software available from tensorflow.org.

[2] R. Biswas, M.K. Sen, V. Das, T. Mukerji, Prestack and poststack inversion using a physics-guided convolutional neural network, Interpret. 7 (2019) SE161–SE174, https://doi.org/10.1190/INT-2018-0236.1.

[3] J. Blechschmidt, O.G. Ernst, Three ways to solve partial differential equations with neural networks — a review, GAMM-Mitt. 44 (2021) e202100006, https://doi.org/10.1002/gamm.202100006, https://onlinelibrary.wiley.com/doi/abs/10.1002/gamm.202100006. (Accessed 9 March 2023), _eprint, https://onlinelibrary.wiley.com/doi/pdf/10.1002/gamm.202100006.

[4] S. Cai, Z. Mao, Z. Wang, M. Yin, G.E. Karniadakis, Physics-informed neural networks (PINNs) for fluid mechanics: a review, https://doi.org/10.48550/arXiv.2105.09506, May 2021, http://arxiv.org/abs/2105.09506. (Accessed 13 January 2023), arXiv:2105.09506 [physics].

[5] S. Cuomo, V.S. di Cola, F. Giampaolo, G. Rozza, M. Raissi, F. Piccialli, Scientific machine learning through physics-informed neural networks: where we are and what's next, arXiv:2201.05624, 2022.

[6] S. Cuomo, V.S. Di Cola, F. Giampaolo, G. Rozza, M. Raissi, F. Piccialli, Scientific machine learning through physics–informed neural networks: where we are and what's next, J. Sci. Comput. 92 (2022) 88, https://doi.org/10.1007/s10915-022-01939-z. (Accessed 26 March 2024).

[7] M.W.M.G. Dissanayake, N. Phan-Thien, Neural-network-based approximations for solving partial differential equations, Commun. Numer. Methods Eng. 10 (1994) 195–201, https://doi.org/10.1002/cnm.1640100303, https://onlinelibrary.wiley.com/doi/abs/10.1002/cnm.1640100303. (Accessed 26 March 2024), _eprint, https://onlinelibrary.wiley.com/doi/pdf/10.1002/cnm.1640100303.

[8] W. E, B. Yu, The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems, Commun. Math. Stat. 6 (2018) 1–12, https://doi.org/10.1007/s40304-018-0127-z. (Accessed 20 February 2023).

[9] M. Eichinger, A. Heinlein, A. Klawonn, Stationary flow predictions using convolutional neural networks, in: F.J. Vermolen, C. Vuik (Eds.), Numerical Mathematics and Advanced Applications ENUMATH 2019, Springer International Publishing, Cham, 2021, pp. 541–549.

[10] M. Eichinger, A. Heinlein, A. Klawonn, Surrogate convolutional neural network models for steady computational fluid dynamics simulations, Electron. Trans. Numer. Anal. 56 (2022) 235–255, https://doi.org/10.1553/etna_vol56s235.

[11] Z. Fang, A high-efficient hybrid physics-informed neural networks based on convolutional neural network, IEEE Trans. Neural Netw. Learn. Syst. (2021) 1–13, https://doi.org/10.1109/TNNLS.2021.3070878.

[12] J. Ferziger, M. Peric, R. Street, Computational Methods for Fluid Dynamics, 4 ed., Springer, 2020.

[13] N.R. Franco, S. Fresca, A. Manzoni, P. Zunino, Approximation bounds for convolutional neural networks in operator learning, https://doi.org/10.48550/arXiv.2207.01546, Jan. 2023, http://arxiv.org/abs/2207.01546. (Accessed 8 March 2023), arXiv:2207.01546 [cs, math].

[14] S. Fresca, L. Dede', A. Manzoni, A comprehensive deep learning-based approach to reduced order modeling of nonlinear time-dependent parametrized PDEs, J. Sci. Comput. 87 (2021) 61, https://doi.org/10.1007/s10915-021-01462-7. (Accessed 8 March 2023).

[15] F.R.S. Karl Pearson, LIII. On lines and planes of closest fit to systems of points in space, Lond. Edinb. Dubl. Philos. Mag. J. Sci. 2 (1901) 559–572, https://doi.org/10.1080/14786440109462720, Publisher: Taylor & Francis.

[16] H. Gao, L. Sun, J. Wang, PhyGeoNet: physics-informed geometry-adaptive convolutional neural networks for solving parameterized steady-state pdes on irregular domain, J. Comput. Phys. 428 (2021) 110079, https://doi.org/10.1016/j.jcp.2020.110079.

[17] H. Gao, L. Sun, J.-X. Wang, Super-resolution and denoising of fluid flow using physics-informed convolutional neural networks without high-resolution labels, Phys. Fluids 33 (2021) 073603, https://doi.org/10.1063/5.0054312.

[18] C. Geuzaine, J.-F. Remacle, Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities, Int. J. Numer. Methods Eng. 79 (2009) 1309–1331, https://doi.org/10.1002/nme.2579.

[19] X. Glorot, A. Bordes, Y. Bengio, Deep sparse rectifier neural networks, in: G. Gordon, D. Dunson, M. Dudík (Eds.), Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, Fort Lauderdale, FL, USA, 11–13 Apr 2011, in: Proceedings of Machine Learning Research, vol. 15, PMLR, 2011, pp. 315–323, https://proceedings.mlr.press/v15/glorot11a.html.

[20] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016.

[21] M. Gori, G. Monfardini, F. Scarselli, A new model for learning in graph domains, in: Proceedings. 2005 IEEE International Joint Conference on Neural Networks, vol. 2, July 2005, 2005, pp. 729–734, ISSN: 2161-4407, https://ieeexplore.ieee.org/document/1555942. (Accessed 17 May 2024).

[22] S. Goswami, A. Bora, Y. Yu, G.E. Karniadakis, Physics-informed deep neural operator networks, https://doi.org/10.48550/arXiv.2207.05748, July 2022, http://arxiv.org/abs/2207.05748. (Accessed 7 July 2023), arXiv:2207.05748 [cs, math].

[23] P.M. Gresho, Incompressible fluid dynamics: some fundamental formulation issues, Annu. Rev. Fluid Mech. 23 (1991) 413–453.

[24] V. Grimm, A. Heinlein, A. Klawonn, A short note on solving partial differential equations using convolutional neural networks, http://www.uni-koeln.de/, Nov. 2022. (Accessed 17 December 2022), Num Pages: 9 Publisher: Universität zu Köln Volume: 2022-07.

[25] V. Grimm, A. Heinlein, A. Klawonn, M. Lanser, J. Weber, Estimating the Time-Dependent Contact Rate of SIR and SEIR Models in Mathematical Epidemiology Using Physics-Informed Neural Networks, Verlag der Österreichischen Akademie der Wissenschaften, 2021, pp. 1–27.
[26] V.H. Grimm, Physics-Aware Convolutional Neural Networks for Computational Fluid Dynamics, doctoral thesis, Universität zu Köln, Aug. 2023, http://www.uni-koeln.de/. (Accessed 7 July 2024).
[27] X. Guo, W. Li, F. Iorio, Convolutional neural networks for steady flow approximation, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16, New York, NY, USA, in: Association for Computing Machinery, 2016, pp. 481–490.
[28] O.J. Hénaff, E.P. Simoncelli, Geodesics of learned representations, arXiv:1511.06394, 2016.
[29] G.E. Karniadakis, I.G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, L. Yang, Physics-informed machine learning, Nat. Rev. Phys. 3 (2021) 422–440, https://doi.org/10.1038/s42254-021-00314-5, https://www.nature.com/articles/s42254-021-00314-5. (Accessed 13 January 2023), Number: 6, Publisher: Nature Publishing Group.
[30] D. Kelshaw, G. Rigas, L. Magri, Physics-informed CNNs for super-resolution of sparse observations on dynamical systems, arXiv preprint, arXiv:2210.17319, 2022.
[31] M. Kemna, A. Heinlein, C. Vuik, Reduced order fluid modeling with generative adversarial networks, Proc. Appl. Math. Mech. 23 (2023) e202200241, https://doi.org/10.1002/pamm.202200241.
[32] E. Kharazmi, Z. Zhang, G.E.M. Karniadakis, hp-VPINNs: variational physics-informed neural networks with domain decomposition, Comput. Methods Appl. Mech. Eng. 374 (2021) 113547, https://doi.org/10.1016/j.cma.2020.113547, https://mathscinet.ams.org/mathscinet-getitem?mr=4183777. (Accessed 17 December 2022).
[33] D.P. Kingma, J. Ba, Adam: a method for stochastic optimization, arXiv preprint, arXiv:1412.6980, 2014.
[34] T.N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, arXiv preprint, arXiv:1609.02907, 2016.
[35] N. Kovachki, Z. Li, B. Liu, K. Azizzadenesheli, K. Bhattacharya, A. Stuart, A. Anandkumar, Neural operator: learning maps between function spaces, https://doi.org/10.48550/arXiv.2108.08481, Oct. 2022, http://arxiv.org/abs/2108.08481. (Accessed 20 February 2023), arXiv:2108.08481 [cs, math].
[36] I. Lagaris, A. Likas, D. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, IEEE Trans. Neural Netw. 9 (1998) 987–1000, https://doi.org/10.1109/72.712178, Conference name: IEEE Transactions on Neural Networks.
[37] T. Lassila, A. Manzoni, A. Quarteroni, G. Rozza, Model order reduction in fluid dynamics: challenges and perspectives, in: Reduced Order Methods for Modeling and Computational Reduction, 2014, pp. 235–273.
[38] Y. LeCun, et al., Generalization and network design strategies, in: Connectionism in Perspective, vol. 19, 1989, p. 18.
[39] K. Lee, K. Carlberg, Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders, https://doi.org/10.48550/arXiv.1812.08373, June 2019, http://arxiv.org/abs/1812.08373. (Accessed 8 March 2023), arXiv:1812.08373 [cs].
[40] R.J. LeVeque, Finite Difference Methods for Ordinary and Partial Differential Equations, Society for Industrial and Applied Mathematics, 2007, https://epubs.siam.org/doi/abs/10.1137/1.9780898717839, https://arxiv.org/abs/https://epubs.siam.org/doi/pdf/10.1137/1.9780898717839.
[41] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, A. Anandkumar, Fourier neural operator for parametric partial differential equations, https://doi.org/10.48550/arXiv.2010.08895, May 2021, arXiv:2010.08895 [cs, math].
[42] Z. Long, Y. Lu, B. Dong, Pde-net 2.0: learning pdes from data with a numeric-symbolic hybrid deep network, J. Comput. Phys. 399 (2019) 108925, https://doi.org/10.1016/j.jcp.2019.108925, https://www.sciencedirect.com/science/article/pii/S0021999119306308.
[43] Z. Long, Y. Lu, X. Ma, B. Dong, PDE-Net: learning PDEs from data, in: Proceedings of the 35th International Conference on Machine Learning, PMLR, July 2018, pp. 3208–3216, https://proceedings.mlr.press/v80/long18a.html, ISSN: 2640-3498.
[44] L. Lu, P. Jin, G.E. Karniadakis, DeepONet: learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators, Nat. Mach. Intell. 3 (2021) 218–229, https://doi.org/10.1038/s42256-021-00302-5, arXiv:1910.03193 [cs, stat].
[45] R. Maulik, B. Lusch, P. Balaprakash, Reduced-order modeling of advection-dominated systems with recurrent neural networks and convolutional autoencoders, Phys. Fluids 33 (2021) 037106, https://doi.org/10.1063/5.0039986, Publisher: American Institute of Physics, https://aip-scitation-org.tudelft.idm.oclc.org/doi/10.1063/5.0039986. (Accessed 8 March 2023).
[46] J.M. McDonough, Lectures in computational fluid dynamics of incompressible flow: mathematics, algorithms and implementations, 2007.
[47] C. Meng, S. Seo, D. Cao, S. Griesemer, Y. Liu, When physics meets machine learning: a survey of physics-informed machine learning, arXiv preprint, arXiv:2203.16797, 2022.
[48] A. Odena, V. Dumoulin, C. Olah, Deconvolution and Checkerboard Artifacts, Distill, 2016.
[49] S.V. Patankar, D.B. Spalding, A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows, in: Numerical Prediction of Flow, Heat Transfer, Turbulence and Combustion, Elsevier, 1983, pp. 54–73.
[50] A. Quarteroni, A. Manzoni, F. Negri, Reduced Basis Methods for Partial Differential Equations, Unitext, vol. 92, Springer, Cham, 2016, https://mathscinet.ams.org/mathscinet-getitem?mr=3379913. (Accessed 17 December 2022).
[51] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics informed deep learning (part I): data-driven solutions of nonlinear partial differential equations, arXiv preprint, arXiv:1711.10561, 2017.
[52] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. 378 (2019) 686–707, https://doi.org/10.1016/j.jcp.2018.10.045.
[53] P. Ramachandran, B. Zoph, Q. Le, Searching for activation functions, arXiv:1511.06394, 2017.
[54] B. Raonić, R. Molinaro, T. De Ryck, T. Rohner, F. Bartolucci, R. Alaifari, S. Mishra, E. de Bézenac, Convolutional neural operators for robust and accurate learning of PDEs, https://doi.org/10.48550/arXiv.2302.01178, http://arxiv.org/abs/2302.01178. (Accessed 1 December 2023).
[55] M. Rathinam, L.R. Petzold, A new look at proper orthogonal decomposition, SIAM J. Numer. Anal. 41 (2003) 1893–1925, https://doi.org/10.1137/S0036142901389049, http://epubs.siam.org/doi/10.1137/S0036142901389049. (Accessed 9 March 2023).
[56] S. Rebay, Efficient unstructured mesh generation by means of Delaunay triangulation and Bowyer-Watson algorithm, J. Comput. Phys. 106 (1993) 125–138.
[57] P. Ren, C. Rao, Y. Liu, J. Wang, H. Sun, PhyCRNet: physics-informed convolutional-recurrent network for solving spatiotemporal pdes, arXiv:2106.14103, 2021.
[58] O. Ronneberger, P. Fischer, T. Brox, U-Net: convolutional networks for biomedical image segmentation, in: N. Navab, J. Hornegger, W.M. Wells, A.F. Frangi (Eds.), Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015, Springer International Publishing, Cham, 2015, pp. 234–241.
[59] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning representations by back-propagating errors, Nature 323 (1986) 533–536.
[60] M. Sadoughi, C. Hu, Physics-based convolutional neural network for fault diagnosis of rolling element bearings, IEEE Sens. J. 19 (2019) 4181–4192, https://doi.org/10.1109/JSEN.2019.2898634.
[61] R. Sharma, A.B. Farimani, J. Gomes, P. Eastman, V. Pande, Weakly-supervised learning of heat transport via physics informed loss, arXiv:1807.11374, 2018.
[62] G.D. Smith, G.D. Smith, Numerical Solution of Partial Differential Equations: Finite Difference Methods, Oxford University Press, 1985.
[63] M. Stender, J. Ohlsen, H. Geisler, A. Chabchoub, N. Hoffmann, A. Schlaefer, $$\hbox {U}ˆp$$-Net: a generic deep learning-based time stepper for parameterized spatio-temporal dynamics, Comput. Mech. 71 (2023) 1227–1249, https://doi.org/10.1007/s00466-023-02295-x.
[64] J.C. Strikwerda, Finite Difference Schemes and Partial Differential Equations, SIAM, 2004.
[65] L. Sun, H. Gao, S. Pan, J. Wang, Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data, Comput. Methods Appl. Mech. Eng. 361 (2020), https://doi.org/10.1016/j.cma.2019.112732.
[66] Y. Sun, C. Moya, G. Lin, M. Yue, DeepGraphONet: a deep graph operator network to learn and zero-shot transfer the dynamic response of networked systems, https://doi.org/10.48550/arXiv.2209.10622, Sept. 2022, http://arxiv.org/abs/2209.10622. (Accessed 30 April 2024), arXiv:2209.10622 [cs].

[67] The OpenFOAM Foundation, OpenFOAM v8 user guide, https://cfd.direct/openfoam/user-guide-v8.
[68] S. Wang, X. Yu, P. Perdikaris, When and why PINNs fail to train: a neural tangent kernel perspective, J. Comput. Phys. 449 (2022) 110768, https://doi.org/10.1016/j.jcp.2021.110768.
[69] J. Willard, X. Jia, S. Xu, M. Steinbach, V. Kumar, Integrating scientific knowledge with machine learning for engineering and environmental systems, arXiv:2003.04919, 2021.
[70] J. Wu, X. Yin, H. Xiao, Seeing permeability from images: fast prediction with convolutional neural networks, Sci. Bull. 63 (2018) 1215–1222, https://doi.org/10.1016/j.scib.2018.08.006.
[71] R. Zhang, Y. Liu, H. Sun, Physics-guided convolutional neural network (PhyCNN) for data-driven seismic response modeling, Eng. Struct. 215 (2020) 110704, https://doi.org/10.1016/j.engstruct.2020.110704.